Isac Gomes de Almeida Silva

Avaliação experimental de algoritmos para localização, planejamento e controle de navegação de robôs móveis com tração diferencial

Campina Grande, Paraíba

Março de 2026

Isac Gomes de Almeida Silva

Avaliação experimental de algoritmos para localização, planejamento e controle de navegação de robôs móveis com tração diferencial

Trabalho de Conclusão de Curso submetido à Coordenadoria do Curso de Graduação em Engenharia Elétrica da Universidade Federal de Campina Grande, *Campus* Campina Grande, como parte dos requisitos necessários para obtenção do Grau de Bacharel em Engenharia Elétrica.

Universidade Federal de Campina Grande - UFCG

Centro de Engenharia Elétrica e Informática - CEEI

Departamento de Engenharia Elétrica - DEE

Orientador: Antonio Marcus Nogueira Lima, Dr.

Campina Grande, Paraíba

Março de 2026

Isac Gomes de Almeida Silva

Avaliação experimental de algoritmos para localização, planejamento e controle de navegação de robôs móveis com tração diferencial

Trabalho de Conclusão de Curso submetido à Coordenadoria do Curso de Graduação em Engenharia Elétrica da Universidade Federal de Campina Grande, *Campus* Campina Grande, como parte dos requisitos necessários para obtenção do Grau de Bacharel em Engenharia Elétrica.

Trabalho aprovado em:

Antonio Marcus Nogueira Lima, Dr.
Orientador

Danilo Freire de Souza Santos, D.Sc
Professor Convidado

Campina Grande, Paraíba
Março de 2026

Agradecimentos

Agradeço primeiramente a Deus, por guiar meus passos ao longo de toda essa jornada e por abrir portas fundamentais que me fizeram chegar até aqui.

À minha família, em especial à minha mãe Alice e ao meu pai Boaventura, *in memoriam*, que tanto me incentivaram. As ligações diárias, regadas a amor, carinho e cuidado, nos mantiveram próximos, mesmo a milhares de quilômetros de distância. As contagens regressivas para a chegada do período de férias e a possibilidade de ir para casa eram sempre recompensadas com abraços calorosos.

Aos amigos que fiz na Paraíba, que tornaram essa caminhada mais leve. Aos amigos que ficaram na Bahia, que mesmo à distância nunca deixaram de oferecer apoio e incentivo, meu muito obrigado.

Agradeço também ao meu orientador Antonio Marcus, pelos conhecimentos transmitidos ao longo da graduação, em especial durante os projetos no laboratório Smart Factoring. Sua orientação, dedicação e comprometimento foram fundamentais para a minha formação acadêmica e profissional.

Resumo

Este trabalho apresenta uma avaliação experimental da estrutura Navigation2 (Nav2) aplicada ao robô móvel TurtleBot3 Waffle Pi em ambiente laboratorial, com foco na comparação entre planejadores globais e controladores locais. Inicialmente, foi realizada a validação do robô, a construção do mapa do ambiente com o *slam_toolbox* e a definição do *footprint*. Em seguida, foram ajustados parâmetros dos *costmaps* e avaliadas diferentes configurações dos planejadores NavFn e Smac, bem como dos controladores DWB e Regulated Pure Pursuit (RPP). Os resultados mostraram que o NavFn apresentou menor tempo de planejamento, enquanto o Smac gerou caminhos mais curtos. Na comparação entre os controladores, o RPP apresentou movimento mais contínuo, enquanto o DWB precisou interromper momentaneamente a navegação em algumas situações de proximidade com obstáculos. Por fim, os ensaios com a arquitetura completa do Nav2 confirmaram a viabilidade funcional da navegação autônoma no cenário avaliado.

Palavras-chave: robô móvel; ROS 2; Navigation2; TurtleBot3; navegação autônoma.

Abstract

This work presents an experimental evaluation of the Navigation2 (Nav2) stack applied to the TurtleBot3 Waffle Pi mobile robot in a laboratory environment, focusing on the comparison of global planners and local controllers. First, the robot was validated, the environment map was built using *slam_toolbox*, and the robot footprint was defined. Next, *costmap* parameters were adjusted, and different configurations of the NavFn and Smac planners, as well as the DWB and Regulated Pure Pursuit (RPP) controllers, were evaluated. The results showed that NavFn achieved lower planning time, while Smac generated shorter paths. In the controller comparison, RPP showed smoother and more continuous motion, whereas DWB had to momentarily stop in some situations near obstacles. Finally, tests with the complete Nav2 architecture confirmed the functional feasibility of autonomous navigation in the evaluated scenario.

Keywords: mobile robot; ROS 2; Navigation2; TurtleBot3; autonomous navigation.

Lista de ilustrações

Figura 1 – Sistema Robótico. 4
Figura 2 – Robôs móveis. A terrestre, B aquático, C aéreo 6
Figura 3 – Sistemas de coordenadas do robô e do ambiente 7
Figura 4 – Representação das partículas no processo de localização probabilística do AMCL. 15
Figura 5 – Mapa métrico e mapa topológico. 16
Figura 6 – Diagrama de grafo 19
Figura 7 – Exemplo de janela dinâmica. 20
Figura 8 – Representação geométrica do método *Pure Pursuit*. 21
Figura 9 – Representação de funcionamento de interação entre nós no ROS. 24
Figura 10 – Exemplo árvore de transformação. 25
Figura 11 – Turtlebot 3 Waffle Pi. 29
Figura 12 – Hardware do OpenCR. 30
Figura 13 – LiDAR LDS-02. 31
Figura 14 – Laboratório Smart Factoring. 33
Figura 15 – Estrutura do Nav2. 34
Figura 16 – Comparação entre a aproximação circular do robô e o *footprint* poligonal adotado. 36
Figura 17 – Procedimento experimental adotado. 39
Figura 18 – Robô posicionado no ponto P1. 41
Figura 19 – Estrutura hierárquica da árvore TF do sistema. 44
Figura 20 – Distância medida e leitura do sensor LiDAR. 45
Figura 21 – Mapa de grade de ocupação do laboratório. 45
Figura 22 – Validação do mapa por sobreposição entre o mapa de ocupação e o *LaserScan*. 46
Figura 23 – Influência do parâmetro *inflation_radius* no mapa de custo e no caminho global planejado. 47
Figura 24 – Caminhos gerados a partir da posição P1. 48
Figura 25 – Caminhos gerados a partir da posição P2. 49
Figura 26 – Trajetórias executadas pelo controlador DWB com diferentes configurações. 51
Figura 27 – Trajetórias executadas pelo controlador RPP com diferentes configurações. 52

Lista de Abreviaturas e Acrônimos

CEEI	Centro de Engenharia Elétrica e Informática.
UFCG	Universidade Federal de Campina Grande.
Embedded	Laboratório de Sistemas Embarcados e Computação Pervasiva.
ROS	*Robot Operating System.*
RViz	*ROS Visualization* (Ferramenta de Visualização do ROS).
IoT	*Internet of Things* (Internet das Coisas).
AMCL	*Adaptive Monte Carlo Localization.*
BT	*Behavior Tree.*
DDS	*Data Distribution Service.*
DWB	*Dynamic Window Approach Based.*
LiDAR	*Light Detection and Ranging.*
MPPI	*Model Predictive Path Integral.*
Nav2	*Navigation2.*
OMPL	*Open Motion Planning Library.*
RPP	*Regulated Pure Pursuit.*
SLAM	*Simultaneous Localization and Mapping.*
TF	*Transform.*
TF2	*Transform Library 2.*
URDF	*Unified Robot Description Format.*

Sumário

1 INTRODUÇÃO 1
1.1 Justificativa 2
1.2 Objetivo Geral 2
1.3 Organização do Documento 3

2 FUNDAMENTAÇÃO TEÓRICA 4
2.1 Sistemas Robóticos em Ambientes Industriais 4
2.1.1 Robô Móvel 5
2.1.1.1 Cinemática de robôs móveis 6
2.2 Percepção 9
2.2.1 Sensores 9
2.2.1.1 Encoders 10
2.2.1.2 IMU 11
2.2.1.3 LiDAR 12
2.3 Localização 12
2.3.1 Odometria e incertezas na estimação de pose 12
2.3.2 Localização probabilística 14
2.4 Mapeamento 15
2.4.1 Grades de ocupação 16
2.4.2 Simultaneous Localization and Mapping 17
2.5 Planejamento de Caminho 18
2.5.1 Algoritmos baseados em grafos 18
2.6 Controle de Movimento 19
2.6.1 Dynamic Window Approach (DWA) 20
2.6.2 Pure Pursuit 21
2.6.2.1 Regulated Pure Pursuit 22
2.7 Robot Operating System 23
2.7.1 Transformações espaciais (TF2) 24
2.7.2 URDF e modelagem do robô 25
2.8 Árvores de Comportamento 26
2.8.1 Estrutura Fundamental 26
2.8.2 Nós de Controle 27

3 METODOLOGIA 28
3.1 Configuração do Ambiente 28
3.1.1 Robô Móvel *Turtlebot3 Waffle Pi* 28

3.1.1.1 OpenCR . . . 30
3.1.1.2 LiDAR . . . 30
3.1.1.3 Encoder . . . 30
3.1.2 Configuração inicial do TurtleBot3 Waffle Pi e integração com o ROS . . . 31
3.2 Construção do mapa . . . 32
3.3 Navegação . . . 34
3.3.1 Localização no ambiente . . . 35
3.3.2 Definição do *footprint* do robô . . . 35
3.3.3 Configuração do *costmap* . . . 36
3.3.4 Planejadores globais no Nav2 . . . 37
3.3.5 Controladores no Nav2 . . . 38
3.4 Síntese do procedimento experimental . . . 38
3.4.1 Procedimentos de validação da plataforma . . . 39
3.4.2 Testes de sintonia do *costmap* . . . 40
3.4.3 Testes de ajuste e comparação entre planejadores . . . 40
3.4.4 Testes de ajuste e comparação entre controladores . . . 42
3.4.5 Teste de navegação completa com a ação *NavigateToPose* . . . 43

4 RESULTADOS . . . 44
4.1 Validação da plataforma e do mapeamento . . . 44
4.2 Sintonia do global *costmap* . . . 46
4.3 Avaliação dos planejadores . . . 47
4.4 Avaliação dos controladores . . . 49
4.4.1 Ajuste do DWBLocalPlanner . . . 49
4.4.2 Ajuste do Regulated Pure Pursuit . . . 50
4.4.3 Discussão dos Resultados . . . 52

5 CONCLUSÃO . . . 55

REFERÊNCIAS . . . 57

1 Introdução

O avanço das tecnologias associadas à Indústria 4.0 tem impulsionado a adoção de sistemas robóticos cada vez mais inteligentes e integrados em ambientes de manufatura. Nesse contexto, robôs móveis são amplamente empregados em tarefas de transporte de materiais e na integração entre diferentes estações de produção, contribuindo para o aumento da flexibilidade dos sistemas produtivos (ZHAO; CHIDAMBARESWARAN, 2023).

O *Robot Operating System*[1] (ROS) constitui um *middleware* amplamente utilizado no desenvolvimento de sistemas robóticos, organizando os diferentes componentes da aplicação em nós que se comunicam por meio da troca de mensagens. Essa abordagem favorece a integração entre sensores, atuadores e algoritmos de processamento, permitindo a construção de aplicações robóticas mais complexas e modulares.

Em trabalhos recentes, arquiteturas baseadas em árvores de comportamento têm sido empregadas para coordenar sistemas robóticos de forma hierárquica e modular, facilitando a integração entre diferentes dispositivos e a reutilização de comportamentos. Devido a essas características, as árvores de comportamento vêm sendo amplamente adotadas em aplicações robóticas que exigem flexibilidade e escalabilidade (COLLEDANCHISE; ÖGREN, 2018).

Nesse contexto, Cardoso (2024) apresentou uma célula de manufatura inteligente composta por um manipulador, um robô móvel e um sistema de visão, cuja coordenação foi implementada por meio de um supervisor baseado em árvores de comportamento no ambiente ROS 2. Esse supervisor foi posteriormente revisado por Batista (2025). Apesar dos avanços obtidos nesses trabalhos, o deslocamento do robô móvel ainda depende de teleoperação, o que limita o nível de autonomia da célula robótica.

Diante desse cenário, torna-se relevante investigar a aplicação de um sistema de navegação autônoma capaz de permitir ao robô móvel deslocar-se de forma independente no ambiente. Entre as alternativas disponíveis no ecossistema ROS 2, destaca-se a estrutura *Navigation2* (Nav2), que reúne funcionalidades de localização, planejamento, seguimento de caminho e desvio de obstáculos, constituindo uma base apropriada para esse tipo de aplicação.

[1] Página oficial do ROS: <www.ros.org>

1.1 Justificativa

Apesar da integração entre os dispositivos robóticos proposta por Cardoso (2024) e das melhorias aplicadas por Batista (2025), o deslocamento do robô móvel ainda depende de teleoperação. Nessa configuração, o robô precisa ser posicionado por um operador humano antes da execução das tarefas, o que reduz o nível de autonomia do sistema e limita o potencial de operação integrada da célula robótica.

Nesse sentido, a implementação de um sistema de navegação autônoma representa uma evolução natural da arquitetura já desenvolvida no laboratório. Ao permitir que o robô móvel se desloque de forma independente no ambiente, torna-se possível ampliar o nível de automação da célula e aproximar sua operação de um cenário mais compatível com os princípios da Indústria 4.0.

A justificativa deste trabalho está, portanto, na necessidade de ampliar a autonomia da célula robótica por meio da avaliação experimental do *framework Navigation2* (Nav2) aplicado ao robô móvel *TurtleBot3 Waffle Pi*[2] Para isso, foram analisadas diferentes configurações de planejadores globais, controladores locais, costmaps e parâmetros de navegação, buscando identificar alternativas adequadas ao ambiente laboratorial e que possam contribuir para uma futura integração autônoma à arquitetura já existente.

1.2 Objetivo Geral

Avaliar experimentalmente o desempenho de planejadores globais e controladores locais da pilha Nav2 no robô TurtleBot3 Waffle Pi, em ambiente laboratorial, considerando a influência da configuração dos *costmaps*, da representação geométrica do robô e da sintonia dos principais parâmetros de navegação, de modo a viabilizar sua futura integração autônoma a uma célula de manufatura inteligente.

Para alcançar o objetivo geral, foram estabelecidos os seguintes objetivos específicos:

- configurar a estrutura de navegação Nav2 para operação no TurtleBot3 Waffle Pi;
- ajustar parâmetros dos costmaps, dos planejadores e dos controladores locais;
- analisar e comparar planejadores globais;
- analisar e comparar controladores locai;
- analisar o comportamento completo da navegação;
- identificar as configurações mais adequadas para o ambiente experimental.

[2] Página oficial do TurtleBot3: <https://emanual.robotis.com/docs/en/platform/turtlebot3/overview/>.

1.3 Organização do Documento

Este trabalho está organizado em cinco capítulos. No Capítulo 1, são apresentados a introdução, a justificativa e os objetivos. O Capítulo 2 apresenta a fundamentação teórica, contemplando os principais conceitos relacionados à navegação autônoma, aos métodos de planejamento e controle analisados e ao sistema ROS 2. No Capítulo 3, é descrita a metodologia adotada, incluindo a caracterização do ambiente experimental, a configuração do robô, a construção do mapa, a definição dos parâmetros de navegação e os procedimentos de avaliação utilizados. O Capítulo 4 apresenta e discute os resultados obtidos nos experimentos com planejadores globais, controladores locais e com a arquitetura completa de navegação. Por fim, no Capítulo 5, são apresentadas as conclusões do trabalho, suas limitações e as propostas para trabalhos futuros.

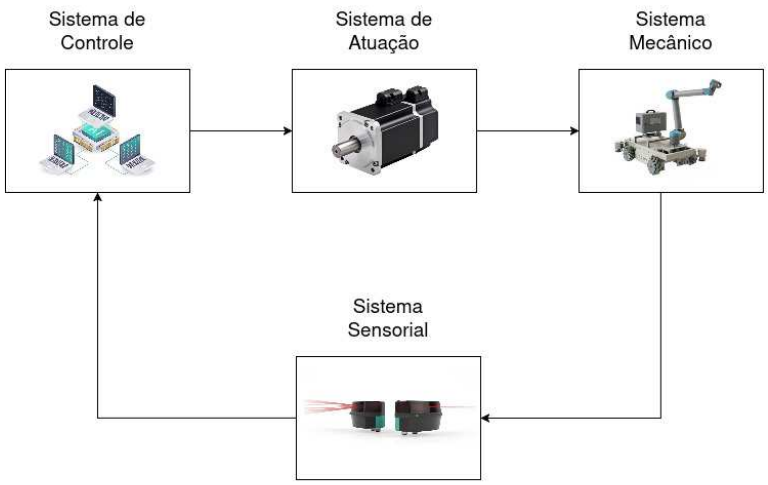
2 Fundamentação Teórica

Este capítulo apresenta os conceitos fundamentais que compõem a base teórica utilizada no desenvolvimento deste trabalho.

2.1 Sistemas Robóticos em Ambientes Industriais

Um sistema robótico é composto por diferentes subsistemas interdependentes, incluindo o sistema mecânico, o sistema de atuação, o sistema sensorial e o sistema de controle, responsáveis respectivamente pela estrutura física, geração de movimento, aquisição de informações e coordenação das ações (SICILIANO et al., 2009). O sistema mecânico define a configuração estrutural e os graus de liberdade disponíveis, enquanto o sistema de atuação transforma sinais de controle em movimento físico. O sistema sensorial fornece informações sobre o robô e sobre o ambiente externo, possibilitando a adaptação do comportamento às condições operacionais. A partir dessas informações, o sistema de controle processa os dados recebidos e determina os comandos necessários para o cumprimento da tarefa. Um sistema robótico é representado na Figura 1.

Figura 1 – Sistema Robótico.



Fonte: Adaptado de (SICILIANO et al., 2009).

O sistema mecânico constitui a base do sistema robótico, pois define sua estrutura física e condiciona a escolha e a configuração dos demais subsistemas. Esse sistema pode incorporar mecanismos de locomoção, como rodas ou esteiras, que possibilitam o desloca-

mento no espaço, ou mecanismos de manipulação, como braços mecânicos e efetuadores finais, destinados à execução de tarefas de interação e manipulação de objetos.

O sistema de atuação é responsável por possibilitar a efetiva movimentação do robô, sendo composto por elementos como servomotores, acionamentos elétricos e mecanismos de transmissão. Esse subsistema atua diretamente sobre os componentes do sistema mecânico, convertendo sinais de controle em movimento físico. Como exemplo, um servomotor pode acionar as rodas de um robô móvel, promovendo seu deslocamento no ambiente, ou movimentar as juntas de um manipulador, permitindo a execução de tarefas de posicionamento e manipulação.

O sistema sensorial é responsável por captar informações sobre o estado interno do robô e sobre o ambiente externo no qual está inserido. Esse subsistema é composto por sensores proprioceptivos, que monitoram variáveis internas, como encoders, sensores de temperatura e sensores inerciais, e sensores exteroceptivos, que captam informações externas, como câmeras e sensores de distância, a exemplo de sensores ultrassônicos e LiDAR, permitindo que o robô perceba o ambiente e interaja adequadamente com ele.

A coordenação entre percepção e ação é realizada pelo sistema de controle, que processa as informações sensoriais e determina os comandos a serem aplicados aos atuadores. Esse sistema opera segundo o princípio da realimentação, no qual os dados coletados são continuamente utilizados para ajustar o comportamento do robô em função dos objetivos da tarefa e das restrições impostas pela dinâmica do sistema e pelo ambiente. A implementação do controle envolve modelagem matemática, técnicas de supervisão e planejamento de movimentos.

2.1.1 Robô Móvel

Os robôs móveis são sistemas robóticos capazes de se deslocar no ambiente, seja sobre a terra, na água ou no ar. Essa característica amplia significativamente seu espaço de atuação, permitindo a execução de tarefas em ambientes extensos e dinâmicos. A mobilidade impõe desafios adicionais relacionados à modelagem, percepção e controle, uma vez que o movimento passa a depender da interação do robô com o meio em que está inserido (SICILIANO et al., 2009).

As primeiras aplicações comerciais de robôs móveis surgiram na década de 1980, com o desenvolvimento dos veículos guiados automaticamente (*Automated Guided Vehicles* – AGVs), utilizados no transporte interno de materiais em ambientes industriais. Desde então, essa tecnologia evoluiu de forma significativa e se consolidou como elemento importante nos sistemas de manufatura automatizada (CORKE, 2011).

Figura 2 – Robôs móveis. `A` terrestre, `B` aquático, `C` aéreo



Fonte: Adaptado de (CORKE, 2011).

2.1.1.1 Cinemática de robôs móveis

A cinemática corresponde ao estudo do movimento dos sistemas mecânicos a partir de grandezas como posição, orientação e velocidade, desconsiderando as forças responsáveis por esse movimento. O estudo da cinemática na robótica móvel é essencial tanto para projetar ou escolher as plataformas adequadas a diferentes tarefas quanto para a formulação de estratégias de controle capazes de conduzir o robô no ambiente. Como o robô móvel se desloca integralmente em relação ao meio, a descrição cinemática é um passo indispensável para compreender as possibilidades de movimento, as restrições impostas pela locomoção e as dificuldades associadas à estimativa de posição ao longo do tempo (SIEGWART; NOURBAKHSH; SCARAMUZZA, 2011).

No caso de robôs terrestres com rodas, como o *Turtlebot3*, a análise cinemática permite estabelecer relações entre a geometria da plataforma, o movimento individual das rodas e a evolução do robô no plano. Com a modelagem é possível representar matematicamente o deslocamento da base móvel, identificar limitações de mobilidade e fornecer suporte teórico para etapas posteriores, como controle de movimento, seguimento de trajetória, localização e navegação autônoma. Entre as configurações de robôs móveis terrestres, o modelo diferencial é amplamente utilizado. Essa arquitetura é composta por duas rodas independentes posicionadas lateralmente em relação ao centro geométrico do robô, além de uma ou mais rodas passivas para suporte estrutural.

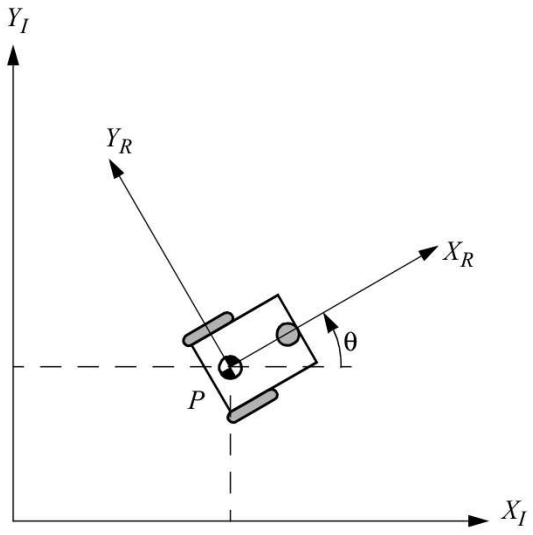
Para elaborar a modelagem cinemática o robô móvel é considerado como um corpo rígido que se desloca sobre um plano horizontal. Nessa condição, sua configuração pode ser descrita por três variáveis independentes: duas associadas à sua posição no plano e uma associada à sua orientação em torno do eixo perpendicular a esse plano. Embora o sistema mecânico real inclua outros elementos móveis, como rodas, eixos e juntas, a descrição cinemática inicial considera o movimento do chassi como um todo.

Para determinar a posição do robô é necessária a definição de dois referenciais: um referencial global, fixo ao ambiente, e um referencial local ao robô. O referencial global é utilizado para expressar a posição da plataforma no ambiente, enquanto o referencial local acompanha seu movimento e sua orientação.

O ponto P no chassi é definido como ponto de referência, de modo que sua posição ($\mathtt{q}$) no plano seja dada pelas coordenadas (x,y) no sistema global, e a orientação da base seja representada pelo ângulo θ, definido entre o eixo longitudinal do robô e o eixo X do referencial global. Conforme Equação 2.1.

$$\mathbf{q} = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} \tag{2.1}$$

Figura 3 – Sistemas de coordenadas do robô e do ambiente



Fonte: (SIEGWART; NOURBAKHSH; SCARAMUZZA, 2011).

Como o robô é descrito por dois referenciais distintos, torna-se necessário estabelecer uma transformação matemática entre eles. Essa relação é expressa por meio de uma matriz de rotação, responsável por converter componentes de movimento de um sistema de coordenadas para o outro.

$$R(\theta) = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.2}$$

Essa transformação permite decompor o movimento do robô em componentes alinhadas aos seus próprios eixos, o que é particularmente útil na formulação do modelo cinemático e na análise das restrições impostas pela locomoção.

A partir dessa relação entre referenciais, pode-se formular o modelo cinemático do robô, estabelecendo a ligação entre sua orientação, sua velocidade linear e sua velocidade angular.

A Equação 2.3 define o vetor de velocidades do robô, sendo v_x velocidade linear ao longo do eixo x, v_y velocidade linear ao longo do eixo y e ω velocidade angular

$$\boldsymbol{\xi}_R = \begin{bmatrix} v_x \\ v_y \\ \omega \end{bmatrix}, \tag{2.3}$$

A partir dessa equação e da inversa matriz de rotação é possível obter as velocidades do ponto de vista global. Conforme Equação 2.4

$$\dot{\mathbf{q}} = R(\theta)^{-1}\,\boldsymbol{\xi}_R \tag{2.4}$$

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} v_x \\ v_y \\ \omega \end{bmatrix} \tag{2.5}$$

Resultando em

$$\dot{x} = v_x\cos\theta - v_y\sin\theta \tag{2.6}$$

$$\dot{y} = v_x\sin\theta + v_y\cos\theta \tag{2.7}$$

$$\dot{\theta} = \omega \tag{2.8}$$

A modelagem da cinemática direta relaciona as velocidades angulares das rodas com o movimento da base móvel. Para o caso de robôs diferenciais e uniciclos que possuem restrição de movimento lateral $v_y = 0$ e $v_x = v$. Considerando ω_r e ω_l como as velocidades

angulares das rodas direita e esquerda, respectivamente, r como o raio das rodas e L como a distância entre elas, obtêm-se:

$$v = \frac{r}{2}(\omega_r + \omega_l) \tag{2.9}$$

$$\omega = \frac{r}{L}(\omega_r - \omega_l). \tag{2.10}$$

Com isso, a evolução da posição e orientação do robô no plano é dada por

$$\dot{x} = v\cos\theta, \qquad \dot{y} = v\sin\theta, \qquad \dot{\theta} = \omega. \tag{2.11}$$

2.2 Percepção

A percepção pode ser entendida como o processo de obtenção de informações sobre um sistema ou sobre o meio em que ele está inserido a partir da observação de grandezas físicas. No contexto de sistemas autônomos, essa etapa é fundamental porque permite transformar sinais medidos em informações úteis para interpretação do ambiente e apoio à tomada de decisão.

A percepção assume papel central na robótica móvel, pois é por meio dela que o robô obtém dados sobre o espaço em que está incluído e sobre o seu próprio movimento. Essas informações são necessárias para que a plataforma possa executar tarefas como localização, mapeamento, detecção de obstáculos e navegação de forma consistente. Conforme discutido por Siegwart, Nourbakhsh e Scaramuzza (2011), a percepção constitui uma das bases da autonomia em robôs móveis, uma vez que conecta a aquisição de dados sensoriais à interpretação do ambiente e à ação do sistema.

Para que esse processo seja possível, os robôs utilizam sensores responsáveis por medir diferentes grandezas de interesse. Esses sensores podem fornecer tanto informações sobre o estado interno, como deslocamento e velocidade, quanto informações sobre o ambiente externo, como distâncias e presença de obstáculos. Assim, o estudo dos sensores empregados no sistema é importante para compreender quais informações estão disponíveis para a navegação e quais limitações podem influenciar o desempenho do robô.

2.2.1 Sensores

Os sensores são dispositivos capazes de medir grandezas físicas e convertê-las em sinais utilizáveis pelo sistema. Em robótica móvel, eles constituem a principal fonte de informação sobre o estado do robô e sobre o ambiente em que ele opera. De modo geral, os sensores podem ser classificados de acordo com a origem da informação fornecida ou com seu modo de operação.

Quanto à origem da informação, os sensores podem ser classificados em proprioceptivos e exteroceptivos. Sensores proprioceptivos medem o estado interno do robô, enquanto sensores exteroceptivos permitem que a plataforma obtenha informações diretamente do ambiente ao seu redor.

Quanto ao modo de operação, os sensores podem ser classificados em ativos e passivos. Sensores ativos são aqueles que realizam medições a partir da emissão de energia em direção ao ambiente, analisando posteriormente a resposta recebida. Já os sensores passivos obtêm informações sem emitir energia própria para esse fim, limitando-se à captação de sinais naturalmente disponíveis no meio.

Essa divisão facilita a compreensão do papel desempenhado por cada sensor no sistema de navegação e permite situar, de forma mais clara, os dispositivos empregados na plataforma analisada.

2.2.1.1 Encoders

Os encoders são sensores proprioceptivos utilizados para medir o movimento rotacional de eixos ou rodas, convertendo esse movimento em sinais elétricos que podem ser processados pelo sistema. Entre os principais tipos, destacam-se os encoders incrementais e os absolutos.

Os encoders incrementais são sensores que fornecem a posição angular do eixo de forma relativa, a partir da geração de pulsos elétricos durante a rotação. Nesse tipo de sensor, a informação de posição não é obtida diretamente como um valor absoluto, mas pela contagem acumulada dos pulsos gerados ao longo do movimento. Encoders incrementais são amplamente utilizados em robôs móveis, especialmente em sistemas de odometria, nos quais a rotação das rodas é medida para estimar o deslocamento da plataforma (SZREK et al., 2020).

No caso do encoder incremental óptico, sua arquitetura é composta por um disco com ranhuras, uma fonte de luz e sensores fotoelétricos posicionados de forma defasada. O disco é fixado ao eixo ou à roda e, durante sua rotação, a luz emitida pela fonte, posicionada de um lado do disco, é alternadamente transmitida através das ranhuras e captada pelos sensores fotoelétricos localizados no lado oposto, ou bloqueada pela superfície do disco. Dessa configuração resultam dois sinais em quadratura, denominados canais A e B, defasados entre si de 90°, o que permite identificar tanto a contagem de pulsos quanto o sentido de rotação. A contagem desses pulsos permite estimar o deslocamento angular da roda, enquanto a frequência dos sinais pode ser associada à sua velocidade de rotação.

Os encoders absolutos são sensores capazes de fornecer diretamente a posição angular do eixo em forma digital, como um valor absoluto associado a um código binário. Diferentemente dos encoders incrementais, nesse tipo de sensor a posição não é obtida

pela contagem acumulada de pulsos, mas pela leitura direta do código correspondente à posição angular do eixo. Quando aplicados em um sistema, esses sensores permitem definir uma posição de referência como origem, passando a informar continuamente o ângulo de rotação em relação a esse ponto.

$$\phi = 2\pi \frac{k}{N} \tag{2.12}$$

em que ϕ representa a posição angular do eixo, k corresponde ao valor discreto lido pelo encoder e N é o número total de posições discretas em uma volta completa.

Entre os encoders absolutos, destacam-se os magnéticos sem contato, cujo funcionamento se baseia na leitura da variação do campo magnético gerada pela rotação de um ímã de dois polos acoplado ao eixo. À medida que esse ímã gira sobre o centro do sensor, a orientação do campo magnético é identificada pelo dispositivo e processada eletronicamente para determinar a posição angular correspondente. Como cada posição do eixo está associada a um valor digital específico, a leitura é realizada de forma absoluta dentro de uma volta completa. Esse princípio permite obter a posição angular de forma imediata, sem necessidade de contagem acumulativa e com boa robustez para aplicações embarcadas.

Na plataforma utilizada neste trabalho, emprega-se encoder absoluto magnético sem contato, o que torna esse princípio de funcionamento particularmente relevante para a compreensão da medição angular associada ao movimento das rodas.

2.2.1.2 IMU

A Unidade de Medição Inercial (*Inertial Measurement Unit* (IMU) é conjunto de sensores proprioceptivos, utilizado para medir grandezas associadas ao movimento da plataforma, como aceleração linear e velocidade angular. Em geral, uma IMU é composta por acelerômetros e giroscópios, em algumas configurações, podem incluir também magnetômetros.

Em implementações baseadas em tecnologia MEMS (*Micro-Electro-Mechanical Systems*), os acelerômetros operam a partir do deslocamento de uma microestrutura interna submetida à aceleração, o que provoca a variação de grandezas elétricas mensuráveis, como a capacitância. Já os giroscópios MEMS medem a velocidade angular a partir do comportamento dinâmico de microestruturas vibratórias, explorando os efeitos inerciais associados à rotação. Dessa forma, a IMU é capaz de fornecer informações instantâneas sobre aceleração e rotação da plataforma em diferentes eixos.

Em sistemas robóticos móveis, a IMU é amplamente utilizada de forma complementar a outros sensores, como encoders. Enquanto os encoders permitem estimar o movimento a partir da rotação das rodas, a IMU contribui para o acompanhamento da

orientação. Essa combinação de diferentes fontes de informação pode tornar a estimação de movimento mais robusta.

2.2.1.3 LiDAR

O LiDAR (*Light Detection and Ranging*) é um sensor exteroceptivo ativo utilizado para medir distâncias até objetos presentes no ambiente. Seu funcionamento está baseado na emissão de feixes de luz e na análise do sinal refletido pelas superfícies ao redor, permitindo estimar a distância entre o sensor e os obstáculos detectados.

Sensores LiDAR medem distâncias com base no princípio do tempo de voo (*time of flight*), em que o intervalo entre a emissão do feixe e a recepção do sinal refletido é utilizado para estimar a distância até o objeto (SIEGWART; NOURBAKHSH; SCARAMUZZA, 2011). Considerando o percurso de ida e volta da luz entre o sensor e a superfície refletora, a distância d pode ser obtida por

$$d = \frac{c\,\Delta t}{2} \tag{2.13}$$

em que c é a velocidade da luz e Δt é o tempo decorrido entre a emissão e a recepção do sinal refletido (BRAND, 2021).

Em aplicações de robótica móvel, os sensores LiDAR 2D são os mais utilizados, o sensor realiza medições ao longo de um plano, produzindo uma varredura angular do ambiente ao redor da plataforma. A partir dessa varredura, o sensor fornece um conjunto de distâncias associadas a diferentes direções, permitindo identificar obstáculos, paredes e regiões livres no ambiente. Por essa razão, o LiDAR é amplamente utilizado em aplicações de mapeamento e localização.

2.3 Localização

A localização em robôs móveis corresponde ao processo de estimar, ao longo do tempo, a posição e a orientação da plataforma no ambiente. Essa etapa é indispensável para a navegação autônoma, pois permite ao robô interpretar sua relação com o espaço ao redor e executar deslocamentos de forma coerente com o ambiente em que está inserido.

Em aplicações reais, essa estimativa não é obtida de forma direta, mas inferida a partir de informações sobre o movimento do próprio robô e das observações do ambiente. Por essa razão, a localização está intimamente relacionada tanto aos sensores proprioceptivos, como encoders e IMU, quanto aos sensores exteroceptivos, como o LiDAR.

2.3.1 Odometria e incertezas na estimação de pose

Uma das formas mais usuais de estimar a pose do robô é por meio da odometria, que utiliza sensores proprioceptivos para acompanhar a evolução do movimento ao longo

do tempo. Em robôs com tração diferencial, essa estimativa é normalmente obtida a partir da rotação das rodas, medida por encoders. A partir dessas informações, torna-se possível estimar continuamente a variação de posição e orientação da plataforma (ULLAH et al., 2024).

O deslocamento linear de cada roda pode ser calculado a partir da variação angular apresentada na Equação 2.12. Dessa forma, os deslocamentos das rodas direita e esquerda são dados por

$$\Delta s_r = r\,\Delta\phi_r \tag{2.14}$$

$$\Delta s_l = r\,\Delta\phi_l \tag{2.15}$$

em que r corresponde ao raio da roda, enquanto Δs_r e Δs_l representam, respectivamente, os deslocamentos lineares das rodas direita e esquerda.

Conhecidos os deslocamentos das duas rodas, o deslocamento linear médio da plataforma pode ser estimado por

$$\Delta s = \frac{\Delta s_r + \Delta s_l}{2} \tag{2.16}$$

enquanto a variação de orientação do robô pode ser calculada por

$$\Delta\theta = \frac{\Delta s_r - \Delta s_l}{L} \tag{2.17}$$

em que L representa a distância entre as rodas.

Quando a IMU é utilizada em conjunto com os encoders, a estimativa odométrica passa a incorporar também as medições inerciais da plataforma. Nesse caso, o deslocamento linear continua sendo obtido a partir da rotação das rodas, enquanto a variação angular pode ser estimada ou corrigida com base na velocidade angular medida pelo giroscópio. Essa combinação tende a tornar a estimação de pose mais robusta, especialmente em situações nas quais a odometria por rodas é afetada por escorregamento ou por pequenas irregularidades no contato com o solo.

Com isso, a pose odométrica do robô pode ser atualizada iterativamente no plano por meio das expressões

$$x_{k+1} = x_k + \Delta s \cos\left(\theta_k + \frac{\Delta\theta}{2}\right) \tag{2.18}$$

$$y_{k+1} = y_k + \Delta s \sin\left(\theta_k + \frac{\Delta\theta}{2}\right) \tag{2.19}$$

$$\theta_{k+1} = \theta_k + \Delta\theta \tag{2.20}$$

em que (x_k, y_k, θ_k) representa a pose do robô no instante k, e $(x_{k+1}, y_{k+1}, \theta_{k+1})$ corresponde à pose estimada no instante seguinte.

Apesar de sua utilidade, a odometria está sujeita a diferentes fontes de erro, como escorregamento das rodas, irregularidades do piso e ruídos de medição. Como esses erros

tendem a se acumular ao longo do deslocamento, a pose estimada por integração odométrica pode apresentar deriva significativa com o passar do tempo (THRUN; BURGARD; FOX, 2005; ULLAH et al., 2024). Dessa forma, a odometria, embora essencial para o acompanhamento do movimento da plataforma, não é suficiente, isoladamente, para garantir uma estimativa confiável da pose em aplicações de navegação mais prolongadas.

2.3.2 Localização probabilística

A literatura apresenta diferentes abordagens para o problema de localização de robôs móveis. Revisões recentes, como a apresentada por Ullah et al. (2024), mostram que essas abordagens podem ser agrupadas, de forma geral, em métodos probabilísticos, métodos baseados no sistema de sensoriamento e na infraestrutura disponível, e métodos baseados em técnicas de inteligência artificial.

A abordagem probabilística fornece a base teórica para métodos de localização em mapa conhecido amplamente empregados em sistemas de navegação móvel (THRUN; BURGARD; FOX, 2005; ULLAH et al., 2024). Nessa perspectiva, a pose do robô não é tratada como um valor exato, mas como uma distribuição de crença sobre os estados possíveis. A localização passa, então, a ser formulada como um problema de estimação de estado sob incerteza, atualizado continuamente a partir de dois elementos principais: o modelo de movimento, que prevê a evolução da pose com base nos comandos aplicados, e o modelo de observação, que corrige essa previsão a partir das medições fornecidas pelos sensores (THRUN; BURGARD; FOX, 2005).

De forma geral, essa atualização pode ser expressa por

$$bel(x_t) = \eta\, p(z_t \mid x_t) \int p(x_t \mid u_t, x_{t-1})\, bel(x_{t-1})\, dx_{t-1} \tag{2.21}$$

em que $bel(x_t)$ representa a crença sobre a pose do robô no instante t, $p(x_t \mid u_t, x_{t-1})$ corresponde ao modelo de movimento, $p(z_t \mid x_t)$ representa o modelo de observação e η é uma constante de normalização. Essa formulação é conhecida como *Markov localization* e constitui a base para diferentes métodos de localização probabilística.

Entre os métodos baseados nessa formulação, destaca-se o *Monte Carlo Localization* (MCL), no qual a distribuição de crença é representada por um conjunto de partículas ponderadas. Nesse método, as partículas são propagadas de acordo com o modelo de movimento e reavaliadas com base na compatibilidade entre as observações do robô e o mapa do ambiente. Essa abordagem é adequada para situações em que múltiplas hipóteses de pose precisam ser consideradas simultaneamente, como em problemas de localização global (THRUN; BURGARD; FOX, 2005).

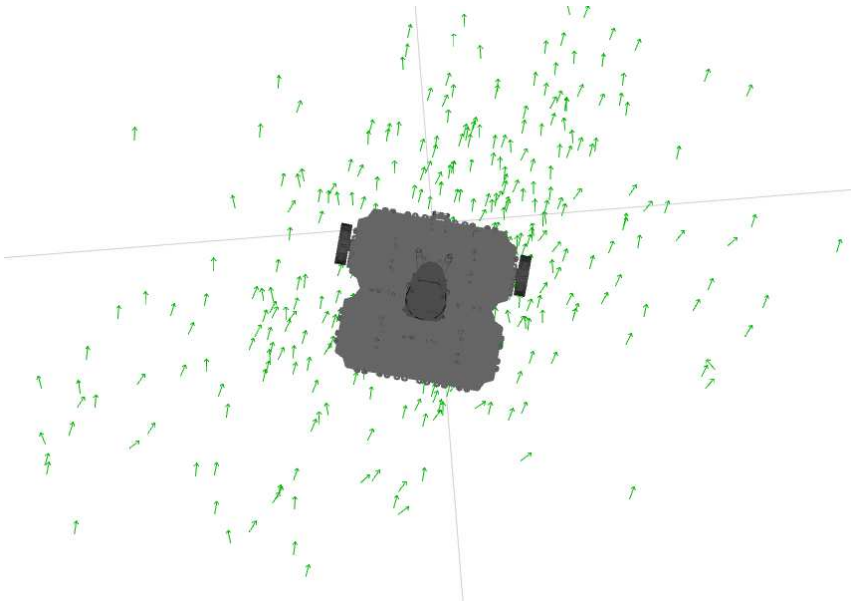
Quando o sistema dispõe de um mapa previamente construído do ambiente, a localização pode ser realizada pela comparação entre as medições atuais dos sensores e a

representação conhecida do espaço. Nesse caso, a estimativa de pose é corrigida continuamente a partir do grau de compatibilidade entre as observações do robô e o mapa disponível.

Entre as abordagens utilizadas nesse contexto, destacam-se os métodos probabilísticos baseados em partículas, dos quais deriva o *Adaptive Monte Carlo Localization* (AMCL). Por ser apropriado à localização em mapa conhecido e amplamente empregado em arquiteturas modernas de navegação, esse método se torna especialmente relevante para a compreensão do sistema utilizado neste trabalho.

A Figura 4 ilustra o processo, no qual as partículas representam diferentes hipóteses de pose distribuídas no ambiente e refinadas sucessivamente durante a localização.

Figura 4 – Representação das partículas no processo de localização probabilística do AMCL.



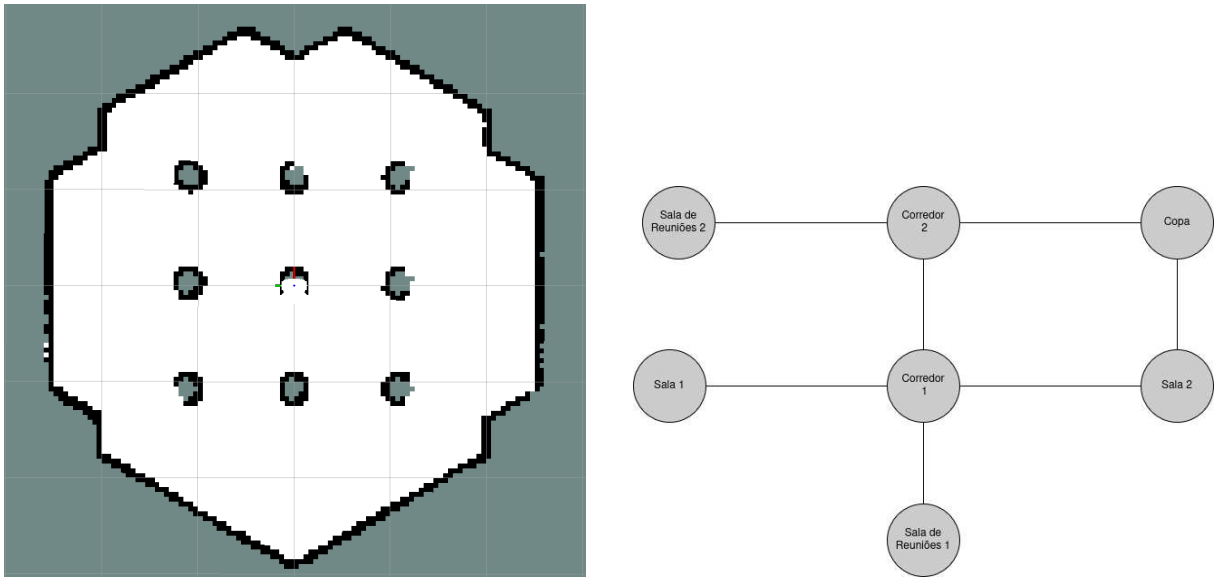
Fonte: Autoria própria.

2.4 Mapeamento

O mapeamento em robôs móveis corresponde ao processo de construção de uma representação do ambiente a partir das informações obtidas pelos sensores. Essa representação permite ao robô identificar regiões livres, ocupadas e desconhecidas, o que é essencial para tarefas de navegação autônoma. O mapa funciona como uma referência que possibilita ao robô interpretar o ambiente e planejar seus deslocamentos de forma coerente (SIEGWART; NOURBAKHSH; SCARAMUZZA, 2011; THRUN; BURGARD; FOX, 2005).

A forma como esse ambiente é representado pode variar de acordo com a aplicação. Em robótica móvel, é comum distinguir mapas em métricos, topológicos e híbridos. A Figura 5 apresenta um mapa métrico e um topológico. Os mapas métricos preservam informações geométricas do ambiente, como posição e distância entre elementos, muito utilizado em tarefas de navegação em ambientes internos. Já os mapas topológicos descrevem o ambiente por meio de conexões entre lugares de interesse, priorizando a estrutura relacional do espaço em vez de sua geometria detalhada. Os mapas híbridos procuram combinar essas duas abordagens, reunindo a precisão geométrica dos mapas métricos com a abstração proporcionada pelas representações topológicas (SIEGWART; NOURBAKHSH; SCARAMUZZA, 2011).

Figura 5 – Mapa métrico e mapa topológico.



((a)) Mapa métrico. ((b)) Mapa topológico.

Fonte: (MACENSKI et al., 2020) e Autoria própria.

2.4.1 Grades de ocupação

Entre as representações métricas mais utilizadas em robótica móvel, destacam-se as grades de ocupação. Nessa abordagem, o ambiente é discretizado em células, e cada célula passa a representar uma pequena região do espaço. Em vez de descrever diretamente objetos ou estruturas geométricas completas, essa forma de representação procura indicar o estado de ocupação de cada célula, permitindo identificar se a região correspondente está livre, ocupada ou ainda desconhecida (ELFES, 1989).

Outra característica importante dessa abordagem é sua capacidade de integrar sucessivas observações realizadas em diferentes posições do robô. À medida que a plataforma se desloca e coleta novas medições, o mapa pode ser refinado e expandido, incorporando informações adicionais sobre o espaço ao redor.

Além disso, por discretizar o espaço em regiões menores e tratá-las individualmente, essa representação facilita a identificação de áreas navegáveis e de obstáculos. Por essa razão, grades de ocupação são amplamente empregadas em robótica móvel, especialmente em sistemas que utilizam sensores como LiDAR para mapear o ambiente e apoiar etapas posteriores de localização e planejamento de movimento.

2.4.2 Simultaneous Localization and Mapping

Quando o robô não tem a disposição um mapa prévio do ambiente, o problema de mapeamento passa a estar diretamente ligado ao problema de localização. Essa condição exige que o robô construa o mapa ao mesmo tempo em que se localiza. O *Simultaneous Localization and Mapping* (SLAM) pode ser entendido como o processo em que o robô, inicialmente em posição desconhecida e inserido em um ambiente desconhecido, constrói gradualmente um mapa consistente enquanto determina sua localização nesse mesmo ambiente (DURRANT-WHYTE; BAILEY, 2006).

Na formulação apresentada por Durrant-Whyte e Bailey (2006), o SLAM é tratado como um problema de estimação conjunta, no qual a pose do robô e os elementos do mapa são atualizados a partir do histórico de controles e das observações realizadas durante o movimento. Isso significa que o sistema precisa usar as informações fornecidas pelos sensores para refinar simultaneamente a trajetória estimada da plataforma e a representação do ambiente.

Outro ponto importante destacado pelos autores é que as observações acumuladas ao longo do deslocamento criam relações entre os diferentes elementos do mapa, já que todos eles dependem, em alguma medida, do erro associado à pose do robô. Por essa razão, a consistência do mapa construído está diretamente ligada à qualidade da estimação da trajetória, o que torna o problema mais complexo e exige abordagens capazes de tratar essas dependências de forma adequada.

Na prática, o SLAM permite que o robô opere em ambientes inicialmente desconhecidos, construindo progressivamente uma representação do espaço à medida que se desloca. A cada nova observação, o mapa pode ser refinado e a estimativa de pose pode ser corrigida, tornando possível o desenvolvimento gradual de uma representação espacial útil para etapas posteriores de localização e navegação.

No contexto deste trabalho, essa discussão é importante porque a construção prévia do mapa constitui uma etapa fundamental para as fases posteriores de localização e navegação autônoma.

2.5 Planejamento de Caminho

O planejamento de caminho consiste em determinar uma sequência de posições que permita ao robô alcançar um destino evitando colisões com os obstáculos presentes no ambiente. Essa etapa constitui um componente essencial dos sistemas de navegação autônoma, pois define como o robô utilizará as informações contidas no mapa para realizar seu deslocamento de maneira segura e eficiente (CHEN et al., 2024).

Neste contexto, é importante distinguir caminho de trajetória. O caminho corresponde à sequência de posições a serem percorridas no ambiente, enquanto a trajetória incorpora também a evolução temporal do movimento, incluindo aspectos como velocidade e aceleração. Em aplicações de navegação móvel baseadas em mapas, o foco é inicialmente no planejamento do caminho, que posteriormente será executado e ajustado pelos módulos responsáveis pela condução do robô.

De modo geral, os métodos de planejamento podem ser classificados em abordagens baseadas em grafos, abordagens baseadas em amostragem e estratégias híbridas que combinam características dessas duas classes. Os métodos baseados em grafos são especialmente adequados quando o ambiente pode ser representado por uma discretização do espaço, enquanto os métodos baseados em amostragem são mais frequentemente empregados em problemas formulados em espaços contínuos ou em sistemas com maior número de graus de liberdade (SÁNCHEZ-IBÁÑEZ; PULGAR; GARCÍA-CEREZO, 2021).

Em aplicações de navegação móvel baseadas em mapas métricos discretizados, como mapas de grade de ocupação, os métodos baseados em grafos tendem a ser mais adequados devido à compatibilidade com a estrutura discreta do mapa. Nesses casos, cada célula ou conjunto de células pode ser associado a uma estrutura de busca, permitindo o cálculo de caminhos livres de colisão entre a pose atual do robô e a meta desejada (LIU et al., 2023).

2.5.1 Algoritmos baseados em grafos

Entre os algoritmos baseados em grafos, destacam-se Dijkstra e A*, amplamente utilizados em navegação robótica. O algoritmo de Dijkstra realiza a busca do caminho de menor custo entre um nó inicial e os demais nós do grafo sem utilizar informação heurística sobre a direção do objetivo. Por essa razão, apresenta caráter mais geral, embora possa demandar maior esforço computacional em espaços de busca extensos. Representação de um diagrama de grafo na Figura 6.

Já o algoritmo A* introduz uma heurística para orientar a busca em direção ao objetivo, combinando o custo acumulado desde a origem com uma estimativa do custo restante até a meta. Essa característica tende a tornar a busca mais eficiente, mantendo a capacidade de encontrar caminhos ótimos quando a heurística adotada é adequada

(HART; NILSSON; RAPHAEL, 1968). Em problemas de navegação móvel, essa propriedade faz do A* uma das abordagens mais difundidas para o cálculo de caminhos sobre mapas discretizados.

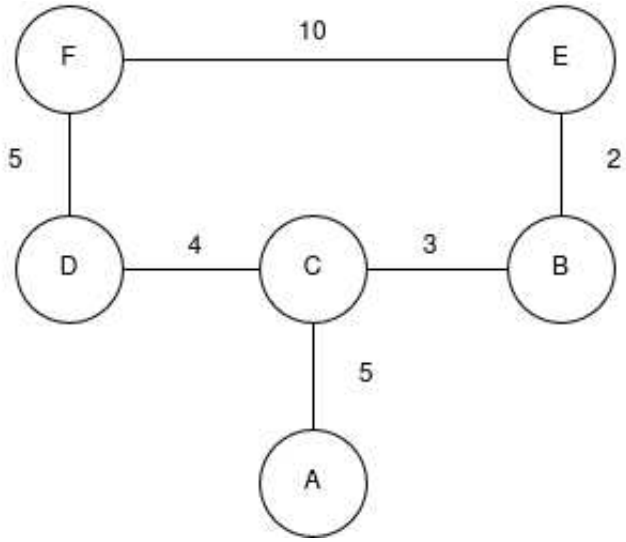
A função de avaliação utilizada pelo A* pode ser expressa por

$$f(n) = g(n) + h(n) \tag{2.22}$$

em que $g(n)$ representa o custo acumulado desde o nó inicial até o nó atual, $h(n)$ corresponde à estimativa do custo restante até o objetivo, e $f(n)$ é o custo total estimado utilizado para orientar a busca.

Essa estratégia permite direcionar a exploração do espaço de busca de forma mais eficiente do que métodos sem heurística, favorecendo o cálculo de caminhos viáveis entre a posição inicial e a meta. Por essa razão, algoritmos dessa classe ocupam papel importante em sistemas modernos de navegação móvel.

Figura 6 – Diagrama de grafo



Fonte: Autoria própria.

2.6 Controle de Movimento

Após o planejamento do caminho, o robô móvel precisa executar esse deslocamento de forma segura e estável, respeitando suas restrições cinemáticas e evitando colisões com obstáculos presentes no ambiente. O controle de movimento corresponde à etapa responsável por transformar o caminho planejado em comandos de atuação aplicados à plataforma robótica, normalmente expressos em termos de velocidade linear e velocidade angular (RAJ et al., 2024).

De modo geral, os métodos de controle recebem como entrada o caminho planejado e a estimativa atual da pose do robô. A partir dessas informações, o controlador calcula

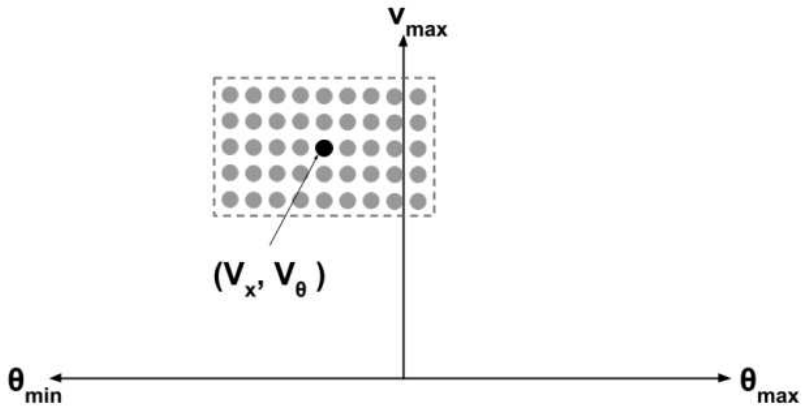
comandos que buscam reduzir o desvio em relação ao caminho desejado, mantendo a estabilidade do movimento e adaptando a navegação às condições do ambiente.

2.6.1 Dynamic Window Approach (DWA)

Entre as abordagens clássicas de controle local para robôs móveis, destaca-se o *Dynamic Window Approach* (DWA). Nesse método, a seleção do comando de controle é realizada diretamente no espaço de velocidades, por meio da amostragem de velocidades linear e angular dinamicamente viáveis. Essa estratégia reduz a complexidade do problema de navegação local e favorece respostas em tempo real para desvio de obstáculos (FOX; BURGARD; THRUN, 1997).

O método assume que, em pequenos intervalos de tempo, o movimento do robô pode ser aproximado por arcos circulares. A busca pelo comando de controle é então restringida em três etapas: às trajetórias circulares definidas por pares (v,ω), às velocidades admissíveis que permitem ao robô frear antes de colidir e à chamada *janela dinâmica*, que limita a busca às velocidades alcançáveis no próximo intervalo de tempo, respeitando as restrições de aceleração (FOX; BURGARD; THRUN, 1997).

Figura 7 – Exemplo de janela dinâmica.



Fonte: (MACENSKI et al., 2023a).

Em sua formulação clássica, a função objetivo do DWA pode ser expressa por

$$G(v,\omega) = \sigma\left(\alpha \cdot heading(v,\omega) + \beta \cdot dist(v,\omega) + \gamma \cdot vel(v,\omega)\right) \tag{2.23}$$

em que $heading(v,\omega)$ representa o progresso em direção ao objetivo, $dist(v,\omega)$ representa a distância ao obstáculo mais próximo ao longo da trajetória prevista, $vel(v,\omega)$ representa a preferência por velocidades mais elevadas, α, β e γ são pesos de ponderação, e $\sigma(\cdot)$ corresponde a uma função de suavização aplicada à combinação desses termos (FOX; BURGARD; THRUN, 1997). Assim, o DWA busca um compromisso entre avanço em direção à meta, segurança em relação aos obstáculos e eficiência de deslocamento.

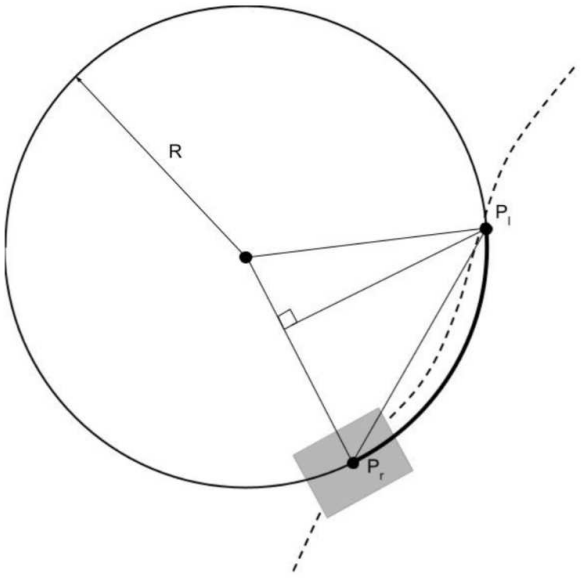
Por considerar simultaneamente restrições dinâmicas, segurança e progresso em direção ao objetivo, o DWA tornou-se uma referência importante entre os métodos de controle local em robótica móvel. Sua formulação influenciou diversas abordagens posteriores voltadas à navegação autônoma em ambientes internos e dinâmicos.

2.6.2 Pure Pursuit

O método *Pure Pursuit*, descrito de forma consolidada por Coulter, é um algoritmo geométrico de seguimento de caminho no qual o robô seleciona um ponto de referência à frente sobre a trajetória, denominado *lookahead point*, e calcula a curvatura necessária para alcançá-lo por meio de um arco circular (COULTER, 1992). Sua principal vantagem está na simplicidade geométrica e no baixo custo computacional, o que favorece aplicações em tempo real.

A Figura 27 ilustra a representação geométrica do método *Pure Pursuit*, no qual o robô segue o caminho a partir da seleção de um ponto de antecipação à frente de sua posição atual. Inicialmente, determina-se o ponto do caminho p_r mais próximo da posição atual do robô. Em seguida, define-se o ponto de antecipação p_l como o primeiro ponto da trajetória cuja distância em relação a p_r seja igual ou superior à distância de *lookahead* L. Matematicamente, essa distância é calculada por $dist(p_i) = \sqrt{(x_r - x_i)^2 + (y_r - y_i)^2}$, e o ponto p_l é escolhido de forma que o ponto anterior ainda esteja a uma distância menor que L, enquanto o ponto atual já satisfaça $dist(p_i) \geq L$.

Figura 8 – Representação geométrica do método *Pure Pursuit*.



Fonte: (MACENSKI et al., 2023b).

No *Pure Pursuit*, a ideia central é fazer com que o robô siga continuamente um ponto alvo localizado a uma distância de antecipação L_d ao longo do caminho. Conside-

rando a geometria do método, a curvatura de comando pode ser expressa por

$$\kappa = \frac{2y_L}{L_d^2} \tag{2.24}$$

em que y_L representa a coordenada lateral do ponto de antecipação no referencial do robô e L_d é a distância entre o robô e esse ponto (COULTER, 1992). A partir da curvatura calculada, obtém-se o comando angular pela relação

$$\omega = v\kappa \tag{2.25}$$

em que v é a velocidade linear do robô e ω sua velocidade angular. Dessa forma, o algoritmo converte um problema geométrico de seguimento de caminho em comandos cinemáticos simples, o que contribui para sua ampla adoção em robótica móvel (COULTER, 1992).

Apesar de sua simplicidade e eficiência, o *Pure Pursuit* clássico apresenta limitações práticas. Seu desempenho depende fortemente da escolha da distância de antecipação. Valores muito pequenos tendem a produzir correções agressivas e oscilações, enquanto valores muito grandes podem reduzir a precisão do seguimento, especialmente em curvas mais acentuadas (COULTER, 1992).

Mesmo com essas limitações, o *Pure Pursuit* continua sendo uma referência importante entre os métodos de seguimento de caminho, tanto pela simplicidade de implementação quanto pela clareza de sua formulação geométrica. Além disso, variantes mais recentes desse método incorporam mecanismos adicionais de regulação, ampliando sua adequação a aplicações de navegação autônoma em ambientes reais.

2.6.2.1 Regulated Pure Pursuit

O *Regulated Pure Pursuit* foi proposto justamente para superar essas limitações em aplicações reais de navegação. O RPP constitui uma evolução do *Pure Pursuit* e do *Adaptive Pure Pursuit*, introduzindo mecanismos de regulação da velocidade linear com foco em segurança, estabilidade e robustez em ambientes estreitos ou parcialmente observáveis (MACENSKI et al., 2023b). Enquanto o *Adaptive Pure Pursuit* já ajusta a distância de antecipação em função da velocidade, o RPP amplia essa lógica ao incorporar heurísticas adicionais para controlar a velocidade de forma mais conservadora quando necessário.

A primeira é a *heurística de curvatura*, usada para reduzir a velocidade em curvas acentuadas:

$$v_t' = \begin{cases} v_t, & \kappa > T_\kappa \\ \dfrac{v_t}{r_{\min}\kappa}, & \kappa \leq T_\kappa \end{cases} \tag{2.26}$$

em que $r_{\min}$ é o raio mínimo para aplicação da heurística e T_κ é o limiar associado à curvatura.

A segunda é a *heurística de proximidade*, empregada para reduzir a velocidade quando o robô se aproxima de obstáculos:

$$v_t' = \begin{cases} v_t \dfrac{\alpha d_O}{d_{prox}}, & d_O \leq d_{prox} \\ v_t, & d_O > d_{prox} \end{cases} \tag{2.27}$$

em que d_{prox} é a distância de proximidade para ativação da heurística, d_O é a distância atual ao obstáculo e α é um ganho de ajuste, com $\alpha \leq 1.0$.

Após a regulação da velocidade linear, a velocidade angular é calculada com base na curvatura e na velocidade já regulada:

$$\omega_t = v_t' \kappa. \tag{2.28}$$

No RPP, o seguimento lateral do caminho continua sendo baseado na geometria do *Pure Pursuit*, mas a velocidade linear passa a ser regulada por critérios adicionais. De forma geral, o controlador reduz a velocidade em situações como: presença de curvatura no caminho e proximidade de obstáculos. Assim, o RPP busca preservar a suavidade e a simplicidade do *Pure Pursuit*, mas adiciona mecanismos que tornam o comportamento mais seguro e mais adequado a aplicações reais de navegação autônoma (MACENSKI et al., 2023b).

2.7 Robot Operating System

O Robot Operating System (ROS) é um *middleware* desenvolvido em 2007 e amplamente utilizado no desenvolvimento de sistemas robóticos, oferecendo um conjunto de bibliotecas e ferramentas. Sua arquitetura baseia-se na organização de funcionalidades em nós, que se comunicam por meio de mecanismos de troca de mensagens, como tópicos, serviços e ações. Essa estrutura modular permite dividir sistemas robóticos complexos em componentes independentes, facilitando o desenvolvimento, a reutilização de software e a integração entre diferentes sensores, atuadores e algoritmos (QUIGLEY et al., 2009).

Os nós representam as unidades básicas de execução em um sistema ROS. Cada nó corresponde a um processo responsável por realizar uma determinada função dentro da aplicação robótica, como adquirir dados de sensores, processar informações ou enviar comandos para atuadores. Na Figura 9 temos dois nós e a representação de comunicação entre eles.

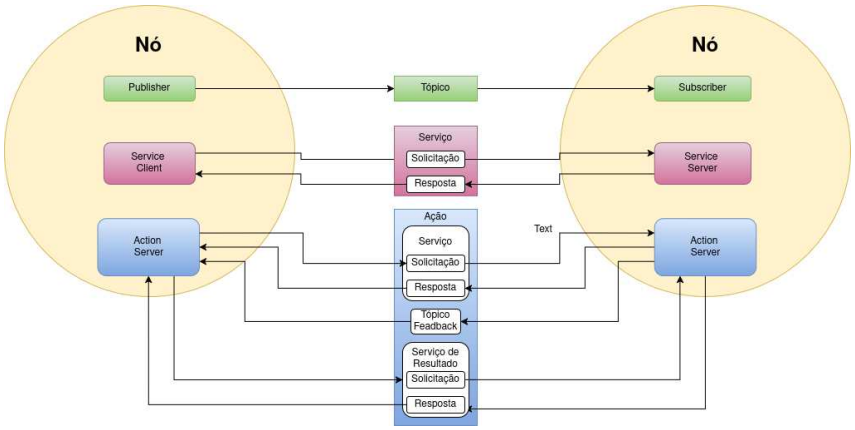
Os tópicos constituem o principal mecanismo de comunicação entre nós no ROS. Esse mecanismo segue o modelo publish–subscribe, no qual um nó pode publicar mensagens em um determinado tópico enquanto outros nós podem se inscrever nesse mesmo tópico para receber as informações transmitidas. Esse modelo é adequado para o envio contínuo de dados, como medições de sensores ou estimativas de posição do robô,

permitindo que diferentes componentes do sistema utilizem simultaneamente as mesmas informações.

Os serviços permitem a comunicação síncrona entre nós por meio de um modelo de requisição e resposta. Nesse caso, um nó cliente envia uma solicitação a um nó servidor e aguarda o retorno da resposta após o processamento da requisição. Esse tipo de interação é normalmente utilizado para operações que exigem uma resposta imediata.

As ações são utilizadas quando uma tarefa exige um tempo maior de execução. Diferentemente dos serviços, as ações permitem acompanhar o progresso da tarefa por meio de mensagens de feedback enviadas ao longo da execução. As ações podem ser canceladas pelo cliente, caso necessário.

Figura 9 – Representação de funcionamento de interação entre nós no ROS.



Fonte: Autoria própria.

Em 2017 foi distribuída uma nova geração do *middleware* denominada ROS 2. Uma das principais diferenças entre o ROS 2 e a primeira versão do ROS está na arquitetura de comunicação. Enquanto o ROS original dependia de um nó central denominado ROS Master, o ROS 2 adota uma arquitetura distribuída em que os nós podem se comunicar diretamente entre si. Essa comunicação é implementada por meio do *middleware Data Distribution Service (DDS)*, que oferece suporte a sistemas distribuídos, mecanismos de controle de qualidade de serviço *(Quality of Service – QoS)* e maior confiabilidade na transmissão de mensagens. Essa mudança arquitetural torna o ROS 2 mais adequado para aplicações robóticas modernas, incluindo sistemas multi-robô e aplicações industriais que exigem maior robustez e escalabilidade (AL-BATATI; HUDA; MOHAMMADI, 2025)

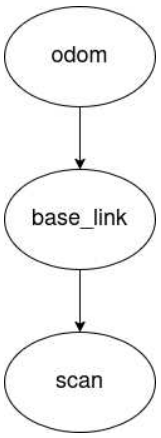
2.7.1 Transformações espaciais (TF2)

Em sistemas robóticos móveis, diferentes informações são obtidas e processadas em referenciais distintos. A pose estimada do robô, as medições dos sensores e a representação do ambiente precisam, portanto, ser relacionadas de forma coerente por meio

de transformações espaciais. No ecossistema ROS 2, essa função é desempenhada pelo *TF2*[1], que permite representar a relação entre diferentes *frames* de referência e consultar transformações entre eles ao longo do tempo.

As transformações espaciais em sistemas robóticos podem ser classificadas em estáticas e dinâmicas. As transformações estáticas são aquelas cuja relação geométrica entre dois *frames* permanece constante ao longo do tempo, como ocorre entre a base do robô e um sensor fixado à sua estrutura. Já as transformações dinâmicas variam continuamente durante a operação, pois estão associadas ao movimento da plataforma no ambiente ou à atualização de sua pose estimada. No contexto da navegação móvel, a transformação entre a base e o Lidar (`base_link` → `scan`) costuma ser estática, enquanto a transformação entre odometria e a base (`odom` → `base_link`) é dinâmica, uma vez que depende do deslocamento do robô. Na Figura 10 temos a visualização dessa árvore de transformaçãp `odom` → `base_link` → `scan`.

Figura 10 – Exemplo árvore de transformação.



Fonte: Autoria própria.

A consistência dessas transformações é essencial para a navegação autônoma. Caso os *frames* estejam incorretos ou mal definidos, o robô pode interpretar de maneira equivocada a posição dos obstáculos, sua própria orientação no ambiente ou a relação entre o mapa e os dados dos sensores. Assim, o TF2 constitui a base espacial, permitindo integrar em um mesmo referencial lógico as estimativas de pose, os sensores e a representação do ambiente utilizada pelos algoritmos de navegação.

2.7.2 URDF e modelagem do robô

A modelagem do robô constitui uma etapa essencial na integração entre a plataforma física e o sistema de navegação. No ROS 2, essa descrição é normalmente realizada

[1] Documentação sobre TF2 no ROS 2: <https://docs.ros.org/en/humble/Tutorials/Intermediate/Tf2/Introduction-To-Tf2.html>

por meio do *URDF* (*Unified Robot Description Format*)[2], um formato que representa a estrutura do robô em termos de *links* e *joints*. Os *links* correspondem aos corpos rígidos do sistema, enquanto os *joints* descrevem as conexões e relações cinemáticas entre esses corpos. Essa modelagem permite ao sistema conhecer a organização estrutural da plataforma e a posição relativa entre seus componentes.

Em robôs móveis, o URDF é importante para representar a base do robô e os sensores embarcados, como LiDAR. Ao descrever a posição e a orientação desses elementos em relação ao *frame* principal da plataforma, o modelo fornece as informações necessárias para que o restante do sistema interprete corretamente os dados sensoriais. Assim, o URDF não se limita a uma representação visual do robô, mas desempenha papel fundamental na coerência geométrica da aplicação.

A publicação das transformações associadas ao modelo do robô é realizada, em geral, pelo nó `robot_state_publisher`[3]. Esse componente utiliza a descrição URDF e o estado das juntas para publicar na árvore TF as transformações entre os diferentes *links* da plataforma. Dessa forma, o `robot_state_publisher` atua como a ponte entre o modelo geométrico do robô e a infraestrutura espacial usada por módulos de percepção, localização e navegação.

2.8 Árvores de Comportamento

As Árvores de Comportamento (*Behavior Trees* – BT) constituem uma arquitetura formal para modelagem de comportamentos reativos e modulares, amplamente utilizada em robótica e sistemas autônomos. Originalmente popularizadas na indústria de jogos digitais, as BT passaram a ser empregadas em aplicações robóticas devido à sua capacidade de estruturar comportamentos complexos de maneira hierárquica e modular (IOVINO et al., 2025).

Uma BT é representada como uma árvore direcionada e acíclica, composta por nós conectados hierarquicamente. O nó raiz define o ponto de entrada da execução, enquanto os nós internos representam operadores de controle e os nós folha representam ações ou condições executáveis.

2.8.1 Estrutura Fundamental

Os principais tipos de nós em uma BT são:

[2] Documentação URDF ROS 2: <https://docs.ros.org/en/humble/Tutorials/Intermediate/URDF/URDF-Main.html>

[3] Documentação Robot State Publisher: <https://docs.ros.org/en/humble/p/robot_state_publisher/>.

- **Nós de Controle**: responsáveis por definir o fluxo de execução, como *Sequence* e *Fallback* (ou Selector);
- **Nós de Ação**: executam comportamentos que produzem efeitos no ambiente;
- **Nós de Condição**: verificam estados do ambiente ou do sistema;
- **Decoradores**: modificam o comportamento de um único nó filho.

A execução de uma BT ocorre por meio de um processo denominado *tick*, enviado periodicamente pelo nó raiz. Esse mecanismo percorre a árvore de cima para baixo e da esquerda para a direita, ativando os nós conforme sua lógica interna.

Cada nó pode retornar um dos seguintes estados:

- **Success**: a execução foi concluída com sucesso;
- **Failure**: a execução falhou;
- **Running**: a execução está em andamento.

Essa tríade de retornos permite que a árvore mantenha reatividade contínua ao ambiente.

2.8.2 Nós de Controle

O nó *Sequence* executa seus filhos em ordem sequencial e retorna *Success* apenas se todos os filhos retornarem sucesso. Caso algum filho retorne *Failure*, a execução é interrompida.

O nó *Fallback* (ou Selector) executa seus filhos até que um deles retorne *Success*. Se todos retornarem *Failure*, o nó também retorna falha.

Essa estrutura hierárquica permite decompor tarefas complexas em subtarefas menores, favorecendo a modularidade do sistema.

3 Metodologia

A metodologia adotada neste trabalho foi estruturada com o objetivo de viabilizar a implementação e a avaliação experimental da navegação autônoma do robô *TurtleBot3 Waffle Pi* em ambiente laboratorial. Para isso, o procedimento foi dividido em etapas que abrangem desde a preparação da plataforma experimental até a realização de ensaios comparativos com algoritmos de planejamento e controle.

Inicialmente, foram realizadas a configuração do robô, sua integração ao ROS 2 e a validação preliminar da plataforma, incluindo a verificação do funcionamento dos sensores, da odometria e da árvore de transformações. Em seguida, realizou-se a construção do mapa do laboratório utilizando a biblioteca *slam_toolbox*. Posteriormente, foi configurada a arquitetura de navegação do Nav2, incluindo a representação geométrica da plataforma, *costmap*, o planejamento global e o controle local.

Após essa etapa de preparação e validação, foram executados os experimentos voltados à análise do impacto da configuração do *costmap* e de diferentes parâmetros e algoritmos sobre o desempenho do planejamento e do controle. Por questões de organização textual, o procedimento experimental e os testes de validação são detalhados nas seções finais deste capítulo.

Os testes e as validações foram realizados no laboratório *Smart Factoring*, localizado no Laboratório de Sistemas Embarcados e Computação Pervasiva (Embedded) da Universidade Federal de Campina Grande (UFCG), ambiente experimental destinado ao desenvolvimento de pesquisas em automação.

3.1 Configuração do Ambiente

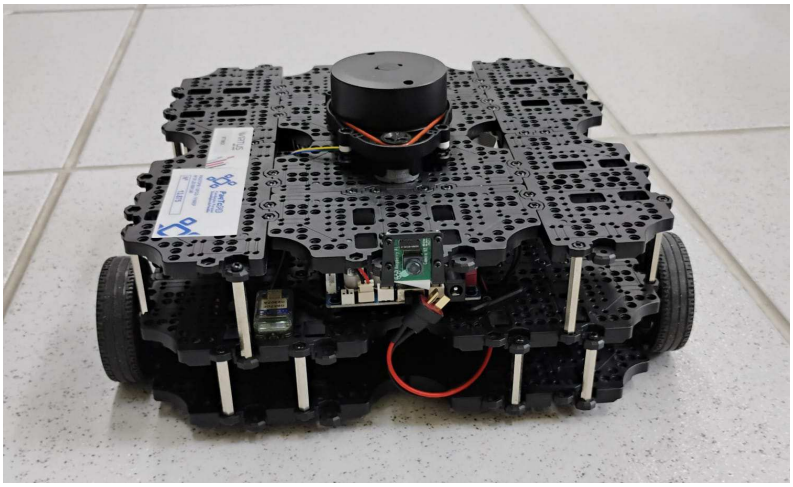
Para o desenvolvimento deste trabalho, foi utilizado o sistema operacional Ubuntu 22.04, escolhido principalmente por sua ampla compatibilidade com ferramentas voltadas ao desenvolvimento de aplicações robóticas. Como *middleware* de comunicação entre os diferentes componentes do sistema, foi empregada a distribuição *ROS 2 Humble*, que fornece a infraestrutura necessária para a implementação de sistemas robóticos distribuídos.

3.1.1 Robô Móvel *Turtlebot3 Waffle Pi*

O robô móvel utilizado neste trabalho foi o *TurtleBot3 Waffle Pi*, uma plataforma robótica de código aberto amplamente empregada em pesquisas e aplicações educacionais na área de robótica móvel. Desenvolvido pela ROBOTIS em colaboração com a Open

Robotics, o TurtleBot3 possibilita o desenvolvimento e a experimentação de algoritmos relacionados à navegação, percepção e controle de robôs móveis (PARK; LEE; OH, 2019).

Figura 11 – Turtlebot 3 Waffle Pi.



Fonte: Autoria própria.

O TurtleBot3 Waffle Pi é equipado com uma unidade computacional baseada em Raspberry Pi, uma placa controladora OpenCR responsável pelo controle dos atuadores e interface com sensores, além de um sensor LiDAR utilizado para percepção do ambiente. O sistema de locomoção utiliza um modelo de acionamento diferencial composto por dois atuadores DYNAMIXEL, permitindo ao robô realizar movimentos de translação e rotação no plano.

As principais especificações técnicas do robô são apresentadas na Tabela 1.

Tabela 1 – Principais especificações do TurtleBot3 Waffle Pi

Especificação	**Valor**
Velocidade translacional máxima	0,26 m/s
Velocidade rotacional máxima	1,82 rad/s (104,27°/s)
Carga útil máxima	30 kg
Dimensões (C × L × A)	281 mm × 306 mm × 141 mm
Peso (com SBC, bateria e sensores)	1,8 kg
Altura máxima de obstáculo	10 mm
Tempo estimado de operação	2 h
Tempo estimado de recarga	2 h 30 min
Computador embarcado	Raspberry Pi 4
Controladora principal	OpenCR 1.0
Atuadores	2 × DYNAMIXEL XM430-W210-T
Sensor LiDAR	LDS-02
Câmera	Raspberry Pi Camera v2.1
Bateria	Li-Po 11,1 V, 1800 mAh

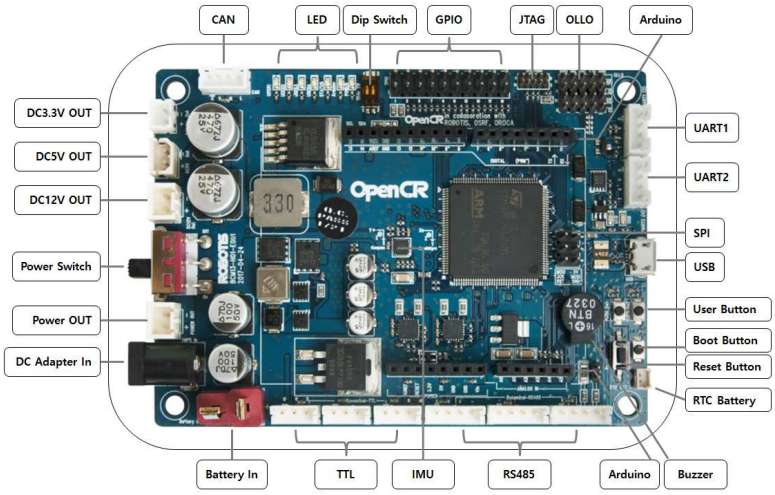
Fonte: Adaptado de (ROBOTIS, 2024).

3.1.1.1 OpenCR

O OpenCR 1.0 é a placa controladora principal do TurtleBot3 Waffle Pi, sendo responsável pela interface entre o computador embarcado e os dispositivos de baixo nível da plataforma.

Do ponto de vista de hardware, o OpenCR oferece pinos de entrada e saída digitais e analógicos, bem como diferentes interfaces de comunicação, como USB, UART, SPI, I2C e CAN, o que facilita sua integração com sensores, atuadores e outros módulos do robô. O OpenCR também fornece saídas de alimentação em 12 V, 5 V e 3,3 V para o computador embarcado e sensores. A alimentação da placa pode ser feita por bateria ou fonte externa.

Figura 12 – Hardware do OpenCR.



Fonte: (ROBOTIS, 2026).

3.1.1.2 LiDAR

O sensor LiDAR utilizado na plataforma é o LDS-02, um scanner a laser bidimensional com varredura de 360°, empregado para obtenção de medições de distância ao redor do robô. No TurtleBot3, esse sensor é utilizado como principal fonte de percepção exteroceptiva para tarefas de mapeamento, localização e navegação.

O LDS-02 opera com alimentação de $5\,V$, taxa de amostragem fixa de $2{,}3\,kHz$, alcance de detecção entre 160 e $8000\,mm$, interface USART de $3{,}3\,V$ a $115200\,bps$ e resolução angular de 1°.

3.1.1.3 Encoder

O sensor AS5045, encoder absoluto magnético sem contato, acoplado ao DYNAMIXEL XM430-W210-T, é utilizado para medição angular em 360°. Esse sensor fornece a posição angular do eixo como um valor absoluto, o que evita a necessidade de contagem acumulada de pulsos para determinar a posição instantânea.

Figura 13 – LiDAR LDS-02.



Fonte: (ROBOTIS, 2026).

O AS5045 tem resolução de 12 bits, correspondendo a 4096 posições discretas por revolução, além de saída por interface SSI e sinal PWM. O sensor opera com tecnologia magnética sem contato, utilizando um ímã simples de dois polos posicionado sobre ou sob o circuito integrado, e oferece ainda ajuste programável da posição de zero.

3.1.2 Configuração inicial do TurtleBot3 Waffle Pi e integração com o ROS

A integração do robô móvel TurtleBot3 Waffle Pi com o ambiente ROS 2 é realizada por meio do conjunto de pacotes disponibilizados pela plataforma *TurtleBot3*, desenvolvida pela empresa *Robotis*. Esses pacotes fornecem a interface entre o hardware do robô e o ecossistema ROS, permitindo a utilização dos sensores e atuadores embarcados em aplicações robóticas como mapeamento, localização e navegação autônoma.

O TurtleBot3 Waffle Pi possui uma arquitetura computacional distribuída composta por dois elementos principais: um computador embarcado instalado no robô, responsável pelo controle direto do hardware, e um computador remoto utilizado para executar aplicações de maior custo computacional. O computador embarcado é um Raspberry Pi responsável pela execução dos nós ROS que realizam a comunicação com os sensores e atuadores do robô, enquanto o computador remoto executa aplicações como geração de mapas, planejamento de trajetórias e monitoramento do sistema. A comunicação entre esses dispositivos ocorre por meio da rede utilizando o *middleware* DDS (*Data Distribution Service*), que constitui o mecanismo de comunicação do ROS2.

A configuração do sistema segue o procedimento recomendado na documentação oficial do TurtleBot3[1], podendo ser dividida nas seguintes etapas principais:

1. **Configuração do computador remoto (PC Setup):**

   Inicialmente, foi realizada a instalação do ROS2 Humble na sua versão desktop no computador que será utilizado para monitoramento e execução das aplicações.

[1] Manual de instalação do TurtleBot3: <https://emanual.robotis.com/docs/en/platform/turtlebot3/quick-start/>

Também foram instalados os pacotes específicos do TurtleBot3 (*DynamixelSDK; turtlebot3_msgs; turtlebot3)*, que permitem a comunicação com o robô por meio da rede.

2. **Configuração do computador embarcado (SBC Setup):**

   A segunda etapa consiste na configuração do computador embarcado no robô, que no caso do TurtleBot3 Waffle Pi é um Raspberry Pi. Inicialmente, realizou a instalação da versão server do Ubuntu 22.04, e na sequência a instalação do ROS 2 em sua versão base, sem ferramentas GUI, em seguida foi realizada a instalação dos pacotes *hls-lfcd-lds-driver; turtlebot3-msgs; dynamixel-sdk; xacro; turtlebot3; ld08_driver; coin_d4_driver*. Essa configuração permite que o Raspberry Pi se comunique com a OpenCR e com o computador remoto.

3. **Configuração da placa controladora (OpenCR Setup):**

   O TurtleBot3 utiliza a placa OpenCR como controlador responsável pela interface direta com os motores e sensores do robô. Durante essa etapa é necessário carregar o firmware específico do TurtleBot3 na placa OpenCR, permitindo que ela se comunique corretamente com o Raspberry Pi por meio de uma interface serial.

4. **Configuração da rede (Network Configuration):**

   Após a configuração do hardware e dos pacotes ROS, é necessário estabelecer a comunicação entre o computador remoto e o Raspberry Pi do robô. Para isso, ambos os dispositivos foram conectados na mesma rede local e definido o *ROS_DOMAIN_ID*.

3.2 Construção do mapa

Para a etapa de mapeamento, foi utilizada a biblioteca *slam_toolbox*[2], que é atualmente é a solução padrão para SLAM 2D no ROS 2. O pacote permite o mapeamento de áreas extensas, a atualização desses mapas ao longo do tempo e a operação em plataformas robóticas com recursos computacionais limitados (MACENSKI; JAMBRECIC, 2021).

Do ponto de vista operacional, o *SLAM Toolbox* disponibiliza três modos principais: mapeamento síncrono, mapeamento assíncrono e localização pura. No modo síncrono, as medições são armazenadas em *buffer* e inseridas no problema de SLAM com ênfase na qualidade do mapa gerado. No modo assíncrono, novas medições são processadas apenas quando a anterior é concluída, favorecendo a manutenção do desempenho em tempo real. Já o modo de localização pura utiliza um mapa previamente obtido e um *buffer* deslizante de medições para localizar o robô sem modificar persistentemente o ambiente mapeado. Nessa aplicação foi aplicado o modo de mapeamento online assíncrono.

[2] Repositório da biblioteca slam_toolbox: <https://github.com/SteveMacenski/slam_toolbox>

A partir da configuração de referência *mapper_params_online_async.yaml*, foram realizadas adaptações pontuais, Tabela 2, com o objetivo de adequar ao ambiente laboratorial e à plataforma TurtleBot3 Waffle Pi.

Tabela 2 – Parâmetros ajustados no processo de mapeamento.

Parâmetro	**Valor inicial**	**Valor ajustado**	**Objetivo do ajuste**
minimum_travel_distance	0,5 m	0,2 m	Melhorar a representação de deslocamentos curtos.
minimum_travel_heading	0,5 rad	0,2 rad	Melhorar a representação de mudanças de orientação em trajetórias curtas.
minimum_time_interval	0,5 s	0,2 s	Permitir maior frequência de atualização do processo de mapeamento.
max_laser_range	20,0 m	8,0 m	Adequar o alcance do LiDAR ao cenário interno e ao modelo LDS-02.

Em testes preliminares, a resolução de 0,05 m por célula mostrou-se suficiente para representar adequadamente o ambiente, não sendo necessário realizar um ajuste mais refinado. Os recursos de *scan matching* e *loop closing* também foram mantidos, preservando as capacidades do *slam_toolbox* de refinar a odometria por alinhamento de varreduras e corrigir o mapa por meio de fechamento de ciclos.

A condução do robô durante a etapa de mapeamento foi realizada por meio de teleoperação. O ambiente a ser mapeado pode ser visto na Figura 14.

Figura 14 – Laboratório Smart Factoring.



Fonte: Autoria própria.

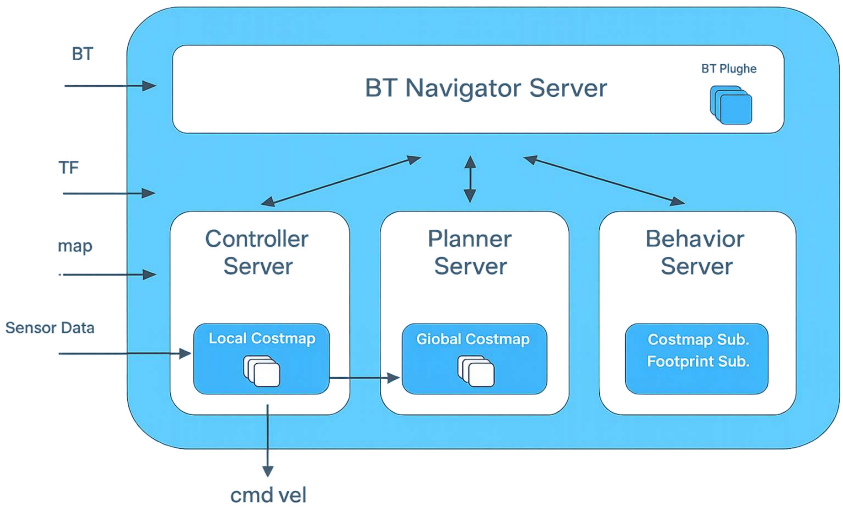
3.3 Navegação

Foi escolhido como *framework* de navegação o Navigation2 (Nav2), que foi desenvolvido para o ROS 2 e considerado o sucessor direto do tradicional *ROS Navigation Stack* utilizado no ROS 1. Esse *framework* foi projetado para atender às demandas de sistemas robóticos modernos, incorporando melhorias relacionadas à modularidade, escalabilidade, desempenho e capacidade de configuração. Essas características tornam o Nav2 adequado para aplicações que envolvem robôs móveis autônomos em ambientes dinâmicos e complexos (MACENSKI et al., 2020).

Diferentemente da implementação presente no ROS 1, o Nav2 foi concebido com uma arquitetura distribuída baseada em nós ROS 2 especializados, denominados servidores de navegação. Cada servidor é responsável por uma funcionalidade específica dentro do processo de navegação. Entre os principais servidores estão o *planner server*, responsável pelo planejamento global da trajetória; o *controller server*, responsável pela execução do seguimento da trajetória; o *behavior server*, responsável por comportamentos e estratégias de recuperação. Há também o *bt navigator server*, atua como supervisor coordenando o fluxo de execução das tarefas de navegação por meio de árvores de comportamento. A comunicação entre servidores é realizada por meio de *ros2 action*. O diagrama da estrutura do do Nav2 pode ser vista na Figura 15.

O funcionamento do sistema requer, como entradas, as transformações espaciais (TFs), uma fonte de mapa, um arquivo XML correspondente à árvore de comportamento e dados sensoriais. A partir dessas informações, o sistema produz, como saída, comandos de velocidade enviados ao robô.

Figura 15 – Estrutura do Nav2.



Fonte: Adaptado de (MACENSKI et al., 2020).

Cada servidor armazena diferentes *plugins*, que correspondem aos algoritmos responsáveis pela execução efetiva de tarefas específicas, como planejamento e controle. Os servidores utilizam mapas de custo (*costmaps)* como representação do ambiente para apoiar a tomada de decisão durante a navegação.

3.3.1 Localização no ambiente

A etapa de localização foi realizada com o pacote `nav2_amcl`[3], utilizado como componente de localização do *framework* Navigation2 em mapas estáticos previamente construídos. Nesse arranjo, a estimativa de pose do robô foi obtida a partir da combinação entre as leituras do sensor LiDAR e a odometria da plataforma, permitindo atualizar continuamente sua posição e orientação no mapa durante a navegação.

3.3.2 Definição do *footprint* do robô

A definição do *footprint* do robô constitui uma das etapas iniciais da configuração da navegação, pois é por meio dele que o sistema representa a geometria da plataforma no plano. Neste trabalho, essa definição foi realizada a partir da geometria do modelo oficial do *TurtleBot3 Waffle Pi*, descrita no arquivo URDF. Como a configuração do Nav2 utiliza o parâmetro *robot_base_frame: base_link*, os vértices do *footprint* foram definidos no plano XY com referência a esse frame.

Inicialmente, foi considerada a caixa de colisão principal da base do robô, cujo centro está posicionado em $(-0{,}064,\ 0)$ e cujas dimensões no plano são $0{,}266 \times 0{,}266$. A partir dessas informações, os limites frontal, traseiro e laterais da base foram obtidos por

$$x_{\max} = c_x + \frac{L_x}{2}, \qquad x_{\min} = c_x - \frac{L_x}{2} \tag{3.1}$$

$$y_{\max} = c_y + \frac{L_y}{2}, \qquad y_{\min} = c_y - \frac{L_y}{2} \tag{3.2}$$

em que (c_x, c_y) representa o centro da caixa de colisão e (L_x, L_y) suas dimensões no plano.

Além da base principal, também foram consideradas as projeções laterais das rodas e o avanço frontal do conjunto da câmera, de modo a obter uma representação mais fiel da geometria do robô. A inclusão desses elementos foi importante para evitar uma aproximação excessivamente simplificada da plataforma, sobretudo em regiões estreitas e em manobras próximas a obstáculos.

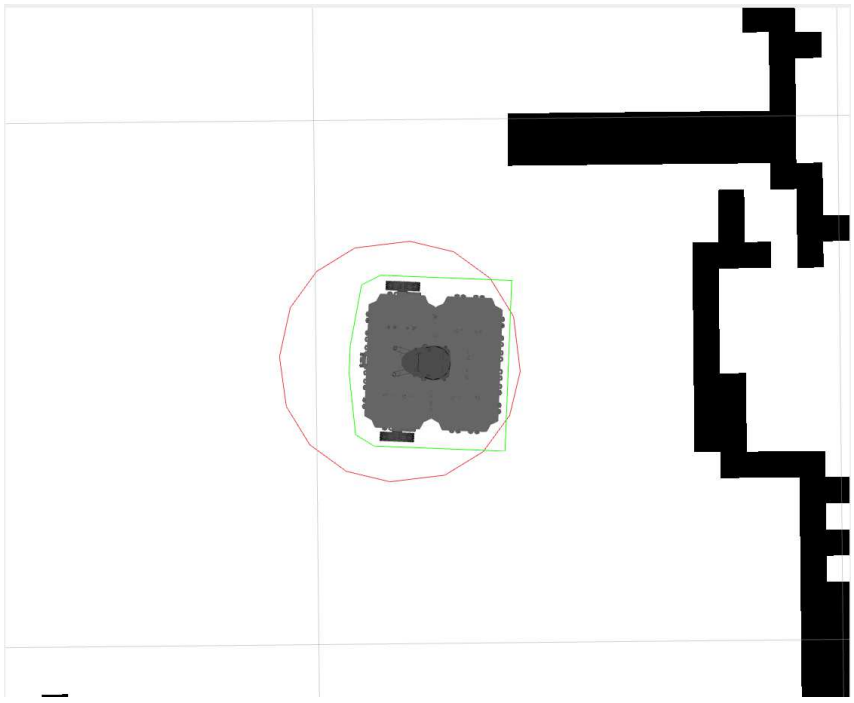
Com base nas dimensões extraídas do modelo, foi definido um *footprint* poligonal convexo com oito vértices, adotando-se uma aproximação conservadora da geometria do

[3] Documentação do pacote amcl no ROS 2: <https://docs.ros.org/en/humble/p/nav2_amcl/>.

robô no plano XY:

$$\mathcal{F} = \begin{bmatrix} 0{,}086 & 0{,}016 \\ 0{,}069 & 0{,}133 \\ 0{,}033 & 0{,}153 \\ -0{,}197 & 0{,}153 \\ -0{,}197 & -0{,}153 \\ 0{,}033 & -0{,}153 \\ 0{,}069 & -0{,}133 \\ 0{,}086 & -0{,}016 \end{bmatrix} \tag{3.3}$$

Figura 16 – Comparação entre a aproximação circular do robô e o *footprint* poligonal adotado.



Fonte: Autoria própria.

3.3.3 Configuração do *costmap*

Na etapa seguinte da configuração da navegação, foi realizada a sintonia da *inflation layer* do *global costmap*. Como essa camada tem a função de expandir o custo ao redor dos obstáculos, induzindo o planejador global a preferir regiões mais centrais e seguras do espaço livre, optou-se por considerar o parâmetro *inflation_radius* como uma das variáveis de interesse na configuração do sistema.

Durante essa etapa, foram mantidos os demais elementos da arquitetura de navegação compatíveis com a configuração previamente definida, incluindo o mapa, o *footprint* do robô e o planejador global. A influência do *inflation_radius* sobre a geração do ca-

minho global foi posteriormente avaliada por meio de testes comparativos, descritos na subseção de procedimentos experimentais.

3.3.4 Planejadores globais no Nav2

O *planner server* é o componente responsável por calcular caminhos globais entre a posição atual do robô e a meta desejada. O Nav2 disponibiliza diferentes *plugins* de planejamento, que podem ser escolhidos conforme as características do ambiente e da plataforma. Neste trabalho, foram utilizados o *NavFn Planner* e o *Smac 2D Planner*.

O *NavFn Planner* implementa um planejador holonômico que pode operar com expansão baseada em Dijkstra ou em A*. No plugin *NavFn Planner*, a seleção entre esses dois modos é realizada pelo parâmetro *use_astar*. Quando *use_astar = true*, a busca é realizada com A*; caso contrário, o planejador utiliza Dijkstra. A documentação oficial do Nav2 não expõe parâmetros específicos para ajuste da heurística do A*, tratando essa componente como parte interna da implementação do planejador.

O *Smac 2D Planner* também implementa um algoritmo A*, porém com maior flexibilidade de configuração. Diferentemente do *NavFn Planner*, no qual o uso do A* depende basicamente do parâmetro *use_astar*, o *Smac 2D Planner* disponibiliza parâmetros que influenciam o comportamento da busca, o tratamento dos custos do mapa e a eficiência do planejamento.

Os parâmetros inicialmente adotados para o *Smac 2D Planner* estão apresentados na Tabela 3. Esses valores foram definidos com base na documentação oficial e utilizados como ponto de partida para os testes realizados posteriormente.

Tabela 3 – Parâmetros iniciais adotados para o *SmacPlanner2D*.

Parâmetro	**Valor adotado**	**Descrição**
tolerance	0.125	Tolerância admitida para alcançar a meta planejada.
downsample_costmap	false	Indica que o mapa de custos não é reduzido antes do planejamento.
downsampling_factor	1	Fator de redução do mapa de custos, mantido unitário por não haver redução.
allow_unknown	true	Permite que o planejador considere regiões desconhecidas durante a busca.
max_iterations	1000000	Número máximo de iterações permitido no processo de planejamento.
max_planning_time	2.0	Tempo máximo permitido para o cálculo do plano.
cost_travel_multiplier	2.0	Fator de ponderação associado ao custo de deslocamento no mapa.

Para facilitar a organização dos testes, foram criados arquivos de configuração

específicos para cada planejador, armazenados no diretório *tb3_navigation/param*. Essa estratégia permitiu testar e avaliar as diferentes abordagens sem sobrescrever configurações anteriores, preservando a comparabilidade entre os experimentos.

3.3.5 Controladores no Nav2

O servidor de controle (*controller server*) é responsável por realizar o controle local do robô, gerando comandos de velocidade linear e angular que permitam seguir o caminho planejado. O Nav2 disponibiliza *plugins* de diferentes tipos de controladores. Neste trabalho foram testados e analisados o controlador reativo *Dynamic Window Approach* (DWB) e o geométrico *Regulated Pure Pursuit* (RPP).

O *Dynamic Window Based Controller* (DWB), utilizado no *Navigation2* (Nav2), é um controlador local reativo baseado nos princípios do *Dynamic Window Approach* (DWA). A principal diferença do DWB em relação ao DWA original está em sua estrutura baseada em *plugins*. Em vez de utilizar uma única função objetivo fixa, o DWB emprega funções de avaliação denominadas *critics*, responsáveis por pontuar as trajetórias candidatas segundo diferentes critérios.(SCHENA, 2024).

Esses *critics* substituem, na prática, a função objetivo fixa do DWA por uma soma ponderada de critérios configuráveis. O DWB preserva a ideia central do DWA, mas amplia essa lógica por meio de uma arquitetura modular.

Assim como no DWA clássico, o DWB amostra comandos de velocidade factíveis dentro da janela dinâmica e projeta trajetórias locais a partir do estado atual do robô. Entretanto, sua arquitetura foi reformulada para ser mais flexível e extensível.

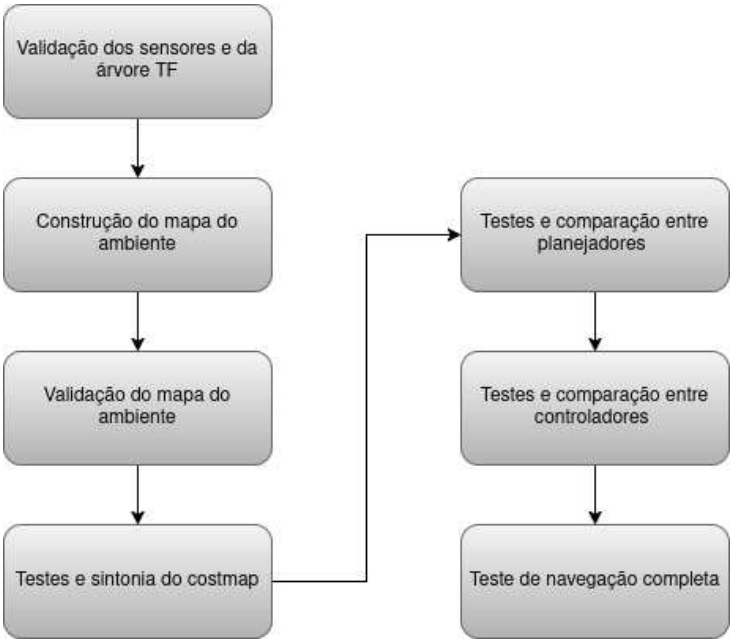
O *Regulated Pure Pursuit* (RPP) é um controlador local de seguimento de trajetórias utilizado no *Navigation2* (Nav2), derivado do algoritmo clássico *Pure Pursuit*. O RPP constitui uma evolução do *Pure Pursuit* e do *Adaptive Pure Pursuit*, introduzindo mecanismos de regulação da velocidade linear com foco em segurança, estabilidade e robustez em ambientes estreitos ou parcialmente observáveis (MACENSKI et al., 2023b).

3.4 Síntese do procedimento experimental

Esta seção apresenta o procedimento experimental adotado para a avaliação da navegação autônoma do robô *TurtleBot3 Waffle Pi*. Os ensaios foram organizados em etapas sucessivas, iniciando pela validação da plataforma, construção e validação do mapa, passando pela sintonia do *costmap*, pelos testes comparativos com planejadores globais e controladores locais e, por fim, pelo teste de navegação completa.

A Figura 17 apresenta o fluxo das atividades desenvolvidas.

Figura 17 – Procedimento experimental adotado.



Fonte: Autoria própria.

3.4.1 Procedimentos de validação da plataforma

Antes da realização dos experimentos de mapeamento e navegação, foram conduzidos procedimentos de validação da plataforma com o objetivo de verificar o funcionamento dos principais elementos envolvidos na percepção e na estimação de movimento do robô. Essa etapa contemplou a avaliação da estrutura de transformações do sistema, do sensor LiDAR, da odometria e da compatibilidade entre o mapa gerado e a percepção laser da plataforma.

A verificação da árvore de transformações do sistema foi realizada por meio da inspeção dos principais referenciais publicados no ROS 2, incluindo *map*, *odom*, *base_footprint*, *base_link* e o referencial do sensor LiDAR. Essa etapa teve como objetivo verificar a coerência da estrutura TF e sua adequação às etapas posteriores de mapeamento, localização e navegação.

A validação do LiDAR foi realizada por meio da comparação entre distâncias de referência previamente definidas no laboratório e as leituras fornecidas pelo tópico */scan*. Para isso, o robô foi posicionado diante de uma parede em três distâncias distintas, correspondentes a 0,50 m, 1,00 m e 1,50 m.

A validação da odometria foi conduzida a partir de deslocamentos lineares controlados realizados por teleoperação. Foram executados percursos de 1 m e 2 m, permitindo comparar o deslocamento efetivamente realizado pela plataforma com a estimativa fornecida pelo sistema odométrico.

Por fim, a validação do mapa foi realizada no RViz, por meio da sobreposição entre

o mapa de ocupação e o *LaserScan*. O critério adotado consistiu na verificação visual do alinhamento entre os pontos do laser e as paredes representadas no mapa, bem como na observação da estabilidade da localização durante o deslocamento do robô.

3.4.2 Testes de sintonia do *costmap*

A avaliação do parâmetro *inflation_radius* foi realizada por meio de testes comparativos com os valores *0.7*, *0.5* e *0.3*. Durante essa etapa, foram mantidos constantes o mapa, o *footprint* do robô e o planejador global, de modo que a única variável analisada fosse o valor de *inflation_radius* no *global costmap*.

A análise foi conduzida por meio da geração do caminho global, sem execução do movimento da plataforma. Para isso, foi utilizado o servidor de ação do Nav2 *ComputePathToPose* em conjunto com a visualização do tópico */plan* no RViz.

Como critério de avaliação, foi observado o comportamento geométrico do caminho planejado em relação aos obstáculos, considerando principalmente o afastamento lateral em relação a paredes e objetos, bem como a suavidade do trajeto gerado.

3.4.3 Testes de ajuste e comparação entre planejadores

Após a definição das configurações iniciais dos planejadores, foram realizados testes sobre o parâmetro *cost_travel_multiplier* do *Smac 2D Planner*, que controla o peso do custo do *costmap* no processo de busca. Para isolar o efeito dessa variável, foram mantidos fixos os demais parâmetros do planejador e do ambiente, variando-se apenas esse fator entre os valores *1.0*, *2.0* e *5.0*. A análise foi realizada a partir dos caminhos globais gerados, observando-se principalmente o afastamento em relação aos obstáculos.

Na sequência, foi estabelecido um protocolo experimental para a comparação entre os planejadores *NavFn* e *Smac 2D Planner*. A análise considerou também duas formas de representação geométrica do robô no *global costmap*: aproximação circular por meio de *robot_radius* e representação poligonal por meio de *footprint*. Dessa forma, foram avaliadas quatro configurações distintas: *NavFn* com *robot_radius*, *NavFn* com *footprint*, *Smac 2D Planner* com *robot_radius* e *Smac 2D Planner* com *footprint*.

Para garantir comparabilidade entre os testes, foram mantidos fixos os demais parâmetros do sistema, incluindo o mapa utilizado, os valores de *inflation_radius* e *cost_scaling_factor* do *global costmap*, a tolerância do planejador e as poses inicial e final em cada cenário. Foi adotado *inflation_radius = 0.5*, *cost_scaling_factor = 3.0* e *tolerance = 0.125* como configuração comum às comparações principais.

Os testes foram organizados a partir de dois pontos de origem e um ponto de destino no mapa, denominados P1, P2 e PD, respectivamente, escolhidos de forma a

representar regiões distintas do ambiente. Para cada configuração e para cada ponto de origem, foram realizadas três repetições, totalizando 24 execuções experimentais.

Figura 18 – Robô posicionado no ponto P1.



Fonte: Autoria própria.

Para a avaliação dos planejadores globais, as métricas foram obtidas de forma automatizada por meio de um script em Python integrado ao ambiente experimental. Esse script enviava requisições à ação */compute_path_to_pose* do Nav2, definindo explicitamente a pose inicial e a pose final. Essa estratégia permitiu garantir que todos os planejadores fossem avaliados sob as mesmas condições de origem e destino, reduzindo a influência de variações da localização estimada do robô durante a comparação.

A determinação de sucesso ou fracasso do planejamento foi feita a partir do status final da ação */compute_path_to_pose*, considerando sucesso quando a ação era concluída com retorno válido. O tempo de planejamento foi obtido diretamente do campo *planning_time* presente no resultado da ação, correspondendo ao intervalo necessário para que o planejador calculasse o caminho global.

A trajetória planejada foi obtida a partir do campo *path* da resposta da ação, representado por uma mensagem do tipo *nav_msgs/Path*, composta por uma sequência ordenada de poses no frame *map*. A partir dessa sequência de poses, o comprimento do caminho foi calculado automaticamente pela soma das distâncias euclidianas entre pontos consecutivos, conforme a Equação 3.4:

$$L = \sum_{i=1}^{N-1} \sqrt{(x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2} \tag{3.4}$$

em que N representa o número total de poses do caminho e (x_i, y_i) corresponde à posição do i-ésimo ponto da trajetória.

3.4.4 Testes de ajuste e comparação entre controladores

Na etapa de avaliação dos controladores foi utilizada a ação */follow_path*, permitindo que o robô executasse diretamente um caminho de referência previamente gerado, sem integração ainda com o supervisor *BT Navigator Server*. Essa escolha teve como objetivo concentrar a análise exclusivamente no comportamento do controlador durante a execução do caminho.

Para todos os testes foi utilizado um único caminho gerado pelo planejador *Smac 2D Planner*.

Foram realizados testes exploratórios variando, de forma isolada, parâmetros do *DWBLocalPlanner*. Entre os parâmetros avaliados, foram considerados o *sim_time*, responsável pelo horizonte temporal da simulação das trajetórias candidatas, e os pesos *PathAlign.scale* e *PathDist.scale*, que influenciam, respectivamente, o alinhamento da orientação do robô com a trajetória e a aderência espacial ao caminho de referência. Para o parâmetro *sim_time*, foram analisados os valores 1,5 e 2,0. Para *PathAlign.scale* e *PathDist.scale*, foram considerados os valores 24 e 32. A estratégia experimental consistiu em modificar um parâmetro por vez, executar novamente e comparar os resultados com a configuração base. No final foram realizados testes com alteração simultânea em mais de um parâmetro.

Após a etapa de ajuste e análise do controlador baseado em *DWB*, foi realizada uma nova rodada experimental com o *Regulated Pure Pursuit* (RPP), com o objetivo de comparar seu comportamento de seguimento de caminho no mesmo ambiente experimental. Foram mantidos os parâmetros de progresso e verificação de meta já utilizados no controlador anterior, de forma que a comparação entre controladores ocorresse sob critérios equivalentes.

Entre os parâmetros avaliados no RPP, foram considerados *rotate_to_heading_angular_vel*, responsável pela velocidade angular empregada na fase de alinhamento inicial com a direção da trajetória, *lookahead_dist*, que define a distância de antecipação utilizada na seleção do ponto de referência ao longo do caminho, e *use_velocity_scaled_lookahead_dist*, que permite ajustar essa distância de forma dinâmica em função da velocidade do robô.

Para o parâmetro *rotate_to_heading_angular_vel*, foram comparados os valores 1,8 rad/s e 1,0 rad/s. Para o *lookahead_dist*, o valor inicial era 0,6 m, sendo posteriormente realizados testes com 0,4 m e 0,8 m. Também foi avaliada a ativação do parâmetro *use_velocity_scaled_lookahead_dist*, alterando seu valor de *false* para *true*, com o objetivo de observar o efeito dessas variações sobre a aderência ao caminho, a suavidade do movimento e o comportamento do robô em curvas.

3.4.5 Teste de navegação completa com a ação *NavigateToPose*

Para avaliar o comportamento integrado da navegação autônoma, utilizou-se a ação *NavigateToPose*, disponibilizada pelo Nav2 para o envio de metas de navegação. Ao receber uma pose objetivo, o sistema inicia a navegação de forma integrada, coordenando a geração do caminho global e o seguimento da trajetória até a meta. Nesse processo, o módulo *BT Navigator* organiza internamente as etapas de planejamento e controle, permitindo executar a tarefa de navegação por meio de uma única interface.

Durante a execução, o sistema também pode atualizar a trajetória quando a lógica de navegação assim o exigir, como em situações de modificação da meta, invalidação do caminho anteriormente gerado ou de acordo com uma periodicidade previamente definida.

Dessa forma, o teste de navegação completa permitiu avaliar o desempenho do sistema em condição integrada de operação. Foram adotadas como métricas de avaliação o sucesso da execução, os tempos associados ao planejamento e ao controle e o número de replanejamentos, este último utilizado como um indicador complementar.

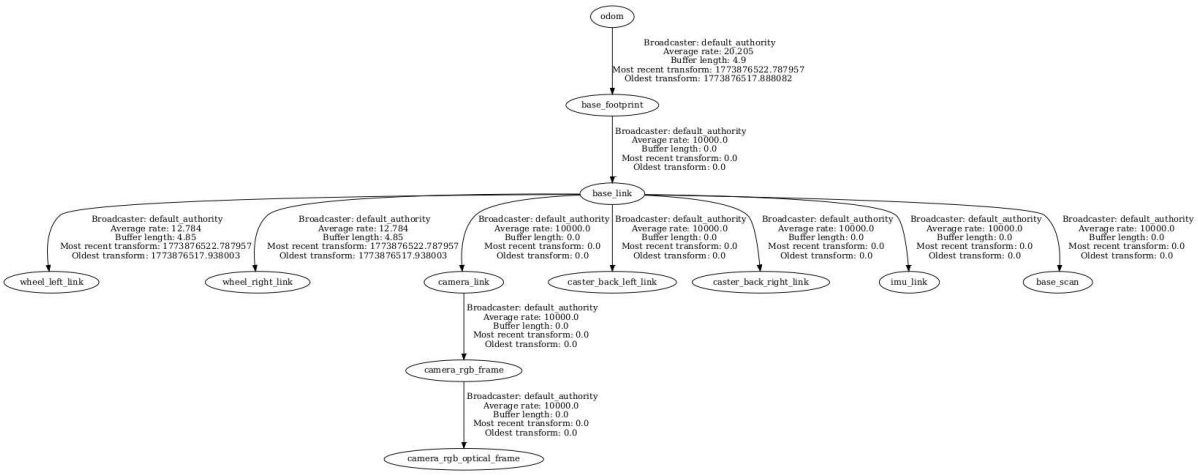
4 Resultados

Este capítulo reúne os resultados obtidos na avaliação experimental da navegação autônoma implementada com o *framework Navigation2* no robô *TurtleBot3 Waffle Pi*. A estrutura acompanha o procedimento experimental descrito no Capítulo 3. Inicialmente, são discutidos os resultados da validação da plataforma e do mapeamento. Em seguida, são apresentados os efeitos da sintonia do *global costmap*. Por fim, são analisados os resultados comparativos entre os planejadores globais e entre os controladores locais avaliados.

4.1 Validação da plataforma e do mapeamento

A inspeção da árvore de transformações do sistema mostrou coerência entre os principais referenciais publicados, incluindo *map*, *odom*, *base_footprint*, *base_link* e o referencial do sensor LiDAR. A estrutura hierárquica observada é apresentada na Figura 19, evidenciando compatibilidade entre os quadros necessários às etapas de mapeamento, localização e navegação.

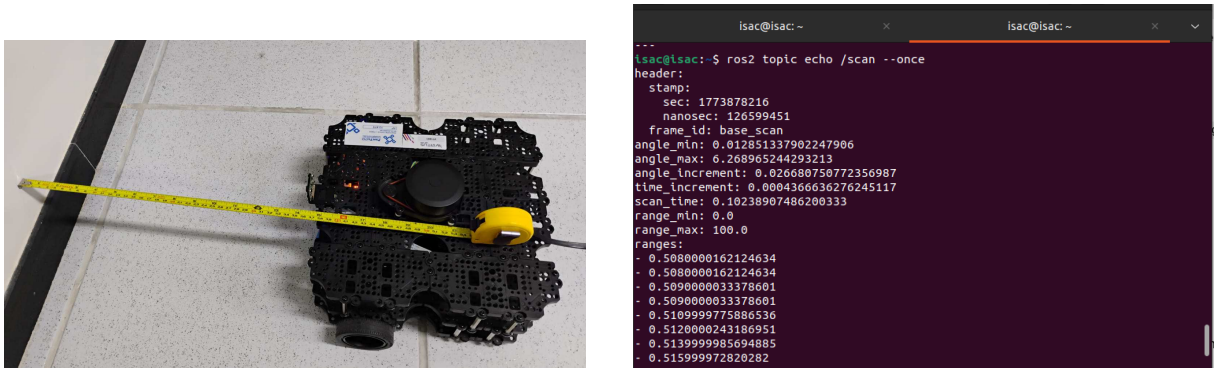
Figura 19 – Estrutura hierárquica da árvore TF do sistema.



Fonte: Autoria própria.

A validação do sensor LiDAR indicou boa correspondência entre as distâncias medidas e os valores de referência adotados no laboratório. Para posicionamentos do robô a 0,50 m, 1,00 m e 1,50 m em relação à parede, observaram-se leituras aproximadas de 0,508 m, 1,003 m e 1,508 m, respectivamente. A Figura 20(a) mostra o robô posicionado a 0,50 m da parede, enquanto a Figura 20(b) apresenta a leitura do LiDAR publicada no tópico */scan.*. Esses resultados indicam funcionamento adequado do sensor para as etapas posteriores de mapeamento e navegação.

Figura 20 – Distância medida e leitura do sensor LiDAR.



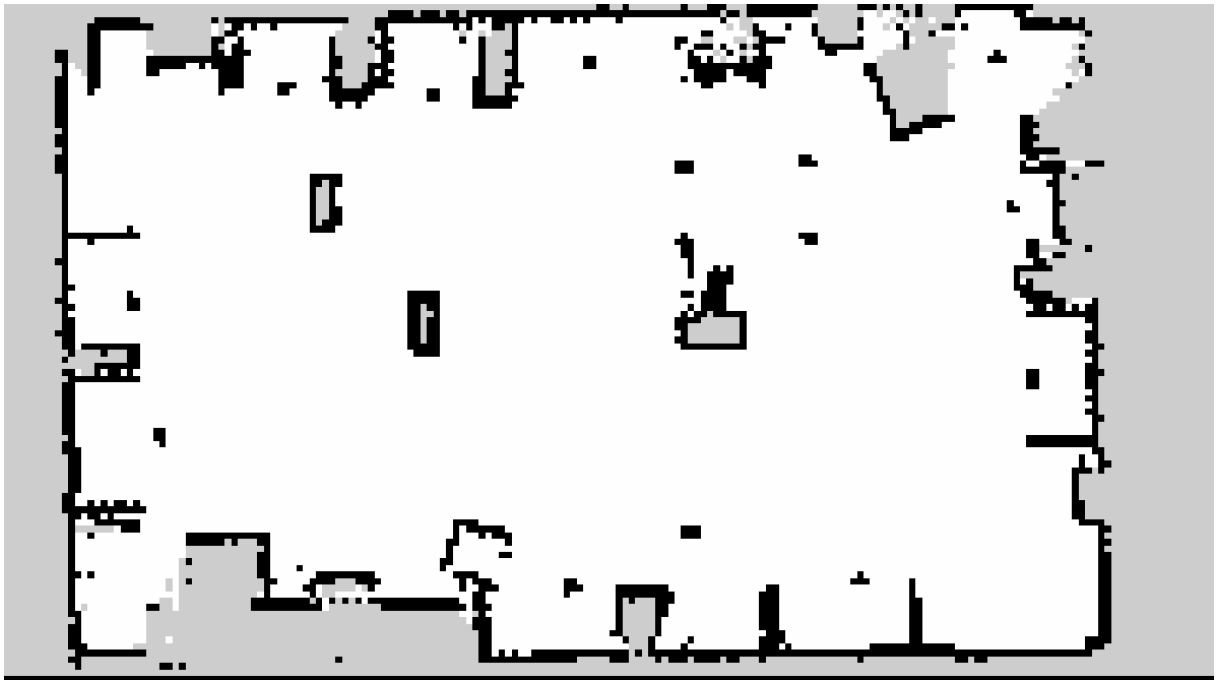
((a)) Distância medida entre LiDAR e parede. ((b)) Leitura do sensor LiDAR.

Fonte: Autoria própria.

Na validação da odometria, observou o deslocamento executado pelo robô e o deslocamento estimado pelo sistema. Nos testes realizados com percursos de 1 m e 2 m, o erro permaneceu inferior a 5%, considerado aceitável para a aplicação proposta.

Concluída a validação inicial da plataforma, foi gerado o mapa de grade de ocupação do laboratório, apresentado na Figura 21.

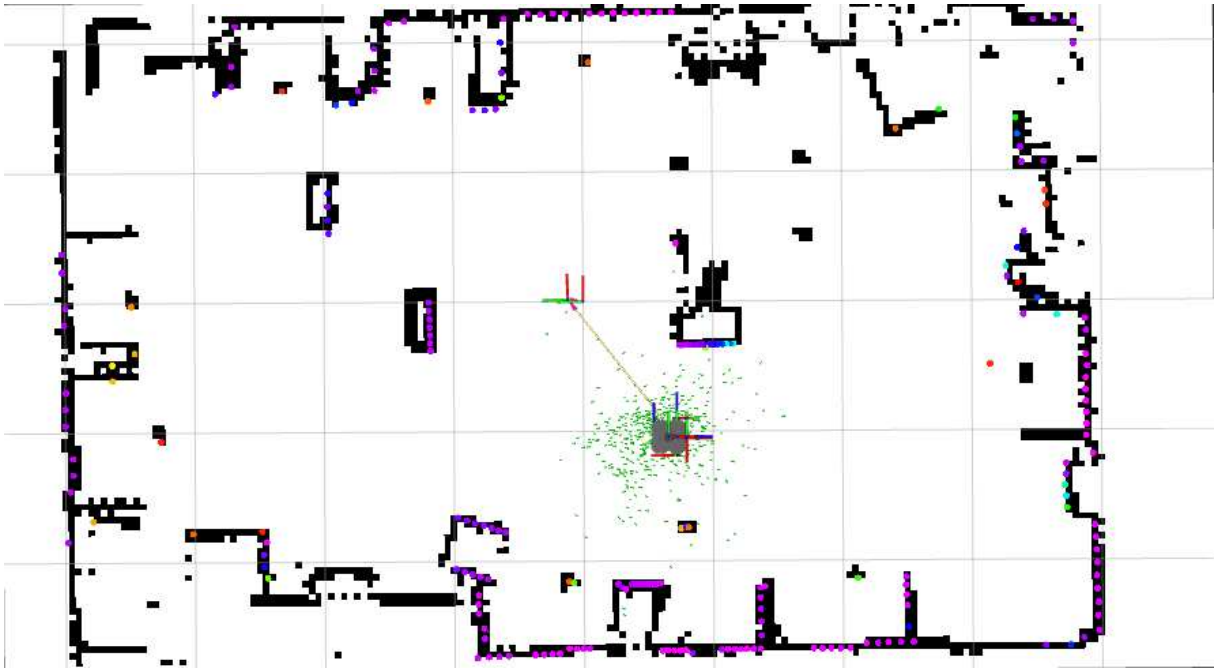
Figura 21 – Mapa de grade de ocupação do laboratório.



Fonte: Autoria própria.

A sobreposição entre o mapa de ocupação e o *LaserScan*, apresentada na Figura 22, mostrou alinhamento entre os pontos coloridos que representam as leituras do laser e as paredes indicadas no mapa.

Figura 22 – Validação do mapa por sobreposição entre o mapa de ocupação e o *LaserScan*.



Fonte: Autoria própria.

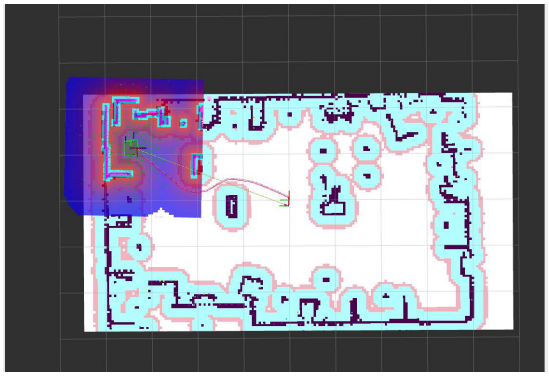
De forma geral, os resultados dessa etapa mostraram que o sensor LiDAR, a odometria, a estrutura TF e o mapa gerado apresentaram comportamento compatível com os requisitos do sistema, fornecendo uma base consistente para os experimentos de localização e navegação realizados na sequência.

4.2 Sintonia do global *costmap*

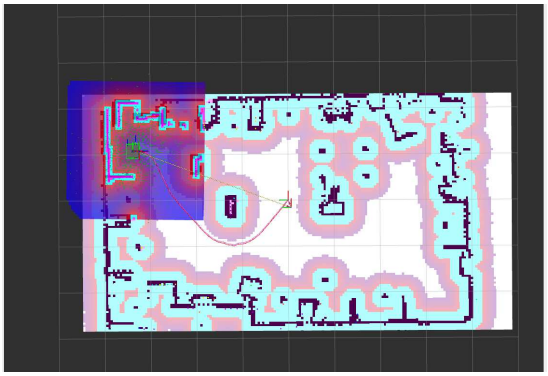
Apresenta-se na Figura 23 o efeito do parâmetro *inflation_radius* sobre o mapa de custo utilizado no planejamento global. Nas imagens, as regiões em preto correspondem aos obstáculos letais, enquanto as áreas em azul claro representam os custos letais inscritos, determinados a partir das dimensões do robô. O gradiente de cores entre vermelho e azul indica o campo de custo potencial gerado pela camada de inflação, ao passo que as regiões livres são representadas em branco.

Observou-se que o aumento do parâmetro *inflation_radius* ampliou a região de influência dos obstáculos no mapa de custo, resultando em caminhos mais conservadores e, neste caso, maiores. Em contrapartida, o valor de *0.3* produziu caminhos mais curtos, porém passando mais próximos dos obstáculos. Já valores intermediários, como *0.5* e *0.7*, resultaram em trajetórias com maior margem de segurança, preservando a viabilidade do planejamento.

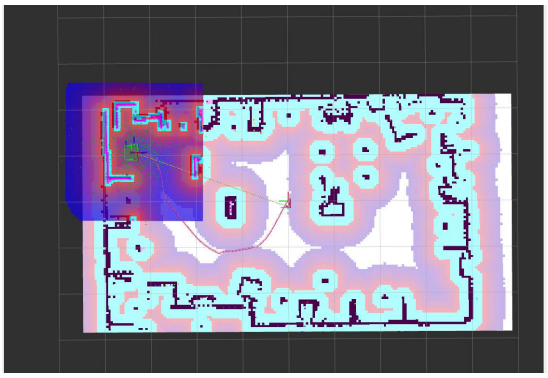
De forma geral, os resultados indicam que a escolha do parâmetro *inflation_radius* deve considerar tanto a geometria do ambiente quanto o comportamento desejado para



((a)) *inflation_radius = 0.3*



((b)) *inflation_radius = 0.5*



((c)) *inflation_radius = 0.7*

Figura 23 – Influência do parâmetro *inflation_radius* no mapa de custo e no caminho global planejado.

Fonte: Autoria própria.

a navegação. Em regiões como corredores, a adoção de valores mais elevados tende a ser vantajosa, pois amplia a zona de influência dos obstáculos no mapa de custo e induz o robô a planejar caminhos mais centralizadas.

Com base nessa análise, foi escolhido um valor intermediário que oferecesse maior segurança para a execução da navegação, sem aumentar excessivamente o comprimento do caminho planejado. Dessa forma, a configuração com *inflation_radius = 0.5* foi adotada nas etapas seguintes de avaliação dos planejadores globais.

4.3 Avaliação dos planejadores

Na Tabela 4, observa-se que todas as configurações avaliadas apresentaram taxa média de sucesso de 100,0%, indicando que tanto o *NavFn* quanto o *Smac 2D Planner* foram capazes de gerar caminhos viáveis em todas as execuções. Em relação ao tempo de planejamento, as menores médias foram obtidas com o *NavFn*, enquanto o *Smac 2D Planner* apresentou os menores comprimentos médios de caminho nas duas posições analisadas.

Tabela 4 – Valores médios dos experimentos com os planners navfn e smac.

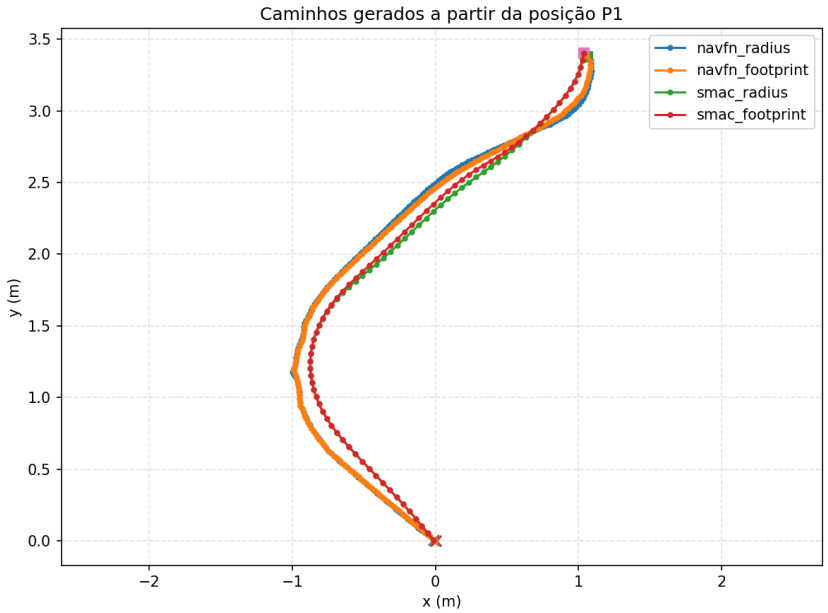
Posição	planner_forma	Sucesso médio (%)	Tempo médio de planejamento (s)	Comprimento médio do caminho (m)
P1	navfn_footprint	100,0	0,000713	4,766019
P1	navfn_radius	100,0	0,000422	4,804538
P1	smac_footprint	100,0	0,003113	4,510914
P1	smac_radius	100,0	0,002633	4,504662
P2	navfn_footprint	100,0	0,000848	3,494339
P2	navfn_radius	100,0	0,000816	3,541655
P2	smac_footprint	100,0	0,002973	3,232000
P2	smac_radius	100,0	0,002861	3,215657

Fonte: Elaborado pelo autor.

Esse comportamento pode ser visualmente confirmado nas Figura 24 e Figura 25, nas quais se nota que os caminhos geradas pelo *Smac 2D Planner* descrevem curvas mais internas no contorno do obstáculo, reduzindo o comprimento total do caminho entre a posição inicial e o objetivo. Em contraste, os produzidas pelo *NavFn* que tendem a ser mais conservadoras e a se afastar mais do obstáculo, o que contribui para o aumento do percurso total. Assim, embora o *NavFn* tenha apresentado menor custo computacional no planejamento, o *Smac 2D Planner* gerou rotas geometricamente mais eficientes no caso analisado.

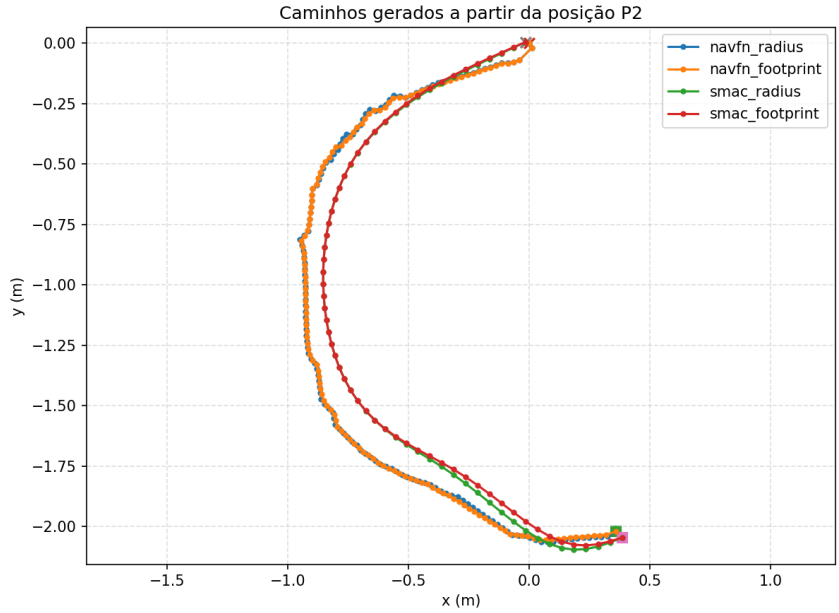
As diferenças entre as modelagens *radius* e *footprint* dentro de um mesmo planejador foram discretas, com caminhos bastante próximos entre si. Isso indica que, nas condições experimentais avaliadas, a forma de representação geométrica do robô exerceu pouca influência sobre o caminho global.

Figura 24 – Caminhos gerados a partir da posição P1.



Fonte: Autoria própria.

Figura 25 – Caminhos gerados a partir da posição P2.



Fonte: Autoria própria.

Considerando o menor comprimento dos caminhos gerados e a manutenção da taxa de sucesso em 100%, o *Smac 2D Planner* foi selecionado para as etapas posteriores de avaliação dos controladores.

4.4 Avaliação dos controladores

Os resultados apresentados nesta seção referem-se aos testes realizados com a ação */follow_path*, de modo a isolar o comportamento do *controller server* durante o seguimento do caminho. Inicialmente, são discutidos os ajustes realizados no *DWBLocalPlanner*. Em seguida, apresentam-se os resultados obtidos com o *Regulated Pure Pursuit* (RPP), buscando comparar o comportamento dos dois controladores sob o mesmo caminho de referência e nas mesmas condições experimentais.

4.4.1 Ajuste do DWBLocalPlanner

No caso do *DWBLocalPlanner*, as trajetórias observadas na Figura 26 permaneceram relativamente próximas entre si e também próximas ao caminho de referência, indicando comportamento globalmente consistente entre as configurações testadas. Entretanto, durante os ensaios com a configuração padrão (*dwb_default*), foi observado um comportamento recorrente em trechos de curva próximos a obstáculos: o robô interrompia o movimento diante do obstáculo e retomava o seguimento em seguida; em algumas execuções, esse comportamento evoluiu para falha.

Entre os parâmetros avaliados, foi observado que a redução de *sim_time* de 2,0 para 1,5 (representada no gráfico por *dwb_sim1.5*) contribuiu para diminuir a hesitação do controlador em regiões de curva, tornando a resposta local mais imediata. De forma semelhante, a redução de *PathAlign.scale* de 32 para 24 também contrinuiu para que o robô executasse a trajetória de maneira contínua.

Por outro lado, as alterações em *PathDist.scale* entre 24 e 32 não produziram mudanças tão evidentes quanto as observadas com *sim_time* e *PathAlign.scale*, mantendo trajetórias bastante próximas à configuração base. De forma geral, os resultados indicam que, no cenário avaliado, os ajustes mais relevantes para o comportamento do DWB estiveram associados ao horizonte de simulação e ao peso de alinhamento com a trajetória.

Ao relacionar esses resultados com a Equação 2.23 apresentada na fundamentação, verifica-se que o parâmetro *sim_time* não se associa diretamente a um dos pesos da função objetivo, mas ao horizonte temporal adotado na simulação das trajetórias candidatas antes de sua avaliação. No DWB a influência análoga ao termo $heading(v,\omega)$ não é representada por um único coeficiente, como ocorre com o peso α na formulação clássica do DWA, mas pela combinação de diferentes parâmetros de ponderação, entre eles o *PathAlign.scale* e o *PathDist.scale*.

Em termos práticos, esses parâmetros regulam a relevância do alinhamento da trajetória com a direção desejada e da manutenção do robô em proximidade adequada ao caminho planejado. Assim, a redução de *PathAlign.scale* diminuiu a influência do critério de alinhamento na escolha da trajetória, enquanto a redução de *sim_time* encurtou o horizonte de previsão e tornou a atuação do controlador mais reativa.

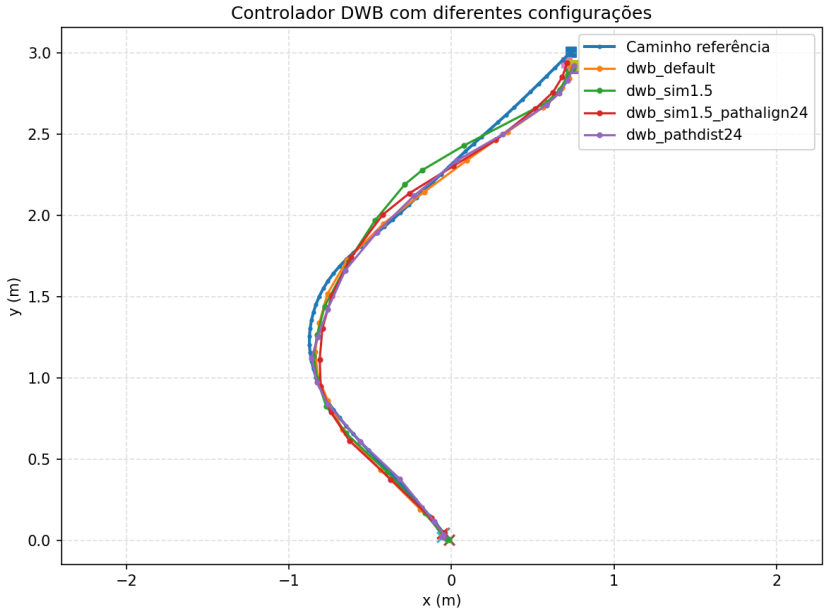
4.4.2 Ajuste do Regulated Pure Pursuit

Os resultados do *Regulated Pure Pursuit* (RPP), apresentados na Figura 27, mostraram comportamento mais sensível às alterações de parâmetro, sobretudo na aproximação inicial ao caminho e na região da curva. A configuração *rpp_default*, associada a *rotate_to_heading_angular_vel = 1,8* rad/s, *lookahead_dist = 0,6* m e *use_velocity_scaled _lookahead_dist = false*, apresentou taxa de sucesso de 100%, porém com maior desvio em relação à rota planejada e com rotação inicial mais agressiva.

Quando o parâmetro *rotate_to_heading_angular_vel* foi reduzido de 1,8 para 1,0 rad/s, observou-se uma entrada mais suave no caminho e melhora no seguimento da trajetória. Esse comportamento foi perceptível na prática, ao deixar de ter rotação agressiva, e bem ilustrado no gráfico.

Em relação ao parâmetro *lookahead_dist*, mantendo*rotate_to_heading_angular _vel* em 1,0, foi verificado que a redução de 0,6 para 0,4 m favoreceu o seguimento do caminho. Mesmo quando o robô iniciava o movimento desalinhado, observou-se rápida

Figura 26 – Trajetórias executadas pelo controlador DWB com diferentes configurações.



Fonte: Autoria própria.

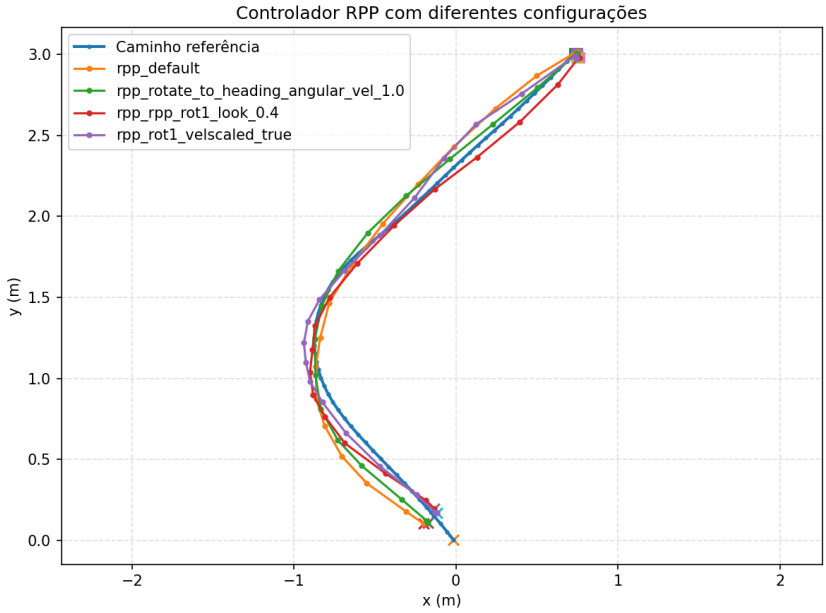
convergência para a trajetória de referência. Já a configuração com *lookahead_dist = 0,8* apresentou falha na execução, indicando que, para o cenário analisado, o aumento da distância de antecipação prejudicou o desempenho do controlador.

Relacionando esse resultado com a Equação 2.24, observa-se que o parâmetro *lookahead_dist* corresponde à distância de antecipação L_d utilizada pelo Pure Pursuit e mantida pelo RPP. Assim, ao reduzir esse valor de 0,6 para 0,4 m, o controlador passou a reagir mais rapidamente aos desvios em relação ao caminho, o que ajuda a explicar a convergência mais rápida observada nos testes. Por outro lado, o aumento de *lookahead_dist* para 0,8 m elevou a distância de antecipação considerada no seguimento, tornando a resposta menos imediata e prejudicando o desempenho do controlador no cenário analisado.

Também se observou melhora no seguimento quando o parâmetro *use_velocity_scaled_lookahead_dist* foi ativado, alterando seu valor de *false* para *true*. Nessa condição, o controlador passou a ajustar dinamicamente a distância de antecipação em função da velocidade do robô, observou-se que foi a configuração em que o robô realizou a curva mais aberta, se afastando mais do obstaculo e do caminho planejado, ao mesmo tempo que foi o que mais se aproximou do caminho planejado na reta final.

Em síntese, os resultados mostram que o RPP apresentou melhor desempenho com a redução de *rotate_to_heading_angular_vel* para 1,0 rad/s, enquanto a configuração com *lookahead_dist* igual a 0,4 favoreceu o seguimento do caminho. Assim, os ajustes mais relevantes no cenário avaliado estiveram associados à suavização da rotação inicial e à redução da distância de antecipação.

Figura 27 – Trajetórias executadas pelo controlador RPP com diferentes configurações.



Fonte: Autoria própria.

4.4.3 Discussão dos Resultados

Considerando os testes realizados com a ação */follow_path*, observou-se que o *DWBLocalPlanner* apresentou comportamento globalmente consistente, porém com episódios de hesitação em curvas próximas a obstáculos, o que afetou sua taxa de sucesso. Já o *Regulated Pure Pursuit* apresentou maior sensibilidade aos parâmetros avaliados, mas também maior potencial de ajuste, permitindo alcançar melhor compromisso entre suavidade de movimento e aderência ao caminho.

A Tabela 5 apresenta o resumo dos testes realizados com diferentes configurações do controlador DWB. O tempo médio de execução foi calculado considerando apenas as execuções com sucesso.

Tabela 5 – Resumo dos testes com diferentes configurações do controlador DWB.

Configuração	**Taxa de sucesso (%)**	**Tempo médio (s)**
`dwb_default`	66,67	26,25
`dwb_sim15`	66,67	28,40
`dwb_sim15_pathalign24`	75,00	27,25
`dwb_pathdist24`	66,67	31,06

Fonte: Elaborado pelo autor.

A Tabela 6 apresenta o resumo dos testes realizados com diferentes configurações do controlador RPP. De forma geral, todas as configurações avaliadas obtiveram sucesso nos ensaios realizados, de modo que a comparação entre elas se concentrou principalmente no tempo médio de execução.

Tabela 6 – Resumo dos testes com diferentes configurações do controlador RPP.

Configuração	**Taxa de sucesso (%)**	**Tempo médio (s)**
`rpp_default`	100	23,07
`rpp_rotate_to_heading_angular_vel1`	100	19,84
`rpp_rpp_rot1_look04`	100	19,91
`rpp_rot1_velscaled`	100	20,20

Fonte: Elaborado pelo autor.

Os resultados resumidos na Tabela 5 e na Tabela 6 reforçam a tendência observada nas trajetórias apresentadas anteriormente. Enquanto o DWB apresentou taxas de sucesso entre 66,67% e 75,00%, o RPP obteve 100% de sucesso em todas as configurações avaliadas, além de menores tempos médios de execução. Esses resultados indicam que, no cenário analisado, o RPP apresentou comportamento mais robusto e estável para o seguimento do caminho gerado pelo planejador global.

Cabe observar que a análise dessas configurações não deve ser feita apenas com base no tempo médio de execução. No caso do DWB, por exemplo, esse tempo foi calculado considerando apenas as execuções com sucesso, de modo que sua interpretação deve ser feita em conjunto com a taxa de sucesso e com o comportamento observado na execução da tarefa.

A escolha do controlador mais adequado considerou simultaneamente a capacidade de concluir a tarefa, a continuidade do movimento e o tempo necessário para executar o percurso. Dessa forma, os resultados indicam que o RPP apresentou o melhor desempenho global no cenário avaliado.

Por fim, realizou-se um teste adicional para avaliar o comportamento completo da arquitetura do Nav2 por meio da ação `NavigateToPose`, utilizando o planejador global *Smac 2D Planner* combinado, separadamente, com os controladores *RPP* e *DWB*. Esse ensaio teve como objetivo complementar a análise isolada dos controladores, permitindo observar o desempenho do sistema em uma condição mais próxima da operação real, na qual planejamento global, seguimento da trajetória, replanejamentos e gerenciamento da navegação atuam de forma integrada.

Os resultados apresentados na Tabela 7 mostram que ambas as configurações foram capazes de concluir a navegação com sucesso, o que confirma a viabilidade funcional das duas abordagens no ambiente experimental adotado.

Tabela 7 – Comparação do comportamento completo da arquitetura do Nav2 com a ação `NavigateToPose`.

Planner	**Controlador**	**Sucesso**	**Tempo de planejamento (s)**	**Tempo de execução (s)**	**Nº de replanejamentos**
Smac2D	RPP	Sim	0,0066	21,16	19
Smac2D	DWB	Sim	0,0059	30,06	24

Fonte: Elaborado pelo autor.

No cenário avaliado, a combinação *Smac2D + RPP* apresentou o melhor desempenho global. As duas configurações conseguiram concluir a navegação, mas o *RPP* executou a tarefa em menos tempo, com redução de cerca de 30% em relação ao *DWB*. De forma complementar, também foi observado menor número de replanejamentos durante a execução.

Como complemento aos resultados apresentados, disponibiliza-se um vídeo referente a esta etapa experimental[1].

[1] Vídeo do experimento de navegação: <https://drive.google.com/file/d/1bnBk-p7_Az88eIxHeXTcgP6iWOyWVHxr/view?usp=drive_link>.

5 Conclusão

Este trabalho apresentou uma avaliação experimental do *framework Navigation2* (Nav2) aplicado ao robô móvel *TurtleBot3 Waffle Pi* em ambiente laboratorial, com foco na comparação entre planejadores globais e controladores locais voltados à navegação autônoma. Os resultados permitiram verificar que diferentes configurações do Nav2 podem ser empregadas com sucesso no cenário estudado, embora com desempenhos distintos em termos de tempo de planejamento, comprimento de caminho e comportamento durante o seguimento da trajetória.

Também se verificou que parâmetros associados ao *costmap* e à representação geométrica do robô influenciam o desempenho da navegação. O *inflation_radius*, em especial, mostrou-se relevante para o ajuste das margens de segurança em torno dos obstáculos, afetando a viabilidade dos caminhos gerados. Já em relação à geometria da base móvel, observou-se que, no cenário analisado, a utilização de *footprint* poligonal ou aproximação circular por *radius* não resultou em diferenças significativas no traçado global das rotas. Ainda assim, tais parâmetros foram importantes para o ajuste fino do sistema e para o controle local.

Uma limitação identificada está relacionada ao sistema de percepção exteroceptiva, uma vez que o uso exclusivo do LiDAR dificulta a detecção de obstáculos pequenos e baixos, como cabos no chão, bem como de obstáculos elevados situados fora de seu plano de varredura. Nessas situações, a percepção incompleta do ambiente pode comprometer o planejamento da trajetória e a robustez da navegação. Nesse sentido, a utilização de câmeras de profundidade em conjunto com o LiDAR apresenta-se como uma alternativa promissora para ampliar a capacidade de percepção do sistema.

Devido às limitações de tempo, não foi possível avaliar outros métodos disponíveis no Nav2, como o planejador *Smac Lattice* e o controlador preditivo *MPPI*. Como trabalho futuro, propõe-se a análise desses métodos em ambiente laboratorial, bem como sua comparação com as configurações implementadas neste trabalho. Também se mostra relevante avançar na integração completa da navegação autônoma ao restante da célula robótica, de modo que o robô móvel possa executar de forma coordenada as tarefas de transporte sem depender de intervenção humana no deslocamento.

Além disso, este trabalho contribui ao estabelecer uma infraestrutura funcional de navegação autônoma no ambiente laboratorial, o que pode viabilizar o desenvolvimento de pesquisas futuras no laboratório. A partir da solução implementada e validada, tornam-se possíveis novos estudos voltados à integração com outros sistemas robóticos, à ampliação das capacidades de percepção e à execução de tarefas mais complexas, contribuindo para

a continuidade e a evolução das aplicações desenvolvidas nesse contexto.

Referências

AL-BATATI, Y.; HUDA, S.; MOHAMMADI, R. Ros2-based distributed architectures for autonomous robotic systems: A review. *Robotics and Autonomous Systems*, Elsevier, v. 180, p. 104782, 2025.

BATISTA, M. O. *Relatório de estágio integrado: Laboratório de sistemas embarcados e computação pervasiva (Embedded)*. Campina Grande, PB, 2025. Disponível em: <https://dspace.sti.ufcg.edu.br/handle/riufcg/44281>.

BRAND, T. *Time of Flight System for Distance Measurement and Object Detection*. 2021. Disponível em: <https://www.analog.com/en/resources/technical-articles/tof-system-for-distance-measurement-and-object-detection.html>.

CARDOSO, S. A. *Controle orientado a eventos de uma fábrica inteligente com base em realimentação visual e uma estrutura de programação distribuída*. Monografia (Trabalho de Conclusão de Curso (Bacharelado em Engenharia Elétrica)), Campina Grande, PB, 2024. Disponível em: <https://dspace.sti.ufcg.edu.br/handle/riufcg/38973>.

CHEN, Y. et al. Recent advances in multi-robot systems: A survey. *Robotics and Autonomous Systems*, Elsevier, v. 174, p. 104616, 2024.

COLLEDANCHISE, M.; ÖGREN, P. *Behavior Trees in Robotics and AI: An Introduction*. Boca Raton: CRC Press, 2018.

CORKE, P. *Robotics, Vision and Control: Fundamental Algorithms in MATLAB*. Berlin, Heidelberg: Springer, 2011. v. 73. (Springer Tracts in Advanced Robotics, v. 73). ISBN 9783642201431.

COULTER, R. C. *Implementation of the Pure Pursuit Path Tracking Algorithm*. Pittsburgh, PA, 1992.

DURRANT-WHYTE, H.; BAILEY, T. Simultaneous localization and mapping: Part i. *IEEE Robotics & Automation Magazine*, v. 13, n. 2, p. 99–110, 2006.

ELFES, A. Occupancy grids: A probabilistic framework for robot perception and navigation. *Computer*, v. 22, n. 6, p. 46–57, 1989.

FOX, D.; BURGARD, W.; THRUN, S. The dynamic window approach to collision avoidance. *IEEE Robotics & Automation Magazine*, v. 4, n. 1, p. 23–33, 1997.

HART, P. E.; NILSSON, N. J.; RAPHAEL, B. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, v. 4, n. 2, p. 100–107, 1968.

IOVINO, M. et al. Comparison between behavior trees and finite state machines. *IEEE Transactions on Automation Science and Engineering*, v. 22, p. 21098–21117, 2025.

LIU, Y. et al. Path planning techniques for autonomous mobile robots: A review. *Robotics and Autonomous Systems*, Elsevier, v. 166, p. 104425, 2023.

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

MACENSKI, S. et al. The marathon 2: A navigation system. *IEEE Robotics and Automation Magazine*, v. 27, n. 2, p. 23–35, 2020.

MACENSKI, S. et al. From the desks of ros maintainers: A survey of modern & capable mobile robotics algorithms in the robot operating system 2. *Robotics and Autonomous Systems*, v. 168, p. 104493, 2023.

MACENSKI, S. et al. Regulated pure pursuit for robot path tracking. *Autonomous Robots*, v. 47, n. 6, p. 685–694, 2023.

PARK, J.; LEE, S.; OH, Y. Turtlebot3: Open-source robot platform for education and research. *IEEE Robotics and Automation Magazine*, v. 26, n. 1, p. 14–20, 2019.

QUIGLEY, M. et al. Ros: an open-source robot operating system. In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA) Workshop on Open Source Software*. Kobe, Japan: [s.n.], 2009.

RAJ, P. et al. Motion control strategies for autonomous mobile robots: A review. *Robotics and Autonomous Systems*, Elsevier, v. 173, p. 104589, 2024.

ROBOTIS. *TurtleBot3 Features*. 2024. Acesso em: 9 mar. 2026. Disponível em: <https://emanual.robotis.com/docs/en/platform/turtlebot3/features/>.

ROBOTIS. *OpenCR1.0*. 2026. Disponível em: <https://emanual.robotis.com/docs/en/platform/turtlebot3/appendix_opencr1_0/>.

SÁNCHEZ-IBÁÑEZ, J. R.; PULGAR, C. J. Pérez-del; GARCÍA-CEREZO, A. Path planning for autonomous mobile robots: A review. *Sensors*, MDPI, v. 21, n. 23, p. 7898, 2021.

SCHENA, F. *Development of an Automated Benchmark for the Analysis of Nav2 Controllers*. Dissertação (Mestrado) — Politecnico di Torino, 2024.

SICILIANO, B. et al. *Robotics: Modelling, Planning and Control*. London: Springer, 2009. ISBN 9781846286414.

SIEGWART, R.; NOURBAKHSH, I. R.; SCARAMUZZA, D. *Introduction to Autonomous Mobile Robots*. 2. ed. Cambridge, MA: MIT Press, 2011.

SZREK, J. et al. Accuracy evaluation of selected mobile inspection robot localization techniques. *Sensors*, v. 21, n. 1, p. 141, 2020.

THRUN, S.; BURGARD, W.; FOX, D. *Probabilistic Robotics*. Cambridge, MA: MIT Press, 2005.

ULLAH, I. et al. Mobile robot localization: Current challenges and future prospective. *Computer Science Review*, v. 53, p. 100651, 2024.

ZHAO, X.; CHIDAMBARESWARAN, T. Autonomous mobile robots in manufacturing operations. In: *Proceedings of the IEEE International Conference on Automation Science and Engineering (CASE)*. Auckland, New Zealand: IEEE, 2023.