UNIVERSIDADE DE CAXIAS DO SUL
ÁREA DO CONHECIMENTO DE CIÊNCIAS EXATAS E ENGENHARIAS

ENZO BUSA TRENTINI

DESENVOLVIMENTO DE UMA APLICAÇÃO ROS2 PARA ROBÔS MÓVEIS AMR: SIMULAÇÃO E APLICAÇÃO VIRTUAL

CAXIAS DO SUL

2024

ENZO BUSA TRENTINI

DESENVOLVIMENTO DE UMA APLICAÇÃO ROS2 PARA ROBÔS MÓVEIS AMR: SIMULAÇÃO E APLICAÇÃO VIRTUAL

Trabalho de Conclusão de Curso apresentado como requisito parcial à obtenção do título de Engenheiro de Controle e Automação na Área do Conhecimento de Ciências Exatas e Engenharias da Universidade de Caxias do Sul.

Orientador: Prof. Me. Daniel Corteletti

CAXIAS DO SUL

2024

ENZO BUSA TRENTINI

DESENVOLVIMENTO DE UMA APLICAÇÃO ROS2 PARA ROBÔS MÓVEIS AMR: SIMULAÇÃO E APLICAÇÃO VIRTUAL

Trabalho de Conclusão de Curso apresentado como requisito parcial à obtenção do título de Engenheiro de Controle e Automação na Área do Conhecimento de Ciências Exatas e Engenharias da Universidade de Caxias do Sul.

Aprovado em 29/11/2024

BANCA EXAMINADORA

_____

Prof. Me. Daniel Corteletti
Universidade de Caxias do Sul - UCS

_____

Prof. Dr. André Luis Martinotto
Universidade de Caxias do Sul - UCS

_____

Prof. Me. Ricardo Leal Costi
Universidade de Caxias do Sul - UCS

AGRADECIMENTOS

Agradeço aos meus pais, Ronei e Marisa, à minha querida "Nonna"Alzira, à dinda Maristela e ao meu irmão Franco, que não mediram esforços para me ajudar nesta etapa tão importante da minha vida, seja de maneira direta ou indireta, sempre mantendo-me em seus pensamentos. A eles, devo todas as minhas conquistas.

Sou grato aos meus amigos e colegas, que me incentivaram todos os dias e ofereceram apoio nos momentos críticos de dificuldade. A amizade é uma virtude que fortalece o espírito e encoraja a enfrentar qualquer desafio.

Agradeço também ao Professor Daniel Corteletti, meu orientador, por suas inúmeras dicas, sessões de mentoria, troca de conhecimento e pela amizade durante toda a minha jornada acadêmica.

E, por último, um agradecimento especial à minha namorada Isadora por, em inúmeras vezes, compreender a situação, sempre me incentivando a ser a minha melhor versão, com seu olhar atencioso e paciente.

“O medo faz parte da vida da gente. Algumas pessoas não sabem como enfrentá-lo. Outras, acho que estou entre elas, aprendem a conviver com ele e o encaram não de forma negativa, mas como um sentimento de autopreservação .”

Ayrton Senna

RESUMO

Este estudo explora o desenvolvimento virtual de um Robô Móvel Autônomo (AMR), fundamentado no Sistema Operacional Robot Operating System 2 (ROS2), com o objetivo de validar essa plataforma como uma ferramenta robusta para modelagem e controle de robôs industriais. A simulação é realizada em um ambiente industrial de forjaria, composto por diversos setores que demandam movimentação autônoma de cargas. A simulação desempenha um papel central, permitindo a verificação e o teste do modelo de *AMR* em ambientes virtuais, onde métodos de simulação realista foram empregados para replicar com precisão as condições do setor de forjaria escolhido, assegurando, assim, a eficácia e confiabilidade do sistema. O processo de desenvolvimento foi estruturado em etapas, começando pela modelagem do *AMR* e sua cinemática, seguido pelo desenvolvimento e adaptação de algoritmos de controle e planejamento de trajetória, por fim integrando sensores ao sistema, fornecendo informações sobre o ambiente circundante. Os resultados demonstraram-se precisos, confirmando que o *AMR* é capaz de movimentar-se de um ponto a outro de maneira totalmente autônoma, sem a necessidade de interação ou controle manual, sendo que o robô foi capaz de detectar e desviar de obstáculos, criando uma rota eficiente de forma independente. Assim, este trabalho contribui para o avanço da robótica, evidenciando como o *ROS2* pode ser aplicado no desenvolvimento de robôs móveis e consolidando seu papel como uma plataforma viável e eficiente para aplicações em ambientes industriais.

Palavras-chave: AMR, ROS 2.

ABSTRACT

This study explores the virtual development of an Autonomous Mobile Robot (AMR), based on the Robot Operating System 2 (ROS2), with the aim of validating this platform as a robust tool for modeling and controlling industrial robots. The simulation is carried out in an industrial forging environment, made up of various sectors that require autonomous handling of loads. Simulation plays a central role, allowing the verification and testing of the *AMR* model in virtual environments, where realistic simulation methods were employed to accurately replicate the conditions of the chosen forging sector, thus ensuring the effectiveness and reliability of the system. The development process was structured in stages, starting with the modeling of the *AMR* and its kinematics, followed by the development and adaptation of control algorithms and trajectory planning, finally integrating sensors into the system, providing information about the surrounding environment. The results proved to be accurate, confirming that the *AMR* is capable of moving from one point to another completely autonomously, without the need for interaction or manual control, and the robot was able to detect and dodge obstacles, creating an efficient route independently. Thus, this work contributes to the advancement of robotics, showing how *ROS2* can be applied in the development of mobile robots and consolidating its role as a viable and efficient platform for applications in industrial environments.

Translated with DeepL.com (free version)

Keywords: AMR, ROS 2.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE ALGORITMOS

LISTA DE ABREVIATURAS E SIGLAS

AGV *Automated Guided Vehicle*

AMCL *Adaptive Monte-Carlo Localizer*

AMR *Autonomous Mobile Robots*

API *Application Programming Interface*

BT *Behavior Trees*

CLP Controlador Lógico Programável

DART *Dynamic Animation and Robotics Toolkit*

DDS *Data Distribution Service*

EKF *Extended Kalman filter*

FBP *Flow-Based Programming*

GIMP *GNU Image Manipulation Program*

HTTP Protocolo de Transferência de Hipertexto

IA Inteligência Artificial

IMU *Inertial measurement unit*

IoT *Internet of Things*

KF *Kalman Filter*

LiDAR *Light Detection and Ranging*

LM *Lean Manufacturing*

LTS *Long-Term Support*

MES *Manufacturing Execution System*

MQTT *Message Queuing Telemetry Transport*

PGM *Portable Gray Map*

POO Programação Orientada a Objetos

RCL *ROS Client Layer*

ROS *Robot Operating System*

ROS2 *Robot Operating System 2*

RViz *ROS Visualization*

SLAM *Simultaneous Localization and Mapping*

SO Sistema Operacional

UWB *Ultra-Wideband*

URDF *Unified Robot Description Format*

WLAN *Wide Local Area Network*

WS *Work Space*

XACRO *XML Macros*

XML *Extensible Markup Language*

SUMÁRIO

1 INTRODUÇÃO

O desenvolvimento acelerado de tecnologias voltadas ao aumento da produtividade impõe novos desafios em diversos setores industriais, caracterizados por uma concorrência intensificada devido ao acesso ampliado à informação e a um número crescente de fornecedores. Consequentemente, as empresas são pressionadas a otimizar suas cadeias produtivas, buscando se diferenciar de concorrentes ao reduzir custos e garantir sustentabilidade financeira (PALANGE; DHATRAK, 2021).

Entre os principais desperdícios apontados pelo sistema *Lean Manufacturing* (LM) estão aqueles decorrentes da superprodução, espera, movimentações e transportes desnecessários, processamento excessivo, acúmulo de inventário, defeitos e subutilização de colaboradores (WAHAB; MUKHTAR; SULAIMAN, 2013). Esses desperdícios representam custos adicionais ao produto final e constituem um desafio relevante para empresas que buscam maximizar a eficiência e minimizar perdas. Nesse cenário, a automação surge como uma importante aliada ao *LM*, fornecendo soluções que mitigam os desperdícios e aumentam a produtividade. A aplicação de tecnologias automatizadas permite realizar atividades logísticas e operacionais de maneira mais ágil e precisa, contribuindo para um fluxo de trabalho mais enxuto e reduzindo a variabilidade nos processos (SOROUSH; BALDEA; EDGAR, 2020).

Especificamente, o uso de tecnologias para otimização da movimentação de cargas, como os veículos guiados automaticamente (*Automated Guided Vehicle* (AGV)) e, mais recentemente, os robôs móveis autônomos (*Autonomous Mobile Robots* (AMR)), emerge como uma solução estratégica para operações logísticas internas mais complexas. A adoção de robôs AMR aumenta a eficiência do processo como um todo, reduzindo significativamente as perdas por espera e minimizando riscos de acidentes com colaboradores humanos (VORONOVA, 2022).

Entre as tecnologias que viabilizam o uso eficiente de AMR, destaca-se o *Robot Operating System 2* (ROS2), uma plataforma que oferece uma infraestrutura de comunicação robusta, essencial para operação em ambientes dinâmicos e integrados em tempo real. O ROS2, com sua arquitetura modular e capacidade de comunicação distribuída, permite integração fluida entre sensores e atuadores de fabricantes variados, facilitando ajustes rápidos e expansões no sistema para atender às demandas variáveis do ambiente industrial (KOUBAA, 2020). Além disso, a segurança e confiabilidade do ROS2, associadas ao suporte oferecido por sua comunidade ativa, tornam essa plataforma uma ferramenta ideal para o desenvolvimento de soluções robóticas voltadas à eficiência e segurança em aplicações industriais.

Essas características fazem do ROS2 uma base tecnológica que agrega valor ao desenvolvimento de *AMRs*, permitindo que as empresas atendam às demandas de produtividade e competitividade, além de promoverem operações logísticas mais seguras e eficientes.

1.1 JUSTIFICATIVA

Em um cenário competitivo, onde diversos fabricantes oferecem soluções similares, o desenvolvimento de uma solução para um robô *AMR* baseado em *ROS2* apresenta vantagens significativas em termos de flexibilidade e escalabilidade. A característica de código livre permite realizar adaptações e melhorias contínuas no projeto, o que seria inviável em uma situação onde a solução fosse adquirida de um fornecedor, sujeito a restrições de propriedade intelectual e direitos autorais. Nessas condições, o cliente se vê limitado em suas possibilidades de alterar o escopo original, devido às barreiras impostas pelo fornecedor.

Além disso, a proposta de um *AMR* fundamentado em tecnologias amplamente estabelecidas no mercado traz consigo uma série de benefícios, como a vasta disponibilidade de documentação e bibliotecas padronizadas que agregam funcionalidades ao projeto. Esse acesso facilita o desenvolvimento, permitindo que o foco seja direcionado à integração de novas tecnologias e à melhoria de processos, em vez de destinar recursos ao desenvolvimento de soluções que já estão consolidadas no mercado, seguindo o princípio de evitar o desenvolvimento redundante de ferramentas já disponíveis e amplamente utilizadas.

Somado a este fator, o uso de simulação em projetos de robótica móvel tem-se mostrado uma abordagem essencial e altamente vantajosa, sobretudo no contexto de redução de custos e otimização de recursos durante o desenvolvimento. A construção de protótipos físicos representa uma das etapas mais dispendiosas desse processo, exigindo componentes de alta precisão e mão de obra especializada, gerando elevados custos financeiros e de tempo. Através da utilização de ferramentas de simulação, é possível testar múltiplos aspectos do sistema, incluindo cinemática, resposta a comandos e interação com o ambiente, de forma totalmente virtual, reduzindo a necessidade de protótipos físicos em fases iniciais do projeto. Craig (2014) destaca que essa capacidade de replicar o ambiente de operação no mundo virtual aumenta a segurança do desenvolvimento e a confiabilidade dos testes, fornecendo uma base sólida para a implementação do sistema em condições reais.

Outro benefício importante do uso de simulação é a possibilidade de testar algoritmos de navegação, controle de movimento e planejamento de trajetória em um ambiente virtual antes de sua implementação em um robô real. Isso permite o ajuste preciso dos parâmetros do sistema em um espaço seguro e controlado, sem riscos de danos ao equipamento, mitigando erros de desenvolvimento. Dessa forma, o número de iterações de prototipagem diminui significativamente, resultando em economia de materiais e tempo de produção (CRAIG, 2014).

1.2 OBJETIVOS

Nesta seção, serão descritos os objetivos gerais e específicos deste trabalho:

1.2.1 Objetivo Geral

Este trabalho tem como objetivo detalhar o desenvolvimento de um sistema *AMR* baseado no sistema operacional *ROS2*, abrangendo desde a concepção até os métodos de programação e simulação em um contexto industrial, com a finalidade de validar essa ferramenta como alternativa viável para soluções robóticas.

1.2.2 Objetivos Específicos

- Dimensionar uma estrutura para um *AMR* de pequeno porte, que atenda os requisitos do trabalho, para fins de testes e validação em *softwares* de simulação;
- Desenvolver um arquivo para descrever de maneira digital as estruturas do robô, como eixos e juntas;
- Desenvolver *software* para controle de todas as instâncias do *AMR*, baseado em *ROS2*;
- Criar ambiente virtual que reproduza um espaço de trabalho industrial, buscando uma fidelidade para os testes;
- Adaptar sistemas de controle, mapeamento e navegação para serem utilizados com o ambiente e o modelo *AMR* criado;
- Integrar todas as ferramentas utilizadas através de uma aplicação única;
- Simular de forma computacional o modelo desenvolvido para validação de seu funcionamento;
- Elaborar uma interface virtual para monitoração remota do *AMR*.

1.3 ESTRUTURA DO TRABALHO

Este trabalho está organizado da seguinte maneira:

- O Capítulo 2 apresenta todos os conceitos utilizados neste trabalho, buscando uma descrição dos componentes abordados.
- O Capítulo 3 apresenta o desenvolvimento deste trabalho, descrevendo as etapas necessárias para que o objetivo proposto seja atingido.
- O Capítulo 4 apresenta os resultados encontrados a partir do projeto desenvolvido, assim como uma análise dos mesmos.
- O Capítulo 5 apresenta as conclusões finais do trabalho e sugestões de trabalhos futuros.

1.4 LIMITAÇÕES DO TRABALHO

Devido a fatores externos impostos ao autor durante o decorrer deste projeto, o presente trabalho limita-se à implementação e simulação de forma virtual do modelo *AMR* , abstendo-se da implementação em meios físicos. Essa aproximação, ao substituir a construção de um modelo físico, permite a utilização e desenvolvimento de diversas outras tecnologias, como métodos de localização e comunicação com sistemas supervisórios para controle e monitoração, fatores sobre os quais talvez não seria possível discorrer em um trabalho focado na construção de um modelo físico de um robô *AMR*.

2 REFERENCIAL TEÓRICO

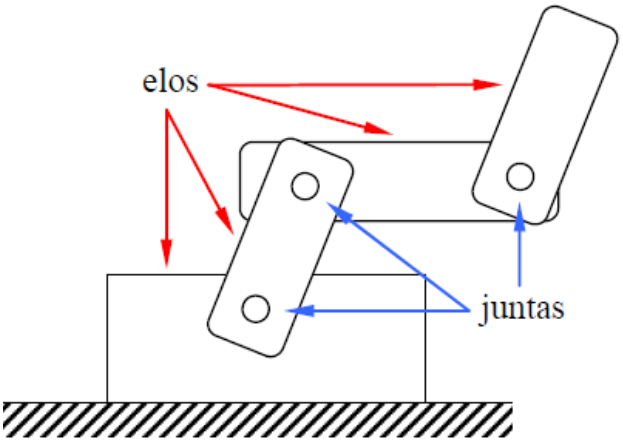
2.1 ROBÓTICA INDUSTRIAL

A robótica industrial, caracterizada pela introdução de eficiência, precisão e automação em larga escala, representa uma das revoluções mais significativas na manufatura moderna. A integração de sistemas robóticos transformou operações antes baseadas no trabalho manual intensivo. Com capacidade de realizar tarefas repetitivas com alta precisão e velocidade, esse campo da robótica contribui não apenas para a otimização da produção, mas também para a segurança dos trabalhadores, especialmente em ambientes perigosos ou insalubres.

Esse campo de estudo abrange várias áreas como mecânica e computação, teoria de controle, eletrônica e, mais recentemente, sistemas de Inteligência Artificial (IA), que permitem criar mecanismos robóticos com configurações específicas para atender a uma ampla gama de aplicações (CARRARA, 2015).

Um exemplo de robô industrial é o braço robótico, também chamado de manipulador, conforme ilustrado na Figura 1. As partes mecânicas, conhecidas como elos, são conectadas por juntas, que podem ser rotativas, prismáticas, cilíndricas, esféricas, entre outras (CARRARA, 2015). A última junta do braço é conhecida como punho, onde são acopladas ferramentas auxiliares, denominadas *End Effectors*.

Figura 1 – Anatomia de um manipulador robótico

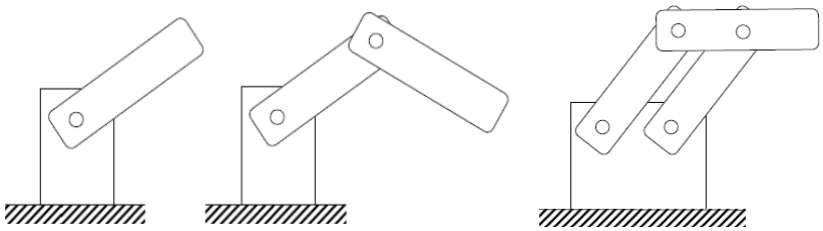


Fonte: Carrara (2015).

O número de juntas define os graus de liberdade (*GL*) de um braço robótico, enquanto sua disposição determina sua cadeia cinemática, o que é essencial para o correto dimensionamento de qualquer solução robótica. Esse dimensionamento considera a capacidade de carga (*payload*) que o braço pode manipular sem comprometer sua precisão.

Conforme Carrara (2015), manipuladores robóticos podem apresentar cadeias de formato aberto, quando há apenas um caminho possível para chegar ao punho, partindo da base. Em uma disposição de cadeia fechada, existem várias formas de percorrer esse caminho, com elos conectados de múltiplas maneiras. Existe ainda uma terceira configuração, parcialmente fechada ou parcialmente aberta. Essas cadeias estão representadas na Figura 2.

Figura 2 – Braços com cadeia aberta (à esquerda) e parcialmente fechada (à direita)



Fonte: Adaptado de Carrara (2015).

2.1.1 Automated Guided Vehicle (AGV)

Além dos modelos tradicionais de braços robóticos, como os de 6 eixos e suas variações, o desenvolvimento de técnicas de controle possibilitou a criação de novas estruturas robóticas. Em setores que demandam grande movimentação de cargas, como centros de distribuição, destacam-se os AGVs, sigla em inglês para Veículos Guiados Autonomamente.

Embora robôs de 6 eixos permitam movimentação em três dimensões, seu alcance espacial é limitado pela base fixa. Nesse quesito, os robôs AGVs apresentam maior mobilidade devido à sua construção com elementos móveis, como rodas e eixos de transmissão, conforme ilustrado na Figura 3. Com estrutura semelhante à de veículos convencionais, sua geometria, tamanho e disposição de controles são adaptados de acordo com a aplicação.

Figura 3 – Exemplo de um robô AGV

Fonte: Sierra-García e Santos (2020).

Esses robôs desempenham papel crucial na automação e eficiência industrial, especialmente em ambientes de produção, logística, armazenamento e distribuição. Operando sem intervenção humana, aumentam significativamente a eficiência operacional em comparação com métodos manuais (TANG; WU, 2023).

A arquitetura de controle de um AGV é fundamental para sua operação eficiente. Geralmente, utiliza-se um modelo de automação híbrida, no qual sistemas de controle e segurança são interligados por uma automação formal estendida. Essa arquitetura permite integrar sistemas de localização e direção, promovendo uma operação segura e eficaz no ambiente industrial (SIERRA-GARCíA; SANTOS, 2020).

2.1.2 Autonomous Mobile Robot (AMR)

A evolução natural da tecnologia AGV resultou nos Robôs Móveis Autônomos (AMRs), projetados para navegar e operar de forma independente em diversos ambientes. Esses robôs utilizam sensores avançados e sistemas de controle para tarefas como transporte, logística e coleta de dados. Graças a sistemas de controle e algoritmos robustos, podem operar autonomamente ou ser controlados remotamente, adaptando-se dinamicamente às mudanças ambientais. Com isso, ferramentas como gestão de tarefas, modelagem ambiental, planejamento de movimento, autolocalização e métodos de evasão de obstáculos tornam-se essenciais em operações em ambientes não estruturados ou dinâmicos (ALATISE; HANCKE, 2020).

A eficácia dos *AMRs* em ambientes complexos é potencializada pela integração de tecnologias de percepção que facilitam a interação em tempo real, como sensores *IMU*, *LiDAR* e outros sensores ambientais (ALATISE; HANCKE, 2020). Esses dispositivos serão discutidos em detalhes ao longo deste trabalho.

2.1.3 Diferenças entre AGV e AMR

Apesar de possuírem semelhanças, AGVs e AMRs oferecem soluções distintas para logística e movimentação de materiais, com características específicas que devem ser consideradas ao iniciar um projeto. Aplicações com AGVs são restritas a rotas fixas definidas por guias físicos como barras magnéticas ou adesivos com códigos QR, sem algoritmos de mapeamento ambiental, onde qualquer alteração no layout do ambiente exige uma reconfiguração do sistema, podendo ser custoso e impactar a produção. Já um AMR, com seu mapeamento contínuo do ambiente, é capaz de se localizar e navegar de forma flexível, sem necessidade de caminhos predefinidos (ZHANG *et al.*, 2023), conforme exemplificado no Quadro 1.

Em relação ao custo, os AGVs requerem manutenção mais simples, enquanto os AMRs, devido à complexidade de seus sensores, demandam um investimento inicial maior e manutenção específica. A escolha entre essas tecnologias deve equilibrar fatores como custo e complexidade do ambiente de aplicação.

Em ambientes dinâmicos, onde as condições mudam constantemente, como um ambiente industrial com movimentação de materiais, a capacidade de decisão autônoma dos AMRs é essencial para ajustar rotas e superar obstáculos inesperados (SIEGWART; NOURBAKHSH; SCARAMUZZA, 2011).

Quadro 1 – Comparações tecnológicas entre AGV e AMR

	AGV	**Lidar AMR**	**Visual AMR**
Localização	Barras magnéticas, Código QR	Tiras reflexivas, *Lidar SLAM*	*Visual SLAM*, Localização semântica visual, Fusão multimodal
Navegação e Evitação	Rota fixa, Para e espera para evitar obstáculos	Navegação livre, Evitação de obstáculos	Navegação livre, Fusão multimodal, Evitação de obstáculos visual
Características	Sem inteligência.	1. Não distingue categorias de obstáculos; previsão de trajetória limitada. 2. Incapaz de interação visual;	1. Distingue categorias de obstáculos e faz rastreamento preciso. 2. Capaz de interação visual;
Custo	Requer manutenção regular	Custo de *Lidar*, substituição e manutenção	Custo da câmera, manutenção especializada

Fonte: Adaptado de Zhang *et al.* (2023).

Pela própria natureza autônoma, AMRs não precisam de controle centralizado para coordenar suas trajetórias, a menos que vários robôs estejam operando em conjunto. Já os AGVs, por seguirem rotas predeterminadas e possuírem uma capacidade de tomada de decisão autônoma limitada, necessitam de um sistema centralizado para monitoramento e controle de rotas.

2.2 LOCOMOÇÃO

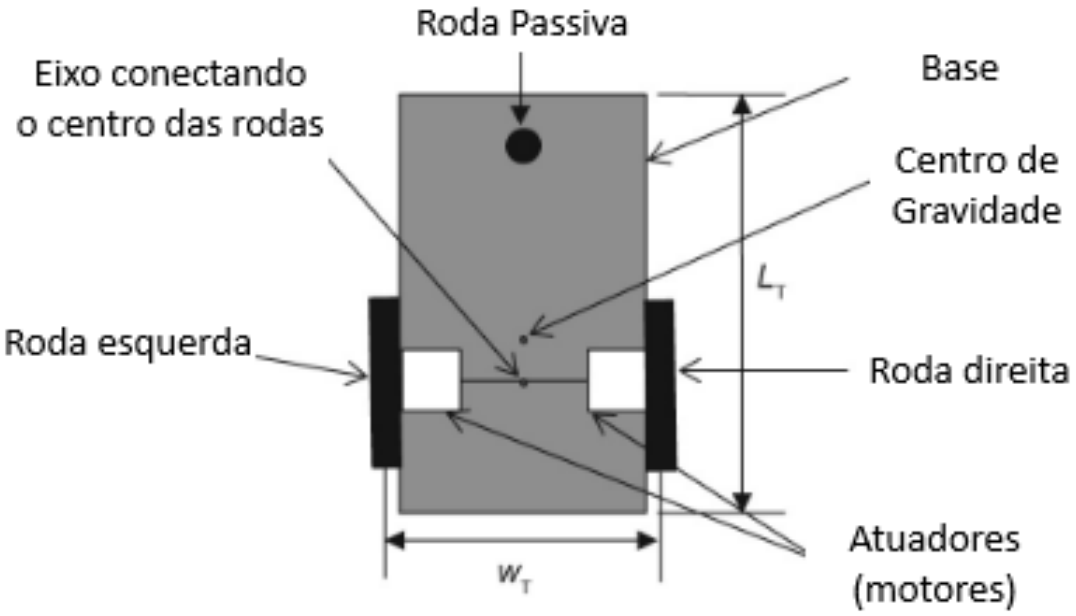
Ao contrário da manipulação, a locomoção permite que o robô altere sua posição no espaço por meio da aplicação de força no ambiente. Buscando alcançar o máximo de estabilidade, velocidade e precisão de movimento, e considerando o tipo de ambiente, aplicações robóticas com "pernas"são, em teoria, mais adequadas. No entanto, devido ao seu alto grau de liberdade e, consequentemente, à complexidade de desenvolvimento, utiliza-se com mais frequência a aplicação de rodas na robótica móvel, especialmente em AMRs (SIEGWART; NOURBAKHSH; SCARAMUZZA, 2011). Nesse contexto, dois modelos de geometria se destacam: Ackermann e Direção Diferencial, ambos oferecendo um equilíbrio entre robustez, manobrabilidade e controle.

2.2.1 Geometria de Direção Diferencial

Também conhecida como modelo Hilare, a direção diferencial é uma das configurações mais comuns para robôs móveis. Nessa geometria, dois motores independentes são alinhados ao longo de um mesmo eixo na seção traseira do robô, enquanto uma roda giratória livre (não motorizada) é instalada na parte frontal para garantir a estabilidade durante a movimentação, conforme ilustrado na Figura 4.

A direção de movimento da base do robô é controlada pela relação de velocidades entre os dois motores, que permanecem paralelos à base. Para realizar rotações mais fechadas, apenas uma das rodas deve ser movimentada, enquanto a outra, posicionada no lado interno da rotação, permanece parada, permitindo a rotação em torno de um ponto comum (ou pivô) sobre a roda

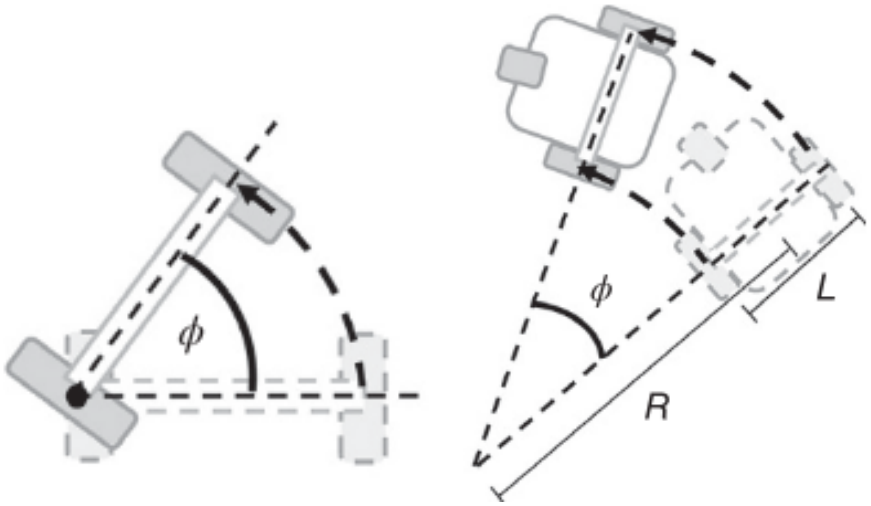
Figura 4 – Geometria Diferencial



Fonte: Mathew e Hiremath (2018).

interna, conforme ilustrado na Figura 5a. Para ângulos de rotação mais amplos, ambas as rodas são acionadas com velocidades diferentes, alterando o centro de rotação, conforme mostra a Figura 5b (KAGAN, 2020). Com essa estrutura, a manobrabilidade do modelo é significativamente melhorada, permitindo rotações em espaços reduzidos, sendo assim um grande diferencial e vantagem para sua utilização em robôs móveis

Figura 5 – Modelo de movimentação para robô com geometria diferencial



(a) Rotação em torno do ponto pivô (b) Rotação com centro de rotação ajustado

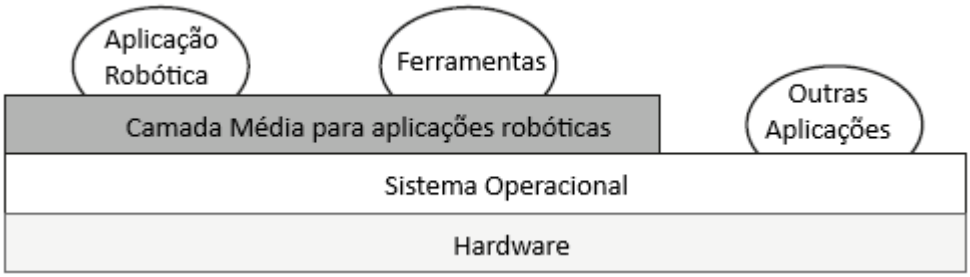
Fonte: Kagan (2020).

2.3 ROS

O *Robot Operating System* (ROS) é um *framework* de software livre para o desenvolvimento de aplicações robóticas, inicialmente apresentado pela *The Open Source Robotics Foundation* e mantido pela comunidade *ROS*. Ele oferece uma estrutura modular e distribuída que facilita a escrita de software para robôs, incluindo bibliotecas e ferramentas para auxiliar os desenvolvedores. Além disso, o *ROS* suporta uma ampla gama de sistemas robóticos e possui um ecossistema de *software* em constante evolução, consolidando-se como uma plataforma padrão no campo da robótica.

Embora frequentemente seja considerado como um Sistema Operacional (SO) que substitui sistemas operacionais como Linux, o ROS é definido como um *middleware*, uma camada de software que integra o sistema operacional e a aplicação que será desenvolvida pelo usuário (Figura 6). Esse *middleware* fornece não somente uma interface para o desenvolvimento, mas também ferramentas para monitoramento e desenvolvimento (RICO, 2022).

Figura 6 – Arquitetura de um *Middleware*



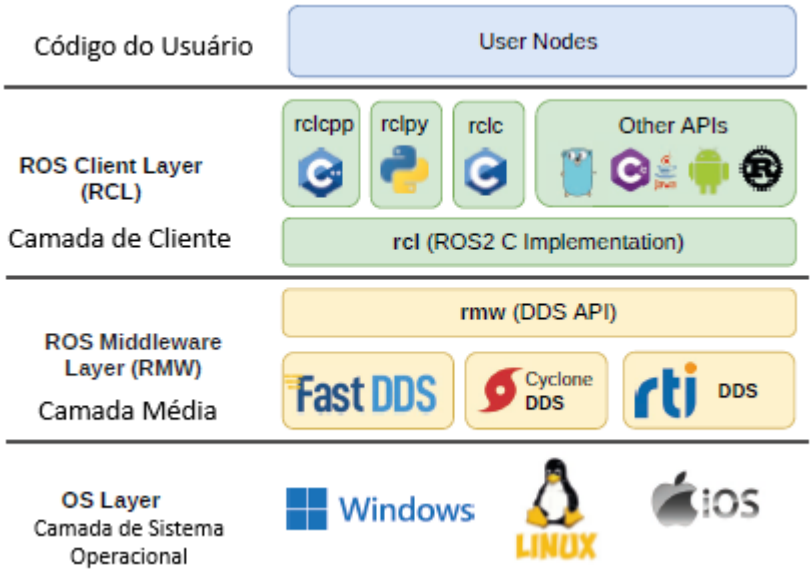
Fonte: Rico (2022).

O interfaceamento entre o programa desenvolvido e o *middleware* é realizado por meio de uma camada chamada *ROS Client Layer* (RCL), como demonstrado na Figura 7. Essa camada fornece uma *Application Programming Interface* (API) que permite o desenvolvimento em linguagens de programação amplamente utilizadas, como C++ e *Python*. Para aplicações em C++, utiliza-se a biblioteca rclpp, enquanto para *Python* é empregada a biblioteca rclpy. Esse *layer* é fundamental na arquitetura do sistema ROS, adaptando as funcionalidades de cada linguagem e facilitando o desenvolvimento da aplicação (RICO, 2022).

2.3.1 ROS2

Projetado para ser uma solução mais robusta e adequada para robótica em ambientes de produção que exigem troca de informações em tempo real, o ROS2 incorpora mudanças significativas que melhoram a segurança, escalabilidade e confiabilidade, atendendo às necessidades de sistemas robóticos mais complexos (MARUYAMA; KATO; AZUMI, 2016).

Figura 7 – Arquitetura ROS



Fonte: Rico (2022).

Uma das principais diferenças entre as versões é que o ROS2 utiliza o protocolo *Data Distribution Service* (DDS), que proporciona melhor suporte para sistemas em tempo real e melhorias em termos de segurança. Isso é especialmente relevante em aplicações industriais onde robôs interagem constantemente com outros objetos, máquinas e pessoas. A arquitetura modular e escalável do *ROS2* permite a comunicação distribuída e a operação com múltiplos robôs, além de integrar funcionalidades de tolerância e gestão de falhas, monitorando constantemente os nós e módulos da aplicação (MARUYAMA; KATO; AZUMI, 2016).

2.3.1.1 Distribuições do ROS 2

Assim como outros *softwares*, o desenvolvimento e aprimoramento do ROS2 seguem uma estratégia de distribuições, ou *releases*, com novas versões sendo lançadas quando mudanças significativas são implementadas. Cada distribuição é composta por bibliotecas, ferramentas e aplicações compatíveis, organizadas para trabalhar de maneira sincronizada e sem erros (RICO, 2022).

Algumas das principais distribuições do ROS2 incluem:

1. **Humble Hawksbill**: Lançada em maio de 2022, é compatível com Ubuntu 22.04 e outras distribuições Linux, com foco em facilidade de uso e estabilidade.

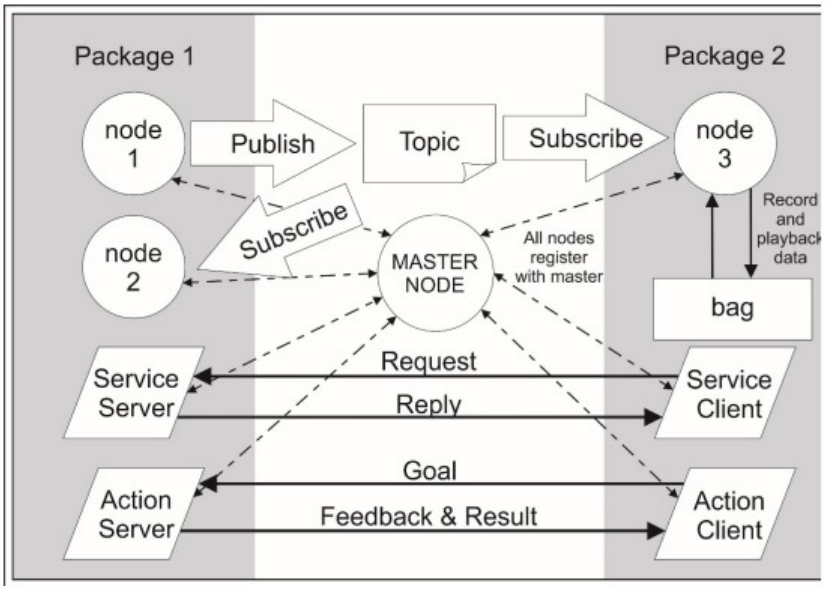
2. **Rolling Ridley**: Diferentemente das outras, não possui data de suporte final. É uma versão de desenvolvimento contínuo, atualizada regularmente com novas funcionalidades e correções, ideal para desenvolvedores que buscam inovação, embora não recomendada para produção devido à sua instabilidade (ROS.ORG, 2024).

A escolha da distribuição de ROS2 sobre a qual o projeto será desenvolvido é fundamental. Para uma versão estável com longo prazo de suporte, recomenda-se a adoção de versões recentes, assegurando estabilidade para aprimoramentos futuros. Além disso, cada distribuição é compatível com uma versão específica do SO, o que significa que a escolha da distribuição de ROS2 também influencia a escolha da versão do SO do projeto.

2.3.1.2 Estrutura de Aplicação ROS

Na organização dos componentes de uma aplicação ROS, destaca-se o conceito de Gráfico de Computação, ou *Computation Graph*. Esse é o termo oficial para a representação visual do comportamento do software durante a execução, demonstrando a interação entre os componentes do sistema, conforme apresentado na Figura 8. Esses componentes são úteis nas etapas de testes e correções de problemas e essenciais para a organização e o planejamento da aplicação (RICO, 2022) Tais componentes podem ser exibidos ao usuário através de ferramentas específicas, como o *RQT Graph*, constituindo assim um componente muito importante durante o desenvolvimento de um projeto, para fins de depuração de erros e comportamento.

Figura 8 – Estrutura de comunicação de uma aplicação ROS2



Fonte: Ramesh (2016).

Duas estruturas principais são utilizadas em uma solução ROS:

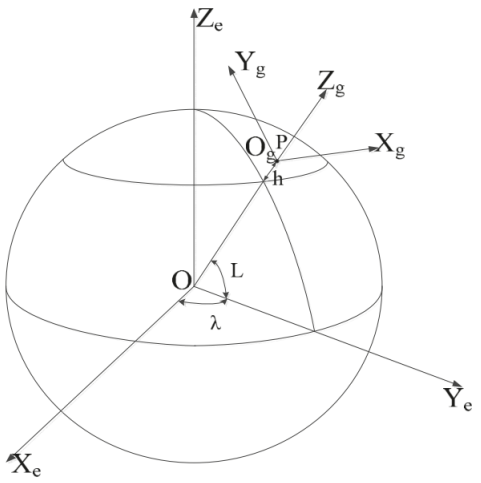
- **Nós**: São os elementos principais de execução, onde as tarefas programadas são realizadas. Em uma analogia com outras linguagens de programação, um nó pode ser visto como um objeto em Programação Orientada a Objetos (POO) ou um bloco de programa em sistemas Controlador Lógico Programável (CLP). A execução pode ser interativa (cíclica), por intervalo determinado ou orientada por evento.

- **Comunicação entre Nós**: Baseada em uma camada de *Data Distribution Service* (DDS), a comunicação entre nós ocorre de três formas distintas, cada qual com características específicas:
  - **Publicação / Assinatura**: Método assíncrono que permite que vários nós acessem simultaneamente a mesma informação. Nessa arquitetura, um nó publica uma informação e outros nós podem assiná-la, acessando-a em tempo real, mas sem alterá-la.
  - **Serviço**: Comunicação síncrona ponto-a-ponto, onde um nó solicita a um segundo nó, aguardando resposta antes de prosseguir. É usada para tarefas que exigem garantia de execução, como limpar dados de memória.
  - **Ação**: Semelhante ao Serviço, mas de forma assíncrona. O nó solicitante não aguarda a confirmação para seguir a execução, ideal para tarefas que demandam tempo prolongado (RICO, 2022).

2.3.1.3 Transformações

Transformações no *ROS2* referem-se à capacidade de definir e manipular a posição e orientação de diferentes partes de um robô (sensores, atuadores, base, etc.) em relação a uma base de referência (KOUBAA, 2020), conforme representado na Figura 9. Para isso, utiliza-se a biblioteca *tf2*, que fornece uma estrutura para publicar, monitorar e manipular transformações espaciais em um sistema de coordenadas, chamado "tree of frames". Um exemplo comum seria a publicação da posição de um sensor de câmera em relação à base do robô, permitindo que outros componentes alinhem os dados do sensor à posição real do robô.

Figura 9 – Demonstração do conceito de transformações



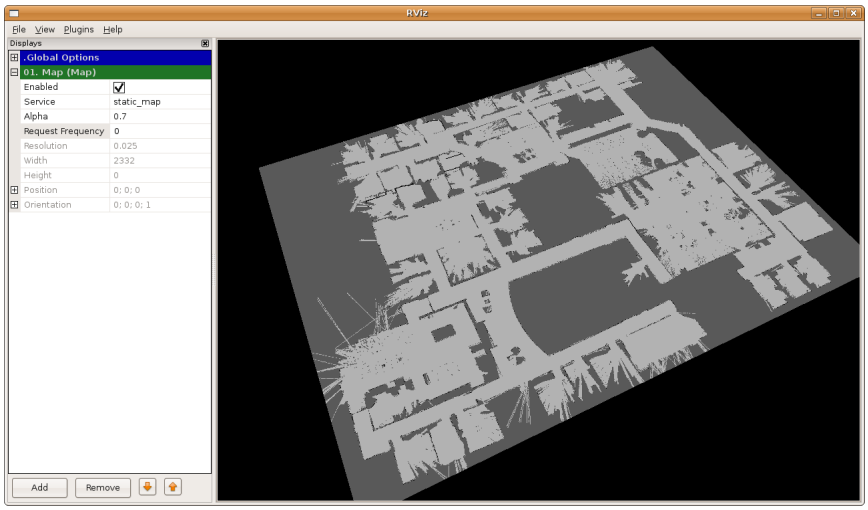
Fonte: Kim (2020).

Além disso, *tf2* oferece funcionalidades de interpolação temporal, permitindo o cálculo preciso da posição de um *frame* em um momento específico. Essa precisão e flexibilidade são particularmente importantes em simulações robóticas (KOUBAA, 2020).

2.3.2 RViz

O *plugin ROS Visualization* (RViz), uma popular ferramenta de visualização 3D no ecossistema *ROS*, auxilia os desenvolvedores na visualização do estado do robô e dos dados de seus sensores durante simulações ou operações reais. Essa visualização inclui sensores como *LIDAR*, câmeras e sensores de profundidade, como demonstrado na Figura 10. Essa funcionalidade facilita a análise do comportamento do robô e dos dados capturados no momento em que o sistema opera, assim permitindo identificar problemas em sistemas complexos, como falhas em sensores, erros de transformação ou configurações incorretas de navegação. Além disso, em projetos que usam o Nav2, o RViz é essencial para visualizar mapas SLAM, a posição do robô, a rota planejada e áreas de navegação permitidas.

Figura 10 – Visualização de dados de um sensor no Rviz



Fonte: OpenRobotics (2024).

Além disso, o RViz permite visualizar a trajetória e o movimento planejado do robô, sendo essencial para a depuração e o desenvolvimento de algoritmos de navegação e manipulação. Ele também é capaz de renderizar modelos 3D do robô e de seu ambiente, facilitando a compreensão espacial das operações robóticas (OPENROBOTICS, 2024).

2.3.3 URDF

Na modelagem física de uma aplicação robótica, o uso de softwares amplamente conhecidos, como o *SolidWorks*, é comum. Contudo, para implementar esse modelo em um ambiente de simulação e integrá-lo com sistemas de controle como o *ROS*, os utiliza-se o formato *Unified Robot Description Format* (URDF), um arquivo *Extensible Markup Language* (XML) que descreve todos os elementos físicos de um robô. O URDF define componentes como *links*, *joints*, sensores e atuadores, permitindo que o *ROS* e outras ferramentas de simulação compreendam sua estrutura e funcionamento (FEDER; GIUSTI; VIDONI, 2022).

Esse formato de arquivo é estruturado como uma árvore, onde o comportamento de cada elemento é descrito em um subgrupo, incluindo características físicas como material, peso, centro de massa e torque. Como discutido na Seção 2.1, em um arquivo URDF cada junta deve estar conectada a dois elos, um anterior e outro posterior, detalhando o tipo de movimento e os limites operacionais, o que ajuda a evitar colisões. Devido à complexidade do código, a criação manual de um arquivo URDF pode ser suscetível a erros. Por isso, algumas ferramentas, como o *XML Macros* (XACRO), auxiliam na geração dessas configurações, tornando o código XML mais legível (FEDER; GIUSTI; VIDONI, 2022).

2.4 GAZEBO

O *Gazebo Simulator* é um software de simulação física de código aberto amplamente utilizado em aplicações robóticas, devido à sua capacidade de incorporar sensores e recriar ambientes 3D complexos, tornando-o ideal para testar conceitos e algoritmos de robótica. Desenvolvido em cooperação com os projetos Player e Stage, o Gazebo oferece controle detalhado e alta fidelidade, proporcionando visualização de dados, simulação de ambientes remotos e possibilidades para engenharia reversa de sistemas fechados (KOENIG; HOWARD, 2004).

Este software apresenta motores físicos avançados para simular interações realistas, como colisões, atrito, gravidade e dinâmica de fluidos, permitindo criar ambientes virtuais ricos que replicam cenários reais, incluindo terrenos irregulares, objetos, obstáculos e condições climáticas, conforme apresentado na Figura 11. Somado a este fator de grande fidelidade de simulação, este software é projetado para se integrar perfeitamente com o ROS2, permitindo que os tópicos, serviços e ações do ROS2 sejam utilizados para controlar robôs e monitorar a simulação, sendo a escolha ideal para o presente trabalho.

Figura 11 – Ambiente industrial simulado no simulador Gazebo

Fonte: O Autor (2024).

Assim como outros softwares de simulação de física, o motor gráfico do Gazebo desempenha um papel crucial para representar virtualmente comportamentos e entidades reais. Para isso, o Gazebo utiliza, por padrão, a ferramenta *Dynamic Animation and Robotics Toolkit* (DART), uma biblioteca *open-source* desenvolvida em C++ pelo *Georgia Institute of Technology* (GAZEBOSIM.ORG, 2024a).

2.4.1 Distribuições do Gazebo

O Gazebo segue um modelo de versionamento para suas atualizações, semelhante ao ROS2, conforme discutido na Seção 2.3.1.1. Cada versão oferece diferentes funcionalidades e nem todas são compatíveis entre si, o que torna essencial selecionar uma versão do Gazebo que seja compatível com a distribuição de ROS2 escolhida para o projeto. Um resumo da compatibilidade de cada versão pode ser encontrado no Quadro 2.

Algumas versões do Gazebo oferecem suporte a longo prazo, indicado pela sigla *Long-Term Support* (LTS). Para garantir compatibilidade futura e evitar alterações frequentes de versões e arquivos, recomenda-se a utilização de uma versão LTS e sua correspondente versão de ROS2, evitando problemas na troca de comunicação entre ambos programas. Para tanto, o presente trabalho utiliza a versão Humble para ROS2, enquanto Fortress foi escolhida como versão do Gazebo

Quadro 2 – Distribuições de Gazebo

	GZ Citadel (LTS)	GZ Fortress (LTS)	GZ Garden	GZ Harmonic (LTS)
ROS 2 Jazzy (LTS)	×	×	±	×
ROS 2 Rolling	×	×	±	±
ROS 2 Iron	×	✓	±	±
ROS 2 Humble (LTS)	×	✓	±	±
ROS 2 Foxy (LTS)	✓	×	×	×
ROS 1 Noetic (LTS)	✓	±	±	×

Legenda:
✓ - Combinação recomendada
× - Incompatível / não possível
± - Possível, mas recomenda-se cautela.

Fonte: Adaptado de Gazebosim.org (2024b).

2.4.2 Integração entre Gazebo e ROS2

A integração entre Gazebo e *ROS2* é facilitada por uma biblioteca fornecida pela *OpenRobotics*, chamada *ros_gz*, oferecendo uma série de funções para facilitar a troca de informações entre os dois ambientes. A funcionalidade mais relevante para este projeto é a *ros_gz_bridge*, que permite a troca de mensagens entre o código ROS2 desenvolvido e o componente de mensagens do Gazebo, chamado Gazebo Transport (GAZEBOSIM.ORG, 2024b).

As mensagens trocadas por meio deste *package* seguem a sintaxe demonstrada no Algoritmo 1.

Algoritmo 1 – Exemplo de comunicação entre ROS2 e Gazebo

```
1 ros2 run ros_gz_bridge parameter_bridge /TOPIC@ROS_MSG@IGN_MSG
```

Fonte: O Autor (2024)

Nesta sintaxe, têm-se os seguintes componentes:

- /TOPIC: Nome do tópico que carrega o dado.
- *ROS_MSG*: Formato da mensagem do lado *ROS*.
- IGN_MSG: Formato da mensagem do lado Gazebo.
- @: Direção da ponte, sendo que:
  - @: Ponte bidirecional.
  - [: Ponte do Gazebo para o *ROS*.
  -]: Ponte do *ROS* para o Gazebo.

2.5 NAVEGAÇÃO E LOCALIZAÇÃO

A navegação e o planejamento de rotas constituem uma área de estudo em constante evolução, com vários autores contribuindo para aprimorar metodologias que permitam maior eficiência, segurança e agilidade em sistemas *AMR*. Autores como Zhang, Lin e Chen (2018) utilizam algoritmos como A*, amplamente reconhecido na robótica móvel para cálculo de caminhos e definição da rota ideal em ambientes estáticos. Neste algoritmo, o destino é encontrado considerando o peso atribuído aos pontos do trajeto, resultando na rota mais curta até o objetivo e proporcionando uma aproximação heurística à resolução do problema.

No entanto, em ambientes reais, é necessário utilizar dados de diversos sensores para adaptar o sistema a características dinâmicas e responder a agentes externos. Técnicas de processamento de sinais são, portanto, essenciais para melhorar o controle e a movimentação de um *AMR* em um ambiente dinâmico ou desconhecido.

2.5.1 SLAM

Uma etapa essencial no desenvolvimento de um projeto de robô autônomo é a escolha do método de localização e navegação, dado que o objetivo central do robô é se deslocar entre pontos, completando as tarefas para as quais foi projetado.

Um método de localização amplamente utilizado em robôs AMR é o *Simultaneous Localization and Mapping* (SLAM), que permite a construção de um mapa de um ambiente desconhecido enquanto o robô se localiza simultaneamente nesse mapa em tempo real. Essa capacidade é fundamental para a navegação autônoma, pois permite ao controlador traçar rotas ideais, evitando obstáculos e zonas desconhecidas, eliminando a necessidade de rotas pré-definidas, como ocorre em robôs AGV ou seguidores de linha (GOBHINATH *et al.*, 2021).

O processo SLAM envolve a coleta de dados de sensores, como *Light Detection and Ranging* (LiDAR). A partir desses dados, o sistema estima sua posição e orientação (pose) e usa essas informações para incrementar o mapa do ambiente. Existem variantes do SLAM, como 2D e 3D, cada uma apropriada a diferentes tipos de ambientes e aplicações. Algoritmos como *Extended Kalman filter* (EKF), *Particle Filter* (ou *Monte Carlo Localization*) e *Graph-Based SLAM* são comumente empregados para implementar SLAM, cada um com vantagens e limitações quanto à precisão, eficiência computacional e facilidade de implementação (GOBHINATH *et al.*, 2021).

Para este conceito, dois elementos se demonstram importantes e complementares, sendo eles o mapa global e o mapa local. O mapa global representa toda a área conhecida do ambiente, sendo utilizado para o planejamento de trajetórias de longo alcance. Ele contém informações sobre obstáculos fixos, paredes e áreas navegáveis. Esse mapa pode ser gerado por ferramentas de mapeamento, como SLAM, ou carregado como um arquivo estático e atualizado dinamicamente em algumas configurações. Por outro lado, o mapa local é uma representação em tempo real da área próxima ao robô, atualizado dinamicamente com base nos dados dos sensores, capturando obstáculos móveis ou mudanças inesperadas no ambiente. Esse mapa é utilizado pelo planejador local para ajustar a trajetória do robô enquanto ele se move, garantindo que obstáculos dinâmicos sejam evitados de forma eficiente (GOBHINATH *et al.*, 2021).

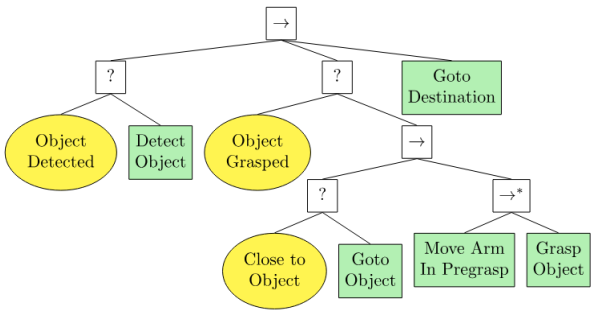
Para sua integração, o mapa global fornece o planejamento inicial, enquanto o mapa local lida com detalhes de curto alcance e mudanças imprevistas durante a execução. Essa abordagem combinar a eficiência do planejamento global com a adaptabilidade e a reatividade do mapa local, necessárias para navegação em ambientes dinâmicos e complexos.

2.5.2 Nav2

Entre os projetos disponíveis na comunidade do ROS2, destaca-se a biblioteca *Nav2*, um conjunto de funções que viabilizam a navegação autônoma de robôs em ambientes desconhecidos, baseado no conceito de SLAM. A biblioteca facilita a integração de algoritmos de localização e mapeamento, permitindo que robôs construam e utilizem mapas enquanto se localizam nesses ambientes. Esse processo envolve a coleta e análise de dados de sensores instalados no robô, como LiDAR e câmeras, para detectar e evitar obstáculos (NAV2.ORG, 2024). Em caso de falha (como bloqueio ou perda de caminho), a Nav2 pode executar ações de recuperação, como voltar ao último ponto seguro ou recalcular a rota.

Em projetos *ROS2*, que frequentemente requerem interação simultânea entre vários nós para executar tarefas complexas, o controle do *Nav2* utiliza o conceito de *Behavior Trees* (BT). Essa estrutura organiza a tomada de decisão e execução de tarefas de acordo com a hierarquia dos nós. Cada ação é determinada pelo fluxo de dados entre os nós que compõem essa árvore (COLLEDANCHISE; NATALE, 2021), conforme ilustrado na Figura 12

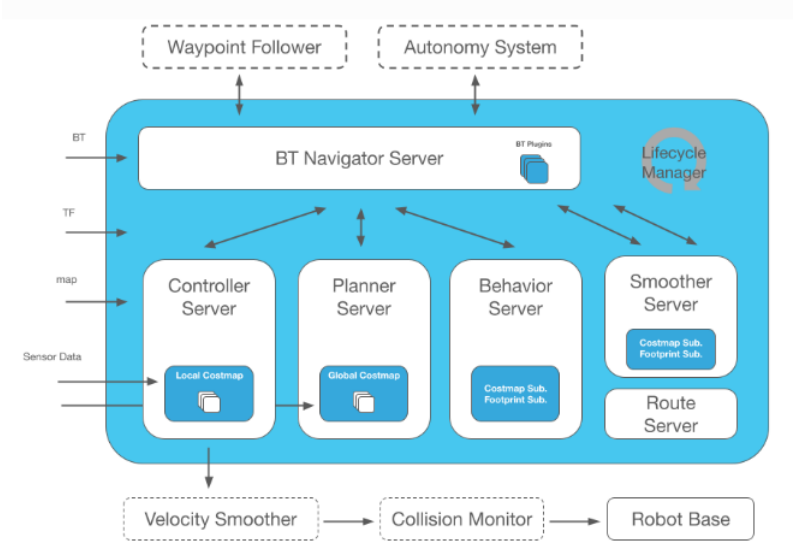
Figura 12 – Estrutura de uma Behavior Tree



Fonte: Colledanchise e Natale (2021).

Usando a biblioteca *BehaviorTree CPP V3*, o *Nav2* desenvolve uma lógica de controle onde várias BT são integradas ao mesmo código, em uma estrutura de sub-árvores. Assim, o projeto pode realizar várias tarefas simultaneamente, como calcular a trajetória ideal para o robô e monitorar seu status, corrigindo eventuais situações, como quando o protótipo fica preso entre obstáculos. Essa estrutura de controle está representada na Figura 13, onde o servidor de navegação central recebe dados de servidores secundários específicos, como o agente responsável pelo planejamento de rota (*Planner Server*). Tal troca e processamento de informações, alinhadas com dados externos como de sensores, são enviadas para o servidor de controle, responsável pela comunicação com elementos de movimentação e acionamento dos motores robô.

Figura 13 – Esquema de Controle da Biblioteca Nav 2



Fonte: Rico (2022).

Para o planejamento de rotas, o *Nav2* utiliza uma série de *plugins* mantidos em um servidor chamado *Planner Server*, que traça o caminho do robô com base nos dados sensoriais. Alguns desses algoritmos exploram o espaço do ambiente, enquanto outros expandem os estados possíveis do robô, considerando a viabilidade do caminho. Dentre eles, o *NavFn Planner* utiliza os algoritmos de Dijkstra ou A*, enquanto o *Smac 2D Planner* aplica um algoritmo A* em 2D, e o *Theta Star Planner* usa linha de visão para criar segmentos de caminho (NAV2.ORG, 2024). Cada algoritmo tem suas vantagens e limitações, sendo que o *NavFn Planner* e o *Smac Planner 2D* são indicados para robôs com controle diferencial, enquanto o *Smac Hybrid* é recomendado para robôs com geometria Ackermann, como mostrado na Quadro 3.

Quadro 3 – Tipos de Robôs Suportados para Vários Plugins de Navegação

Nome do Plugin	**Tipos de Robôs Suportados**
NavFn Planner	Diferencial Circular, Omnidirecional Circular
Smac Planner 2D	Diferencial Circular, Omnidirecional Circular
Theta Star Planner	Ackermann Circular ou Não Circular, Perna Circular ou Não Circular
Smac Hybrid-A* Planner	Ackermann Circular ou Não Circular, Perna Circular ou Não Circular
Smac Lattice Planner	Diferencial Não Circular, Omnidirecional Não Circular

Fonte: Adaptado de nav2.org (2024).

2.5.3 Sensoriamento

Em robótica móvel, os sensores são componentes fundamentais pois permitem que os sistemas interajam com o ambiente ao redor de maneira eficaz, detectando obstáculos, mapeando o ambiente e possibilitando a localização do robô. Dessa forma, os sensores garantem que o robô responda a mudanças no ambiente e interaja com objetos, otimizando o desempenho de suas tarefas, além de garantirem a segurança da aplicação (ALATISE; HANCKE, 2020).

Diferentes tipos de sensores podem ser necessários para adaptar o robô a tarefas específicas, como ilustrado na Quadro 4. Entre eles, destacam-se os sensores táteis, como interruptores de contato e barreiras ópticas, empregados para detectar a proximidade de objetos. Os encoders de roda, são essenciais para medir a distância e a velocidade percorrida pelo robô, além de monitorar as rotações das rodas, auxiliando na orientação do movimento. Sensores ópticos, como infravermelho e LiDAR, são utilizados para estimativas de distância com base na propagação da luz, enquanto os sensores de direção permitem medir a velocidade angular e a orientação do robô, proporcionando maior controle na navegação. Além disso, sensores baseados em visão capturam informações do ambiente e possibilitam interações inteligentes em cenários dinâmicos, enquanto os sensores de medição ativa, como ultrassônicos e telemetria a laser, contribuem para uma navegação ao gerar medições exatas de distância entre o sensor e o alvo.

Devido à grande variedade de equipamentos e aplicações, nem todos os sensores listados são necessários em uma aplicação *AMR*; a escolha depende da complexidade dos movimentos e do ambiente. Seguindo essas considerações, as próximas seções abordarão apenas os sensores utilizados neste projeto.

Quadro 4 – Tecnologias de Sensores para *AMRs*

Classificação	Sistema de sensor	Funções
Sensores táteis:	Interruptores de contato, para-choques	Detectam e determinam a posição exata de objetos a curta distância via contato direto. Também usados para medir variações de força. Usados principalmente para medir a força aplicada pelos efetores finais do robô.
	Barreiras ópticas	
	Sensores de proximidade sem contato	
Encoders de roda:	Óptico	Medem a distância ou velocidade percorrida pelo robô, contando as revoluções e a orientação de cada roda.
	Magnético	
Sensor óptico:	Infravermelho	Baseados em luz, produzem estimativas de alcance pelo tempo que a luz leva para alcançar o alvo e retornar.
	LiDAR	
Sensor de direção	Giroscópio	Sensor confiável que mede velocidades angulares e orientação.
Sensores baseados em visão	Câmera(s) CCD/CMOS	Oferecem vasta quantidade de informações sobre o ambiente e permitem interação inteligente em ambientes dinâmicos.
Sensores de medição ativa	Ultrassônico	Auxiliam a navegação, gerando medições precisas da distância entre o sensor e o alvo, parte da modelagem de localização e ambiental.
	Telêmetro laser	
	Triangulação óptica	

Legenda:
CCD - Charge-Coupled Device (Dispositivo de carga acoplada)
CMOS - Complementary Metal Oxide Semiconductor

Fonte: Adaptado de Alatise e Hancke (2020).

2.5.3.1 Fusão de Sensores

Para lidar com a quantidade de sensores utilizados em uma única aplicação, que mensuram grandezas distintas e cujos dados devem ser processados de forma integrada pelo controlador do robô, o conceito de fusão de sensores é essencial. Este processo envolve o uso cooperativo de informações de múltiplos sensores para uma função específica, em vez de depender de um único dispositivo, que pode ter limitações e ser suscetível a erros. Esse método permite que informações de diferentes fontes sejam integradas, complementando-se e corrigindo-se mutuamente (ALATISE; HANCKE, 2020). A fusão de sensores também simplifica o sistema, processando e integrando os dados antes de utilizá-los nos controles, o que possibilita uma entrada de dados padronizada para o laço de controle, independentemente do tipo e número de sensores.

Com diversos algoritmos de fusão, como filtro *Kalman Filter* (KF), EKF e o algoritmo de Dempster-Shafer, um AMR pode obter uma visão mais completa e precisa do ambiente, permitindo detecção e identificação de obstáculos, compreensão do contexto e navegação eficiente em ambientes complexos (ALATISE; HANCKE, 2020).

2.5.3.2 LiDAR

Um sensor LiDAR é uma tecnologia fundamental na robótica autônoma, georreferenciamento, mapeamento e outras aplicações que exigem captação precisa de informações sobre a distância e a forma dos objetos no ambiente. Essa tecnologia utiliza pulsos de luz laser para medir distâncias com alta precisão, calculando o tempo que o pulso refletido leva para retornar ao sensor, convertido em medidas de distância. Combinando essas distâncias com a direção exata dos feixes emitidos, o *LiDAR* constrói uma representação tridimensional detalhada do ambiente. Esse processo ocorre em milissegundos, podendo ser repetido diversas vezes por segundo, o que permite ao dispositivo capturar uma visão dinâmica do seu entorno (BEHROOZPOUR *et al.*, 2017).

Diversos fabricantes oferecem soluções inovadoras para *LiDAR*, como sensores aerotransportados e formatos de instalação variados, explorando funcionalidades em novos produtos. Entre as principais vantagens do *LiDAR* estão sua alta precisão e a capacidade de operar em diferentes condições ambientais, como baixa visibilidade e noite. Contudo, a tecnologia enfrenta desafios em superfícies com baixa capacidade de reflexão, como materiais absorventes ou escuros. Além disso, os sistemas *LiDAR* tendem a ser mais caros que outras tecnologias de sensores, como radar e câmeras (BEHROOZPOUR *et al.*, 2017).

2.5.3.3 IMU

Essenciais em sistemas robóticos, os sensores *Inertial measurement unit* (IMU) medem e rastreiam movimentos, incluindo aceleração, orientação e velocidade angular. Para isso, eles são compostos por acelerômetros, giroscópios e, frequentemente, magnetômetros, permitindo a coleta de dados tridimensionais sobre a posição e movimentação do robô. Em aplicações de navegação, os dados da IMU auxiliam na estabilização e no controle de trajetória, fornecendo informações fundamentais para algoritmos de navegação e posicionamento (FAISAL; PURBOYO; ANSORI, 2019).

Cada componente da IMU mede uma grandeza específica: o acelerômetro coleta a aceleração linear nos três eixos (x, y, z), que é a variação da velocidade do objeto; o giroscópio mede a velocidade angular ou taxa de rotação ao longo dos mesmos eixos, complementando as medições do acelerômetro e eliminando ruídos e erros constantes; e o magnetômetro mede o campo magnético em relação ao norte, determinando a orientação do objeto (ALATISE; HANCKE, 2020).

Contudo, em ambientes industriais, a presença de materiais metálicos pode afetar a precisão dos dados do magnetômetro, demandando outras formas de determinação da orientação. Alternativas incluem redes *Ultra-Wideband* (UWB), *Wide Local Area Network* (WLAN) ou *Bluetooth*, embora cada uma tenha limitações, como interferências, sendo necessário um estudo para determinar o melhor sistema para cada caso (ALATISE; HANCKE, 2020).

2.6 NODE-RED

Node-RED é uma plataforma de desenvolvimento baseada em fluxo, ou *Flow-Based Programming* (FBP), construída sobre o código aberto multiplataforma *Node.js*, que permite a criação e integração de aplicações de forma visual. Originalmente desenvolvida pela IBM, essa ferramenta foi projetada para simplificar o desenvolvimento de soluções em Internet das Coisas (*Internet of Things* (IoT)), automação e integração de sistemas. Seu funcionamento ocorre por meio de uma interface gráfica, na qual o desenvolvedor conecta blocos, conhecidos como *nodes*, para criar fluxos de dados que executam uma sequência lógica (HAGINO; O'LEARY, 2021).

Cada *node* em Node-RED representa uma tarefa específica, como manipulação de dados, comunicação com dispositivos ou interações com API. Os *nodes* são conectados por linhas que representam o fluxo de dados, permitindo ao desenvolvedor visualizar e modificar o comportamento do sistema de forma intuitiva. A plataforma possibilita a combinação de diversas funcionalidades, como sensores, protocolos de comunicação (*Message Queuing Telemetry Transport* (MQTT) e Protocolo de Transferência de Hipertexto (HTTP)) e armazenamento de dados, simplificando o desenvolvimento de sistemas de automação.

Aproveitando-se do ecossistema JavaScript, Node-RED permite a integração com uma ampla gama de bibliotecas e módulos. Essa combinação de usabilidade, extensibilidade e robustez torna o Node-RED uma solução ideal para a criação rápida de protótipos e a integração de sistemas complexos com soluções web, como sistemas supervisórios ou de monitoramento (HAGINO; O'LEARY, 2021).

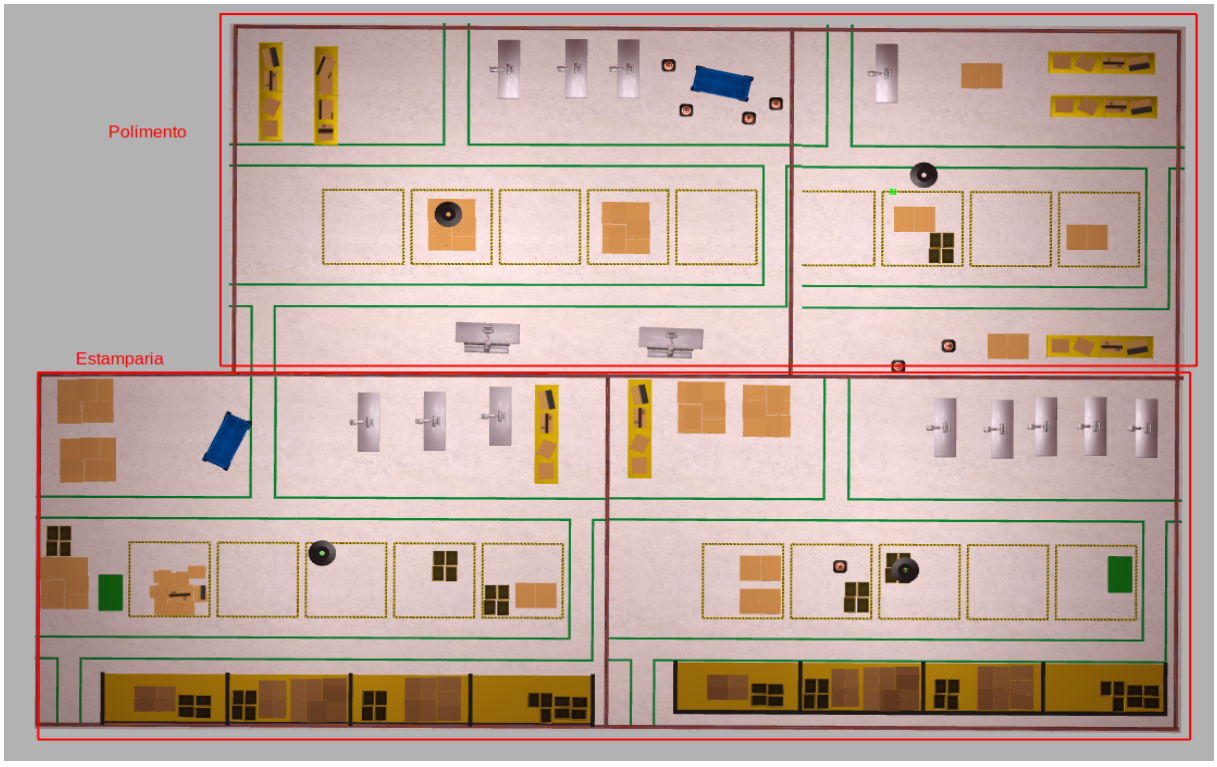
3 SOLUÇÃO PROPOSTA

3.1 ORGANIZAÇÃO DO PROJETO

3.1.1 Ambiente Simulado

A metodologia de desenvolvimento adotada segue uma estruturação modular e integrada, permitindo testar, desenvolver e monitorar o comportamento do robô em um ambiente simulado, com o objetivo de validar o modelo. O projeto será utilizado em um ambiente virtual que simule fielmente um ambiente industrial físico. Para isso, foi criado um cenário industrial com quatro ambientes, organizados em dois setores distintos. O setor de origem, denominado Estamparia, onde o material produzido será coletado pelo robô, e o setor de destino, intitulado Polimento, conforme ilustrado na Figura 14.

Figura 14 – Vista superior do ambiente simulado



Fonte: O Autor (2024).

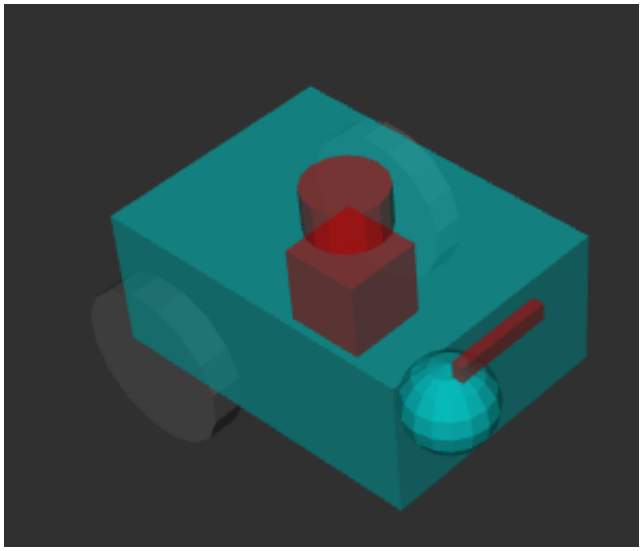
Após a criação do *layout* do ambiente, o cenário foi preenchido com componentes e equipamentos industriais, reproduzindo as condições em que o robô deverá operar. Para isso, foram utilizados modelos *3D* gratuitos do portal *Gazebosim*, reduzindo o tempo de desenvolvimento. Esses modelos foram organizados conforme a disposição atual do ambiente físico que inspirou o projeto, além de incluir obstáculos adicionais para aumentar a complexidade dos movimentos e rotas possíveis para o *AMR*.

3.1.2 Modelo AMR

Conforme apresentado na Seção 2.2, diversas geometrias podem ser utilizadas na modelagem do robô, sendo fundamental analisar os componentes envolvidos durante a etapa de *design*. Assim, foram considerados diversos fatores que levaram à escolha da geometria de direção diferencial para este projeto.

Os *plugins* de movimentação clássicos utilizados no *ROS*, como o popular *diff_drive*, são projetados especificamente para robôs com essa geometria, oferecendo uma solução já consolidada e de fácil adaptação às dimensões do robô. Além disso, o ambiente simulado no qual o *AMR* operará apresenta muitos obstáculos e rotas restritas, exigindo manobras em espaços limitados e giros de 90 °para seguir o trajeto planejado. Dessa forma, a direção diferencial mostrou-se a opção mais lógica para atender a esses requisitos, resultando no modelo apresentado na Figura 15.

Figura 15 – Visualização 3D do modelo *AMR* desenvolvido



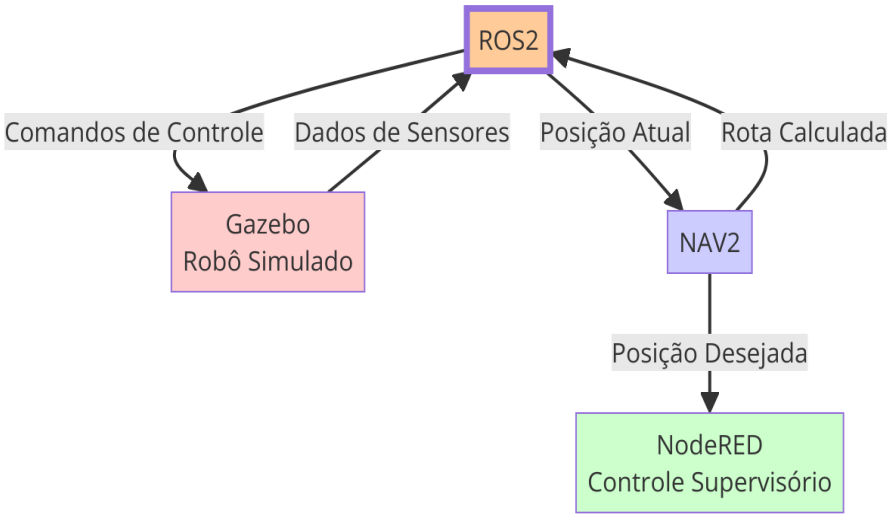
Fonte: O Autor (2024).

3.1.3 Topologia do Projeto

Utilizando os componentes apresentados no Capítulo 2, como o Gazebo para simulação, o *NAV2* para navegação e o NodeRED para controle supervisório, é possível integrar e coordenar as diferentes partes do projeto, garantindo um desenvolvimento eficiente e robusto do sistema robótico.

O desenvolvimento do projeto segue a lógica apresentada na Figura 16, onde o *middleware* ROS2 é responsável pelo controle da aplicação e pela comunicação entre os diferentes componentes do sistema. A figura mostra o fluxo de dados entre os principais componentes do projeto, detalhados a seguir.

Figura 16 – Topologia do projeto



Fonte: O Autor (2024).

O simulador Gazebo recebe comandos de navegação e retorna dados dos sensores simulados para o ROS2, permitindo a análise e tomada de decisões pelo sistema. Esses dados são interpretados pela aplicação ROS2, que os utiliza para movimentar o robô e, se necessário, acionar os eixos e realizar paradas em caso de detecção de objetos no caminho.

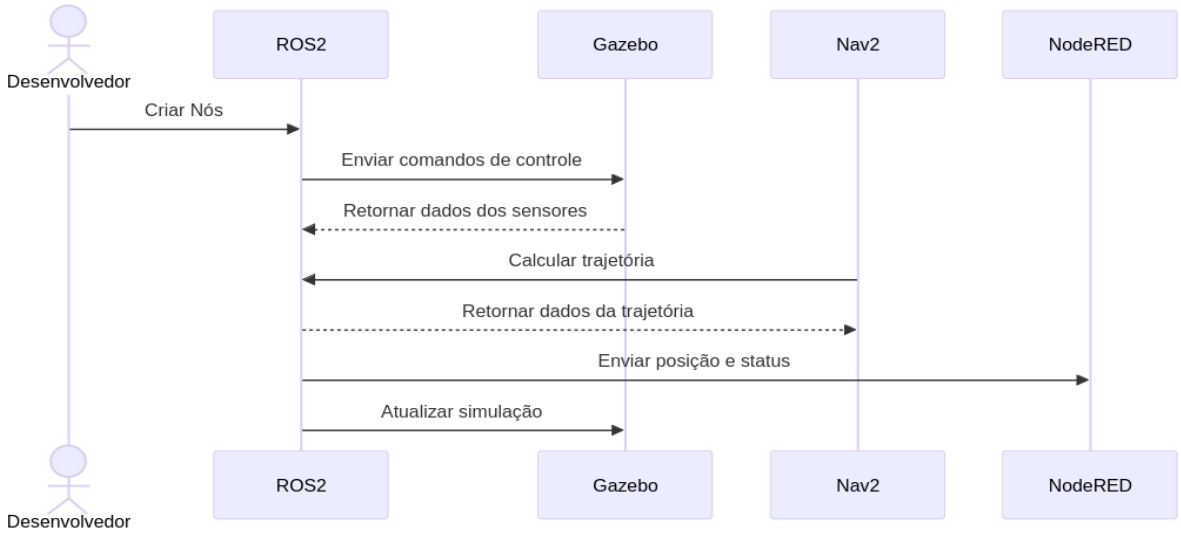
O *NAV2* é responsável pela definição da trajetória de navegação do robô, calculando rotas e monitorando a posição atual do robô. Esse nó envia a posição do robô para o ROS2 e fornece a rota calculada com base na posição desejada e outros parâmetros de navegação, como rotas otimizadas e distâncias dos obstáculos. O Node-RED atua como interface de controle supervisório, permitindo a supervisão do sistema. Este componente recebe os dados do robô, tais como posição e velocidade, além de mensagens de status e posição de origem definida, apresentando esses dados de maneira gráfica ao usuário.

Seguindo práticas de desenvolvimento de *software*, a interação entre os componentes do sistema é definida no diagrama de sequência da Figura 17. Este diagrama mostra a interação entre o desenvolvedor e sistemas como ROS2, Gazebo, Nav2 e NodeRED em um fluxo de simulação e controle de robôs. Nesta sequência, o desenvolvedor cria nós no ROS2, que enviam comandos de controle para o Gazebo, onde a simulação é atualizada, e recebe dados de sensores. Esses dados são usados pelo Nav2 para calcular trajetórias e enviar a posição e o status do robô, que podem ser visualizados ou integrados por meio do NodeRED.

3.1.4 Navegação

Como descrito na seção 2.5.2, a movimentação do robô é baseada nas coordenadas enviadas pelo *Nav2*, inicialmente configurado com o mapa gerado a partir do ambiente simulado no Gazebo.

Figura 17 – Diagrama de Sequência da Aplicação



Fonte: O Autor (2024).

Para realizar esse mapeamento inicial, é necessário desenvolver um controle manual do robô, permitindo que ele percorra o ambiente e identifique todos os pontos, cantos, paredes e trajetos. O controle manual também é útil na aplicação final para situações onde o robô fica preso por condições externas, permitindo ao operador movê-lo manualmente. Inicialmente, o controle foi implementado com as teclas direcionais do teclado, mas uma futura integração com o sistema supervisório é viável.

Após o mapeamento, é possível gerar um arquivo em formato de imagem, que será utilizado pelo Nav2 para navegação autônoma. Os pontos de destino serão definidos pelo autor para simular um cenário onde as coordenadas são informadas por um sistema *Manufacturing Execution System* (MES), que reconhece as demandas de produção e instrui o AMR sobre qual máquina deve atender.

3.1.5 Sistema Operacional

Para garantir compatibilidade com o *ROS2*, o sistema operacional escolhido foi o Linux, em sua distribuição Ubuntu 24.04. Conhecido pela facilidade de uso e ampla adoção em desktops, servidores e, mais recentemente, dispositivos IoT e computação em nuvem, o Ubuntu utiliza o ambiente de desktop GNOME como padrão em suas versões recentes, apresentando um conjunto de ferramentas e programas exclusivos (RAGGI; THOMAS; VUGT, 2011).

A ampla comunidade e suporte são outros fatores que contribuem para o sucesso do Ubuntu. Com uma das maiores comunidades de usuários e desenvolvedores entre as distribuições Linux, o Ubuntu conta com suporte robusto e desenvolvimento contínuo de soluções para problemas comuns, além de documentação detalhada e tutoriais para usuários de todos os níveis, garantindo suporte de longo prazo, essencial para aplicações que demandam estabilidade.

3.2 IMPLEMENTAÇÃO

3.2.1 Organização de Pastas

Uma etapa fundamental no desenvolvimento de uma solução *ROS2* é a organização adequada das pastas, conhecidas como *Work Space* (WS), ou ambiente de trabalho, o qual refere-se ao diretório onde todos os arquivos da aplicação ROS2 serão instalados. A estrutura completa de pastas utilizadas para este trabalho pode ser encontrada no Anexo A.

Um elemento fundamental do WS são os chamados *package*, os componentes que contêm os executáveis e as bibliotecas. Esses arquivos podem ser criados e editados diretamente pelo usuário, porém também podem ser acessados de outras fontes, como GitHub, dada a característica de código aberto de ROS2.

3.2.2 Modelagem URDF

O desenvolvimento do modelo *AMR* descrito na Seção 3.1.2 foi realizado por meio de um arquivo URDF, onde são definidos todos os componentes físicos do robô, como dimensões, fatores de inércia para simulação, e a disposição das rodas e sensores. O código segue uma estrutura lógica, começando pela declaração das propriedades físicas. Esse arquivo utiliza a ferramenta XACRO para simplificar a declaração de tais propriedades.

A declaração das juntas e elos segue a sintaxe mostrada no Algoritmo 2, e é utilizada para definir todos os componentes mecânicos e sensores do robô. Tais informações, como inércia e área de colisão, serão utilizadas durante a simulação no ambiente Gazebo. Como o código completo é uma repetição dessas estruturas com nomes e valores variados, ele não será abordado integralmente aqui, mas está disponível no repositório GitHub do autor[1].

Algoritmo 2 – Exemplo de declaração de elos e juntas em arquivo URDF

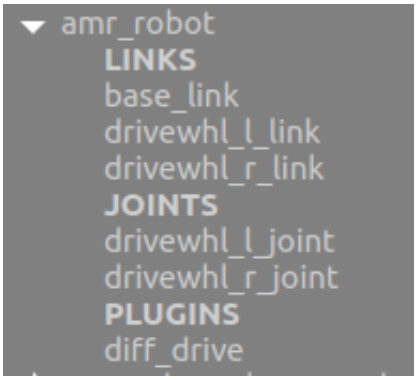
```
1  <link name="base_link">
2    <visual>
3      <geometry><box size="${base_length} ${base_width} ${base_height}"/>
4      </geometry>
5      <material name="Cyan"><color rgba="0 1.0 1.0 1.0"/></material>
6    </visual>
7    <collision>
8      <geometry><box size="${base_length} ${base_width} ${base_height}"/>
9      </geometry></collision>
10   <xacro:box_inertia m="15" w="${base_width}" d="${base_length}"
11   h="${base_height}"/>
12 </link>
```

Fonte: O Autor (2024)

[1] Disponível em <https://github.com/DaHU3nzo/TCC_repo/tree/main/nav2_amr_ros_ws>

Este arquivo também inclui a declaração dos sensores utilizados pelo modelo AMR. Foram incluídos um sensor IMU para cálculo de velocidade e direção, um LiDAR para mapeamento e detecção de obstáculos, e o *plugin* de movimentação diferencial *diff_drive*, que permite a movimentação manual e automática do robô, controlando os eixos diferenciais traseiros.O resultado final deste arquivo é o modelo visualizado na Figura 15, com as estruturas descritas na Figura 18.

Figura 18 – Estrutura do modelo *AMR* gerado pelo arquivo *URDF*



Fonte: O Autor (2024).

3.2.3 Mapeamento SLAM

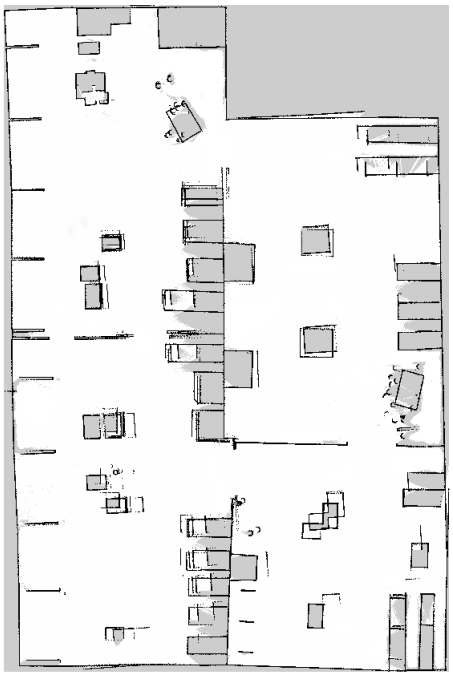
Após o desenvolvimento do mundo e do modelo *AMR*, a etapa seguinte é o mapeamento do ambiente para possibilitar a navegação, utilizando técnicas de SLAM, conforme descrito na Seção 2.5.1. Para isso, uma das ferramentas utilizadas foi o pacote *slam_toolbox*, distribuído gratuitamente por Macenski e Jambrecic (2021).

O objetivo dessa ferramenta é gerar um mapa bidimensional do ambiente real (ou simulado) a partir dos sensores do robô. Para isso, é necessário movimentar manualmente o robô pelo mapa proposto, utilizando o *plugin diff_drive* mencionado anteriormente. Com base nos dados do sensor LiDAR, a ferramenta traça as áreas livres e ocupadas do ambiente, resultando em um arquivo *Portable Gray Map* (PGM), em escala de cinza.

Neste arquivo, as áreas livres (onde o robô pode se mover) são representadas em branco, enquanto as áreas ocupadas e obstáculos aparecem em cinza. No entanto, o mapa apresenta algumas falhas e inconsistências devido ao alcance limitado do sensor LiDAR usado no modelo *AMR*. Como o *slam_toolbox* gera o mapa com base nos obstáculos detectados, em grandes áreas livres onde o LiDAR não detecta objetos dentro de seu alcance máximo, o *plugin* marca essas áreas como ocupadas (cinza) por falta de dados.

Portanto, ajustes manuais são necessários para corrigir o arquivo PGM e representar com precisão o ambiente simulado. Para isso, é possível utilizar softwares de edição gráfica, como Adobe Photoshop, ou alternativas para Ubuntu, como o *GNU Image Manipulation Program* (GIMP). A versão final do mapa, após edição, é apresentada na Figura 19.

Figura 19 – Mapa SLAM editado



Fonte: O Autor (2024).

3.2.4 Arquivo de Fusão de Sensores

Com a grande quantidade de dados processados pela aplicação *ROS2* até esta etapa, conclui-se que é necessário um método para organizar e consolidar essas informações, o que demanda uma técnica de fusão de sensores, conforme discutido na Seção 2.5.3.1. Para isso, foi utilizado um arquivo de configuração chamado *ekf.yaml*, adaptado a partir do exemplo fornecido pelo repositório do Nav2. O código simplificado desse arquivo é apresentado no Algoritmo 3.

Algoritmo 3 – Código YAML para fusão de sensores

```
ekf_filter_node:
    ros__parameters:
        frequency: 30.0      # Frequencia de atualizacao
        odom_frame: odom
        base_link_frame: base_link  # Base do robo
        world_frame: odom           # Odometria fornecida por diff_drive
        odom0: demo/odom
        odom0_config: [true,  true,  true, false, false, false,
                       false, false, false, false, false, true,
                       false, false, false]
        imu0: demo/imu
        imu0_config: [false, false, false, true,  true,  true,
                      false, false, false, false, false, false,
                      false, false, false]
```

Fonte: O Autor (2024)

Este código recebe dados do sensor *IMU* por meio do nó *demo/imu* e dados de odometria do *plugin diff_drive* por meio do tópico *demo/odom*. Cada campo da configuração define quais valores de cada sensor serão usados, onde cada posição corresponde a um dado específico.

Devido a limitações do *software* de navegação e controle, não é possível fundir múltiplos dados derivados entre si. Assim, para a odometria, são configurados como *true* os campos correspondentes à velocidade nos eixos X, Y e Z, além da velocidade angular no eixo Z (*Vyaw*), que determina a velocidade de rotação do modelo. Para o *IMU*, são utilizados os valores de posição angular nos eixos X, Y e Z (*Roll*, *Pitch* e *Yaw*).

3.2.5 Arquivos Launch

Para integrar todos os componentes descritos até esta etapa do projeto, um arquivo *launch* do *ROS* é essencial, permitindo a criação de um arquivo em *Python* capaz de iniciar diversos componentes com caminhos e parâmetros específicos.

No presente projeto, foi criado o arquivo *display.launch.py*, que também reconfigura alguns tópicos com nomes diferentes entre *plugins*, além de definir parâmetros de simulação e caminhos de diretórios. Um trecho desse código pode ser visto no Algoritmo 4.

Algoritmo 4 – Código do arquivo launch

```
1  def generate_launch_description():
2      pkg_share = launch_ros.substitutions.FindPackageShare(
3          package='amr_robot').find('amr_robot')
4      default_model_path = os.path.join(
5          pkg_share, 'src/description/amr_robot.urdf')
6      default_rviz_config_path=os.path.join(pkg_share,'rviz/urdf_config.rviz')
7      world_path = os.path.join(pkg_share, 'world/fabrica_expandida.sdf')
8      launch_dir = os.path.join(pkg_share, 'launch')
9      spawn_entity = launch_ros.actions.Node(
10         package='gazebo_ros', executable='spawn_entity.py',
11         arguments=['-entity', 'amr_robot', '-topic', 'robot_description'],
12         output='screen')
13     localization_launch = IncludeLaunchDescription(
14         PythonLaunchDescriptionSource(
15             os.path.join(launch_dir, 'localization_launch.py'))).items()
```

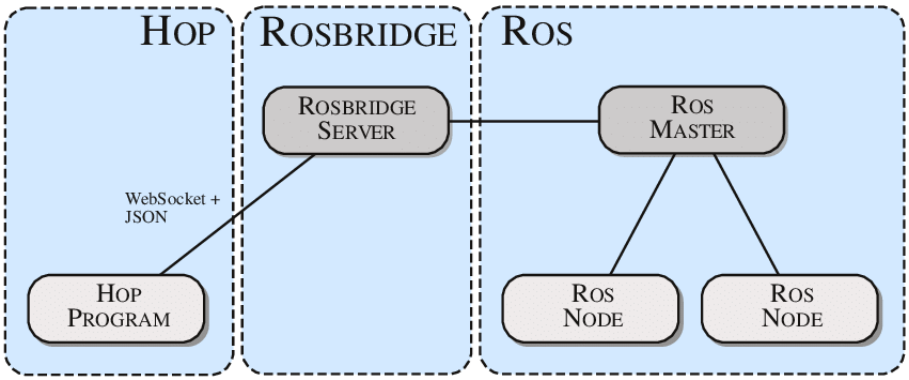
Fonte: O Autor (2024)

Este código conta com algumas diretrizes para inicialização da simulação, tais como o caminho para o arquivo contendo a descrição do robô e do ambiente a ser simulado, o arquivo de configuração da ferramenta RViz, e alguns parâmetros de inserção do robô no mundo simulado, contendo o nome do modelo e tópico que serão utilizados. Com esse arquivo, é possível executar o projeto completo com uma única linha de comando no terminal Ubuntu, facilitando testes e controle centralizado do sistema.

3.2.6 Desenvolvimento do Sistema Supervisório

Para implementar o sistema de monitoramento remoto do robô, algumas ferramentas são necessárias para possibilitar a troca de informações entre o Node-RED e os dados publicados em tópicos específicos pelo ROS2. Primeiramente, é essencial estabelecer a comunicação entre ambos os agentes utilizando o pacote *rosbridge-server*, que emprega *WebSockets* para tornar a estrutura de tópicos acessível para uma aplicação web, como ilustrado na Figura 20.

Figura 20 – Demonstração de comunicação estabelecida através do Rosbridge



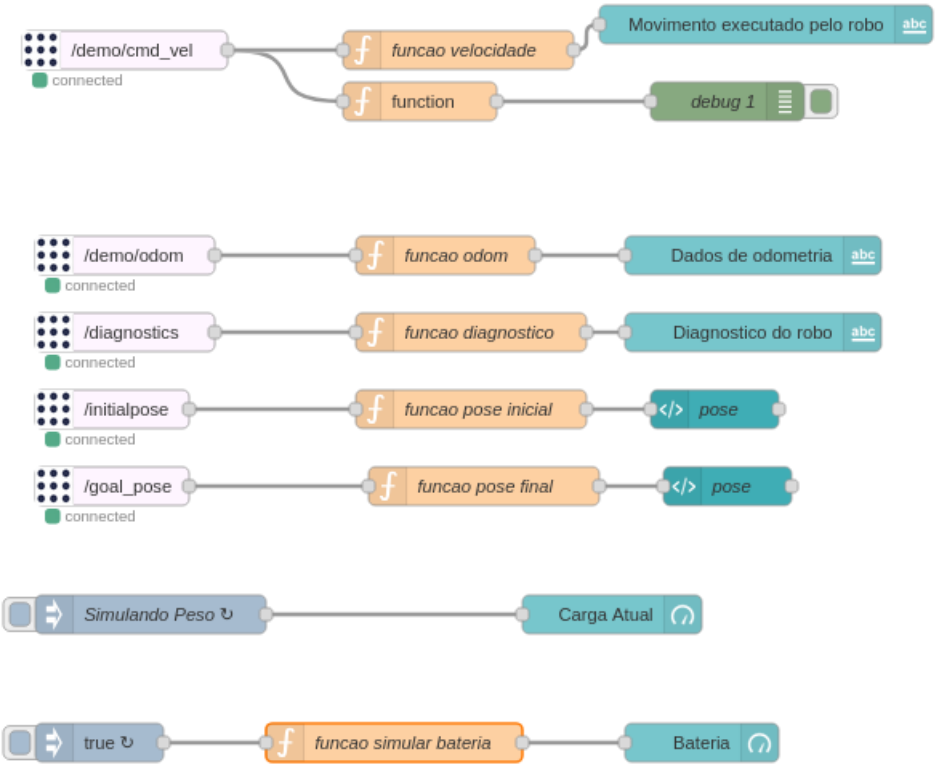
Fonte: Szlenk *et al.* (2015)

Com a conexão estabelecida, os tópicos podem ser acessados como nós na plataforma Node-RED. Como a aplicação *ROS* desenvolvida envolve diversos componentes e uma grande quantidade de dados em transferência, é fundamental utilizar o nome exato do tópico desejado para garantir uma leitura correta. Para isso, é possível listar todos os tópicos ativos por meio de um terminal Ubuntu, resultando na lista exibida no Anexo B.

Com os dados devidamente mapeados para seus respectivos tópicos, é possível desenvolver um sistema de monitoramento remoto em Node-RED utilizando o pacote *Dashboard*. A Figura 21 mostra como esses tópicos são usados como entradas para funções que convertem os dados em um formato legível, realizando ajustes de unidades e organizando-os em linhas separadas.

É importante frisar que para alguns dados, como nível de carga e estado da bateria, foi atribuído um valor inicial. Isso ocorre porque essas informações só estariam disponíveis em uma aplicação real, onde sensores específicos de carga e medição da bateria estariam presentes. Esses dados foram, entretanto, incluídos no painel de monitoramento com fins de demonstração.

Figura 21 – Estrutura de código para o sistema de monitoramento



Fonte: Szlenk *et al.* (2015)

4 ANÁLISE DE TESTES E RESULTADOS

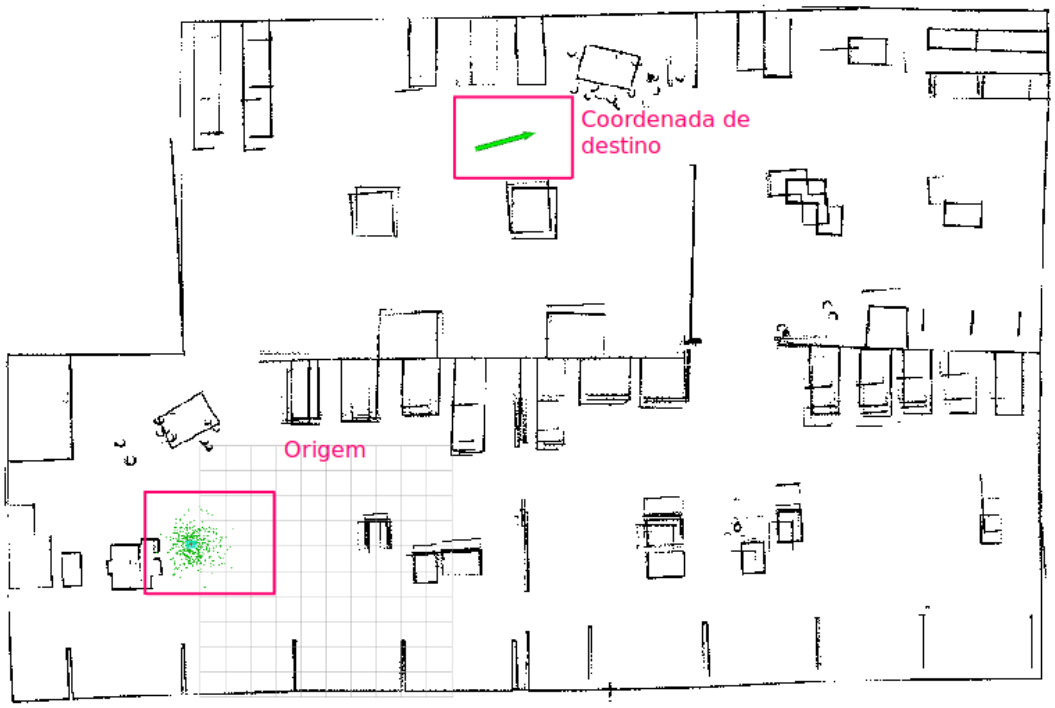
Após o desenvolvimento da aplicação, é necessária uma análise crítica para avaliar sua funcionalidade e precisão. Este capítulo descreve os resultados obtidos, visando validar sua aplicabilidade.

4.1 FUNCIONALIDADE

Para iniciar a simulação corretamente, é necessário definir o ponto de inserção do robô no mapa, atribuindo uma posição de origem via *Rviz*. Esse passo, embora essencial, não exige precisão extrema, pois o sistema de controle é capaz de corrigir a posição do robô comparando os dados dos sensores com o mapa global.

Os pontos de destino são então definidos pelo usuário, simulando um processo de transporte entre setores da fábrica. Esses pontos são inseridos diretamente no mapa do *Rviz* de forma gráfica, usando o botão *Nav2 Goal*, conforme apresentado na Figura 22. O servidor de planejamento de rota da biblioteca *Nav2* calcula a rota ideal entre a posição atual e a desejada, de acordo com o algoritmo de planejamento selecionado, conforme descrito na Seção 3.1.4.

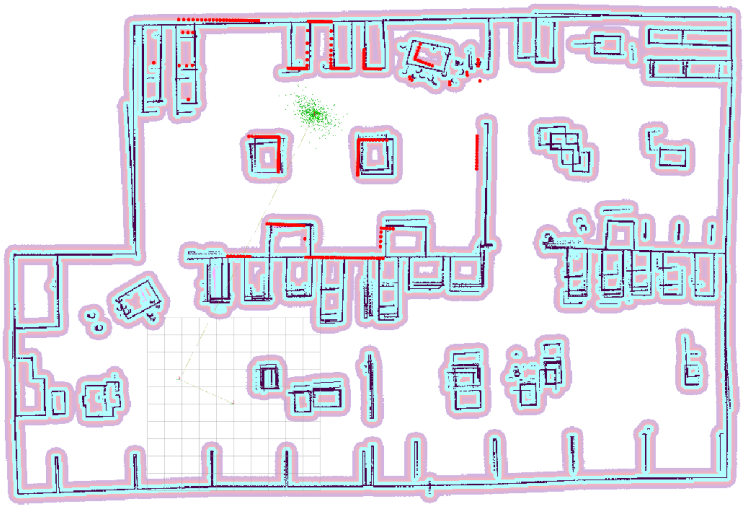
Figura 22 – Definição gráfica das coordenadas de origem e destino



Fonte: O autor (2024).

Para controle, a simulação é configurada para iniciar em modo de pausa, permitindo ajustes iniciais na posição do robô e modificações visuais no *Rviz*. Após iniciar a execução, os valores dos sensores *Lidar* recebidos através do mapa local são exibidos em contraste com o mapa global. Quando o ponto de inserção inicial é corretamente definido, o resultado é semelhante ao mostrado na Figura 23, com os pontos do *Lidar* (em vermelho) alinhados com o mapa.

Figura 23 – Alinhamento dos dados do Lidar com o mapa



Fonte: O autor (2024).

Simultaneamente, a simulação é executada no ambiente virtual do *Gazebo*, onde o robô é representado em tempo real, incluindo a emissão do laser do *Lidar*. A Figura 24 mostra a representação visual da movimentação do robô no ambiente simulado.

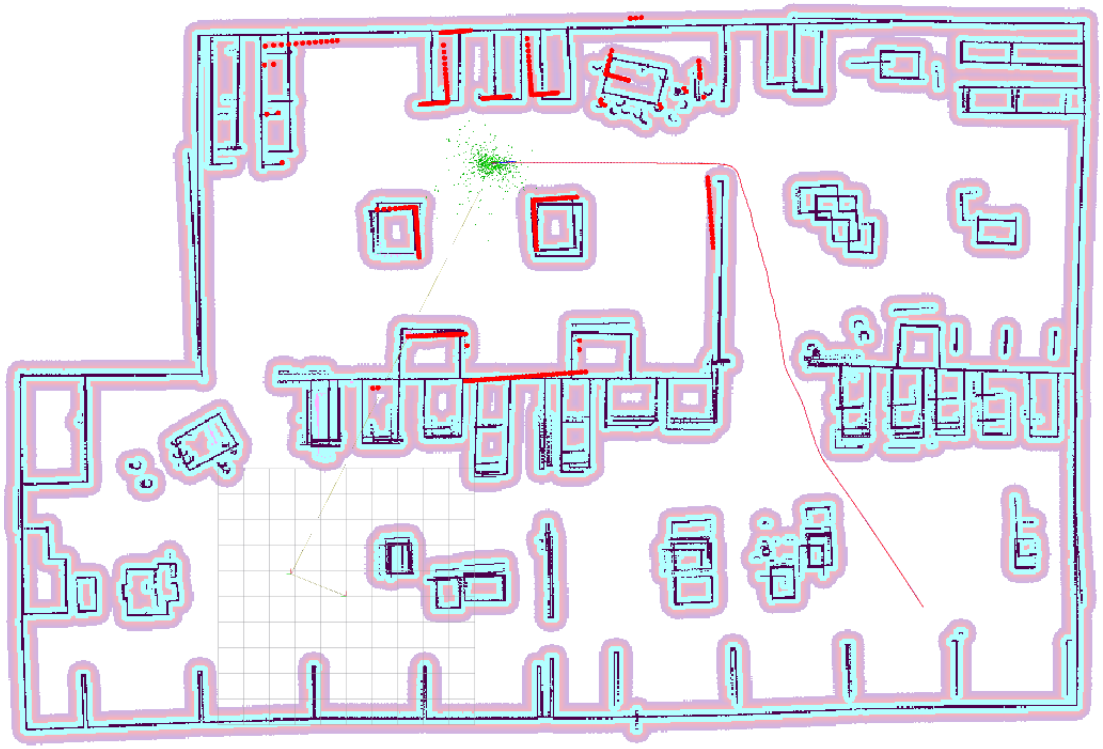
Figura 24 – Representação do robô AMR no ambiente simulado

Fonte: O autor (2024).

4.1.1 Movimentação

O fornecimento de dados de origem e destino ao módulo *Nav2* e a execução da simulação permitem que o sistema calcule a menor rota entre esses pontos, considerando as condições do ambiente. Essa decisão é tomada pelo algoritmo configurado. Observou-se que o *Adaptive Monte-Carlo Localizer* (AMCL) atende de forma satisfatória, como ilustrado na Figura 25, onde a rota planejada é representada pela linha vermelha.

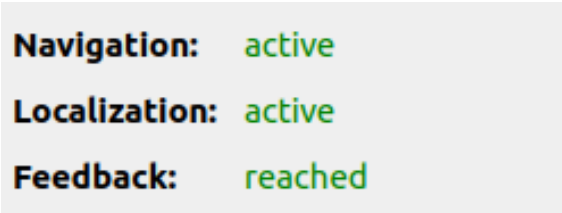
Figura 25 – Planejamento de rota com o algoritmo AMCL



Fonte: O autor (2024).

Durante o percurso entre origem e destino, o *plugin Nav2* para *Rviz* exibe o tempo de deslocamento e a distância restante até o destino, funcionando como uma ferramenta de monitoramento. Quando o destino é alcançado, conforme a tolerância configurada no *Nav2*, o status exibe a mensagem "Alcançado", como mostrado na Figura 26.

Figura 26 – Mensagem indicando que o objetivo foi alcançado

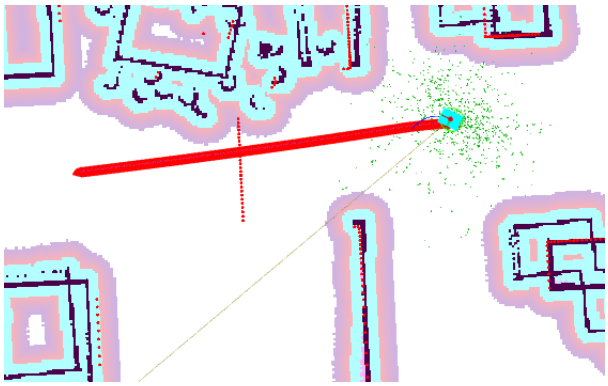


Fonte: O autor (2024).

4.1.2 Capacidade de evitar obstáculos

Outra capacidade importante de um robô AMR, e que portanto deve ser testada a fim de validação do método de controle, é a capacidade de desviar de objetos dinâmicos impostos ao longo da trajetória calculada pelo robô. Para isso, foram adicionados alguns objetos novos ao ambiente simulado, os quais não estavam presente durante a etapa de mapeamento original. Como demonstrado pela Figura 27, a rota calculada não considera tais objetos, uma vez que os mesmos não são previamente conhecidos pelo servidor de planejamento de rota, assim traçando um caminho através de tal objeto.

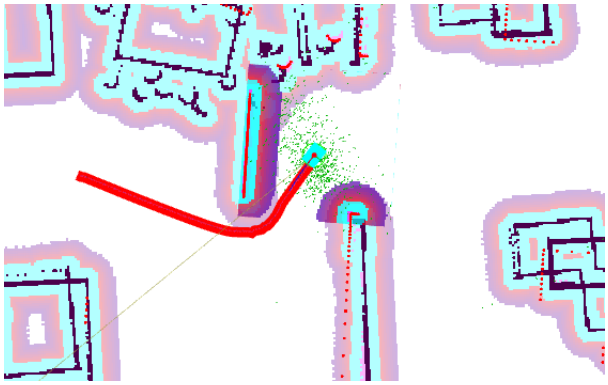
Figura 27 – Demonstração de rota calculada original



Fonte: O autor (2024).

Contudo, a Figura 28 demonstra que conforme o robô se aproxima do objeto desconhecido, o seu sensor LiDAR é capaz de detectar o mesmo, passando a exibir o obstáculo no mapa local. Com isso, a rota original é modificada para evitar o novo obstáculo, recalculando o percurso mais curto para desviar de tal objeto e assim poder atingir o objetivo final.

Figura 28 – Demonstração de rota modificada



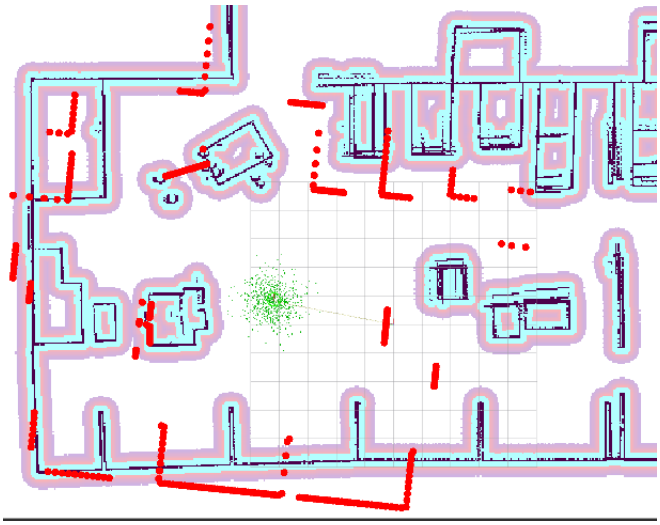
Fonte: O autor (2024).

Tais figuras comprovam a adaptabilidade do sistema de controle proposto, garantindo a eficiência de movimentação enquanto permite a utilização de rotas flexíveis.

4.2 PRECISÃO

A precisão de movimentação é uma característica crucial de um robô AMR e dos sistemas de controle envolvidos. Para assegurar a consistência entre o mapa gerado e os dados dos sensores em tempo real, o *Nav2* corrige automaticamente a posição do robô. Para testar essa funcionalidade, o robô foi propositalmente posicionado incorretamente, como mostrado na Figura 29, gerando assim uma situação em que os dados recebidos pelo sensor não coincidem com o mapa global esperado.

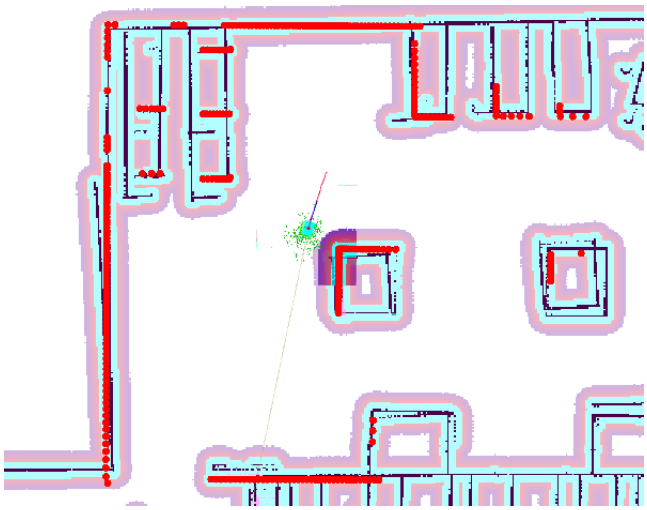
Figura 29 – Posição inicial incorreta para teste de correção



Fonte: O autor (2024).

O robô foi então movimentado para uma posição aleatória, possibilitando a interação entre os dados do Lidar e o mapa. Após alguns momentos, observou-se, como mostra a Figura 30, que o sistema corrigiu a posição do robô, alinhando-a com o mapa.

Figura 30 – Correção da posição com base nos dados do Lidar



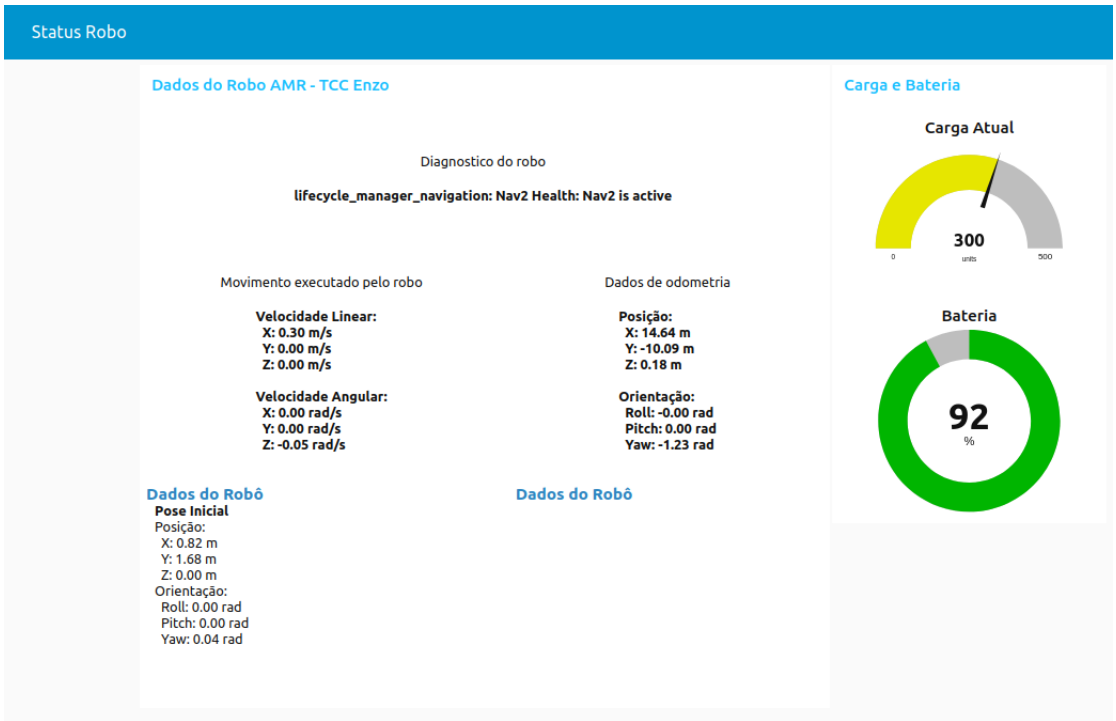
Fonte: O autor (2024).

4.3 SISTEMA DE MONITORAMENTO

Durante a simulação, os tópicos listados na Figura 33 são preenchidos com dados recebidos pelo *Gazebo*. Isso permite que os dados sejam processados pelo Node-RED e exibidos de forma clara e legível no painel de monitoramento. Dados como velocidade são apresentados em metros ou radianos por segundo, e a posição é exibida em metros a partir da origem (coordenada 0,0,0 no ambiente simulado). O painel também exibe diagnósticos do robô, incluindo mensagens de controle e navegação.

Os dados de bateria e carga são apresentados para simular condições reais, onde o robô transportaria uma carga específica. Esse monitoramento permite avaliar o esforço realizado, assegurando que a carga não ultrapasse um valor seguro. A porcentagem de bateria também está representada para fins de demonstração.

Figura 31 – Painel de monitoramento remoto



Fonte: O autor (2024).

5 CONSIDERAÇÕES FINAIS

Com base nos resultados obtidos neste projeto, algumas considerações podem ser feitas a respeito da eficácia e aplicabilidade da solução desenvolvida. A escolha pela simulação no Gazebo, em conjunto com o uso de ferramentas como *ROS2* e suas bibliotecas, foi fundamental para validar a funcionalidade do sistema em um ambiente virtual. Essa abordagem possibilitou ajustes e aprimoramentos importantes antes de qualquer implementação física, promovendo uma base mais segura e otimizada para futuras aplicações.

O modelo cinemático diferencial do robô, aliado ao uso de sensores como LiDAR e IMU, demonstrou alta confiabilidade na integração de dados sensoriais, permitindo ajustes dinâmicos na trajetória e evitando colisões com obstáculos móveis e estáticos. A fusão de sensores, realizada por meio de algoritmos de predição e correção, garantiu a consistência dos dados coletados, resultando em trajetórias suaves e otimizadas para o robô. Além disso, o sistema de planejamento de rotas, implementado com algoritmo Dijkstra, apresentou elevada eficiência no cálculo de trajetórias globais e locais, utilizando mapas estáticos e dinâmicos para lidar com as complexidades do ambiente industrial. O sistema de monitoramento remoto, por sua vez, provou ser funcional na supervisão do estado do robô. Isso confirma a viabilidade da solução para operações industriais que exigem integração entre supervisores humanos e sistemas robóticos.

A simulação permitiu a análise de diversos aspectos críticos da navegação autônoma, incluindo a precisão da localização e a capacidade de desvio de obstáculos, ambos essenciais para o sucesso de um sistema *AMR* em ambientes industriais complexos e dinâmicos, conforme ilustrado neste trabalho. Esse processo de simulação e ajuste não só aproximou o sistema simulado do cenário real, mas também proporcionou uma redução potencial de custos e de tempo para implementações futuras, dado o alto nível de confiabilidade das ferramentas utilizadas.

Os resultados indicam que o robô foi capaz de executar tarefas de movimentação e navegação autônomas, detectando e evitando obstáculos de forma eficiente, utilizando algoritmos de localização e mapeamento baseados em SLAM. Isso valida o potencial do ROS2 como uma solução modular e escalável, adequada para a integração de sensores e controle distribuído em ambientes industriais dinâmicos.

5.1 TRABALHOS FUTUROS

Devido ao caráter de simulação do presente trabalho, destaca-se a necessidade de implementar fisicamente o modelo desenvolvido . Essa transição permitirá validar os resultados obtidos nas simulações em condições reais, enfrentando desafios como interferências no sensoriamento e a integração entre hardware e software, sendo assim um passo essencial para demonstrar a aplicabilidade do sistema no ambiente industrial e avaliar sua performance prática.

Além disso, os algoritmos de navegação e controle de movimento podem ser aprimorados para melhorar a eficiência e a capacidade de resposta do robô em tempo real. Especialmente em cenários dinâmicos e complexos, o refinamento desses algoritmos pode reduzir o tempo de processamento e ampliar a segurança operacional do sistema, garantindo que os sistemas embarcado escolhido seja capaz de processar todos os dados do projeto. A utilização de técnicas avançadas, como aprendizado de máquina, também pode ser explorada para otimizar a navegação e a adaptação do robô a mudanças no ambiente.

Outro aspecto promissor é a integração do sistema com tecnologias de IoT, permitindo o monitoramento remoto e o controle em tempo real. Isso abrirá novas possibilidades para a gestão de frotas de AMRs, possibilitando a sincronização de múltiplos robôs em operações logísticas complexas. Essa conectividade também pode fornecer dados operacionais valiosos para análise e otimização contínua do sistema.

Por fim, é recomendada a expansão dos testes e simulações para cenários industriais mais diversos, como armazéns de múltiplos níveis Essa abordagem permitirá avaliar a robustez e a adaptabilidade do sistema em diferentes condições. O desenvolvimento de interfaces mais intuitivas e avançadas para monitoramento, incluindo visualizações em tempo real e controle via dispositivos móveis, também poderá ampliar a aplicabilidade e a aceitação do sistema por usuários finais em ambientes industriais.

REFERÊNCIAS

ALATISE, M. B.; HANCKE, G. P. A review on challenges of autonomous mobile robot and sensor fusion methods. **IEEE Access**, v. 8, p. 39830–39846, 2020.

BEHROOZPOUR, B. *et al.* Lidar system architectures and circuits. **IEEE Communications Magazine**, v. 55, n. 10, p. 135–142, 2017.

CARRARA, V. **INTRODUÇÃO À ROBÓTICA INDUSTRIAL**. [s.n.], 2015. Disponível em: <http://urlib.net/8JMKD3MGP3W34P/3K5JPL8>. Acesso em: 02 abr 2024.

COLLEDANCHISE, M.; NATALE, L. On the implementation of behavior trees in robotics. **IEEE Robotics and Automation Letters**, v. 6, n. 3, p. 5929–5936, 2021.

CRAIG, J. J. **Introduction to robotics : mechanics and control**. 3rd international ed.. ed. Pearson Higher Education, 2014. ISBN 9781292040042; 1292040041. Disponível em: <libgen.li/file.php?md5=a68ae21d06c80f1ff8ad0251f6d113e0>.

FAISAL, I. A.; PURBOYO, T. W.; ANSORI, A. S. R. A review of accelerometer sensor and gyroscope sensor in imu sensors on motion capture. **J. Eng. Appl. Sci**, v. 15, n. 3, p. 826–829, 2019.

FEDER, M.; GIUSTI, A.; VIDONI, R. An approach for automatic generation of the urdf file of modular robots from modules designed using solidworks. **Procedia Computer Science**, v. 200, p. 858–864, 2022. ISSN 1877-0509. 3rd International Conference on Industry 4.0 and Smart Manufacturing. Disponível em: <https://www.sciencedirect.com/science/article/pii/S1877050922002927>.

GAZEBOSIM.ORG. 2024. Disponível em: <https://gazebosim.org/docs/fortress/ros2_integration>.

____. 2024. Disponível em: <https://gazebosim.org/docs/latest/ros_installation#gazebo-garden-with-ros-2-humble-iron-or-rolling-use-with-caution->.

GOBHINATH, S. *et al.* Simultaneous localization and mapping [slam] of robotic operating system for mobile robots. In: **2021 7th International Conference on Advanced Computing and Communication Systems (ICACCS)**. [S.l.: s.n.], 2021. v. 1, p. 577–580.

HAGINO, T.; O'LEARY, N. **Practical Node-RED Programming: Learn powerful visual programming techniques and best practices for the web and IoT**. Packt Publishing, 2021. ISBN 9781800207660. Disponível em: <https://books.google.se/books?id=nlwlEAAAQBAJ>.

KAGAN, E. **Autonomous mobile robots and multi-robot systems : motion-planning, communication, and swarming**. Wiley, 2020. ISBN 9781119213154; 1119213150; 9781119213161; 1119213169; 9781119213178; 1119213177; 9781119212867; 1119212863. Disponível em: <libgen.li/file.php?md5=bc17c3b7bae701e444d28581d312a9e0>.

KIM, D. **Advanced Mobile Robotics: Volume 3**. MDPI - Multidisciplinary Digital Publishing Institute, 2020. ISBN 9783039219469; 3039219464; 9783039219476; 3039219472; 9783039219421; 3039219421; 9783039219438; 303921943X. Disponível em: <libgen.li/file.php?md5=de745d7fa0d59b772ddc7bbf38d8c9f6>.

KOENIG, N.; HOWARD, A. Design and use paradigms for gazebo, an open-source multi-robot simulator. In: **2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)**. [S.l.: s.n.], 2004. v. 3, p. 2149–2154 vol.3.

KOUBAA, A. **Robot Operating System (ROS): The Complete Reference (Volume 5)**. Springer International Publishing, 2020. (Studies in Computational Intelligence). ISBN 9783030459567. Disponível em: <https://books.google.se/books?id=8OD4DwAAQBAJ>.

MACENSKI, S.; JAMBRECIC, I. Slam toolbox: Slam for the dynamic world. **Journal of Open Source Software**, The Open Journal, v. 6, n. 61, p. 2783, 2021. Disponível em: <https://doi.org/10.21105/joss.02783>.

MARUYAMA, Y.; KATO, S.; AZUMI, T. Exploring the performance of ros2. In: **Proceedings of the 13th International Conference on Embedded Software**. New York, NY, USA: Association for Computing Machinery, 2016. (EMSOFT '16). ISBN 9781450344852. Disponível em: <https://doi.org/10.1145/2968478.2968502>.

MATHEW, R.; HIREMATH, S. S. Control of velocity-constrained stepper motor-driven hilare robot for waypoint navigation. **Engineering**, v. 4, n. 4, p. 491–499, 2018. ISSN 2095-8099. Disponível em: <https://www.sciencedirect.com/science/article/pii/S2095809917303922>.

NAV2.ORG. 2024. Disponível em: <https://docs.nav2.org/>.

OPENROBOTICS. 2024. Disponível em: <http://wiki.ros.org/rviz>.

PALANGE, A.; DHATRAK, P. Lean manufacturing a vital tool to enhance productivity in manufacturing. In: . [S.l.]: Elsevier, 2021. v. 46, p. 729–736. ISSN 2214-7853.

RAGGI, E.; THOMAS, K.; VUGT, S. V. **Beginning ubuntu linux**. [S.l.]: Springer, 2011.

RAMESH, N. Y. N. Ros based communication system for agvs : A service oriented architecture (soa) approach. In: . [s.n.], 2016. Disponível em: <https://api.semanticscholar.org/CorpusID:70004763>.

RICO, F. M. **A Concise Introduction to Robot Programming with ROS2**. 1. ed. Chapman and Hall/CRC, 2022. ISBN 9781032267203; 1032267208. Disponível em: <libgen.li/file.php?md5=cad8e6cc37b30b832453dba239465c47>.

ROS.ORG. **Ros2 Documentation**. 2024. Disponível em: <https://www.ros.org>.

SIEGWART, R.; NOURBAKHSH, I.; SCARAMUZZA, D. **Introduction to Autonomous Mobile Robots, second edition**. MIT Press, 2011. (Intelligent Robotics and Autonomous Agents series). ISBN 9780262015356. Disponível em: <https://books.google.se/books?id=4of6AQAAQBAJ>.

SIERRA-GARCíA, J. E.; SANTOS, M. Mechatronic modelling of industrial agvs: A complex system architecture. **Complexity**, Hindawi Limited, v. 2020, 2020. ISSN 10990526.

SOROUSH, M.; BALDEA, M.; EDGAR, T. **Smart Manufacturing: Concepts and Methods**. Elsevier, 2020. ISBN 9780128203804. Disponível em: <https://books.google.se/books?id=ZSjJDwAAQBAJ>.

SZLENK, M. *et al.* Reconfigurable agent architecture for robots utilising cloud computing. **Advances in Intelligent Systems and Computing**, v. 351, p. 253–264, 01 2015.

TANG, Y.-Y.; WU, Q. Design and realization of magnetic and laser hybrid navigation in agv system. v. 12709, p. 127094F – 127094F–11, 2023.

VORONOVA, O. Improvement of warehouse logistics based on the introduction of lean manufacturing principles. In: . [S.l.]: Elsevier, 2022. v. 63, p. 919–928. ISSN 2352-1465.

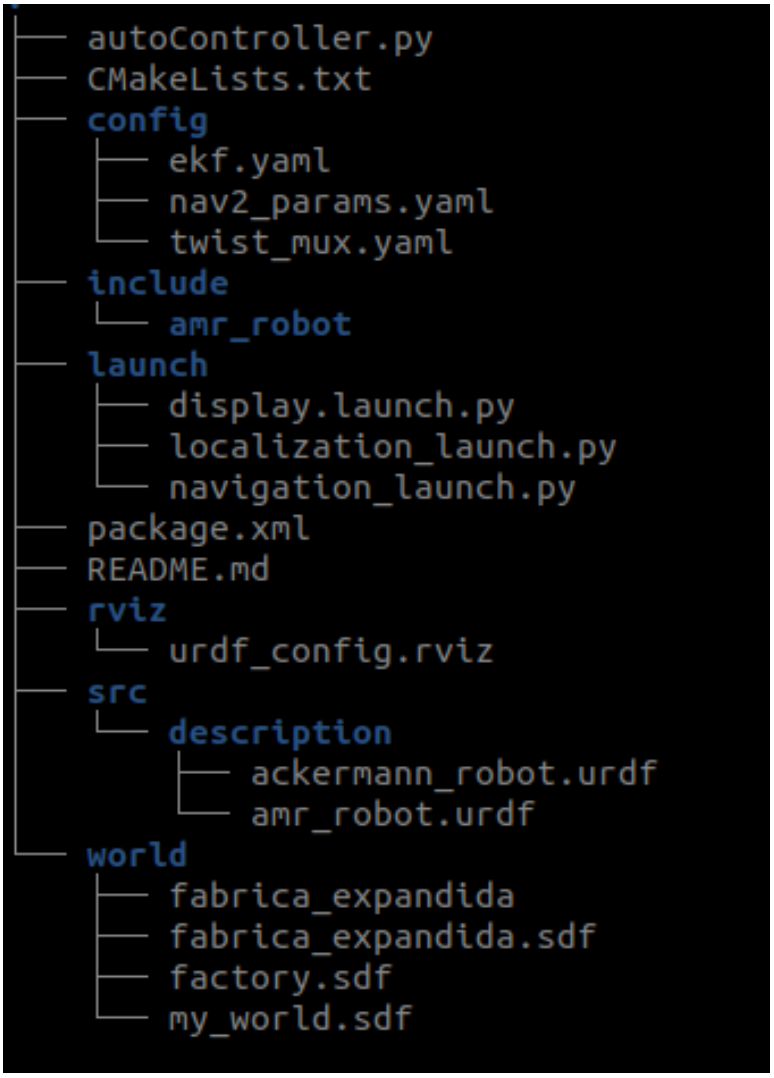
WAHAB, A. N. A.; MUKHTAR, M.; SULAIMAN, R. A conceptual model of lean manufacturing dimensions. In: . [S.l.]: Elsevier, 2013. v. 11, p. 1292–1298. ISSN 2212-0173.

ZHANG, H.-y.; LIN, W.-m.; CHEN, A.-x. Path planning for the mobile robot: A review. **Symmetry**, v. 10, n. 10, 2018. ISSN 2073-8994. Disponível em: <https://www.mdpi.com/2073-8994/10/10/450>.

ZHANG, J. *et al.* Automated guided vehicles and autonomous mobile robots for recognition and tracking in civil engineering. **Automation in Construction**, v. 146, p. 104699, 2023. ISSN 0926-5805. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0926580522005696>.

ANEXO A – ESTRUTURAÇÃO DE PASTAS UTILIZADAS PARA O PROJETO

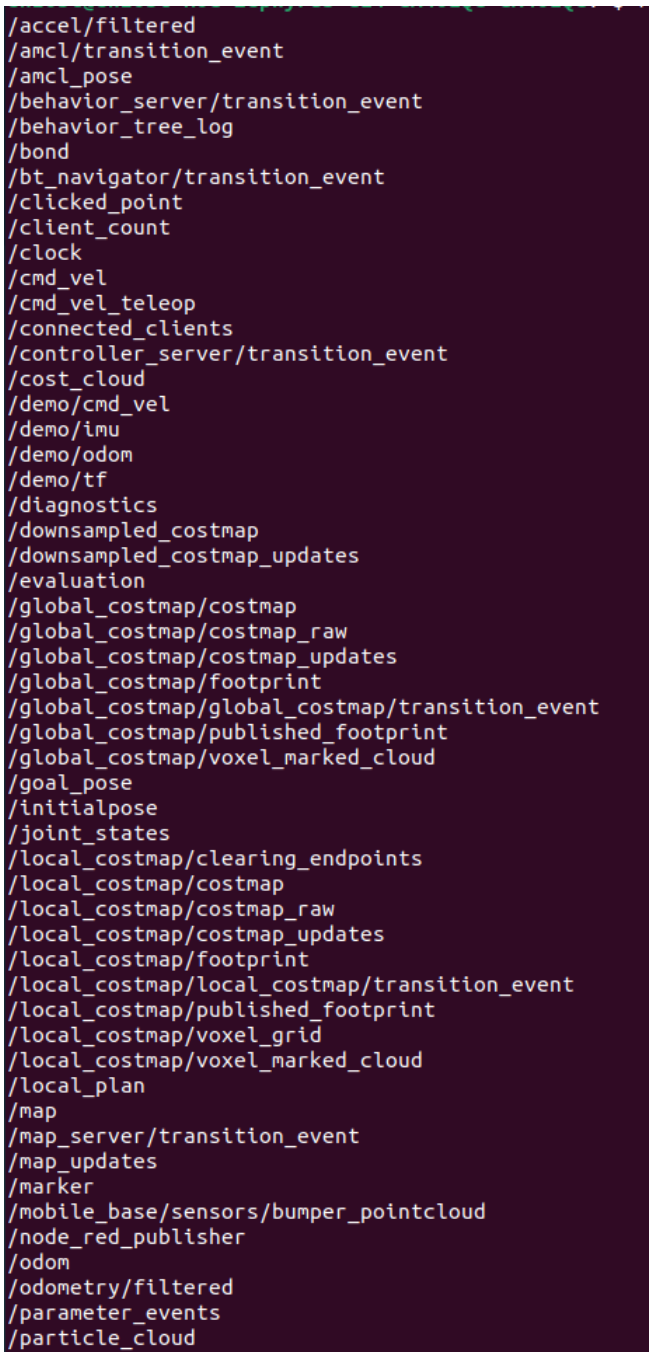
Figura 32 – Estrutura de arquivos para solução *ROS*



Fonte: O Autor (2024).

ANEXO B – DEMONSTRAÇÃO DE TÓPICOS ATIVOS DURANTE A SIMULAÇÃO

Figura 33 – Lista de tópicos ativos no projeto



Fonte: O autor (2024)