



Instituto Federal de Educação, Ciência e Tecnologia do Ceará
Programa de Pós-Graduação em Engenharia de Telecomunicações

Antônio Sávio Silva Oliveira

**Análise comparativa de métodos de localização
para robôs móveis autônomos utilizando o
Robot Operating System 2**

Fortaleza - CE

2023

Antônio Sávio Silva Oliveira

Análise comparativa de métodos de localização para robôs móveis autônomos utilizando o Robot Operating System 2

Dissertação apresentada como requisito parcial para obtenção do título de Mestre em Engenharia de Telecomunicações, ao Programa de Pós-Graduação em Engenharia de Telecomunicações, do Instituto Federal de Educação, Ciência e Tecnologia do Ceará.

Programa de Pós-Graduação em Engenharia de Telecomunicações

Instituto Federal de Educação, Ciência e Tecnologia do Ceará

Orientador: Prof. Dr. Auzuir Ripardo de Alexandria

Fortaleza - CE

2023

Antônio Sávio Silva Oliveira

**Análise comparativa de métodos de localização para robôs móveis autônomos
utilizando o Robot Operating System 2**

Dissertação apresentada como requisito parcial para obtenção do título de Mestre em Engenharia de Telecomunicações, ao Programa de Pós-Graduação em Engenharia de Telecomunicações, do Instituto Federal de Educação, Ciência e Tecnologia do Ceará.

Aprovado em 09/02/2023

BANCA EXAMINADORA

Prof. Dr. Auzuir Ripardo de Alexandria
(Orientador)
Instituto Federal de Educação, Ciência e
Tecnologia do Ceará - IFCE

Prof. Dr. Francisco José Alves de Aquino
Instituto Federal de Educação, Ciência e
Tecnologia do Ceará - IFCE

Profa. Dra. Jéssyca Almeida Bessa
Instituto Federal de Educação, Ciência e
Tecnologia do Ceará - IFCE Maranguape

Dedico este trabalho, primeiramente, a Deus e aos meus pais, pelo total apoio, dedicação e incentivo em todas as etapas da vida.

Agradecimentos

Primeiro agradeço a Deus por todas as oportunidades a mim dadas;

Aos meus pais, Luiz e Maria, pelo contínuo incentivo;

A minha esposa, Shirley, por sua paciência e cumplicidade;

Ao meu orientador, Auzuir, pela confiança, ensinamentos e apoio;

Ao IFCE, Instituto Federal de Educação, Ciência e Tecnologia do Ceará, por todo o repasse de conhecimento;

Aos amigos e professores do curso de Pós Graduação em Engenharia de Telecomunicações, por todas as lições aprendidas;

"Aprendizagem nunca esgota a mente."

(Leonardo da Vinci)

Resumo

A localização é um requisito básico de um robô móvel. Para poder navegar de forma autônoma e exercer suas atividades ele deve estimar com precisão sua posição no ambiente ao longo do tempo. O objetivo deste trabalho consiste em analisar e comparar os métodos de localização de robôs móveis escolhidos durante a revisão de literatura: Localização de Monte Carlo, AMCL (Localização de Monte Carlo Adaptativa), Fusão de sensores e o método combinado de AMCL com fusão de sensores, utilizando o Robot Operating System (ROS). O trabalho foi desenvolvido no simulador Webots, em que os algoritmos foram implementados utilizando o ROS 2 e a linguagem Python. Cada método foi avaliado de modo a obter informações de eficiência (com relação a precisão da localização) e desempenho (com relação ao consumo de *hardware*). O método que obteve o melhor desempenho com relação aos erros de posição foi o método combinado de fusão de sensores com AMCL apresentando erro médio RMSE de 4 a 5 cm e pontuação no R2 Square de 95,10% a 99,79% no cenário 01 e de 68,20% a 89,65% no cenário 02, e em segundo lugar o método de fusão de sensores com erro médio de 4 a 7 cm e pontuação no R2 Square de 94,22% a 99,69% no cenário 01 e 52,27% a 81,42% no cenário 02. Com relação ao consumo de *hardware* o melhor método foi o de fusão de sensores consumindo cerca de 17% de CPU e 32 MB de RAM e em segundo lugar ficou o AMCL com 21% de CPU e 40 MB de RAM. Dentre as principais contribuições desse trabalho destacam-se: i. A utilização de diversas técnicas para o problema de localização, aplicadas em cenários específicos de modo a possibilitar um estudo comparativo entre elas. ii. A utilização do ROS e a disponibilização dos algoritmos e resultados em um repositório do GitHub, que pode ser utilizado para estudos do simulador Webots, do ROS2 e de técnicas de localização em robótica.

Palavras-chave: Sistema Operacional de Robôs. Localização. Robôs Móveis.

Abstract

Localization is a basic requirement of a mobile robot. To be able to navigate autonomously and carry out its activities, it must accurately estimate its position in the environment over time. The objective of this work is to analyze and compare the mobile robot localization methods chosen during the literature review: Monte Carlo Localization, AMCL (Adaptive Monte Carlo Localization), Sensor Fusion and the combined method of AMCL with Sensor Fusion, using the Robot Operating System (ROS). The work was developed in the Webots simulator, in which the algorithms were implemented using ROS 2 and the Python language. Each method was evaluated in order to obtain information on efficiency (with regard to location accuracy) and performance (with regard to *hardware* consumption). The method that obtained the best performance regarding position errors was the combined method of sensor fusion with AMCL, presenting a mean RMSE error of 4 to 5 cm and a R2 Square score of 95.10% to 99.79% in the scenario 01 and from 68.20% to 89.65% in scenario 02, and secondly the sensor fusion method with an average error of 4 to 7 cm and a score in R2 Square of 94.22% to 99.69% in scenario 01 and 52.27% to 81.42% in scenario 02. Regarding *hardware* consumption, the best method was sensor fusion consuming about 17% of CPU and 32 MB of RAM and in second place was AMCL with 21% of CPU and 40 MB of RAM. Among the main contributions of this work are: i. The use of different techniques for the location problem, applied in specific scenarios in order to allow a comparative study between them. ii. The use of ROS and the availability of algorithms and results in a GitHub repository, which can be used for studies of the Webots simulator, ROS2 and localization techniques in robotics.

Keywords: Robot Operating System. Localization. Mobile Robots.

Lista de ilustrações

Figura 1 – Diagrama de módulos componentes de um robô móvel autônomo. . . .	18
Figura 2 – Arquitetura baseada em mapas	21
Figura 3 – Esquema geral de localização para robôs móveis	22
Figura 4 – Representação de crença de posição	24
Figura 5 – O Problema de SLAM	25
Figura 6 – Rede de Nós	26
Figura 7 – Rede de Nós se comunicando por tópicos	27
Figura 8 – Serviços ROS	28
Figura 9 – Ações ROS	29
Figura 10 – Epuck Mobile Robot	37
Figura 11 – Cenário 01	39
Figura 12 – Mapa relativo ao cenário 01	39
Figura 13 – Cenário 02	40
Figura 14 – Mapa relativo ao cenário 02	40
Figura 15 – Gráfico ROS da aplicação	46
Figura 16 – Monte Carlo Cenário 01	47
Figura 17 – Comparativo entre Localização de Monte Carlo e Odometria	48
Figura 18 – Monte Carlo Cenário 02	49
Figura 19 – Erros de posição da Localização de Monte Carlo Cenário no 01	50
Figura 20 – Erros de posição da Localização de Monte Carlo Cenário no 02	51
Figura 21 – Consumo de Memória e CPU - Localização de Monte Carlo	51
Figura 22 – Erros de posição do método de fusão de sensores no Cenário 01	52
Figura 23 – Erros de posição do método de fusão de sensores no Cenário 02	53
Figura 24 – Consumo de Memória e CPU - Fusão de Sensores com EKF	53
Figura 25 – Erros de posição do método AMCL no Cenário 01	54
Figura 26 – Erros de posição do método AMCL no Cenário 02	55
Figura 27 – Consumo de Memória e CPU - AMCL	55
Figura 28 – Consumo de Memória e CPU - Método combinado de AMCL e Fusão de Sensores com EKF	56
Figura 29 – Erros de posição do método combinado de AMCL e fusão de sensores no Cenário 01	57
Figura 30 – Erros de posição do método combinado de AMCL e fusão de sensores no Cenário 02	58
Figura 31 – Processo de identificação, triagem e seleção de estudos para revisão sistemática	67
Figura 32 – Lista de palavras-chave mais usadas nos artigos	68

Lista de tabelas

Tabela 1 – Principais Métodos de Localização	30
Tabela 2 – Erros de posição RMSE - Método MCL	50
Tabela 3 – Erros de posição RMSE - Método MCL com 2000 partículas	51
Tabela 4 – Erros de posição RMSE - Método Fusão de Sensores com EKF	52
Tabela 5 – Erros de posição RMSE - Método AMCL	54
Tabela 6 – Erros de posição RMSE - Método combinado de AMCL com Fusão de Sensores	56
Tabela 7 – Desempenho dos métodos de localização no R2 Square	59
Tabela 8 – Resumo métodos de localização	59

Lista de abreviaturas e siglas

GPS	<i>Global Positioning System</i>
SLAM	<i>Simultaneous Localization And Mapping</i>
ROS	<i>Robot Operating System</i>
CSV	<i>Comma-Separated Values</i>
MCL	<i>Monte Carlo Localization</i>
AMCL	<i>Adaptive Monte Carlo Localization</i>
CPU	<i>Central Processing Unit</i>
RAM	<i>Random Access Memory</i>
EKF	<i>Extended Kalman Filter</i>
UKF	<i>Unscented Kalman Filter</i>
IMU	<i>Inertial Measurement Unit</i>
RMSE	<i>Root Mean Squared Error</i>

Sumário

1	INTRODUÇÃO	14
1.1	Objetivos	15
1.1.1	Objetivos Específicos	15
1.2	Contribuições	15
1.3	Produção Científica	16
1.4	Organização do Trabalho	16
2	FUNDAMENTAÇÃO TEÓRICA	17
2.1	Robótica Móvel	17
2.2	Localização	19
2.2.1	Localização baseada em mapa	20
2.2.2	Representação de crença	23
2.2.3	Localização e Mapeamento simultâneo	25
2.3	Robot Operating System	25
2.3.1	Nós	26
2.3.2	Serviços	27
2.3.3	Ações	28
2.4	Métodos de Localização utilizando o ROS	29
2.4.1	Trabalhos Seleccionados	32
2.5	Considerações Finais	35
3	METODOLOGIA	36
3.1	Robô Móvel Utilizado	36
3.2	Ambiente para testes	38
3.3	Localização de Monte Carlo	40
3.3.1	Modelo de movimento	41
3.3.2	Modelo de medição	42
3.3.3	Filtro de Partículas	42
3.4	Fusão de Sensores	43
3.5	Nav2 AMCL	43
3.6	Método Combinado de AMCL com Fusão de Sensores EKF	44
3.7	Análise de desempenho	44
3.8	Considerações Finais	45
4	RESULTADOS E DISCUSSÕES	46
4.1	Localização de Monte Carlo	46

4.2	Fusão de Sensores	52
4.3	AMCL	54
4.4	Método combinado de AMCL com Fusão de Sensores	56
5	CONSIDERAÇÕES FINAIS E TRABALHOS FUTUROS	60
	REFERÊNCIAS	61
6	APÊNDICE	66
6.1	Método utilizado para a revisão de literatura	66
6.2	Instalação do ROS	68
6.3	ROS 2 Navigation Stack	69
6.4	Webots ROS2 Interface	70
6.5	Robot_Localization	71
6.6	PlotJuggler	71

1 Introdução

A constante modernização da indústria e de suas técnicas de produção aliadas as tecnologias da indústria 4.0¹ e principalmente questões competitivas mostram a necessidade de adaptação das fábricas e dos meios de produção ditos “tradicionais” para métodos confiáveis e mais econômicos (ODEBRECHT et al., 2020). Diante disso, pode-se refletir sobre as vantagens na substituição dos equipamentos e robôs tradicionais que são grandes, fixos e dependentes de um operador por robôs móveis modernos.

Segundo Ferreira (2020), os robôs móveis são mais versáteis porque não precisam estar fixados a unidades de trabalho e podem ser usados para tarefas que não possuem restrições geográficas. Os robôs movimentam-se por pernas ou rodas, podem ser autônomos ou não, é possível programá-los em baixo ou alto nível e sentem o ambiente ao seu redor por meio de sensores.

De acordo com Pio, Castro e Castro Júnior (2006), pode-se definir um robô móvel como sendo um agente autônomo capaz de extrair informações do ambiente e utilizar esse conhecimento do mundo para deslocar-se com segurança de modo significativo e intencional, atuando e executando tarefas. Para que o robô consiga executar essas tarefas é necessário o uso de sensores (dispositivos que permitem ao robô coletar as informações do ambiente), pode-se fazer uma analogia com o ser humano e dizer que o sensor para o robô é como um dos sentidos para o humano (visão, audição, olfato, etc.), em outras palavras, é através do sensor que o robô consegue captar as informações do ambiente.

Algo essencial para um robô móvel é a capacidade de se localizar no ambiente, sempre que for necessário se locomover de um ponto a outro, o robô deve ser capaz de detectar obstáculos, desviar-se deles e cumprir o seu objetivo. Se o ambiente for conhecido, esse problema pode ser resumido como um planejamento de trajetória, já se o ambiente for desconhecido, o robô deve agir reativamente ao que for detectado por meio dos sensores (MARCHI et al., 2001).

Segundo Cousins (2010) devido a complexidade dos robôs móveis autônomos modernos, é necessário simplificar a integração de sistemas e possuir um conjunto de ferramentas que consigam gerenciar essa complexidade para facilitar o desenvolvimento do robô. O ROS (*Robot Operating System*) fornece funcionalidade para abstração de *hardware*, *drivers* de dispositivo, comunicação entre processos em várias máquinas, ferramentas para teste e visualização, entre outras. Além disso, uma aplicação para robô móvel utilizando o

¹ A Indústria 4.0 é um conceito que representa a automação industrial e a integração de diferentes tecnologias como inteligência artificial, robótica, internet das coisas e computação em nuvem com o objetivo de promover a digitalização das atividades industriais melhorando os processos e aumentando a produtividade.

ROS pode ser facilmente adaptada para trabalhar com outros tipos de robôs.

Diante disso, essa dissertação visa realizar uma revisão da literatura existente sobre os métodos utilizados para localização de robôs móveis com o ROS, implementar alguns em um simulador para obter resultados que possam descrever o desempenho deles de maneira quantitativa e qualitativamente e comparar os resultados obtidos para fornecer uma visão geral para os novos pesquisadores da área e servir como material auxiliar no ensino da robótica móvel.

1.1 Objetivos

O objetivo geral deste trabalho é analisar e comparar métodos de localização de robôs móveis utilizando o ROS.

1.1.1 Objetivos Específicos

Especificamente, este trabalho busca os seguintes objetivos:

- Identificar os métodos de localização mais utilizados com o ROS através de uma revisão de literatura e selecionar 03 (três) deles para estudo mais aprofundado.
- Combinar dois métodos visando melhorar a precisão de posição.
- Implementar as técnicas selecionadas utilizando o ROS em um simulador.
- Analisar os resultados obtidos na implementação dessas técnicas.

1.2 Contribuições

Destacam-se como principais contribuições:

- A utilização de diferentes técnicas de localização para robôs móveis, aplicadas em 02 (dois) cenários distintos (Figuras 11 e 13) de modo a possibilitar um estudo comparativo entre elas.
- A utilização do ROS, que é um *middleware* para robótica móvel muito utilizado devido a sua modularidade e capacidade de abstrair um problema complexo em etapas menores e realizar a comunicação entre os diferentes sensores do robô.
- A disponibilização dos algoritmos e resultados em um repositório do GitHub ², que pode ser utilizada para estudos do simulador utilizado, do ROS2 e de técnicas de localização em robótica.

² <https://github.com/savio-dev/analise_localizacao_ros>

1.3 Produção Científica

Em termos de produção científica, durante a realização deste trabalho, foi publicado o seguinte artigo:

OLIVEIRA, A. S. S.; REIS, M. C. dos; MOTA, F. A. X. da; MARTINEZ, M. E. M.; ALEXANDRIA, A. R. New trends on computer vision applied to mobile robot localization. *Internet of Things and Cyber-Physical Systems*, v. 2, p. 63–69, 2022. ISSN 2667-3452. Disponível em: <<https://www.sciencedirect.com/science/article/pii/S2667345222000128>>

1.4 Organização do Trabalho

Este trabalho está estruturado da seguinte forma:

- No Capítulo 2, a fundamentação teórica das técnicas utilizadas é descrita. São abordados conceitos de robótica móvel, ROS e as principais técnicas de localização de robôs móveis.
- No Capítulo 3, são apresentadas as etapas adotadas que compõem a metodologia proposta para este trabalho.
- No Capítulo 4, os resultados obtidos são descritos e discussões são realizadas em relação aos experimentos.
- No Capítulo 5, são apresentadas, as considerações finais sobre os resultados e trabalhos futuros.
- No Apêndice constam: O método utilizado para a revisão de literatura, a instalação do ROS, da interface do Webots_ROS 2, do pacote robot_localization, do PlotJuggler e do Nav2.

2 Fundamentação Teórica

Neste capítulo são apresentados alguns conceitos como Robótica Móvel, ROS e as principais técnicas de localização utilizadas para robôs móveis.

2.1 Robótica Móvel

Um robô móvel autônomo é um sistema que opera em um ambiente parcialmente desconhecido ou não estruturado. Isto significa que o robô deve conseguir navegar mesmo na presença de ruídos e evitar os possíveis obstáculos ([ALATISE; HANCKE, 2020](#)). Os robôs podem sentir o ambiente e tomar decisões que lhes permitam realizar suas tarefas de forma eficiente, eficaz e segura. Devido a essas características, a implantação desses robôs ocorreu em vários campos como residências pessoais, indústrias, hospitais, escolas, exploração de terras desconhecidas, operações de resgate e muitas outras áreas ([NILOY et al., 2021](#)).

Podem-se classificar as arquiteturas de robôs móveis de acordo com algumas características como estrutura (centralizadas e distribuídas), raciocínio (reativa, deliberativa e híbrida), decomposição e encapsulamento (baseadas em comportamento e baseadas em módulos funcionais) ([NAKHAEINIA et al., 2011](#)).

Em robôs móveis com arquitetura centralizada, todas as decisões são feitas em âmbito local, ou seja, para tomar qualquer decisão serão analisados os dados dos sensores do próprio robô, ele age de forma autônoma baseado única e exclusivamente nas informações colhidas durante sua incursão no ambiente ([TZAFESTAS, 2018](#)).

Quando o robô não age de forma isolada no ambiente, temos a ideia de distribuição, pois nesse caso o robô faz parte de uma sociedade de agentes onde cada um tem um papel a ser exercido. Existem vários motivos para distribuir sistemas inteligentes como melhoria na adaptabilidade, confiabilidade e autonomia do sistema, reduzir custos, aumentar a eficiência e permitir a integração desses sistemas ([SHAO; ZHANG; ZHANG, 2022](#)).

Entende-se por reação a forma como alguém ou algo se comporta diante de um determinado estímulo e por reativo aquilo que produz uma reação. De acordo com [Ferreira e Girardi \(2002\)](#), uma arquitetura reativa é composta por uma série de comportamentos que relacionam certas condições sensoriais a um conjunto de ações do robô, além de ser dotada de métodos que garantam a devida coordenação destes comportamentos

Planejar (ou deliberar) uma tarefa consiste em determinar as ações e sequências necessárias para conseguir sua execução no contexto de um certo modelo de mundo.

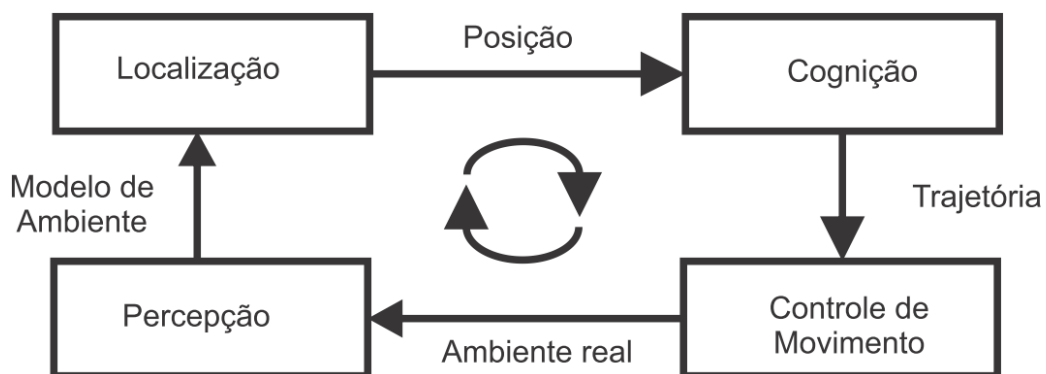
Segundo [Pieri \(2002\)](#), em arquiteturas baseadas em módulos funcionais as respostas comportamentais não estão explícitas na arquitetura, mas são criadas pelo planejador de acordo com o modelo de mundo particular que foi construído a partir dos dados sensoriais. Em sistemas baseados em comportamento dá-se o inverso, ou seja, as respostas devem estar explícitas na arquitetura.

De acordo com [Santos et al. \(2017\)](#), na arquitetura de subsunção os módulos são agrupados em camadas de competência que reflete uma hierárquica de inteligência, onde os níveis mais baixos representam funções de “sobrevivência” e os níveis mais altos representam funções dirigidas aos objetivos da tarefa. Essa arquitetura possui um mecanismo de prioridade de comportamentos onde o nível mais alto pode suprimir ou inibir o nível mais baixo.

De acordo com [Silva, Roche e Kondoz \(2018\)](#), um robô/veículo móvel autônomo típico é composto por três componentes tecnológicos principais: um sistema de mapeamento que é responsável por detectar e entender os objetos ao redor ambiente; um sistema de localização com o qual o robô conhece sua localização atual a qualquer tempo, e o terceiro componente responsável pela tomada de decisão. Outros autores como [Panigrahi e Bisoy \(2021\)](#) e [Siegwart, Nourbakhsh e Scaramuzza \(2011\)](#) consideram quatro componentes: percepção, localização, cognição e controle de movimento.

No módulo da percepção o robô extrai dados significativos interpretando seus sensores. Na fase de localização, o robô estima sua localização atual no ambiente por meio das informações obtidas pelos sensores. No módulo de cognição, o robô deve realizar algum tipo de cálculo para planejar a rota que ele deve percorrer, sendo que a rota escolhida precisa evitar os obstáculos no meio do caminho. A fase de controle de movimento permite que o robô atinja sua trajetória desejada modificando suas saídas do motor. Esse modelo com quatro módulos está representado na Figura 1.

Figura 1 – Diagrama de módulos componentes de um robô móvel autônomo.



Fonte: [Bessa \(2015\)](#).

Pode-se observar a partir da Figura 1 que essas quatro fases não ocorrem apenas uma única vez, mas sim de forma cíclica, enquanto o robô estiver em funcionamento sempre que terminar suas ações planejadas ele retorna para a fase de percepção, em que capta as informações do ambiente e pode utilizá-las para planejar seus próximos passos ou verificar se já atingiu seu objetivo. Pode-se considerar essas duas visões como equivalentes, sendo que a única diferença é o nível de detalhamento de cada fase. Percebe-se que o robô móvel precisa constantemente saber a sua localização atual para conseguir navegar com precisão no ambiente e realizar o trajeto planejado. Segundo Niloy et al. (2021), a navegação oferece os desafios mais difíceis para os robôs móveis. Esta pode ser descrita em quatro etapas:

- I. Modelagem Ambiental (Mapeamento);
- II. Localização;
- III. Planejamento de caminho;
- IV. Evitar obstáculos.

2.2 Localização

A localização é um requisito básico de um robô móvel. Para poder navegar de forma autônoma e exercer suas atividades ele deve compreender com precisão sua posição no ambiente ao longo do tempo (DOURADO JUNIOR et al., 2019). O desafio da localização consiste em estimar a posição e orientação do robô por meio das informações adquiridas pelos sensores e outros sistemas (ALATISE; HANCKE, 2020).

Existem muitas estratégias para resolver ou até mesmo evitar esse problema de localização. Por exemplo, segundo Siegwart, Nourbakhsh e Scaramuzza (2011), com a utilização de uma solução pré-definida, pode-se fazer um robô ir de um ponto A para um ponto B sem que o robô precise saber da sua posição atual. Porém esse tipo de solução é altamente dependente do ambiente, ou seja, somente se aplica a um determinado ambiente e caso seja necessário mudar o robô de local é preciso realizar toda a programação novamente, o que torna inviável para boa parte das aplicações de robôs móveis.

Campbell et al. (2020) apresentam algumas abordagens comuns para localização de robôs, dentre as quais pode-se destacar:

- I. GPS, que funciona por meio de sistemas de navegação por satélite;
- II. Odometria, que calcula a posição do robô por meio do uso de um codificador que mede a rotação das rodas;
- III. Sistema de navegação inercial (INS), que funciona de maneira semelhante a odometria, possui a vantagem de não necessitar nenhum ponto de referencia inicial, porém,

não é adequado para realizar a localização durante grandes períodos devido aos erros de acumulação, sendo frequentemente utilizado para complementar outros métodos de localização como sistemas por visão ou GPS;

IV. Localização baseada em Mapa.

De acordo com [Siegwart, Nourbakhsh e Scaramuzza \(2011\)](#), caso o sistema de GPS (*Global Positioning System*) possuísse uma precisão na faixa de centímetros ou milímetros, boa parte do problema de localização seria evitado. O GPS informaria ao robô sua posição exata, dentro e fora de casa e a resposta da pergunta "onde estou?" estaria sempre disponível. Entretanto, a precisão atualmente é de vários metros e isso é algo inviável para robôs móveis, em escala humana ou em miniatura.

Além disso, o GPS não funciona em ambientes fechados ou em áreas obstruídas devido aos potenciais obstáculos que podem bloquear a ligação com os satélites. Mas, se analisasse além dessas limitações observa-se que a localização do robô implica mais do que apenas saber a posição absoluta de alguém na Terra. Considerando um robô que está interagindo com humanos, ele precisaria identificar sua posição absoluta, mas sua posição relativa aos humanos, outros robôs e objetos, é igualmente importante.

2.2.1 Localização baseada em mapa

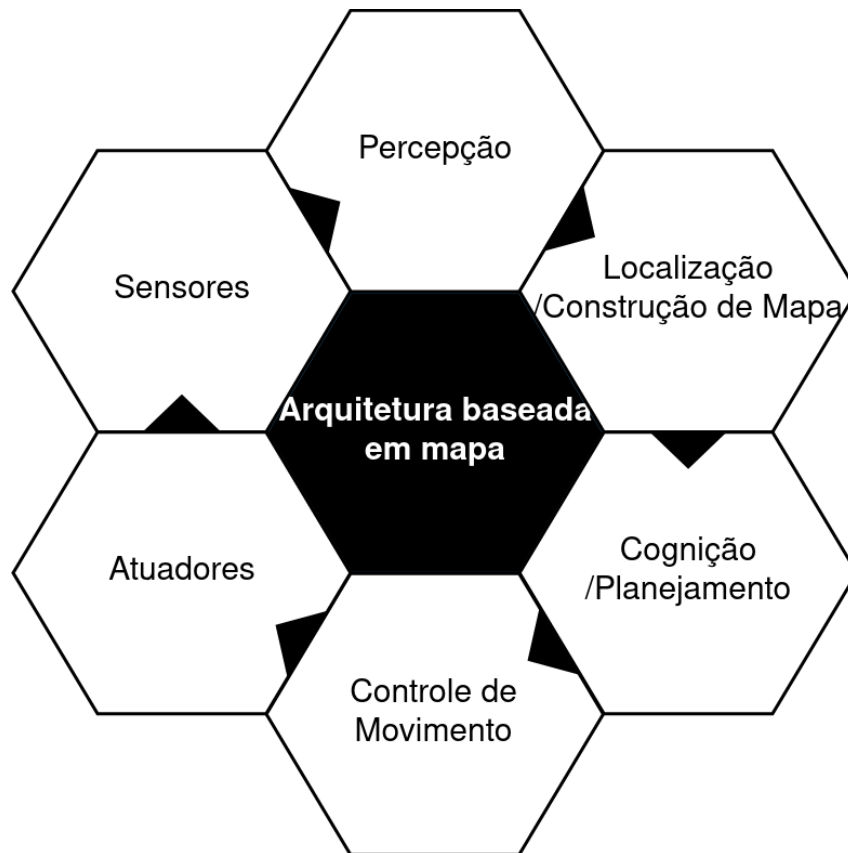
Uma possível solução para o problema de localização é a utilização de técnicas baseadas em mapa, nessa abordagem o robô tenta se localizar por meio de dados coletados do sensor, que são utilizados para atualizar uma crença (estimativa) sobre sua posição em relação a um mapa do ambiente. De acordo com [Siegwart, Nourbakhsh e Scaramuzza \(2011\)](#), as principais vantagens dos sistemas baseados em mapas são as seguintes:

- O conceito explícito de posição baseado em mapa torna a crença do sistema sobre a posição disponível de forma transparente para os operadores humanos.
- A existência do mapa em si representa um meio de comunicação entre humanos e robô: o humano pode simplesmente dar ao robô um novo mapa se o robô for para um novo ambiente e caso o robô crie um mapa ele também pode ser usado por humanos.

As desvantagens do posicionamento baseado em mapa são as exigências para a exatidão dos sensores e a necessidade de que existam características estacionárias e facilmente distinguíveis que possam ser usadas para comparação.

Um exemplo da arquitetura baseada em mapas é mostrado na Figura 2 pode-se observar que os dados obtidos pelos sensores são interpretados na camada de percepção, e essas informações são usadas para a construção do mapa e localização do robô nesse mapa, após essas etapas o robô consegue realizar um planejamento do trajeto e se movimentar pelo ambiente utilizando seus atuadores.

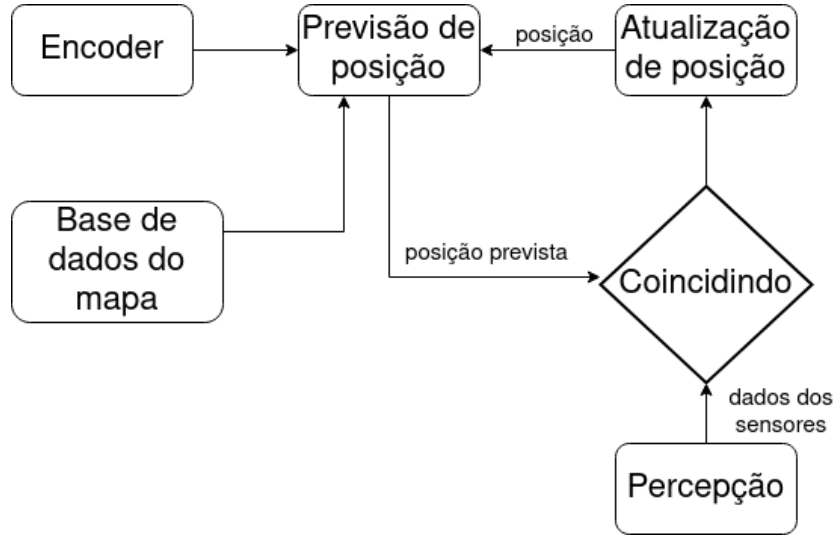
Figura 2 – Arquitetura baseada em mapas



Fonte: Autor.

Um esquema geral de localização para robôs móveis por mapas está representado na Figura 3, conforme dito anteriormente, o problema da localização consiste em estimar a posição e orientação do robô. Para realizar essa tarefa o robô deve possuir informações sobre o mapa do ambiente, utilizar alguma técnica, como por exemplo a odometria, para realizar a previsão da posição do robô, obter informações dos sensores que possam auxiliar no processo de verificar se a posição estimada confere com as características observadas no ambiente e assim possa atualizar sua estimativa de posição.

Figura 3 – Esquema geral de localização para robôs móveis



Fonte: Adaptado de [Siegwart, Nourbakhsh e Scaramuzza \(2011\)](#).

Pode-se definir a posição p do robô conforme:

$$p = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix}, \quad (2.1)$$

em que x representa o eixo horizontal, y representa o eixo vertical e θ representa o ângulo.

Para um robô com rodas diferenciais (como o robô e-puck utilizado nesse trabalho) a posição atual pode ser estimada a partir de uma posição inicial conhecida, somando as distâncias de deslocamento incrementais:

$$\Delta x = \Delta s \cos\left(\theta + \frac{\Delta\theta}{2}\right), \quad (2.2)$$

$$\Delta y = \Delta s \sin\left(\theta + \frac{\Delta\theta}{2}\right), \quad (2.3)$$

$$\Delta\theta = \frac{(\Delta S_r - \Delta S_l)}{b}, \quad (2.4)$$

$$\Delta s = \frac{(\Delta S_r + \Delta S_l)}{2}, \quad (2.5)$$

em que $(\Delta x; \Delta y; \Delta\theta)$ representam o caminho percorrido no último intervalo de amostragem Δt , a distância percorrida pela roda direita é representado por ΔS_r e a distância percorrida pela roda esquerda é representado por ΔS_l e b é a distância entre as rodas.

A posição atualizada p' pode ser calculada da seguinte forma:

$$p' = \begin{bmatrix} x' \\ y' \\ \theta' \end{bmatrix} = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} + \begin{bmatrix} \Delta s \cos(\theta + \frac{\Delta\theta}{2}) \\ \Delta s \sin(\theta + \frac{\Delta\theta}{2}) \\ \Delta\theta \end{bmatrix} \quad (2.6)$$

Substituindo $\Delta\theta$ e Δs , temos:

$$p' = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} + \begin{bmatrix} \frac{(\Delta S_r + \Delta S_l)}{2} \cos(\theta + \frac{(\Delta S_r - \Delta S_l)}{2b}) \\ \frac{(\Delta S_r + \Delta S_l)}{2} \sin(\theta + \frac{(\Delta S_r - \Delta S_l)}{2b}) \\ \frac{(\Delta S_r - \Delta S_l)}{b} \end{bmatrix} \quad (2.7)$$

O erro de posição cresce ao longo do tempo devido a erros de integração das incertezas de p e os erros de movimento que ocorreram durante o incremento de movimento $(\Delta S_r; \Delta S_l)$. (SIEGWART; NOURBAKHSH; SCARAMUZZA, 2011).

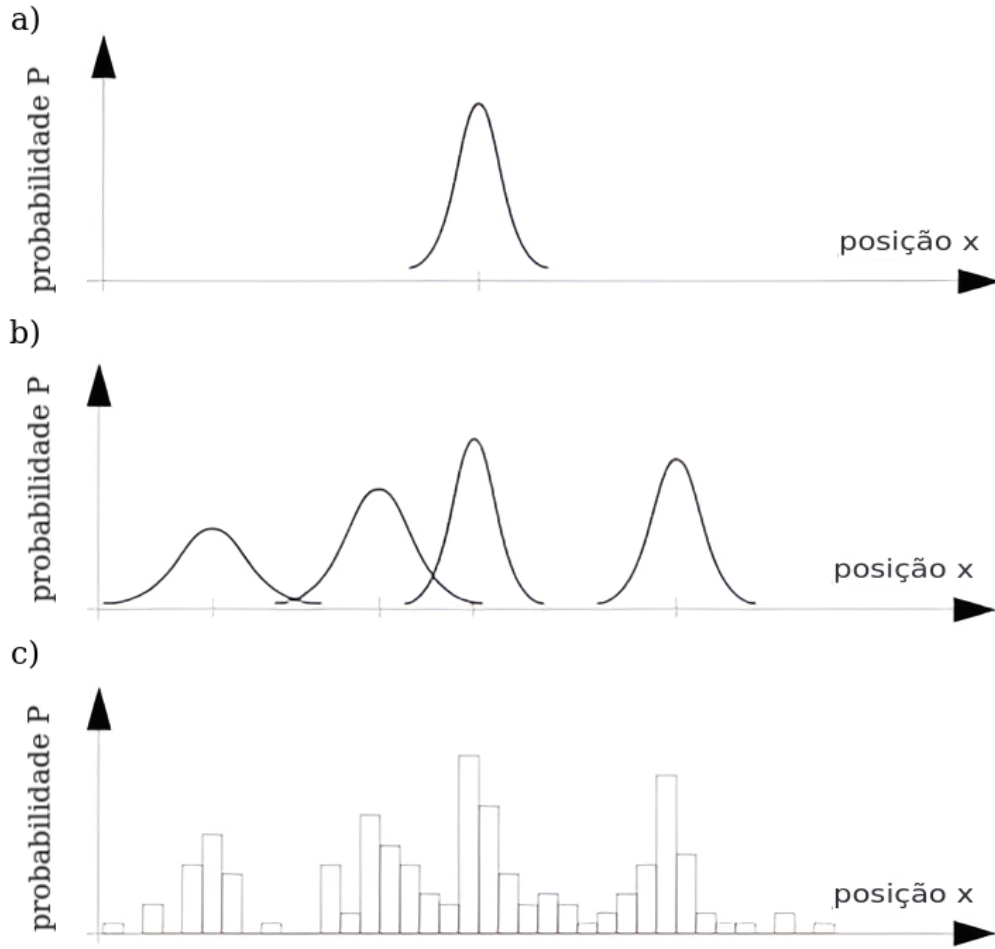
2.2.2 Representação de crença

Segundo Siegwart, Nourbakhsh e Scaramuzza (2011), uma questão fundamental que diferencia os sistemas de localização baseados em mapas é a representação. Existem dois conceitos específicos que o robô deve representar, o mapa e a sua crença (estimativa) em relação a posição no mapa. Para o desenho do mapa é preciso questionar quais aspectos do ambiente devem ser representados e o nível de fidelidade do mapa em relação ao mesmo. Para a representação da crença em relação à sua posição no mapa, é necessário decidir se o robô identificará uma única posição como sendo a atual ou se descreve sua posição em termos de um conjunto de posições possíveis. Na Figura 4 pode-se observar modelos de representação de crença com hipótese única (a) e com hipótese múltipla (b) e (c).

A robótica probabilística representa crenças por meio de distribuições de probabilidade condicional. Uma distribuição de crença atribui uma probabilidade (ou valor de densidade) a cada hipótese possível em relação ao estado verdadeiro. As distribuições de crenças são probabilidades posteriores sobre variáveis de estado condicionadas aos dados disponíveis. Vamos denotar crença sobre um estado variável x_t por: $bel(x_t) = p(x_t | z_{1:t}, u_{1:t})$, em que os valores de z_t representam os dados de medição, obtidos de imagens da câmera, varreduras de alcance a laser etc. E os valores de u_t representam dados de controle obtidos de informações como a velocidade do robô e de odômetros (THRUN; BURGARD; FOX, 2006).

Segundo Thrun, Burgard e Fox (2006) o algoritmo mais geral para calcular crenças é dado pelo algoritmo do filtro de Bayes. Este realiza o cálculo de uma crença ($bel(x_t)$), feito geralmente a partir das medidas e dos dados de controle. Além disso, é um filtro recursivo, o que significa que a crença em um determinado instante t é calculado com base na crença de um instante passado $t - 1$. Conforme pode-se observar no Algoritmo 1, o

Figura 4 – Representação de crença de posição



Fonte: Adaptado de [Siegwart, Nourbakhsh e Scaramuzza \(2011\)](#).

filtro de Bayes é composto de duas etapas principais que são a etapa de predição: $\overline{bel}(x_t)$ e a etapa de correção: $bel(x_t)$

Algoritmo 1: Filtro de Bayes

```

Function Filtro_de_Bayes( $bel(X_{t-1}), ut, zt$ ):
    for all  $x_t$  do
         $\overline{bel}(x_t) = \int p(x_t|u_t, x_{t-1})bel(x_{t-1})dx$  ;
         $bel(x_t) = \eta p(z_t|x_t)\overline{bel}(x_t)$  ;
    end
    return  $bel(x_t)$  ;

```

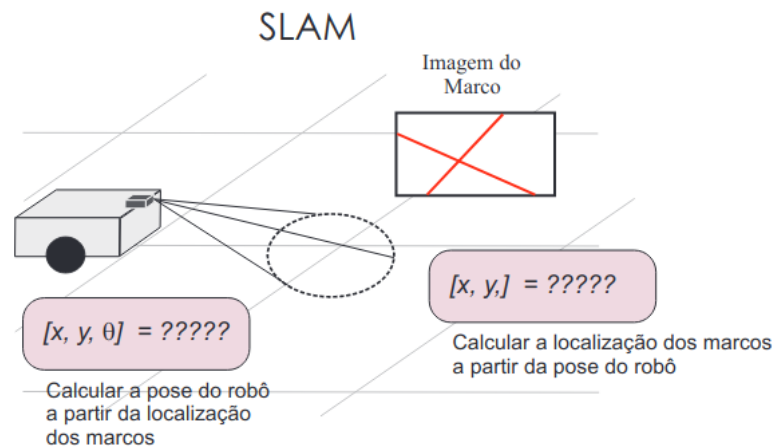
Outros exemplos de localização probabilística são a Localização de Markov e a Localização do filtro de Kalman, em ambas as abordagens, o teorema da probabilidade total é aplicado na fase de atualização da previsão. No entanto, a regra de Bayes é usada para atualização da percepção ([SIEGWART; NOURBAKHSH; SCARAMUZZA, 2011](#)).

2.2.3 Localização e Mapeamento simultâneo

Localização e Mapeamento simultâneo (Simultaneous Localization and Mapping, SLAM) é uma técnica utilizada para que o robô móvel construa um mapa do ambiente enquanto este é explorado. Após esse processo, o mapa gerado é utilizado para que o robô consiga determinar sua localização em relação ao ambiente (KHAIRUDDIN; TALIB; HARON, 2015). A maior vantagem dessa técnica é o fato de que o robô consegue se localizar em um ambiente desconhecido, do qual ele não possui um mapa.

De acordo com Santana (2011), o problema de SLAM pode ser pensado como um “problema do ovo e da galinha” (*chicken or egg problem*), pois: para a localização é necessário o conhecimento do mapa enquanto que uma estimativa exata da pose do robô é necessária para construir esse mapa. O problema de SLAM é ilustrado na Figura 5.

Figura 5 – O Problema de SLAM



Fonte: Santana (2011).

Segundo Camara Neto e Campos (2009), a dificuldade da integração do aspecto exploratório ao SLAM deve-se principalmente ao fato de que não é possível planejar a trajetória com antecedência, já que o próprio mapa do ambiente é desconhecido.

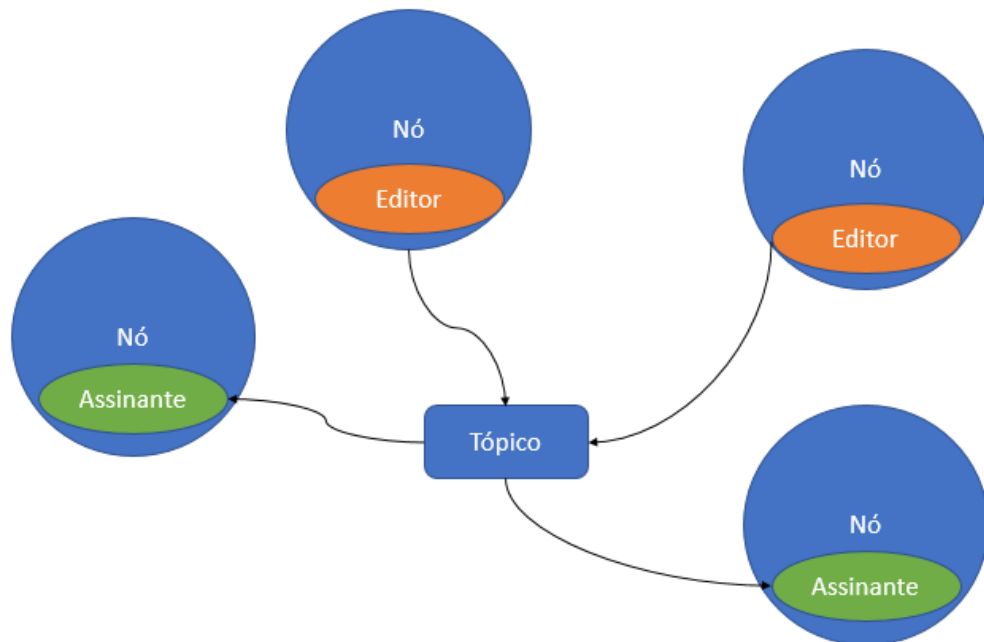
2.3 Robot Operating System

O ROS (*Robot Operating System*) é um *software* de código aberto para desenvolvimento de aplicações robóticas, ROS oferece uma plataforma de software padronizada para desenvolvedores em todos os setores que os levarão da pesquisa e prototipagem até a implantação e produção (MACENSKI et al., 2022).

Dentre as principais características do ROS destaca-se o fato de possuir uma comunidade global de desenvolvedores por mais de 10 anos, possui confiabilidade comprovada por diversas pesquisas em robótica e fornecer um conjunto de ferramentas, bibliotecas

como um barramento para os nós trocarem mensagens. Um nó pode publicar dados para qualquer número de tópicos e simultaneamente ter assinaturas para qualquer número de tópicos.

Figura 7 – Rede de Nós se comunicando por tópicos

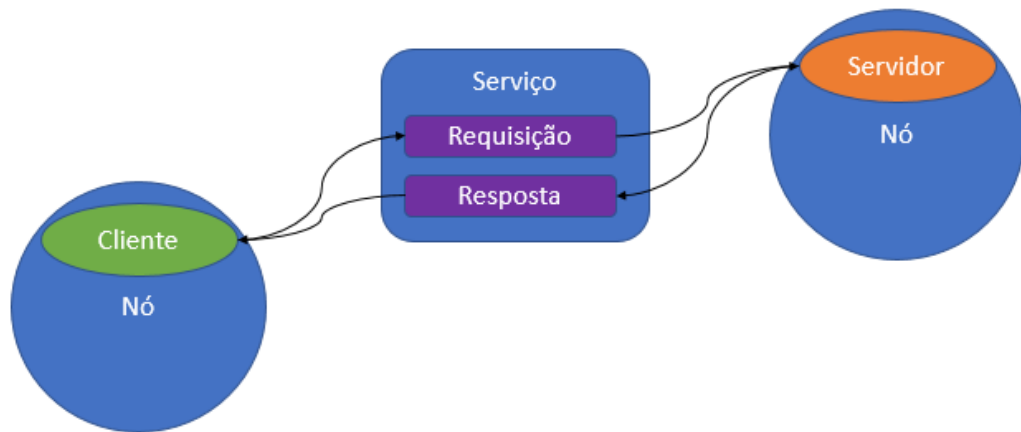


Fonte: Adaptado de ROS (2022).

2.3.2 Serviços

Os serviços são um outro método de comunicação para os nós no gráfico ROS. Os serviços são baseados em um modelo de solicitação/resposta, versus o modelo de editor-assinante dos tópicos. Enquanto os tópicos permitem que os nós assinem fluxos de dados e obtenham atualizações contínuas, os serviços só fornecem dados quando são especificamente chamados por um cliente, conforme a Figura 8.

Figura 8 – Serviços ROS



Fonte: Adaptado de ROS (2022).

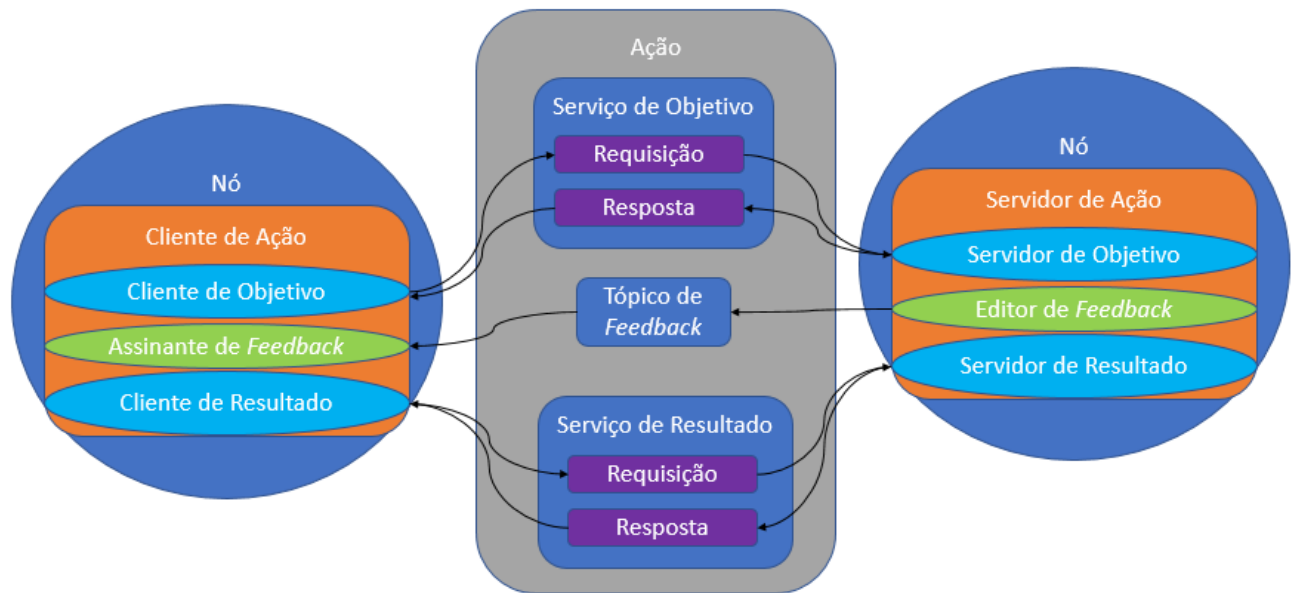
2.3.3 Ações

As ações são um dos tipos de comunicação no ROS 2 e destinam-se a tarefas de longa duração. Eles consistem em três partes: um objetivo, *feedback* e um resultado.

As ações são construídas em tópicos e serviços. Sua funcionalidade é semelhante aos serviços, exceto que as ações são preemptivas (você pode cancelá-las durante a execução). Eles também fornecem *feedback* constante, ao contrário de serviços que retornam uma única resposta.

As ações utilizam o modelo cliente-servidor, um nó “cliente de ação” envia um objetivo para um nó “servidor de ação” que reconhece o objetivo e retorna um fluxo de *feedback* e um resultado, conforme a Figura 9.

Figura 9 – Ações ROS



Fonte: Adaptado de ROS (2022).

2.4 Métodos de Localização utilizando o ROS

Com o objetivo de selecionar os métodos de localização mais utilizados com o ROS, foi realizada uma revisão sistemática da literatura científica nas plataformas Web of Science, Science Direct e IEEE Xplore e Scopus seguindo o método descrito no capítulo 6 (Apêndice). Os trabalhos selecionados foram analisados e comparados seguindo alguns critérios descritos na Tabela 1.

Tabela 1 – Principais Métodos de Localização

Autor	Ambiente	Metodo	Sensor	Plataforma
Belkin, Abramenko e Yudin (2021)	Externo	Monte Carlo Localization adaptado	Lidar	Nvidia Jetson AGX Xavier
Chikurtev et al. (2021)	Interno	AMCL	Lidar e GPS	Não Informado
Chung e Lin (2022)	Interno	AMCL adaptado	Lidar	NVIDIA Tegra K1 Quad Core 2.1GHz CPU, 2G RAM
Chame, Santos e Botelho (2018)	Externo	Rede neural com fusão de sensores	USBL, DGPS E SONAR	Ubuntu 16.04
Franchi et al. (2021)	Externo	MAP	USBL e GPS	PC i7-6700HQ CPU@2.60GHz com 16 GB RAM
Guan et al. (2020)	Interno	VLP	Câmera CMOS	Raspberry Pi 3 Core 1.2 GHz, 1 GB RAM
Gunasagaran et al. (2018)	Interno	AMCL	Laser	Dell laptop com windows 8 e ubuntu 14.04
Housein et al. (2022)	Interno	Fusão de sensores EKF	Odometria e Lidar	Não informado

Jiang et al. (2018)	Interno	EKF adaptado	Kinect e Laser	Laptop Intel Celeron 847 Dual Core
Jonasson, Pinto e Vale (2021)	Interno	ALOAM	RGBD, Lidar e RADAR	Nvidia Jetson Nano
Kayhani et al. (2022)	Interno	Fusão de sensores EKF	Câmera e IMU	Não informado
King et al. (2021)	Externo	Localização sonora	Microfone e Lidar	Não informado
Li et al. (2021)	Interno	SLAM	Câmera	Desktop com 4 núcleos de 3,3 GHz e 16 GB de RAM
Oğuz-Ekim (2020)	Interno	SRD-LS	Lidar	Xeon 3.2 GHz Quad-Core CPU e 8 GB de RAM
Oishi et al. (2019)	Externo	SeqSLAM++	3D Lidar e Câmera	Não informado
Peel et al. (2018)	Externo	AMCL e Hector SLAM	Lidar	Raspberry Pi e NVIDIA Jetson TX1
Rizzo et al. (2021)	Externo	Radio Frequência (RF)	Transmissor e receptor RF	Não informado

Saeedvand, Aghdasi e Baltes (2018)	Externo	LHOL	IMU	Intel Core i5, 8 GB de RAM
Sullivan, Grainger e Cazzolato (2018)	Interno	Cooperative Localization e EKF	3D Lidar	Não informado
Xiao et al. (2019)	Interno e Externo	e Dynamic-SLAM	Câmera	Intel Core i5, 8 GB de RAM e GTX1050Ti
Yilmaz e Temeltas (2019)	Interno	SA-MCL	Lidar	Não informado

Fonte: Autor.

2.4.1 Trabalhos Selecionados

Belkin, Abramenko e Yudin (2021) discutem os principais recursos da abordagem modular proposta para a localização baseada em lidar e propõem um novo procedimento para avaliação da posição inicial do robô baseado no método de localização de Monte Carlo modificado. A aplicação da abordagem modular proposta mostrou um erro de posição de 10 cm e de orientação de 0,7 graus.

Chikurtev et al. (2021) Apresentam um método que utiliza os dados dos sensores Lidar e GPS para realizar a localização com o AMCL e atingiram a precisão de posição em +/- 0,93 m e orientação em 2,53 graus.

Chung e Lin (2022) propõem uma melhoria no AMCL e utiliza informações de laser 2D, e telêmetro para realizar a tarefa de localização. A partir dos resultados experimentais, os autores afirmam que o algoritmo AMCL aprimorado pode melhorar a precisão de posicionamento do robô em relação ao AMCL original.

Chame, Santos e Botelho (2018) propõem um modelo de rede neural denominado B-PR-F que realiza heurísticamente a fusão adaptável de informações, baseado no princípio da antecipação contextual do sinal de localização com um processamento de vizinhança ordenada. Um experimento em simulador mostra que o modelo supera o Filtro de Kalman e os algoritmos de Localização Monte Carlo Aumentada (*Augmented Monte Carlo Localization*) na tarefa de localização.

Franchi et al. (2021) apresentam um estimador MAP (*Maximum A Posteriori*) adaptado para localização de AUVs (Veículos Subaquáticos Autônomos) na presença de medições USBL (*Ultra-Short BaseLine*), a solução apresentada mostrou um erro horizontal máximo de 2,5 m.

Guan et al. (2020) apresentam pela primeira vez um sistema de localização de posicionamento por luz visível (VLP). Um sistema protótipo foi implementado em um Turtlebot3. Os resultados experimentais mostram que o sistema proposto pode fornecer precisão de posição na faixa de 1 cm e um tempo computacional médio de apenas 0,08 s.

Housein et al. (2022) utilizam O filtro de Kalman estendido (EKF) para realizar a fusão de sensores do LIDAR e a Odometria. Os resultados experimentais no robô “turtlebot” usando o ROS mostram uma melhora significativa na pose do robô em comparação com a Odometria.

Jiang et al. (2018) desenvolveram um sistema cooperativo de localização humana em ambientes internos utilizando um robô móvel e *smartphones*. Um algoritmo de localização dinâmica baseado em EKF foi desenvolvido para fundir as medições de distância do sensor Kinect e do alcance acústico baseado em smartphone, para que as posições de destino possam ser estimadas iterativamente. Experimentos mostraram que o algoritmo de posicionamento foi capaz de localizar e rastrear alvos humanos em movimento em diferentes ambientes internos. A precisão da estimativa mediana varia de 0,43 m a 1,12 m.

Kayhani et al. (2022) propõem um método de localização visual inercial baseado em *tags* para UAVs comuns com apenas uma câmera e uma unidade de medida inercial (IMU). o método proposto estima a pose global do UAV fundindo dados inerciais e medições de tags usando EKF. A raiz do erro quadrático médio (RMSE) varia de 2 a 5 cm.

King et al. (2021) examinam a viabilidade de uma metodologia audiovisual para localização de fontes sonoras ocultas. Uma matriz de microfone de quatro canais é usada em conjunto com mapeamento LiDAR e 2D/3D para mesclar ângulos de chegada estimados com parâmetros de sala para localização de fonte de som. Os testes foram realizados em ambientes controlados e não controlados e o método foi capaz de encontrar a fonte oculta com precisão de 0,5 m.

Li et al. (2021) propõem uma estratégia de descarga (*offloading*) de computação parcial visual SLAM multi-robô, fazendo com que parte do processamento seja realizado em nuvem ao invés de localmente. O melhor ponto de descarga é dado para reduzir o consumo de energia e custo de tempo de todo o sistema SLAM visual.

Oğuz-Ekim (2020) propõem um novo algoritmo para localização de robôs baseado no método dos mínimos quadrados. A abordagem descreve a localização do robô em coordenadas polares, levando a um algoritmo não convexo cuja solução pode ser encontrada com eficiência. A raiz do erro quadrático médio (RMSE) varia de 2 cm a 2.2 m.

Oishi et al. (2019) apresentam uma nova abordagem para localização e navegação baseadas em visão para ambientes externos. O novo método denominado SeqSLAM++, é uma extensão do SeqSLAM convencional para que além de estimar de forma robusta a posição do robô comparando sequências de imagens, também possa lidar com mudanças no rumo e velocidade de um robô, bem como visualizar mudanças usando imagens angulares e um esquema de localização de Markov. A confiabilidade foi demonstrada em um ambiente externo e alcançou uma taxa de sucesso de localização de 95,8%.

Peel et al. (2018) utilizam a Localização de Monte Carlo Adaptativa em um mapa conhecido e esse método forneceu resultados comparáveis ao Hector-SLAM, com taxas de erro inferiores a um limite definido de 10 cm.

Rizzo et al. (2021) estimam a localização de um robô ao longo de um tubo com uma abordagem alternativa de Radiofrequência (RF). O método é capaz de localizar um robô ao longo de um tubo por meio da geração e detecção de um padrão de desvanecimento de sinal periódico, sem a necessidade de um mapa previamente conhecido do cenário ou qualquer modificação substancial da infraestrutura existente.

Saeedvand, Aghdasi e Baltes (2018) apresentam um método de aprendizado robusto para localizar um robô humanoide chamado *Lightweight Humanoid robot Odometric Learning method* (LHOL). O aprendizado básico do método é baseado na rede neural artificial (ANN) que usa cálculos cinemáticos, IMU (dados de rotação e inclinação) e os dados de carga presentes internos dos atuadores do robô como dados de entrada. A taxa de erro média apresentada foi menor que 9 cm.

Xiao et al. (2019) apresentam um *framework* SLAM denominado DynamicSLAM, que é um sistema semântico monocular visual simultâneo de localização e mapeamento usando aprendizado profundo para melhorar o desempenho em ambiente dinâmico. A precisão de localização do Dynamic-SLAM é maior 7,48% – 62,33% do que o sistema ORBSLAM2 de última geração e o desempenho da operação é melhorado em cerca de 10%.

Yilmaz e Temeltas (2019) propõem um novo modelo de energia baseado em elipse para o método SA-MCL (*Self Adaptive Monte Carlo Localization*) para eliminar a restrição de posicionamento uniforme do sensor do SA-MCL padrão e tornar o método adequado para AGVs (*Autonomous Guided Vehicles*) com LIDARs 2D ou 3D. Observa-se que o algoritmo estimou a posição do AGV com taxas de erro de 6,69 cm e 4,74 cm RMSE nas direções x e y, respectivamente.

Conforme pode-se observar na Tabela 1, os métodos de Monte Carlo e de fusão de sensores com EKF, são os mais utilizados para localização de robôs móveis com o ROS e por esse motivo esses métodos são abordados de maneira mais aprofundada nesse trabalho e são implementados no simulador Webots utilizando o ROS 2.

2.5 Considerações Finais

Ciente da importância do ROS e da localização para robôs móveis, realizou-se uma revisão sobre a arquitetura do ROS e os principais métodos de localização. No próximo capítulo, é apresentada a implementação de alguns métodos de localização de robôs móveis.

3 Metodologia

Esse trabalho busca estudar e implementar técnicas de localização de robôs móveis em um ambiente simulado e compará-las, para isso foi utilizada a plataforma de simulação Webots, esse simulador permite que os controladores sejam programados em C/C++ (LANCHEROS; SANABRIA; CASTILLO, 2016), Python (FISHER, 2022), Java (FARLEY; WANG; MARSHALL, 2022), Matlab (MA; LIANG; TIAN, 2020) ou ROS (DOBROKVAISHINA et al., 2022). Neste trabalho utilizamos a API (*Applications Protocol Interface*) para ROS (Robot Operating System), com relação à versão do ROS foi utilizada a versão 2 LTS (Long-Term Support) mais recente, codinome (Foxy Fitzroy), em conjunto com a linguagem Python versão 3.8 para a implementação dos métodos de localização no simulador. Com relação ao ambiente para teste, todas as técnicas foram avaliadas nos mesmos cenários, o ambiente foi preparado de modo que não interfira de maneira negativa para nenhuma técnica e possa ser analisada a eficiência de cada uma dessas técnicas de localização e comparar os resultados obtidos com relação às linhas de código, tempo de processamento, consumo de CPU, consumo de memória e dificuldade de implementação.

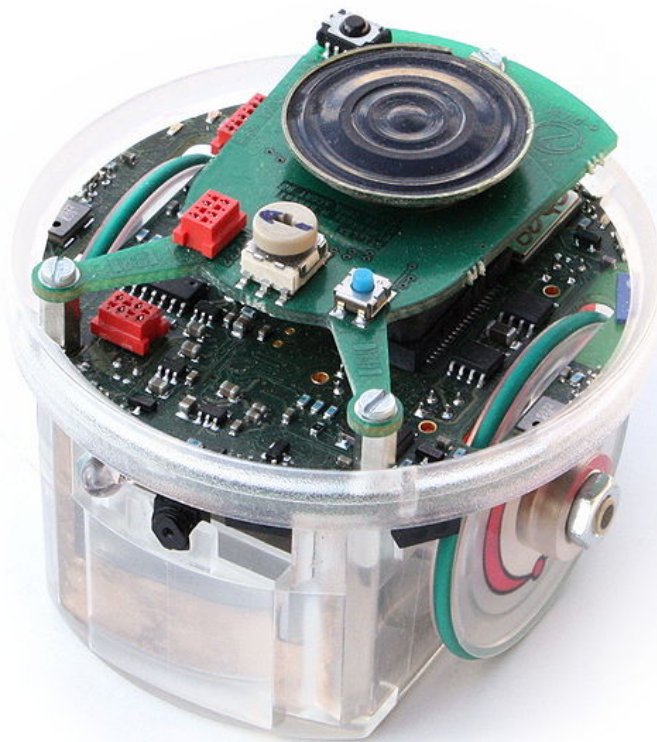
Todo o processo foi realizado em um notebook com processador Intel Core i3 com 2.4 GHz, 4 GB de memória RAM DDR3, 240 GB de SSD e sistema operacional Ubuntu Linux — Focal Fossa (20.04) de 64 bits.

3.1 Robô Móvel Utilizado

O robô utilizado nos experimentos foi o E-puck Mobile Robot (Figura 10). O E-puck é um robô móvel em miniatura originalmente desenvolvido no EPFL (Instituto Federal Suíço de Tecnologia em Lausanne) para fins de ensino pelos designers do bem-sucedido robô Khepera. O *hardware* e *software* do e-puck é totalmente de código aberto, fornecendo acesso de baixo nível a todos os dispositivos eletrônicos e oferecendo varias possibilidades de extensão. O robô e-puck possui 7,4 cm de diâmetro, 4,5 cm de altura e peso de 150 g.

O modelo no Webots inclui suporte para os motores com rodas diferenciais (os codificadores também são simulados, como sensores de posição), os sensores infravermelhos para medições de proximidade e luz, o acelerômetro, o giroscópio, a câmera, LEDs e comunicação bluetooth. Como extensão foi adicionado ao robô um módulo de GPS e um modulo de IMU (Inertial Measurement Unit) para serem utilizados com os métodos de localização.

Figura 10 – Epuck Mobile Robot



Fonte: Autor.

O pacote `webots_ros2` fornece a modelagem URDF(Unified Robot Description Format) padrão do robô E-puck, sendo necessário alterar apenas para incluir os novos sensores que foram adicionados na extensão como o GPS e o IMU.

O pacote `webots_ros2_driver` cria automaticamente uma interface ROS 2 para quase todos os dispositivos Webots (exceto o IMU, pois ele é uma combinação de vários dispositivos).

O IMU é composto de três dispositivos no Webots:

- Um dispositivo de unidade inercial (No simulador a unidade inercial retorna apenas uma orientação).
- Um dispositivo giroscópio para medir velocidades angulares.
- Um dispositivo acelerômetro para medir acelerações.

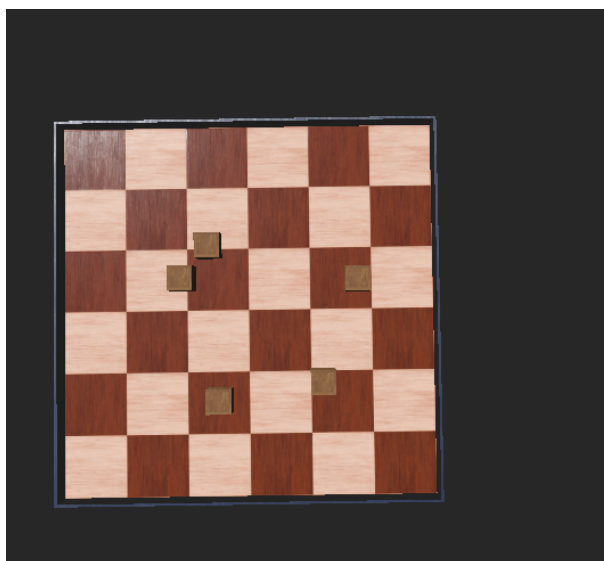
Como a interface para o dispositivo IMU não é criada automaticamente, para que o ROS consiga reconhecer o dispositivo corretamente, foi necessário adicionar o seguinte plugin ao arquivo URDF:

```
<robot name="Epuck_Webots">
  <webots>
    <plugin type="webots_ros2_driver::Ros2IMU">
      <enabled>true</enabled>
      <updateRate>20</updateRate>
      <topicName>/e_puck/imu</topicName>
      <alwaysOn>false</alwaysOn>
      <frameName>imu_link</frameName>
      <inertialUnitName>inertial_unit</inertialUnitName>
      <gyroName>gyro</gyroName>
      <accelerometerName>accelerometer</accelerometerName>
    </plugin>
  </webots>
</robot>
```

3.2 Ambiente para testes

Para a realização dos testes foi criado o cenário 01 no Simulador Webots, conforme pode ser visto na Figura 11, que se trata de um ambiente com 1,5 metros de comprimento e 1,5 metros de largura, possui quatro paredes e um conjunto de obstáculos (caixas de madeira). A partir do ambiente criado foi gerado um mapa para servir de referência para as técnicas de localização, pode-se observar que o mapa mostrado na Figura 12 possui menos detalhes do ambiente se compararmos ao cenário mostrado na Figura 11, isso acontece pois no mapa deve ter apenas as informações mais relevantes para a localização, por exemplo a localização das paredes e obstáculos, outras informações como por exemplo a textura ou cor do piso e paredes não estão presentes no mapa pois não são relevantes para a localização do robô móvel.

Figura 11 – Cenário 01



Fonte: Autor.

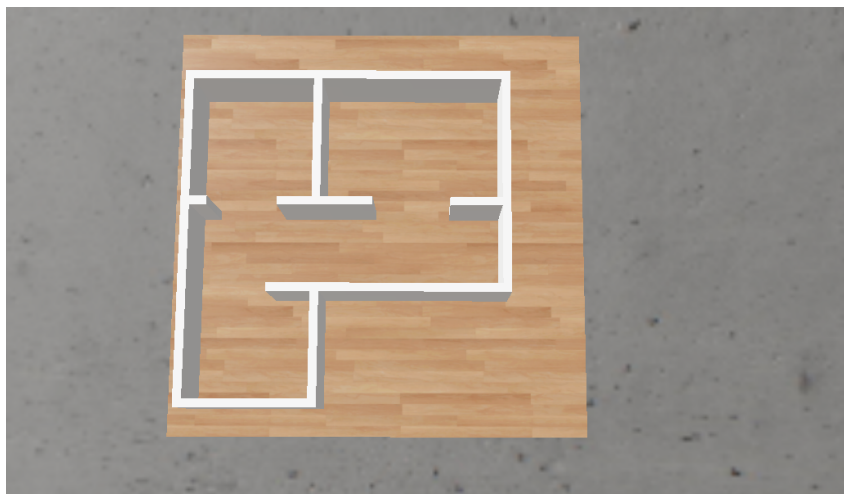
Figura 12 – Mapa relativo ao cenário 01



Fonte: Autor.

Para fins de comparação com o cenário 01, foi criado o cenário 02 que se trata de um ambiente semelhante a uma planta de uma casa, possuindo paredes que separam cada comodo, conforme pode ser observado na Figura 13. O Cenário 02 possui 0,8 metros de comprimento e 0,8 metros de largura, então é consideravelmente menor do que o Cenário 01. O mapa gerado para esse Cenário é mostrado na Figura 14, observe que no cenário 02 os principais obstáculos são as paredes e no cenário 01, os principais obstáculos são as caixas, embora também existam paredes.

Figura 13 – Cenário 02



Fonte: Autor.

Figura 14 – Mapa relativo ao cenário 02



Fonte: Autor.

3.3 Localização de Monte Carlo

A localização de Monte Carlo ou também chamado de filtro de partículas é derivado do filtro de Bayes. A ideia chave é usar amostras aleatórias extraídas da probabilidade posterior $bel(xt)$. Para implementar a localização do filtro de partículas, precisa-se dos seguintes componentes:

- Modelo de movimento para atualização de previsão;
- Modelo de medição para atualização de medição;
- Filtro de partículas para toda a atualização recursiva.

3.3.1 Modelo de movimento

A maioria dos robôs está equipada com um codificador de roda para determinar até que ponto as rodas giraram. Ao usar esta informação o robô pode prever onde está posicionado depois de se mover por um período de tempo.

O modelo de odometria desenvolvido usa informações de movimento relativo que é o delta entre as poses anteriores e atuais e é calculado da forma a seguir:

$$\delta_{rot1} = atan2(\bar{y}' - \bar{y}, \bar{x}' - \bar{x}) - \bar{\theta}, \quad (3.1)$$

$$\delta_{trans} = \sqrt{(\bar{x} - \bar{x}')^2 + (\bar{y} - \bar{y}')^2}, \quad (3.2)$$

$$\delta_{rot2} = \bar{\theta}' - \bar{\theta} - \delta_{rot1}, \quad (3.3)$$

em que δ_{rot1} representa a variação na rotação 1, δ_{trans} é a variação na translação, δ_{rot2} é a variação na rotação 2, x representa o eixo horizontal e y representa o eixo vertical.

Pode-se prever a pose atual, adicionando algumas incertezas ao processo para modelar os erros de odometria. Modelamos nosso ruído como gaussiana de média zero conforme [Thrun, Burgard e Fox \(2006\)](#):

$$N(\mu = 0, \sigma^2), \quad (3.4)$$

em que σ^2 é a variância e μ é a média de distribuição. O ruído do modelo de movimento é calculado pelas funções a seguir, que possuem como parâmetros de erro α_1 a α_4 (erros específicos do robô e determinam o erro acumulado com o movimento):

$$\delta_{rot1} = \delta_{rot1} - ruido(\alpha_1 \delta_{rot1}^2 + \alpha_2 \delta_{trans}^2), \quad (3.5)$$

$$\delta_{trans} = \delta_{trans} - ruido(\alpha_3 \delta_{trans}^2 + \alpha_4 \delta_{rot1}^2 + \alpha_4 \delta_{rot2}^2), \quad (3.6)$$

$$\delta_{rot2} = \delta_{rot2} - ruido(\alpha_1 \delta_{rot2}^2 + \alpha_2 \delta_{trans}^2), \quad (3.7)$$

em que $\alpha_1, \alpha_2, \alpha_3$ e α_4 possuem os seguintes valores respectivamente para esse modelo específico (0,001; 0,001; 0,01; 0,001).

O estado previsto é calculado da seguinte forma:

$$x' = x + \hat{\delta}_{trans} \cos(\theta + \hat{\delta}_{rot1}), \quad (3.8)$$

$$y' = y + \hat{\delta}_{trans} \sin(\theta + \hat{\delta}_{rot1}), \quad (3.9)$$

$$\theta' = \theta + \hat{\delta}_{rot1} + \hat{\delta}_{rot2}, \quad (3.10)$$

3.3.2 Modelo de medição

Os modelos de medição descrevem o processo de formação pelo qual as medições do sensor são geradas no mundo físico. Formalmente, é definido como uma distribuição de probabilidade condicional: $p(z_t|x_t, m)$, em que x_t é a posição do robô, z_t é a medição no tempo t , e m é o mapa do ambiente.

Quando sabe-se onde estão os pontos de extremidade no mapa, pode-se calcular a probabilidade de eles serem verdadeiros. Quando não há obstáculos, a informação não tem significado e, portanto, é descartada.

Para calcular os pontos finais no mapa, foi utilizada a seguinte equação:

$$\begin{pmatrix} x_{z_t}^k \\ y_{z_t}^k \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} \cos \theta - \sin \theta \\ \sin \theta \cos \theta \end{pmatrix} + \begin{pmatrix} x_k, sens \\ y_k, sens \end{pmatrix} + z_t^k \begin{pmatrix} \cos(\theta + \theta_k, sens) \\ \sin(\theta + \theta_k, sens) \end{pmatrix}, \quad (3.11)$$

em que z_t^k é a medição do sensor, $(x_k, sens, y_k, sens)$ é a localização relativa do sensor em relação ao robô e $(\theta_k, sens)$ é a orientação do sensor em relação a direção do robô.

3.3.3 Filtro de Partículas

O filtro de partículas funciona estimando a pose verdadeira usando um conjunto de partículas/amostras. Quanto mais densa a área, maior a probabilidade de estar próxima da pose verdadeira. O algoritmo utilizado do filtro de partículas está representado a seguir, em que ocorrem basicamente duas etapas principais, a de previsão do movimento e de atualização da medição do sensor.

Algoritmo 2: Filtro de partículas

Function Filtro_de_Partículas(X_{t-1}, ut, zt, m):

```

 $X_t = 0$  ;
for  $m = 1 \rightarrow M$  do
     $x_t^m = atualizacao\_de\_movimento(ut, x_t^m)$  ;
     $w_t^m = atualizacao\_do\_sensor(zt, x_t^m, m)$  ;
end
return  $X_t$  ;
```

3.4 Fusão de Sensores

Para realizar a fusão de sensores com o ROS foi utilizado o pacote `robot_localization`, esse pacote contém nós de estimativa de estado como o `EKF_Localization` (*Extended Kalman filter*) e `UKF_Localization` (*Unscented Kalman Filter*). Além disso ele permite a fusão de um número arbitrário de sensores e é possível configurar a personalização de dados de entrada por sensor, ou seja, se uma determinada mensagem do sensor contiver dados que você não deseja incluir em sua estimativa de estado, os nós de estimativa de estado em `robot_localization` permitem que você exclua esses dados por sensor. Para utilizar esse pacote e realizar a fusão de sensores, o primeiro passo é verificar se os dados dos sensores obedecem aos padrões ROS, os dois REPs (ROS Enhancement Proposals) mais importantes a serem considerados são:

- REP-103 (Unidades Padrão de Medida e Convenções de Coordenadas)
- REP-105 (Convenções de quadro de coordenadas)

A configuração dos sensores é realizada no arquivo `ekf.yaml`, em que é necessário definir se está trabalhando em 2D ou 3D. A frequência de saída após a fusão, dentre outras configurações que pode-se destacar a configuração dos quadros de coordenada: `map_frame`, `odom_frame`, `base_link_frame` e `world_frame`; A associação dos sensores aos respectivos tópicos ROS e a configuração individual de cada um dos sensores conforme:

```
odom0_config:      [true, true, true,
                    false, false, false,
                    true, true, true,
                    false, false, true,
                    false, false, false]
```

A ordem dos valores booleanos é `[x, y, z, roll, pitch, yaw, vx, vy, vz, vroll, vpitch, vyaw, ax, ay, az]`, em que `x, y` e `z` representam a posição do robô, `roll, pitch` e `yaw` representam a orientação, `vx, vy` e `vz` representam a velocidade linear, `vroll, vpitch` e `vyaw` representam a velocidade angular e `ax, ay` e `az` representam a aceleração linear. O valor de obtido de cada sensor apenas será considerado na fusão caso esteja configurado como `true`, dessa forma é possível personalizar a configuração para cada sensor e obter uma melhor estimativa de posição pois quando obtemos duas ou mais fontes de dados de orientação é possível fundir apenas os dados do sensor mais preciso.

3.5 Nav2 AMCL

O ROS 2 Navigation Stack (Também conhecido como Nav2) é um *framework* de navegação gratuito e de código aberto que auxilia na busca de encontrar uma maneira

segura de fazer com que um robô móvel se mova do ponto A para o ponto B. Em outras palavras, utilizando o Nav2 é possível movimentar o robô sem utilizar o pacote `teleop_twist_keyboard` (pacote incluso com o ROS que movimenta o robô baseado na tecla pressionada) a vantagem do Nav2 é que o usuário informa o ponto de destino e o robô consegue traçar a rota e se locomover em direção a ele.

O AMCL (Localização de Monte Carlo Adaptativa) é fornecido junto com o Nav2, seu funcionamento é bem similar a implementação de Monte Carlo, porém o numero de partículas é variável.

O AMCL recebe um mapa, varreduras a laser e mensagens de transformação, e sua saída representa a estimativa de posição. Na inicialização, seu filtro de partículas é iniciado de acordo com os parâmetros fornecidos. Observe que, devido aos padrões, se nenhum parâmetro for definido, o estado inicial do filtro é uma nuvem de partículas de tamanho moderado centrada em $(0,0,0)$.

Para utilizar o AMCL é necessário instalar o Nav2 (Veja no Apêndice 6 e configurar um inicializador (Launcher) fornecendo informações sobre o mapa, configurando a posição inicial do robô e os parâmetros utilizados.

3.6 Método Combinado de AMCL com Fusão de Sensores EKF

A implementação do método combinado de AMCL com fusão de sensores foi realizada utilizando a saída da estimativa de posição pelo AMCL no tópico `/amcl_pose`, como entrada para o método de fusão de sensores. A estimativa de posição do AMCL e os sensores GPS e IMU foram combinados com o objetivo de melhorar a precisão e diminuir a taxa de erros de posição em relação aos métodos isolados.

3.7 Análise de desempenho

As técnicas escolhidas para análise fazem parte de um subconjunto das encontradas durante a revisão de literatura, visando aprofundar no estudo dessas técnicas e aproveitar o tempo hábil disponível, apenas alguns métodos serão implementados e analisados durante essa pesquisa. Os pontos de análise das técnicas serão listados a seguir:

- Análise Quantitativa:
 - Eficiência
 - Consumo de CPU
 - Consumo de memória RAM
 - Tempo médio de resposta

- Analise Qualitativa:
 - Facilidade ou dificuldade de implementação
 - Principais problemas encontrados

A visualização do robô durante a movimentação pelo mapa foi realizada com o RViz (*ROS 3D Robot Visualizer*) que já é incluído na instalação padrão do ROS.

Para analisar o consumo de memória RAM e CPU de cada experimento foi utilizado o seguinte script em que utiliza o comando top do Linux para captar informações, após isso converte para o formato CSV que é compatível com Excel, e finalmente no Excel foi gerado os gráficos a partir dos dados obtidos.

Algoritmo 3: Obtenção de dados de desempenho

```
while sleep 0.1 ;
do top -b -n 3 | sed -n '8,12{s/^*//;s/_*$//;s/_ */;/gp;};12q'
>> saida.csv;
done
```

Para a verificação dos erros de posição em relação a posição real do robô foi utilizado o RMSE (*root mean squared error*), calculado da forma a seguir:

$$RMSE = \sqrt{\frac{1}{n} * \sum_{i=1}^n (real_i - previsto_i)^2} \quad (3.12)$$

O desempenho dos métodos também foi avaliado utilizando o R2 Square (Coeficiente de determinação)

$$R2 = 1 - \frac{\sum_{i=1}^n (Y_i - \hat{Y}_i)^2}{\sum_{i=1}^n (Y_i - \bar{Y})^2}, \bar{Y} = \frac{1}{N} \sum_{i=1}^n Y_i, \quad (3.13)$$

em que Y_i representa a posição real, $\hat{Y}_i$ representa a previsão da posição e $\bar{Y}$ representa a média dos valores de posição reais.

Para a visualização dos gráficos de erro foi utilizado o PlotJuggler, que é uma das melhores ferramentas para visualização de séries temporais com o ROS (ver Apêndice).

3.8 Considerações Finais

A metodologia apresentada neste capítulo segue o seguinte fluxo: preparação do cenário, implementação dos algoritmos de localização e avaliação do desempenho. No próximo capítulo, os resultados obtidos com as respectivas discussões são apresentados.

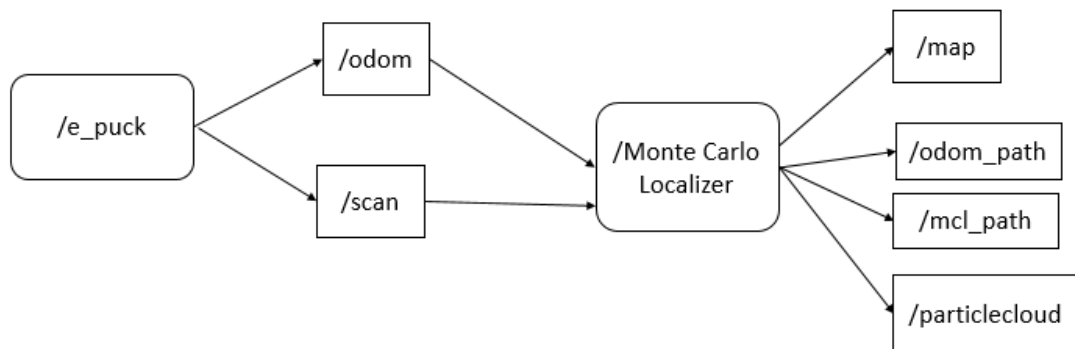
4 Resultados e Discussões

Neste capítulo são apresentados e discutidos os resultados obtidos com a implementação dos métodos de localização para robôs móveis utilizando o ROS 2 no simulador Webots. A visualização da crença de posição em relação ao mapa foi extraída a partir do RViz, todos os métodos de localização implementados são comparados entre si.

Após instalação do Webots e do ROS2, foi utilizado o pacote `webots_ros2`, esse pacote provê as interfaces necessárias para simular um robô no simulador Webots e realizar a integração ao ROS2 usando mensagens, serviços e ações.

O Gráfico ROS com a arquitetura da aplicação e a relação dos nós ROS está representado na Figura 15. O Nó `e_puck` é responsável pelo acesso ao *hardware* do robô e todos os sensores, o nó `monte_carlo_localizer` recebe os dados dos sensores de odometria e scan a laser e com base nessas informações, a posição do robô é calculada no nó `particle_cloud`, que foi desenvolvido conforme o algoritmo mostrado na metodologia e atualiza a sua referência de posição em relação ao mapa. Os nós `odom_path` e `mcl_path` são responsáveis por mostrar o caminho percorrido pelo robô de acordo com seu histórico de localização de odometria e do algoritmo de monte carlo respectivamente.

Figura 15 – Gráfico ROS da aplicação



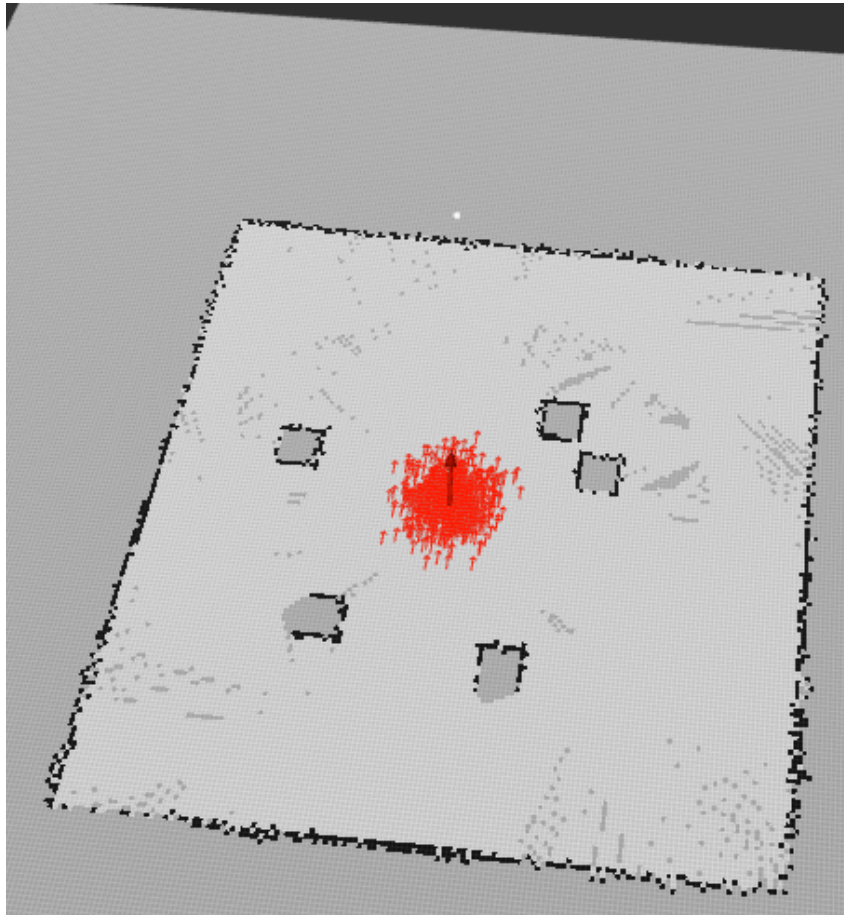
Fonte: Autor.

4.1 Localização de Monte Carlo

O MCL funciona estimando a pose verdadeira usando um conjunto de partículas ou amostras. Quanto mais densa a área, maior a probabilidade de estar próxima da pose verdadeira. Veja o exemplo na Figura 16, as pequenas setas vermelhas representam as partículas e a grande seta vermelha é a pose de odometria. Observe que devido ao fato

de o MCL ser um algoritmo probabilístico não há uma crença de posição única, mas um conjunto de possíveis posições e a melhor probabilidade de posição é baseado na quantidade de partículas com valor similar.

Figura 16 – Monte Carlo Cenário 01



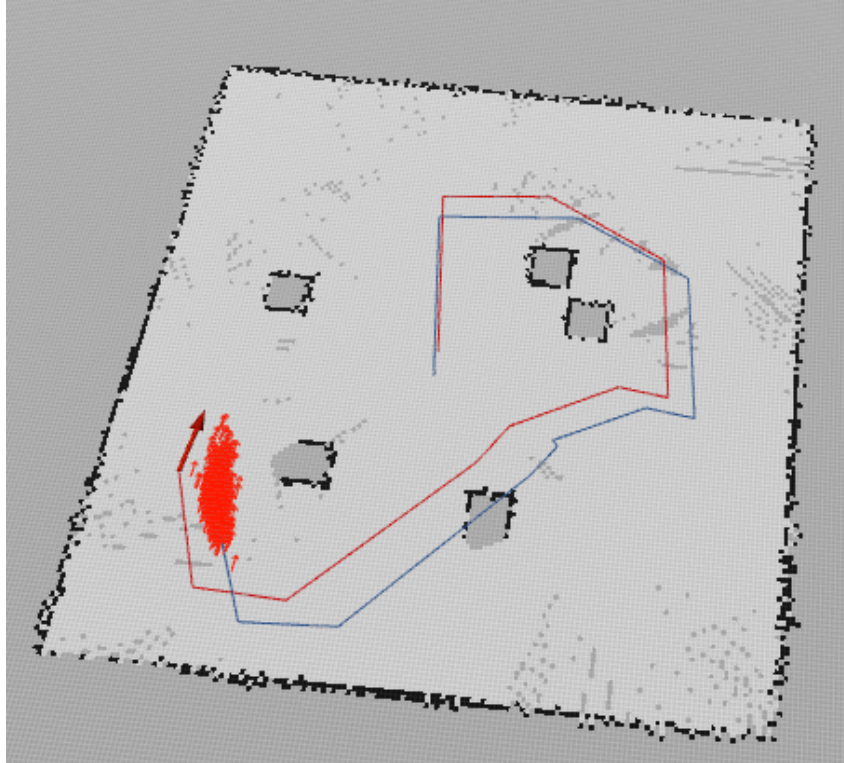
Fonte: Autor.

Para realizar a estimativa de localização por meio do MCL a pose inicial do robô é conhecida, mas não precisa, além disso, foi adicionado um ruído a odometria para ficar mais semelhante ao que é encontrado no desenvolvimento usando robôs reais. Para inicializar as partículas, foi utilizada a distribuição Gaussiana, definido o número inicial de 1000 partículas e como usamos um filtro de partículas não adaptativo, o número de partículas permanece o mesmo ao longo da execução.

Na Figura 17 os dados de localização da odometria e do MCL são comparados. A linha azul é o caminho do Filtro de Partículas (baseado na informação da melhor partícula) e a linha vermelha é o caminho da Odometria. observe que em alguns pontos o Filtro de Partículas ajusta a pose do robô, na maioria das vezes de maneira a melhorar a informação sobre a localização, porém, ocorreu situações em que a melhor partícula não estava de acordo com a posição do robô, como pode-se observar a linha azul atravessando

um obstáculo, que no caso era impossível de ser realizado pelo robô, ou seja, temos um erro de posição nesse local.

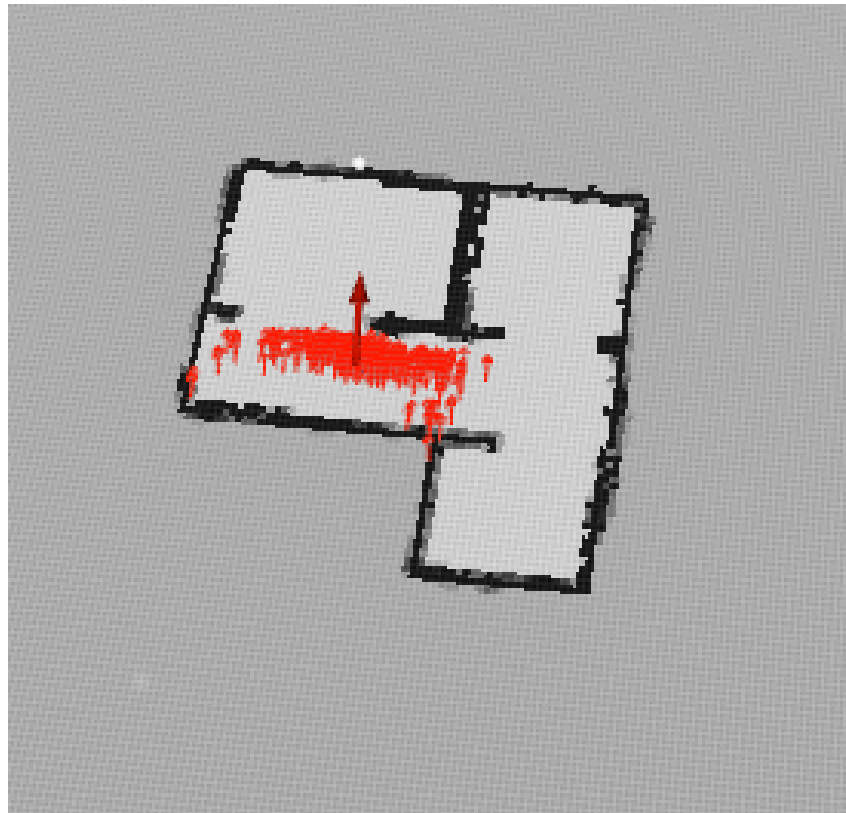
Figura 17 – Comparativo entre Localização de Monte Carlo e Odometria



Fonte: Autor.

Para movimentar o robô pelo mapa e observar o comportamento do algoritmo de localização, foi utilizado o pacote `teleop_twist_keyboard` do ROS 2, que permite controlar o robô remotamente a partir de comandos enviados pelo teclado. Ao alterar a posição do robô a posição das partículas no mapa foi alterada conforme a nova posição, o que indica que o algoritmo consegue atualizar a probabilidade da posição do robô de acordo com as informações dos sensores.

Figura 18 – Monte Carlo Cenário 02



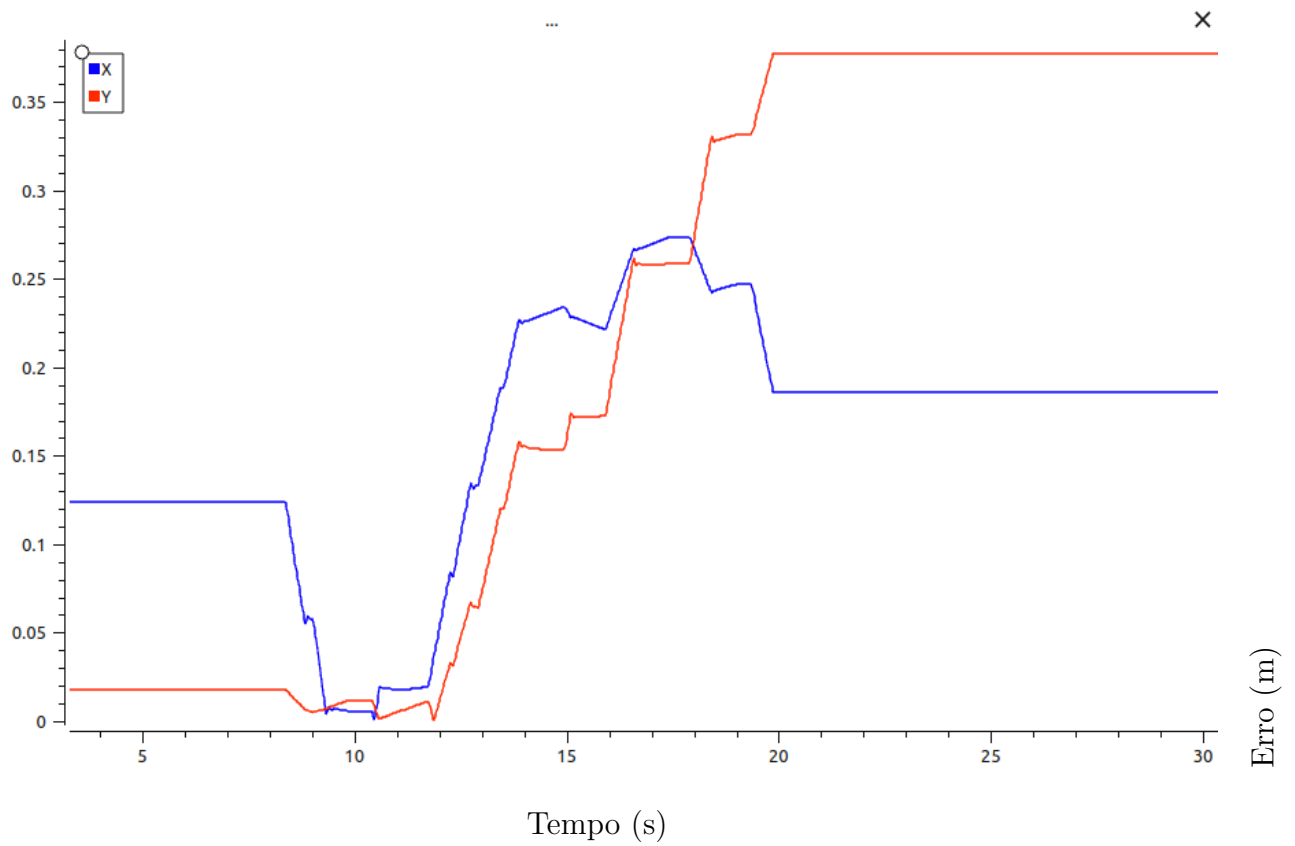
Fonte: Autor.

No Cenário 02, mostrado na Figura 18, devido a ser espaço mais fechado e menor, foi observado uma maior incerteza de posição do robô em relação ao cenário 01, que é demonstrada pelo número de partículas isoladas e um número maior com relação ao erro máximo na Tabela 2. Apesar do maior grau de incerteza o algoritmo conseguiu realizar a localização de maneira adequada e o erro médio ficou na faixa de 14 cm a 16 cm.

Os erros de posição em relação a posição real do robô foram calculados usando o RMSE (Raiz quadrada do erro-médio) e são mostrados nas Figuras 19 e 20, que representam respectivamente os erros de posição nos cenários 01 e 02. Além disso, os erros de posição são mostrados de forma mais resumida na Tabela 2. Pode-se observar que a precisão do algoritmo é bem elevada já que mesmo no pior caso, a média dos erros de posição chegaram apenas a 21 cm, o que demonstra a eficiência do mesmo.

Com o objetivo de verificar a influência do número de partículas na precisão da localização do MCL, foi configurado o número de partículas para 2000 (o dobro do experimento anterior) e os resultados são demonstrados na Tabela 3. Pode-se observar que não houve uma grande melhora na precisão da localização do robô, pois apesar de ter diminuído o erro médio em x no cenário 02 e em y no cenário 01, o erro médio de x no cenário 01 e y no cenário 02 aumentaram. Além disso, ocorreu um aumento no custo computacional e durante a simulação ocorreu uma lentidão maior e pequenos travamentos

Figura 19 – Erros de posição da Localização de Monte Carlo Cenário no 01



Fonte: Autor.

Tabela 2 – Erros de posição RMSE - Método MCL

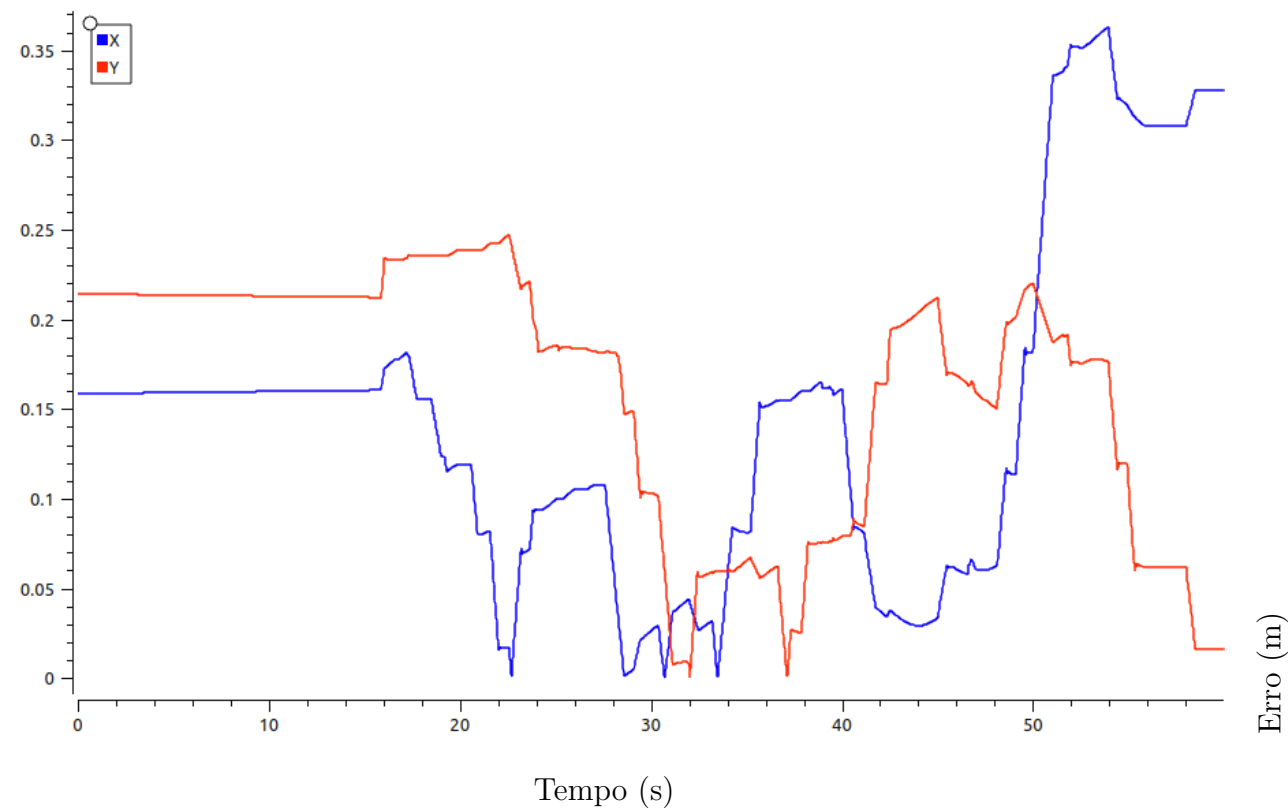
Cenário	Eixo	Erro Mínimo (m)	Erro Máximo (m)	Erro Médio (m)
01	x	0,000687	0,273623	0,163671
01	y	0,000824	0,377224	0,211272
02	x	0,000228	0,362798	0,145356
02	y	0,000059	0,247096	0,160695

Fonte: Autor.

devido a esse alto número de partículas. Essa lentidão se deve ao aumento de consumo da CPU que ficou próximo a 100% de utilização. Então percebe-se que aumentar o número de partículas não significa necessariamente aumentar a precisão de localização.

O consumo de CPU e Memória RAM da implementação do método de Monte Carlo (com 1000 partículas) está representada na Figura 21, que mostra a porcentagem de utilização em relação ao tempo:

Figura 20 – Erros de posição da Localização de Monte Carlo Cenário no 02



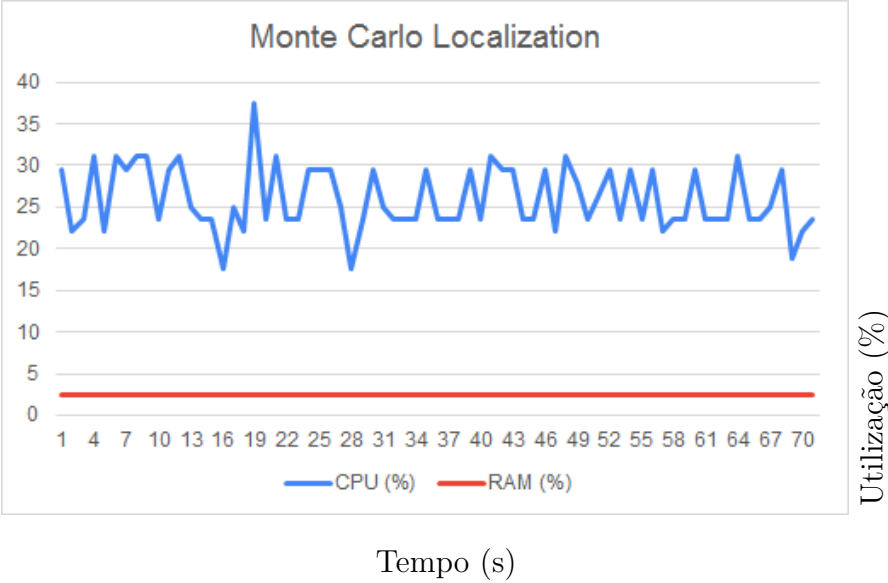
Fonte: Autor.

Tabela 3 – Erros de posição RMSE - Método MCL com 2000 partículas

Cenário	Eixo	Erro Mínimo (m)	Erro Máximo (m)	Erro Médio (m)
01	x	0,000070	0,522438	0,180753
01	y	0,000014	0,397050	0,126784
02	x	0,000063	0,421328	0,135680
02	y	0,000006	1,005860	0,484136

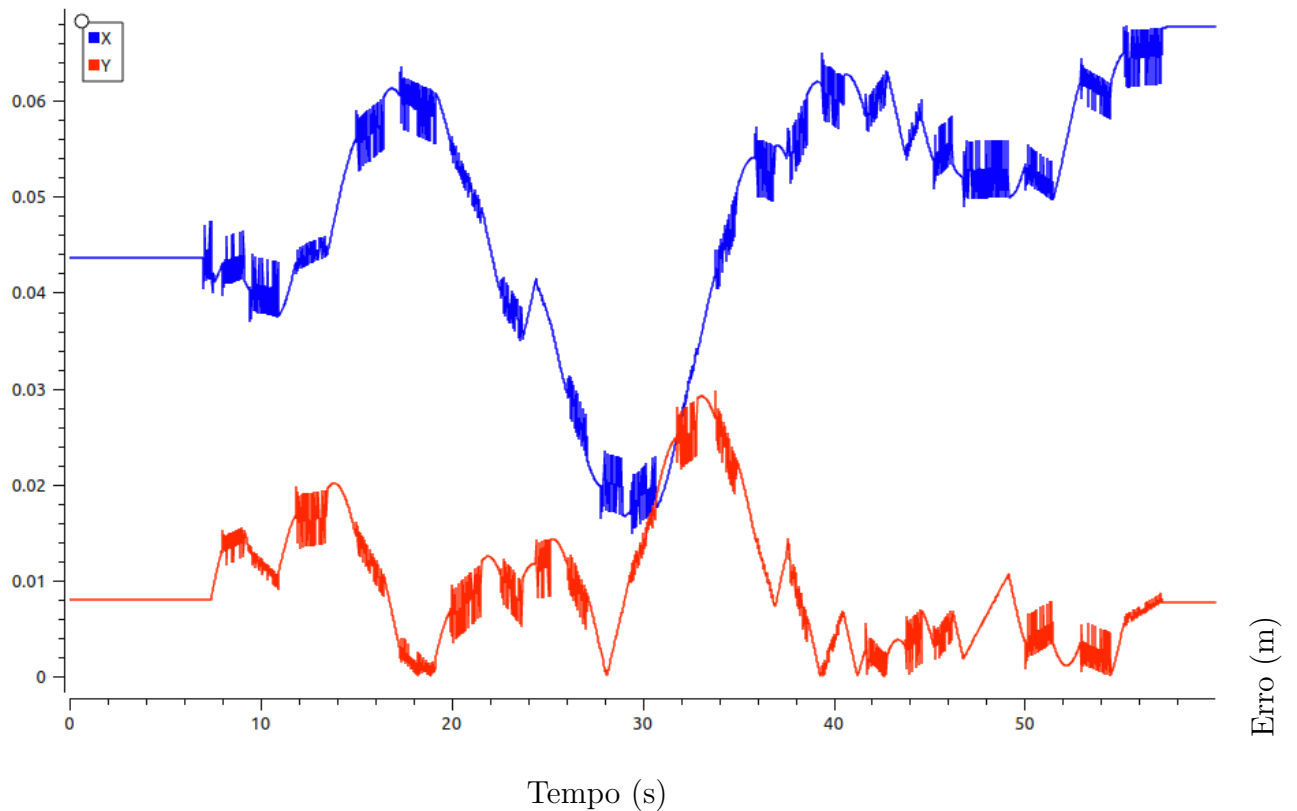
Fonte: Autor.

Figura 21 – Consumo de Memória e CPU - Localização de Monte Carlo



Fonte: Autor.

Figura 22 – Erros de posição do método de fusão de sensores no Cenário 01



Fonte: Autor.

Tabela 4 – Erros de posição RMSE - Método Fusão de Sensores com EKF

Cenário	Eixo	Erro Mínimo (m)	Erro Máximo (m)	Erro Médio (m)
01	x	0,014792	0,067875	0,048411
01	y	0,000001	0,029779	0,009210
02	x	0,000032	0,067991	0,018865
02	y	0,043279	0,109811	0,067829

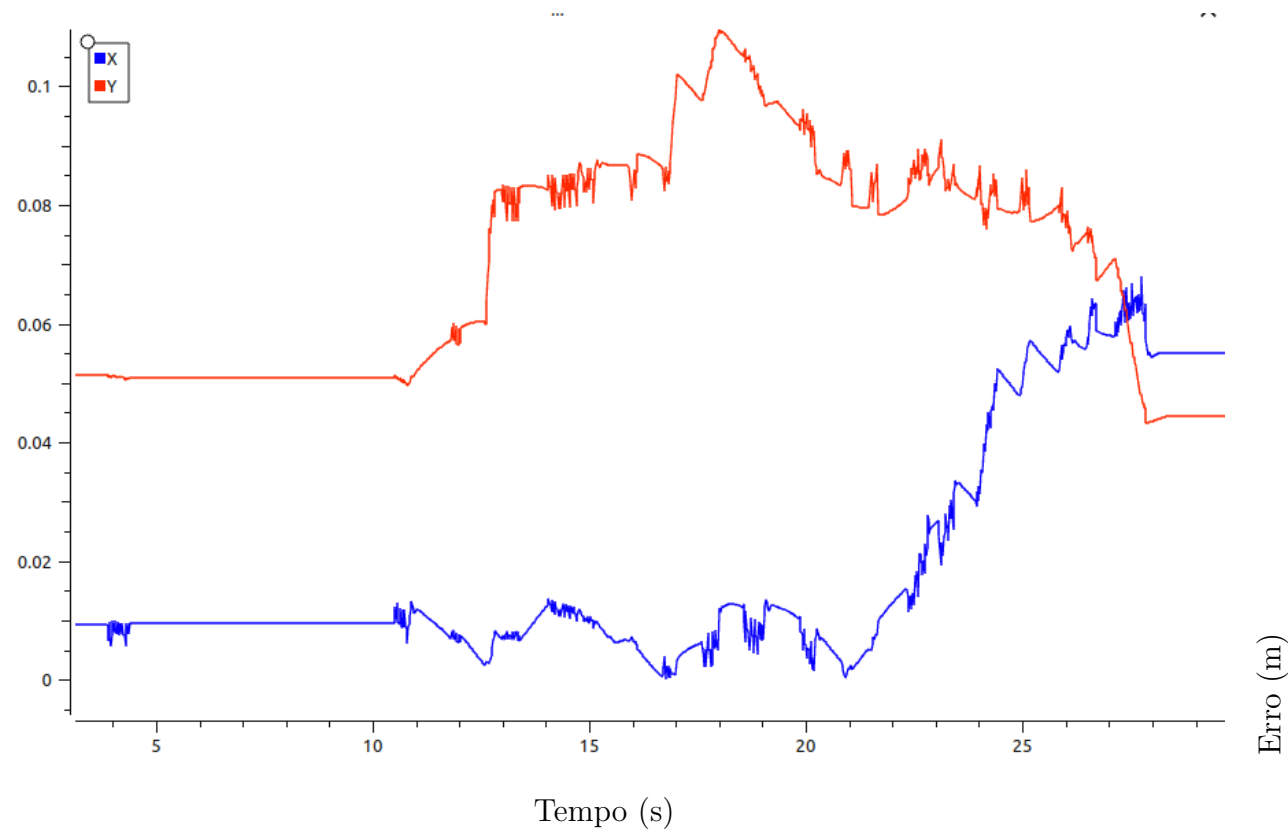
Fonte: Autor.

4.2 Fusão de Sensores

O método de fusão de sensores foi implementado combinando as informações dos sensores de GPS, IMU e a informação da odometria. Os erros de posição são mostrados nas Figuras 22 e 23 e na Tabela 4. Pode-se observar que esse método também possui uma boa precisão, com o erro médio entre 1,8 cm a 6,7 cm e o erro máximo de 10,9 cm.

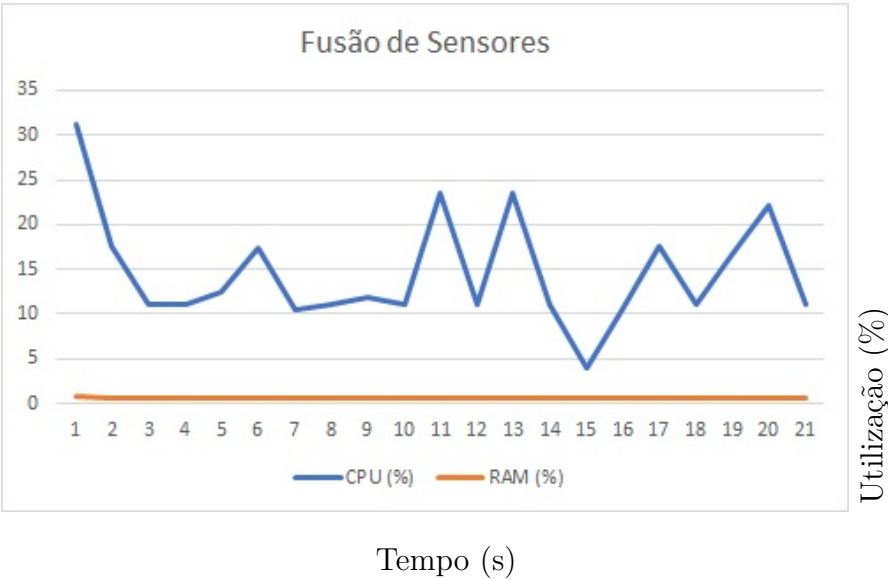
O consumo de Memória e CPU desse método é mostrado na Figura 24, pode-se verificar que há um menor consumo de *hardware* em relação ao MCL. Então esse método pode ser considerado uma ótima opção para ser utilizado em um robô móvel autônomo de baixo custo, pois além de ter uma boa precisão ele foi o que apresentou o menor consumo.

Figura 23 – Erros de posição do método de fusão de sensores no Cenário 02



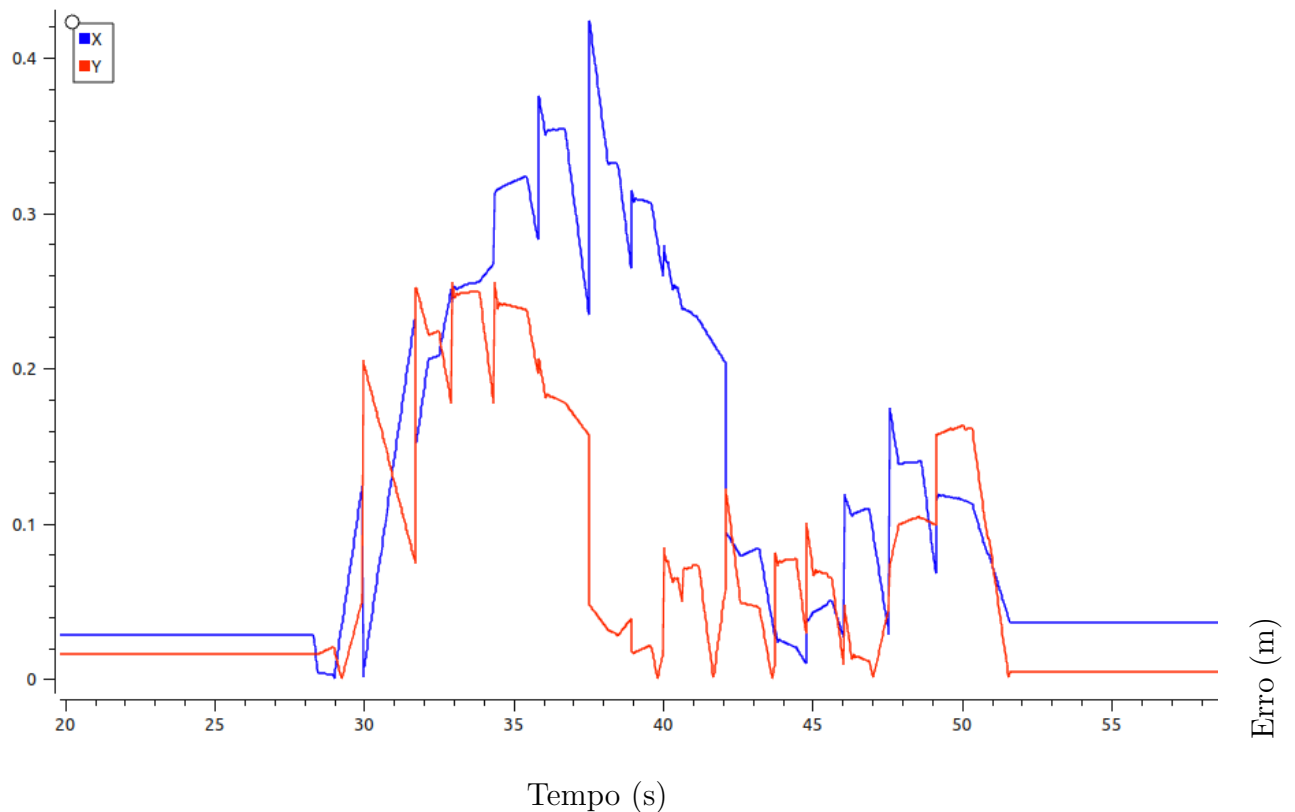
Fonte: Autor.

Figura 24 – Consumo de Memória e CPU - Fusão de Sensores com EKF



Fonte: Autor.

Figura 25 – Erros de posição do método AMCL no Cenário 01



Fonte: Autor.

Tabela 5 – Erros de posição RMSE - Método AMCL

Cenário	Eixo	Erro Mínimo (m)	Erro Máximo (m)	Erro Médio (m)
01	x	0,000381	0,423892	0,115521
01	y	0,000201	0,255508	0,066626
02	x	0,000000	0,163444	0,055063
02	y	0,027483	0,194471	0,081509

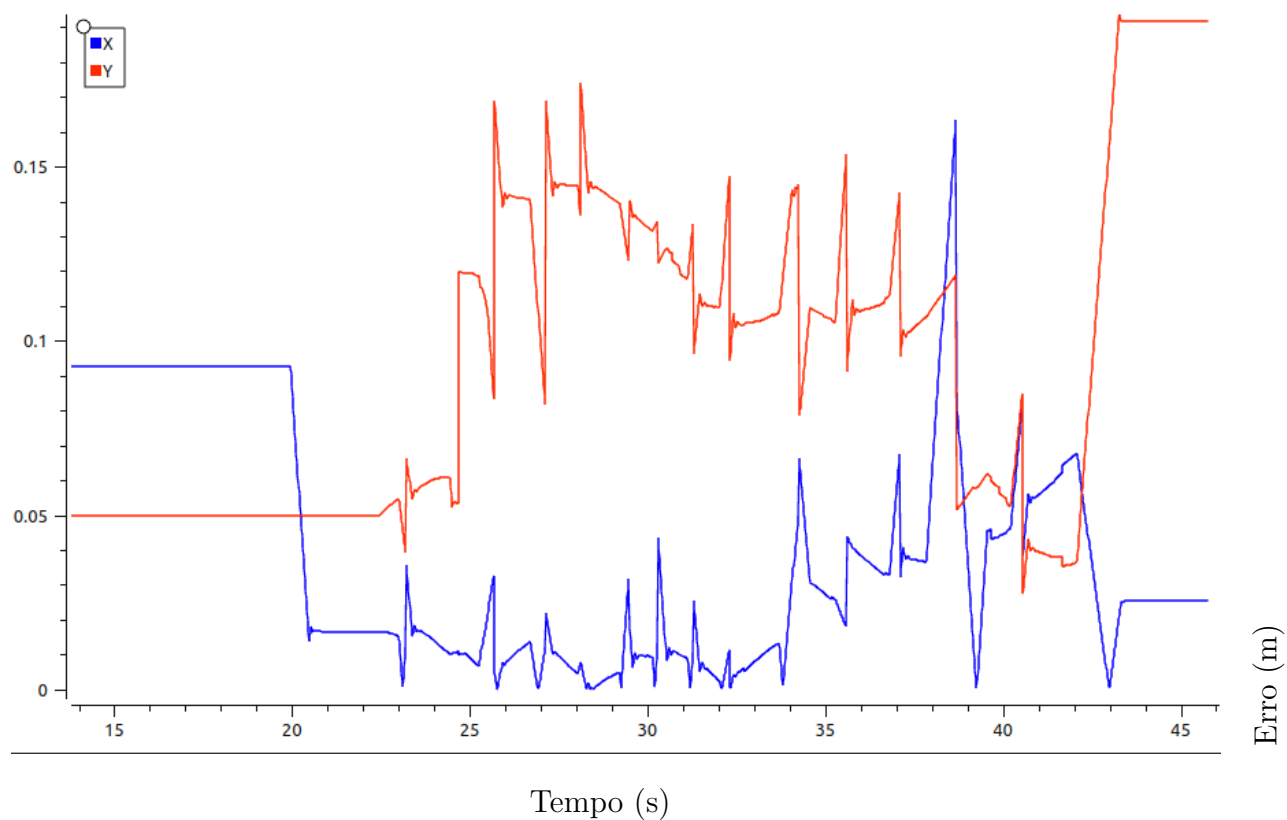
Fonte: Autor.

4.3 AMCL

Para a implementação do AMCL foi definido como número mínimo de partículas o valor de 60 e como máximo 2000 partículas. Com relação à precisão da estimativa de posição, esse método apresentou resultados similares ao método de localização de Monte Carlo que foi implementado seguindo as equações, conforme pode ser observado nas Figuras 25 e 26 e na Tabela 5, os valores são próximos a implementação do método de Monte Carlo utilizando 2000 partículas de forma fixa. Entretanto o AMCL se apresentou mais eficiente do que o MCL normal devido a possuir esse número variável de partículas, o que influencia diretamente no desempenho e faz com ele tenha um menor consumo de *hardware* sem aumentar muito os erros de posição.

Com relação ao consumo de Memória e CPU (Figura 27) apesar de o AMCL

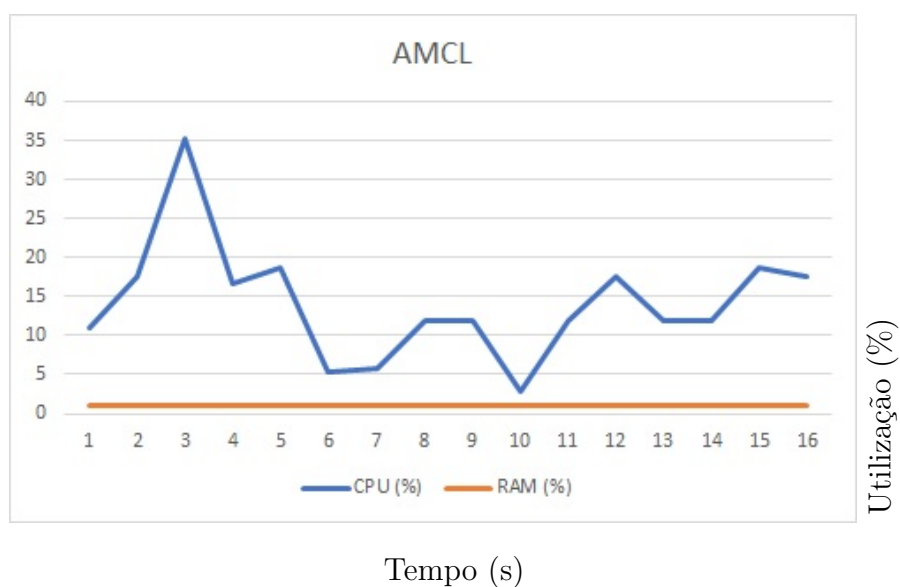
Figura 26 – Erros de posição do método AMCL no Cenário 02



Fonte: Autor.

consumir menos que o algoritmo MCL implementado, ele ainda consome um pouco mais do que o método de fusão de sensores.

Figura 27 – Consumo de Memória e CPU - AMCL

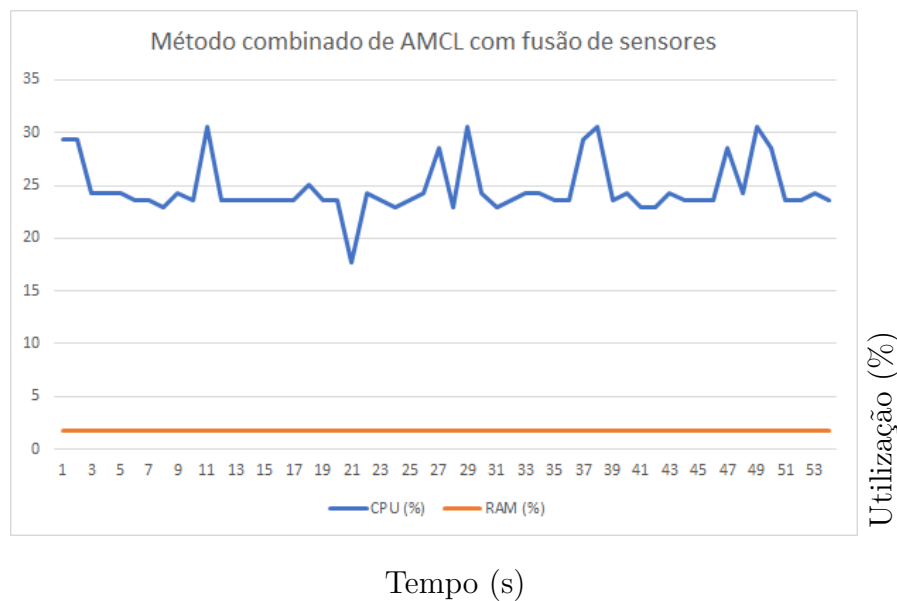


Fonte: Autor.

4.4 Método combinado de AMCL com Fusão de Sensores

Para a implementação desse método foi utilizado a estimativa de posição do AMCL como entrada para o método de fusão de sensores com EKF e então essa informação de posição foi combinada com o GPS, IMU e odometria. O consumo de *hardware* desse método é mostrado na Figura 28. Pode-se observar que ele possui um baixo consumo, mesmo sendo resultado da combinação de dois métodos diferentes.

Figura 28 – Consumo de Memória e CPU - Método combinado de AMCL e Fusão de Sensores com EKF



Fonte: Autor.

Os erros de posição são mostrados nas Figuras 29 e 30 que representam o cenário 01 e 02 respectivamente. Além disso, também é apresentado de maneira resumida na Tabela 6. Pode-se observar que a combinação desses dois métodos foi capaz de melhorar a precisão da localização, já que mostrou erros bem menores em relação ao AMCL e o EKF isoladamente. Além disso, essa melhoria da precisão não gerou um grande aumento no processamento, mantendo um bom desempenho.

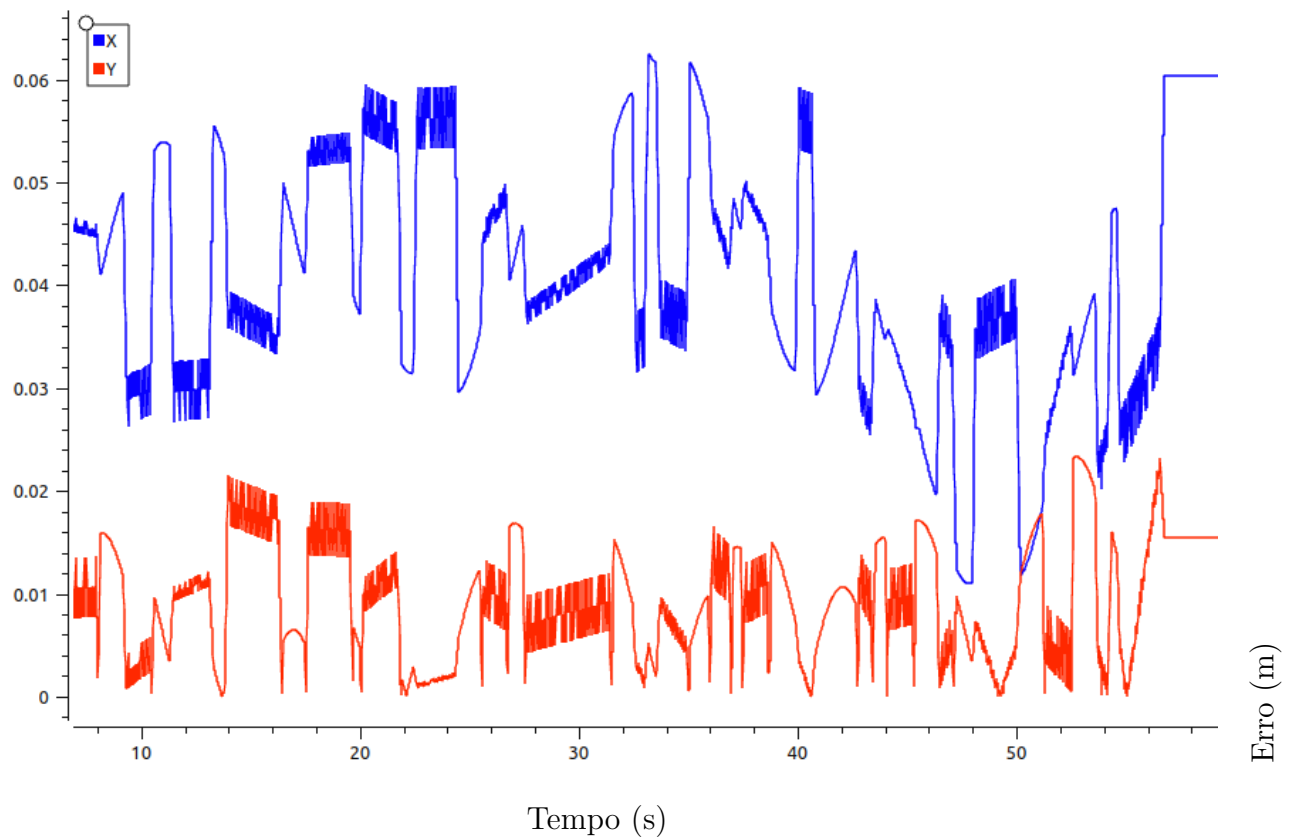
Tabela 6 – Erros de posição RMSE - Método combinado de AMCL com Fusão de Sensores

Cenário	Eixo	Erro Mínimo (m)	Erro Máximo (m)	Erro Médio (m)
01	x	0,011004	0,062481	0,041003
01	y	0,000021	0,023302	0,009547
02	x	0,000466	0,086590	0,031224
02	y	0,020877	0,077292	0,050768

Fonte: Autor.

Na Tabela 8 temos o resumo do desempenho dos métodos analisados nos Cenários 01 e 02. O método que obteve o melhor desempenho com relação aos erros de posição

Figura 29 – Erros de posição do método combinado de AMCL e fusão de sensores no Cenário 01

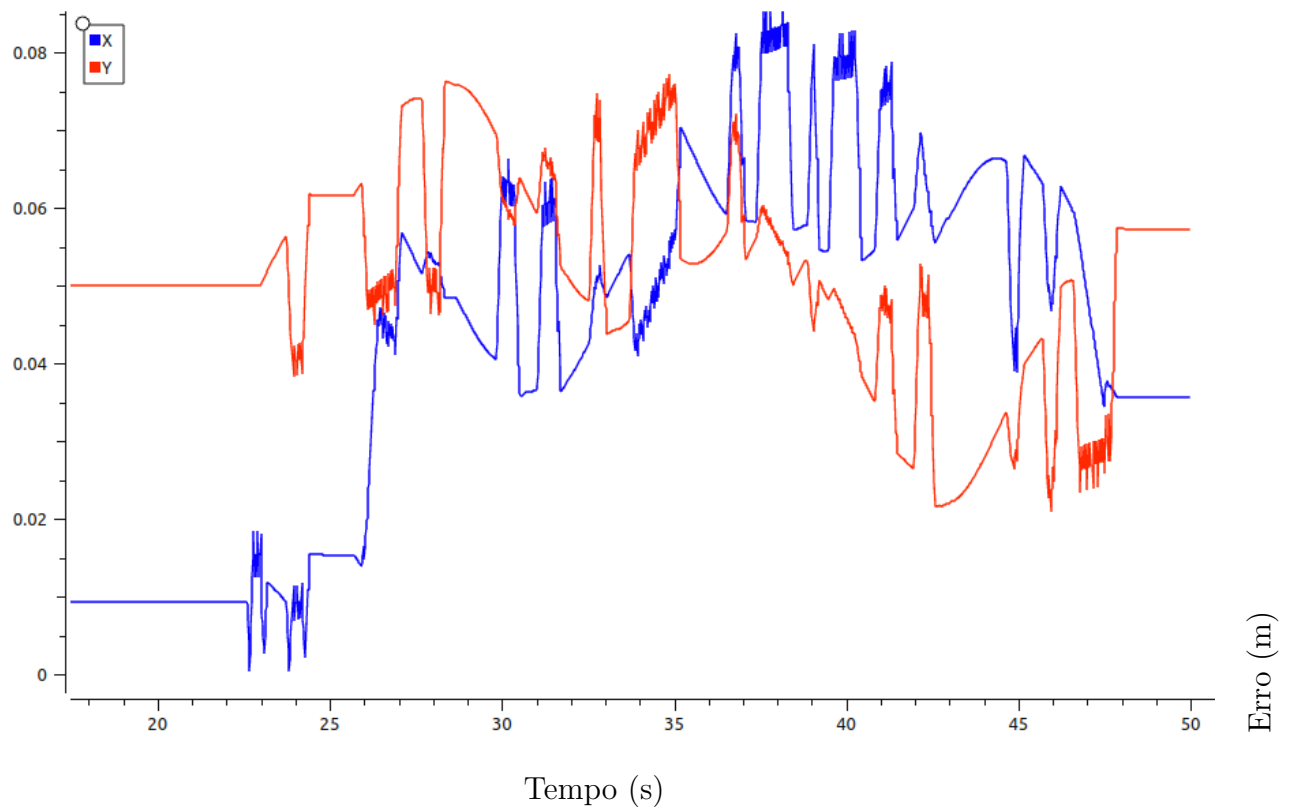


Fonte: Autor.

foi o método combinado de fusão de sensores com AMCL, e em segundo lugar o método AMCL. Já com relação ao consumo de *hardware* e tempo médio de resposta (TMR) o método que apresentou o melhor desempenho foi o de fusão de sensores. Na Tabela 7 é apresentado o desempenho dos métodos para o R2 Square. Pode-se observar que o método combinado de Fusão de sensores com AMCL apresenta o melhor desempenho em ambos os cenários (considerando que para essa métrica a melhor pontuação possível é 100%). Com relação aos métodos MCL e AMCL embora tenham um valor baixo no Score R2, isso não significa que sejam métodos de localização ruins, pois apesar do valor no R2 Square ambos apresentam erros de posição inferiores a 22 cm RMSE. Entende-se que essas porcentagens baixas apenas informam que a relação entre o valor real e o valor previsto não é tão próxima quanto o método de fusão de sensores ou o método combinado de fusão com AMCL.

Comparando os resultados obtidos com a implementação dos métodos e os encontrados na revisão de literatura pode-se observar que o método MCL embora tenha tido o pior resultado entre os métodos analisados com erro máximo de 21 cm, apresentou uma melhor precisão do que [Chikurtev et al. \(2021\)](#) que apresentou erro de $\pm 0,93$ m e [Franchi et al. \(2021\)](#) que apresentou erro de 2,5 m. O método AMCL demonstrou uma precisão similar a

Figura 30 – Erros de posição do método combinado de AMCL e fusão de sensores no Cenário 02



Fonte: Autor.

Belkin, Abramenko e Yudin (2021) que apresentou erro de posição de 10 cm e o método de fusão de sensores apresentou resultados similares a Kayhani et al. (2022) que apresentou erros de 2 a 5 cm e Yilmaz e Temeltas (2019) que teve taxas de erro de 4,74 cm a 6,69 cm. Além disso, a fusão de sensores apresentou um resultado melhor que Oğuz-Ekim (2020), em que os erros variam de 2 cm a 2,2 m. O método combinado de fusão de sensores e AMCL foi o que chegou mais próximo em relação ao desempenho de Guan et al. (2020) que apresentou erros na faixa de 1 cm.

Tabela 7 – Desempenho dos métodos de localização no R2 Square

Método	Cenário	Eixo	R2 Square
MCL	01	x	-11,51%
MCL	01	y	-76,07%
AMCL	01	x	-19,10%
AMCL	01	y	60,14%
Fusão de sensores	01	x	94,22%
Fusão de sensores	01	y	99,69%
Combinação de fusão com AMCL	01	x	95,10%
Combinação de fusão com AMCL	01	y	99,79%
MCL	02	x	-600,00%
MCL	02	y	38,14%
AMCL	02	x	95,56%
AMCL	02	y	8,89%
Fusão de sensores	02	x	81,42%
Fusão de sensores	02	y	52,27%
Combinação de fusão com AMCL	02	x	89,65%
Combinação de fusão com AMCL	02	y	68,20%

Fonte: Autor.

Tabela 8 – Resumo métodos de localização

Nó	CPU (%)	RAM (MB)	TMR (s)	Eixo	Erro Médio C1 (m)	Erro Médio C2 (m)
Localização de Monte Carlo	30	102,40	0,10	x	0,163671	0,145356
				y	0,211272	0,160695
Fusão de Sensores (EKF)	17	32,76	0,05	x	0,048411	0,018865
				y	0,009210	0,067829
AMCL (Nav2)	21	40,96	0,06	x	0,115521	0,055063
				y	0,066626	0,081509
Combinação de AMCL e Fusão de Sensores	33	73,72	0,07	x	0,041003	0,031224
				y	0,009547	0,050768

Fonte: Autor.

5 Considerações Finais e Trabalhos Futuros

Para que um robô móvel autônomo consiga realizar suas tarefas de maneira adequada, é essencial que esse robô possua a capacidade de se localizar no ambiente. Diante desse problema essa dissertação abordou diferentes métodos de localização aplicados a robôs móveis fazendo uma revisão da literatura sobre os métodos mais utilizados, além disso, foram selecionados alguns para serem implementados e comparados utilizando o simulador Webots, o robô e-puck e o ROS 2. Os métodos escolhidos foram: localização de Monte Carlo (MCL), Fusão de Sensores (EKF), Localização Adaptativa de Monte Carlo (AMCL) e o método combinado de AMCL com fusão de sensores. O método que obteve o melhor desempenho com relação aos erros de posição foi o método combinado de fusão de sensores com AMCL com erro médio RMSE de 4 a 5 cm e pontuação de até 99,79% no R2 Square, e em segundo lugar o método de fusão de sensores com erro médio de 4 a 7 cm e pontuação de até 99,69% no R2 Square. Com relação ao consumo de *hardware* o melhor método foi o de fusão de sensores consumindo cerca de 17% de CPU e 32 MB de RAM e em segundo lugar ficou o AMCL com 21% de CPU e 40 MB de RAM.

As principais contribuições desse trabalho são a utilização de diferentes técnicas de localização para robôs móveis, aplicadas em 02 (dois) cenários distintos (Figuras 11 e 13) de modo a possibilitar um estudo comparativo entre elas. A utilização do ROS 2 e a disponibilização dos algoritmos e resultados em um repositório do GitHub ¹, que pode ser utilizado para estudos do simulador Webots, do ROS2 e de técnicas de localização em robótica e a revisão de literatura realizada.

Como trabalhos futuros sugerem-se:

- Análise de desempenho de diferentes métodos de SLAM aplicados ao ROS;
- Análise da implementação no simulador com relação a robôs em um cenário real;
- Combinação das técnicas utilizadas com novos métodos de localização visando melhorar a eficiência.
- Comparação com outros métodos que não foram testados.
- Análise do menor valor necessário de partículas para fazer a estimativa da posição em cada cenário.

¹ <https://github.com/savio-dev/analise_localizacao_ros>

Referências

- ALATISE, M. B.; HANCKE, G. P. A review on challenges of autonomous mobile robot and sensor fusion methods. *IEEE Access*, IEEE, v. 8, p. 39830–39846, 2020.
- BELKIN, I.; ABRAMENKO, A.; YUDIN, D. Real-time lidar-based localization of mobile ground robot. *Procedia Computer Science*, v. 186, p. 440–448, 2021. ISSN 1877-0509. Disponível em: <<https://www.sciencedirect.com/science/article/pii/S1877050921009996>>.
- BESSA, J. A. *Localização de robôs móveis através de reconhecimento de padrões em imagens omnidirecionais*. Dissertação (Mestrado) — Instituto Federal do Ceará / Campus Fortaleza, 2015.
- CAMARA NETO, V.; CAMPOS, M. F. Localização e mapeamento simultâneos em robótica cooperativa com planejamento: Uma revisão. *IX Simpósio Brasileiro de Automação Inteligente (SBAI 2009)*, 2009.
- CAMPBELL, S.; O'MAHONY, N.; CARVALHO, A.; KRPALKOVA, L.; RIORDAN, D.; WALSH, J. Where am i? localization techniques for mobile robots a review. In: IEEE. *2020 6th International Conference on Mechatronics and Robotics Engineering (ICMRE)*. [S.l.], 2020. p. 43–47.
- CHAME, H. F.; SANTOS, M. M. dos; BOTELHO, S. S. da C. Neural network for black-box fusion of underwater robot localization under unmodeled noise. *Robotics and Autonomous Systems*, v. 110, p. 57–72, 2018. ISSN 0921-8890. Disponível em: <<https://www.sciencedirect.com/science/article/pii/S0921889018302926>>.
- CHIKURTEV, D.; CHIVAROV, N.; CHIVAROV, S.; CHIKURTEVA, A. Mobile robot localization and navigation using lidar and indoor gps. *IFAC-PapersOnLine*, v. 54, n. 13, p. 351–356, 2021. ISSN 2405-8963. Disponível em: <<https://www.sciencedirect.com/science/article/pii/S2405896321019091>>.
- CHUNG, M. A.; LIN, C. W. An improved localization of mobile robotic system based on amcl algorithm. *IEEE Sensors Journal*, v. 22, n. 1, p. 900–908, 2022. ISSN 1558-1748. Disponível em: <<https://ieeexplore.ieee.org/document/9606907>>.
- COUSINS, S. Welcome to ros topics [ros topics]. *IEEE Robotics & Automation Magazine*, v. 17, n. 1, p. 13–14, 2010.
- DOBROKOVASHINA, A.; LAVRENOV, R.; BAI, Y.; SVININ, M.; MAGID, E. Sensors modelling for servosila engineer crawler robot in webots simulator. In: IEEE. *2022 Moscow Workshop on Electronic and Networking Technologies (MWENT)*. [S.l.], 2022. p. 1–5.
- DOURADO JUNIOR, C. M.; SILVA, S. P. da; NOBREGA, R. V. da; BARROS, A. C.; SANGAIAH, A. K.; FILHO, P. P. R.; ALBUQUERQUE, V. H. C. de. A new approach for mobile robot localization based on an online iot system. *Future Generation Computer Systems*, Elsevier, v. 100, p. 859–881, 2019.

- FARLEY, A.; WANG, J.; MARSHALL, J. A. How to pick a mobile robot simulator: A quantitative comparison of coppeliasim, gazebo, morse and webots with a focus on accuracy of motion. *Simulation Modelling Practice and Theory*, Elsevier, v. 120, p. 102629, 2022.
- FERREIRA, J. A. B. Redes neurais artificiais aplicadas em aprendizagem de trajetória em robótica móvel. 2020.
- FERREIRA, S. L. C.; GIRARDI, R. Arquiteturas de software baseadas em agentes: do nível global ao detalhado. *Revista Eletrônica de Iniciação Científica da SBC*, 2002.
- FISHER, J. C. A new python api for webots robotics simulations. 2022.
- FRANCHI, M.; BUCCI, A.; ZACCHINI, L.; RIDOLFI, A.; BRESCIANI, M.; PERALTA, G.; COSTANZI, R. Maximum a posteriori estimation for auv localization with usbl measurements**the research leading to these results has been partially supported by the european project eumarinerobots (eumr), which received funding from the european unions horizon 2020 research and innovation program under grant agreement no 731103. *IFAC-PapersOnLine*, v. 54, n. 16, p. 307–313, 2021. ISSN 2405-8963. Disponível em: <<https://www.sciencedirect.com/science/article/pii/S2405896321015111>>.
- GUAN, W.; CHEN, S.; WEN, S.; TAN, Z.; SONG, H.; HOU, W. High-accuracy robot indoor localization scheme based on robot operating system using visible light positioning. *IEEE Photonics Journal*, v. 12, n. 2, p. 1–16, 2020. ISSN 1943-0655. Disponível em: <<https://ieeexplore.ieee.org/ielx7/4563994/9018355/09040589.pdf?tp=&arnumber=9040589&isnumber=9018355&ref=>>>.
- GUNASAGARAN, S.; KAMARUDIN, K.; YEON, A. S. A.; VISVANATHAN, R.; MAMDUH, S. M.; ZAKARIA, A. Implementation of mobile robot localisation and path planning for navigation in known map. *Journal of Telecommunication, Electronic and Computer Engineering*, v. 10, n. 1-15, p. 67–73, 2018. Disponível em: <<https://www.scopus.com/inward/record.uri?eid=2-s2.0-85048596958&partnerID=40&md5=57cb39eea69b1f41361176ca9c6016d7>>.
- HOUSEIN, A. A.; GAO, X. Y.; LI, W. M.; HUANG, Y. Extended kalman filter sensor fusion in practice for mobile robot localization. *INTERNATIONAL JOURNAL OF ADVANCED COMPUTER SCIENCE AND APPLICATIONS*, v. 13, n. 2, p. 33–38, 2022. ISSN 2158-107X 2156-5570 J9 - INT J ADV COMPUT SC.
- JIANG, C.; FAHAD, M.; GUO, Y.; CHEN, Y. Robot-assisted smartphone localization for human indoor tracking. *Robotics and Autonomous Systems*, v. 106, p. 82–94, 2018. ISSN 0921-8890. Disponível em: <<https://www.sciencedirect.com/science/article/pii/S0921889016304201>>.
- JONASSON, E. T.; PINTO, L. R.; VALE, A. Comparison of three key remote sensing technologies for mobile robot localization in nuclear facilities. *Fusion Engineering and Design*, v. 172, p. 112691, 2021. ISSN 0920-3796. Disponível em: <<https://www.sciencedirect.com/science/article/pii/S0920379621004671>>.
- KAYHANI, N.; ZHAO, W.; MCCABE, B.; SCHOELLIG, A. P. Tag-based visual-inertial localization of unmanned aerial vehicles in indoor construction environments using an on-manifold extended kalman filter. *Automation in*

Construction, v. 135, p. 104112, 2022. ISSN 0926-5805. Disponível em: <<https://www.sciencedirect.com/science/article/pii/S092658052100563X>>.

KHAIRUDDIN, A. R.; TALIB, M. S.; HARON, H. Review on simultaneous localization and mapping (slam). In: IEEE. *2015 IEEE international conference on control system, computing and engineering (ICCSC)*. [S.l.], 2015. p. 85–90.

KING, E. A.; TATOGLU, A.; IGLESIAS, D.; MATRISS, A. Audio-visual based non-line-of-sight sound source localization: A feasibility study. *Applied Acoustics*, v. 171, p. 107674, 2021. ISSN 0003-682X. Disponível em: <<https://www.sciencedirect.com/science/article/pii/S0003682X20307787>>.

LANCHEROS, P. N.; SANABRIA, L. B.; CASTILLO, R. A. Simulation of modular robotic system mecabot in caterpillar and snake configurations using webots software. In: IEEE. *2016 IEEE Colombian Conference on Robotics and Automation (CCRA)*. [S.l.], 2016. p. 1–6.

LI, B.; MI, Z.; GUO, Y.; YANG, Y.; OBAIDAT, M. S. A high efficient multi-robot simultaneous localization and mapping system using partial computing offloading assisted cloud point registration strategy. *Journal of Parallel and Distributed Computing*, v. 149, p. 89–102, 2021. ISSN 0743-7315. Disponível em: <<https://www.sciencedirect.com/science/article/pii/S0743731520304019>>.

MA, Z.; LIANG, Y.; TIAN, H. Research on gait planning algorithm of quadruped robot based on central pattern generator. In: IEEE. *2020 39th Chinese Control Conference (CCC)*. [S.l.], 2020. p. 3948–3953.

MACENSKI, S.; FOOTE, T.; GERKEY, B.; LALANCETTE, C.; WOODALL, W. Robot operating system 2: Design, architecture, and uses in the wild. *Science Robotics*, v. 7, n. 66, p. eabm6074, 2022. Disponível em: <<https://www.science.org/doi/abs/10.1126/scirobotics.abm6074>>.

MACENSKI, S.; MARTÍN, F.; WHITE, R.; CLAVERO, J. G. The marathon 2: A navigation system. In: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. [s.n.], 2020. Disponível em: <<https://github.com/ros-planning/navigation2>>.

MARCHI, J. et al. Navegação de robôs móveis autônomos: estudo implementação de abordagens. Florianópolis, SC, 2001.

NAKHAEINIA, D.; TANG, S. H.; NOOR, S. M.; MOTLAGH, O. A review of control architectures for autonomous navigation of mobile robots. *International Journal of the Physical Sciences*, v. 6, n. 2, p. 169–174, 2011.

NILOY, A.; SHAMA, A.; CHAKRABORTTY, R. K.; RYAN, M. J.; BADAL, F. R.; TASNEEM, Z.; AHAMED, M. H.; MOYEEN, S. I.; DAS, S. K.; ALI, M. F. et al. Critical design and control issues of indoor autonomous mobile robots: A review. *IEEE Access*, IEEE, 2021.

ODEBRECHT, V. A. et al. Otimização de usinagem robótica através de análises estatísticas. Florianópolis, SC., 2020.

- OISHI, S.; INOUE, Y.; MIURA, J.; TANAKA, S. Seqslam++: View-based robot localization and navigation. *Robotics and Autonomous Systems*, v. 112, p. 13–21, 2019. ISSN 0921-8890. Disponível em: <<https://www.sciencedirect.com/science/article/pii/S092188901730684X>>.
- OğUZ-EKIM, P. Tdoa based localization and its application to the initialization of lidar based autonomous robots. *Robotics and Autonomous Systems*, v. 131, p. 103590, 2020. ISSN 0921-8890. Disponível em: <<https://www.sciencedirect.com/science/article/pii/S0921889020304309>>.
- PANIGRAHI, P. K.; BISOY, S. K. Localization strategies for autonomous mobile robots: A review. *Journal of King Saud University-Computer and Information Sciences*, Elsevier, 2021.
- PEEL, H.; LUO, S.; COHN, A. G.; FUENTES, R. Localisation of a mobile robot for bridge bearing inspection. *Automation in Construction*, v. 94, p. 244–256, 2018. ISSN 0926-5805. Disponível em: <<https://www.sciencedirect.com/science/article/pii/S0926580517303916>>.
- PIERI, E. R. de. Curso de robótica móvel. *Santa Catarina, Florianópolis*, 2002.
- PIO, J. L. de S.; CASTRO, T. H. C. de; CASTRO JÚNIOR, A. N. de. A robótica móvel como instrumento de apoio à aprendizagem de computação. In: *Brazilian Symposium on Computers in Education (Simpósio Brasileiro de Informática na Educação-SBIE)*. [S.l.: s.n.], 2006. v. 1, n. 1, p. 497–506.
- RIZZO, C.; SECO, T.; ESPELOSIN, J.; LERA, F.; VILLARROEL, J. L. An alternative approach for robot localization inside pipes using rf spatial fadings. *Robotics and Autonomous Systems*, v. 136, p. 103702, 2021. ISSN 0921-8890. Disponível em: <<https://www.sciencedirect.com/science/article/pii/S092188902030542X>>.
- ROS. *Documentação Oficial*. 2022. <<https://docs.ros.org/en/foxy/Tutorials/Beginner-CLI-Tools.html>>.
- SAEEDVAND, S.; AGHDASI, H. S.; BALTES, J. Novel lightweight odometric learning method for humanoid robot localization. *Mechatronics*, v. 55, p. 38–53, 2018. ISSN 0957-4158. Disponível em: <<https://www.sciencedirect.com/science/article/pii/S0957415818301338>>.
- SANTANA, A. M. Localização e mapeamento simultâneos de ambientes planos usando visão monocular e representação híbrida do ambiente. Universidade Federal do Rio Grande do Norte, 2011.
- SANTOS, P. C. et al. Arquitetura de subsunção baseada em objetivo de controle principal. Universidade Federal de Sergipe, 2017.
- SHAO, X.; ZHANG, J.; ZHANG, W. Distributed cooperative surrounding control for mobile robots with uncertainties and aperiodic sampling. *IEEE Transactions on Intelligent Transportation Systems*, IEEE, 2022.
- SIEGWART, R.; NOURBAKHSI, I. R.; SCARAMUZZA, D. *Introduction to autonomous mobile robots*. [S.l.]: MIT press, 2011.

- SILVA, V. D.; ROCHE, J.; KONDOZ, A. Robust fusion of lidar and wide-angle camera data for autonomous mobile robots. *Sensors*, Multidisciplinary Digital Publishing Institute, v. 18, n. 8, p. 2730, 2018.
- SULLIVAN, N.; GRAINGER, S.; CAZZOLATO, B. Analysis of cooperative localisation performance under varying sensor qualities and communication rates. *Robotics and Autonomous Systems*, v. 110, p. 73–84, 2018. ISSN 0921-8890. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0921889018301969>.
- THRUN, S.; BURGARD, W.; FOX, D. Probabilistic robotics. *Kybernetes*, Emerald Group Publishing Limited, 2006.
- TZAFESTAS, S. G. Mobile robot control and navigation: A global overview. *Journal of Intelligent & Robotic Systems*, Springer, v. 91, n. 1, p. 35–58, 2018.
- XIAO, L.; WANG, J.; QIU, X.; RONG, Z.; ZOU, X. Dynamic-slam: Semantic monocular visual localization and mapping based on deep learning in dynamic environment. *Robotics and Autonomous Systems*, v. 117, p. 1–16, 2019. ISSN 0921-8890. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0921889018308029>.
- YILMAZ, A.; TEMELTAS, H. Self-adaptive monte carlo method for indoor localization of smart agvs using lidar data. *Robotics and Autonomous Systems*, v. 122, p. 103285, 2019. ISSN 0921-8890. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0921889019302106>.

6 Apêndice

6.1 Método utilizado para a revisão de literatura

A revisão de literatura foi realizada nas plataformas Web of Science, Science Direct e IEEE Xplore e Scopus. Para efetuar a busca nas plataformas foram utilizadas palavras-chave com uma combinação de termos referentes a localização de robôs móveis e adaptadas a sintaxe de cada um dos mecanismos de buscas: ("mobile robots") AND ("robot localization") AND (("robot operating system") OR ("ROS")).

O levantamento incluiu artigos publicados nos últimos 5 anos até 10 de Outubro de 2022.

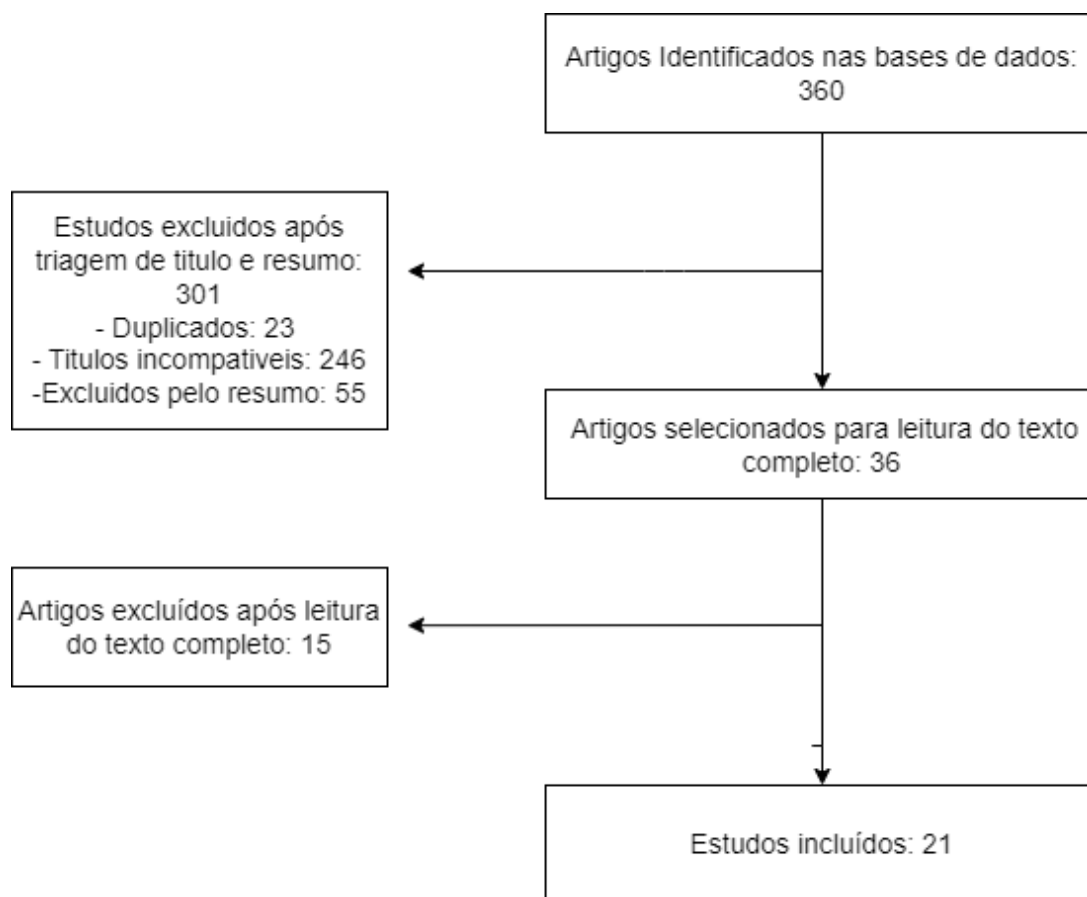
Foram pesquisados artigos em inglês que apresentassem técnicas de localização de robôs móveis com o ROS, artigos com foco exclusivo em navegação foram excluídos. Além disso, quando observado dois trabalhos sobre o mesmo tema apenas o mais recente foi incluído.

Inicialmente, 360 artigos foram identificados, sendo 2 na Base IEEE, 7 no Web of Science, 79 no Science Direct e 272 no Scopus.

Após revisão dos títulos e resumos, foram excluídos 339 artigos, restando 21 para leitura do texto completo, conforme representado na Figura 31. Nesse refinamento inicial, 23 trabalhos foram excluídos por estarem duplicados nas bases de dados, 280 por incompatibilidade de títulos e 36 após leitura do resumo.

Dos 47 títulos que permaneceram para leitura do texto completo, foram excluídos 24 deles por não apresentarem uma técnica para localização de robôs móveis ou por apresentar informações confusas ou incompletas. Os 21 trabalhos selecionados foram analisados e comparados seguindo alguns critérios descritos na Tabela 1.

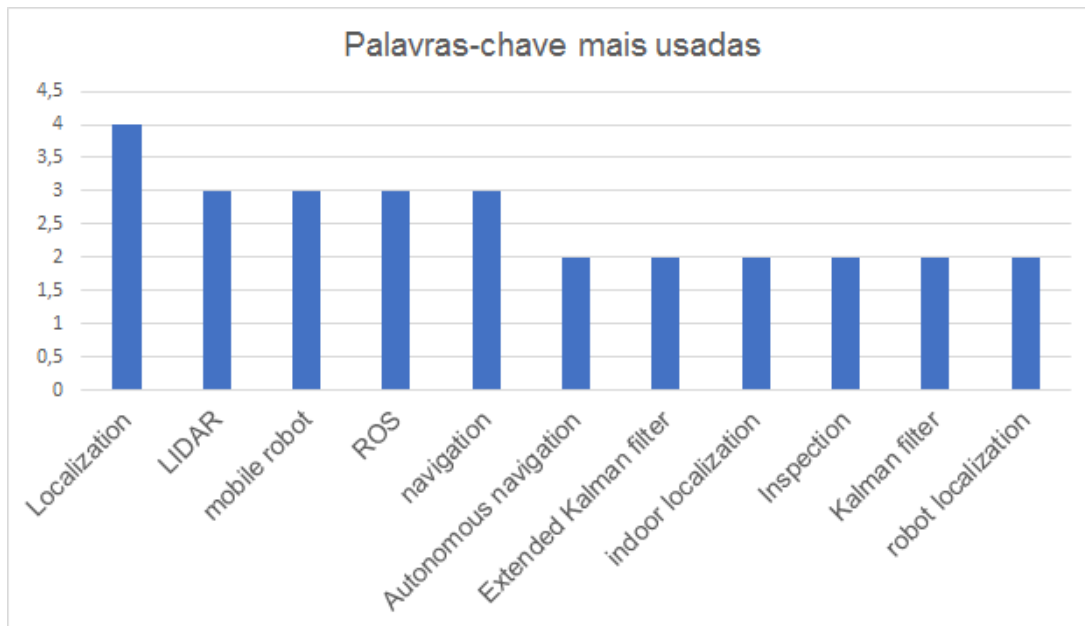
Figura 31 – Processo de identificação, triagem e seleção de estudos para revisão sistemática



Fonte: Autor.

Considerando os artigos selecionados foi realizado uma análise estatística das palavras-chave mais utilizadas e que podem ser usadas para realizar buscas com melhores resultados, conforme pode-se observar na Figura 32

Figura 32 – Lista de palavras-chave mais usadas nos artigos



Fonte: Autor.

6.2 Instalação do ROS

O ROS é lançado como distribuições ou "distros", a diferença entre elas ocorre com relação a estabilidade, adição de novos recursos e/ou compatibilidade com plataformas e linguagens de programação. Atualmente o ROS 1 LTS é suportado somente no Ubuntu Linux e o ROS 2 é suportado no Linux, MacOS ou Windows 10. O ROS 2 foi escolhido por ser a versão mais nova e apesar de existir uma versão mais atual (Galactic Geochelone), a versão Foxy Fitzroy apresenta melhor estabilidade por ser a versão LTS e ter passado por mais testes.

A instalação do ROS no ubuntu linux pode ser realizada de duas formas, sendo compilado a partir do código fonte ou a partir de pacotes binários pré-compilados. O método mais recomendado pela própria documentação do ROS é a partir dos pacotes binários, pois é mais prático, já que instala as dependências automaticamente e facilita a atualização.

Para iniciar a instalação é necessário adicionar os repositórios APT (*Advanced Package Tool*) do ROS 2 ao seu sistema. Para fazer isso, primeiro autorize a chave GPG com o apt:

```
sudo apt update && sudo apt install curl gnupg2 lsb-release sudo  
curl -sSL https://raw.githubusercontent.com/ros/rosdistro/master/ros.key -o  
/usr/share/keyrings/ros-archive-keyring.gpg
```

E, em seguida, adicione o repositório à sua lista de fontes:

```
echo "deb [arch=$(dpkg --print-architecture)]signed-by=/usr/share/keyrings/ros-
archive-keyring.gpg]http://packages.ros.org/ros2/ubuntu(source /etc/os-release
&& echo $UBUNTU_CODENAME) main sudo tee /etc/apt/sources.list.d/ros2.list
> /dev/null
```

Atualize seus caches de repositório apt depois de configurar os repositórios.

```
sudo apt update
```

e instale a versão desktop do ROS, que inclui ROS, RViz, demonstrações e tutoriais.

```
sudo apt install ros-foxy-desktop
```

Também é possível instalar somente a versão base com bibliotecas de comunicação, pacotes de mensagens, ferramentas de linha de comando. Porém, sem nenhuma ferramenta GUI (Interface Gráfica de Usuário).

```
sudo apt install ros-foxy-ros-base
```

Antes de iniciar o ROS é necessário configurar o ambiente no terminal com o seguinte comando:

```
source /opt/ros/foxy/setup.bash
```

Porém caso não deseje realizar essa configuração sempre que iniciar um novo terminal, é possível adicionar esse comando para ser inicializado automaticamente da forma a seguir:

```
echo "source /opt/ros/foxy/setup.bash" > ~/.bashrc
```

6.3 ROS 2 Navigation Stack

O ROS 2 Navigation Stack (Também conhecido como Nav2) é um framework de navegação gratuito e de código aberto que auxilia na busca de encontrar uma maneira segura de fazer com que um robô móvel se mova do ponto A para o ponto B. De acordo com [Macenski et al. \(2020\)](#), o Navigation 2 usa uma árvore de comportamento para orquestrar tarefas do navegador e emprega novos métodos projetados para ambientes dinâmicos aplicáveis a uma ampla variedade de sensores modernos. Para realizar a instalação do Nav2 a partir dos arquivos pré-compilados deve ser utilizado o seguinte comando:

```
sudo apt install ros-<distro>-navigation2 ros-<distro>-nav2-bringup
ros-<distro>-turtlebot3*
```

onde <distro> deve ser substituído pelo nome da versão do ROS que está sendo utilizada, como no caso desse trabalho substituímos por foxy, referente a versão do ROS Foxy Fitzroy. No caso da instalação ser feita a partir do código fonte deve ser usado os comandos abaixo:

```
mkdir -p /nav2_ws/src
cd /nav2_ws/src
git clone https://github.com/ros-planning/navigation2.git --branch <ros2-
distro>-devel cd /nav2_ws
rosdep install -y -r -q --from-paths src --ignore-src --rosdistro <ros2-
distro>
colcon build --symlink-install
```

onde <ros2-distro> deve ser substituído pelo nome da versão do ROS, será criada uma pasta para o workspace(ambiente de trabalho), em seguida feito o download dos arquivos necessários e o pacote será compilado através do colcon build (a mesma ferramenta utilizada para compilação dos demais pacotes no ROS). Para a configuração das variáveis de ambiente, adicione os comandos abaixo ao final do arquivo bash ou insira-os no terminal:

```
source /nav2_ws/install/setup.bash
export TURTLEBOT3_MODEL=waffle
export GAZEBO_MODEL_PATH=$GAZEBO_MODEL_PATH:
/opt/ros/foxy/share/turtlebot3_gazebo/models
source /usr/share/gazebo/setup.sh
```

Para testar se a instalação ocorreu sem problemas, utilize o comando abaixo e será iniciado o Rviz2 juntamente com o simulador Gazebo com um projeto de exemplo.

```
cd /nav2_ws ros2 launch nav2_bringup tb3_simulation_launch.py
```

6.4 Webots ROS2 Interface

webots_ros2 é um pacote que fornece as interfaces necessárias para simular um robô no simulador Webots. Ele se integra ao ROS2 usando mensagens, serviços e ações do ROS2. Para realizar a instalação desse pacote insira o comando a seguir no terminal:

```
sudo apt-get install ros-$ROS_DISTRO-webots-ros2
```

Para verificar se a instalação ocorreu com sucesso execute um projeto de exemplo:

```
ros2 launch webots_ros2_universal_robot multirobot_launch.py
```

Para visualizar tutoriais e exemplos de utilização da interface webots ros2, acesse o repositório oficial:

<https://github.com/cyberbotics/webots_ros2/>

6.5 Robot_Localization

O Robot_Localization é uma coleção de estimadores de estado não linear para robôs que se movem no espaço 3D (ou 2D). Cada um dos estimadores de estado pode fundir um número arbitrário de sensores (IMUs, hodômetros, sistemas de localização interna, receptores GPS...)

Para a instalação desse pacote utilize o seguinte comando:

```
sudo apt install ros-foxy-robot-localization
```

A documentação oficial está disponível em:

[<http://docs.ros.org/en/noetic/api/robot_localization/html/index.html>](http://docs.ros.org/en/noetic/api/robot_localization/html/index.html)

6.6 PlotJuggler

PlotJuggler é uma ferramenta para visualizar séries temporais que é rápida, poderosa e intuitiva.

Recursos notáveis:

- Interface de usuário simples de arrastar e soltar.
- Carregar dados de arquivo (CSV, JSON, CBOR, BSON).
- Conecte-se à transmissão ao vivo de dados.
- Salve o layout e as configurações de visualização para reutilizá-los posteriormente.
- Pode lidar com milhares de séries temporais e milhões de pontos de dados.
- Bem integrado com o ROS: abra rosbags e/ou assine os tópicos do ROS (tanto ROS1 quanto ROS2).
- Pode ser facilmente estendido usando plugins.

Para instalar utilize o seguinte comando:

```
sudo apt install ros-$ROS_DISTRO-plotjuggler-ros
```

Para iniciar o PlotJuggler no ROS, use o comando:

```
ros2 run plotjuggler plotjuggler
```