

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL  
ESCOLA DE ENGENHARIA  
ENG. DE CONTROLE E AUTOMAÇÃO

**HENRIQUE SCHARLAU COELHO - 243627**

**MAPEAMENTO DE AMBIENTE  
PARA NAVEGAÇÃO DE UM ROBÔ  
MÓVEL**

Porto Alegre  
2024

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL  
ESCOLA DE ENGENHARIA  
ENG. DE CONTROLE E AUTOMAÇÃO

**HENRIQUE SCHARLAU COELHO - 243627**

**MAPEAMENTO DE AMBIENTE  
PARA NAVEGAÇÃO DE UM ROBÔ  
MÓVEL**

Trabalho de Conclusão de Curso (TCC-CCA)  
apresentado à COMGRAD-CCA da Universi-  
dade Federal do Rio Grande do Sul como parte  
dos requisitos para a obtenção do título de *Ba-  
charel em Eng. de Controle e Automação* .

ORIENTADOR:

Prof. Dr. Walter Fetter Lages

Porto Alegre  
2024

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL  
ESCOLA DE ENGENHARIA  
ENG. DE CONTROLE E AUTOMAÇÃO

**HENRIQUE SCHARLAU COELHO - 243627**

**MAPEAMENTO DE AMBIENTE  
PARA NAVEGAÇÃO DE UM ROBÔ  
MÓVEL**

Este Trabalho de Conclusão de Curso foi julgado adequado para a obtenção dos créditos da Disciplina de TCC do curso *Eng. de Controle e Automação* e aprovado em sua forma final pelo Orientador e pela Banca Examinadora.

Orientador: \_\_\_\_\_

Prof. Dr. Walter Fetter Lages, UFRGS

Doutor pelo Instituto Tecnológico de Aeronáutica – São José dos Campos, Brasil

Banca Examinadora:

Prof. Dr. Walter Fetter Lages, UFRGS

Doutor pelo Instituto Tecnológico de Aeronáutica – São José dos Campos, Brasil

Prof. Dr. Rafael Antônio Comparsi Laranja, UFRGS

Doutor pela Universidade Federal do Rio Grande do Sul – Porto Alegre, Brasil

Prof. Dr. Eduardo André Perondi, UFRGS

Doutor pela Universidade Federal de Santa Catarina – Florianópolis, Brasil

---

Alceu Heinke Frigeri  
Coordenador de Curso  
Eng. de Controle e Automação

Porto Alegre, Fevereiro 2024

## RESUMO

O presente trabalho apresenta um sistema de navegação autônoma, aplicado no robô móvel Twil. Aproveitando o desenvolvimento de trabalhos anteriores, este sistema foi estendido, com adição do mapeamento do espaço onde o robô se movimenta. O mapa criado é utilizado para aprimorar a estimativa de posição do robô, e para replanejar trajetos em tempo real, evitando colisões. Isto foi realizado com auxílio de ferramentas do ROS 2, como a biblioteca de navegação autônoma *Navigation2* e pacotes que incorporam técnicas de mapeamento e localização simultâneas. O desempenho dos sistemas desenvolvidos foi avaliado através de testes em um ambiente simulado, com gráficos medindo a estimativa de posição, comparação do mapa obtido com o mapa real, e análise qualitativa do planejador de trajetórias.

**Palavras-chave:** Robótica móvel, navegação autônoma, localização e mapeamento simultâneos.

## ABSTRACT

This paper presents an autonomous navigation system, applied to the Twil mobile robot. Taking advantage of the development of previous works, this system was extended, with the addition of mapping of the space in which the robot navigates. The generated map is used to improve the robot's pose estimation, and to replan trajectories in real time, avoiding collisions. This was done with the aid of tools made for ROS 2, such as the *Navigation2* navigation framework and packages that perform simultaneous mapping and localization. The performance of the developed systems was evaluated through tests in a simulated environment, with graphs measuring the estimated position, comparison of the obtained map with the real map, and qualitative analysis of the path planner.

**Keywords:** Mobile robotics, autonomous navigation, simultaneous localization and mapping.

## **LISTA DE ILUSTRAÇÕES**

1	Mercado global de robôs autônomos de 2016 a 2021, com projeção até 2028.	8
2	Modelo do robô utilizado.	9
3	Relação entre os sistemas de coordenadas utilizados na localização de um robô móvel.	11
4	Modelo gráfico do problema do SLAM.	12
5	Exemplo de configuração de camadas de um mapa de custo.	15
6	Arquitetura do <i>Navigation2</i> .	16
7	Arquitetura do simplificada do sistema desenvolvido.	17
8	Planta do 1º andar do prédio Centenário da EE-UFRGS.	19
9	Ambiente Gazebo com modelo do prédio Centenário da EE-UFRGS.	19
10	Trajetórias criadas com diferentes configurações da camada de inflação	25
11	Mapeamento de obstáculos da camada <i>voxel</i>	26
12	Robô durante o teste realizado.	27
13	Resultado da odometria das rodas.	28
14	Resultado da odometria visual.	28
15	Resultado do pacote <code>rtabmap_slam</code> .	29
16	Resultado do pacote <code>slam_toolbox</code> .	30
17	Mapa construído pelo <code>rtabmap_slam</code> .	31
18	Mapa construído pelo <code>slam_toolbox</code> .	31

## **LISTA DE TABELAS**

1	Sistemas de odometria .....	21
2	Sistemas de localização.....	21

## **LISTA DE ABREVIATURAS**

<b>BT</b>	<i>Behavior Tree</i> (árvore de Comportamento)
<b>RGB-D</b>	<i>Red Green Blue - Depth</i> (vermelho verde azul - profundidade)
<b>ROS</b>	<i>Robot Operating System</i> (sistema operacional de robôs)
<b>SLAM</b>	<i>Simultaneous Localization and Mapping</i> (localização e mapeamento simultâneos)
<b>UFRGS</b>	Universidade Federal do Rio Grande do Sul

# SUMÁRIO

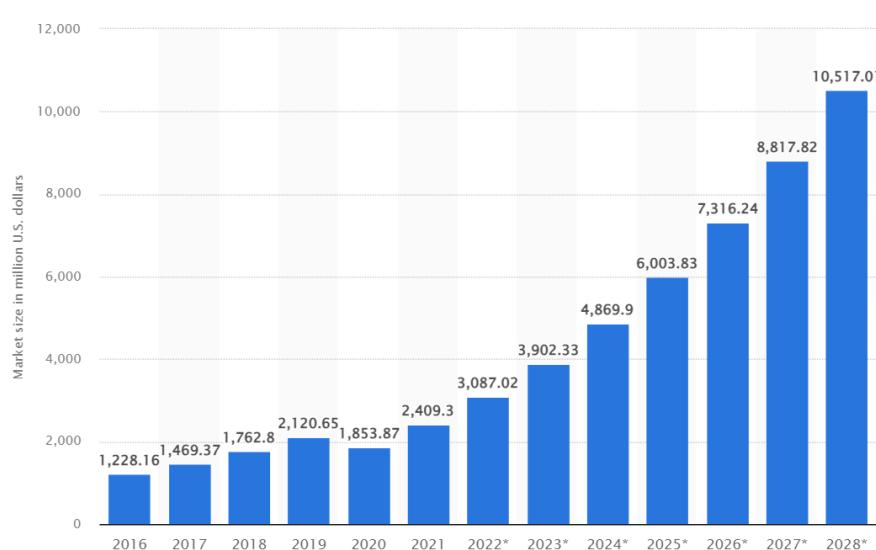
1	INTRODUÇÃO .....	8
2	REVISÃO DA LITERATURA.....	10
2.1	Robot Operating System 2 (ROS 2).....	10
2.2	Estimativa de posição de um robô móvel.....	10
2.2.1	Odometria .....	11
2.2.2	Localização.....	11
2.3	Localização e Mapeamento Simultâneo (SLAM).....	12
2.3.1	SLAM baseado em grafo de poses.....	13
2.3.2	Aplicações do SLAM no ROS .....	13
2.4	Fusão de dados de sensores .....	13
2.5	Mapas de custo .....	14
2.6	Navigation2 .....	15
3	METODOLOGIA .....	16
3.1	Configuração do robô .....	18
3.2	Ambiente de simulação.....	18
3.3	Estimativa de posição do robô.....	20
3.3.1	Odometria .....	20
3.3.2	Localização.....	21
3.4	Mapeamento .....	22
3.4.1	Mapeamento de custo.....	22
3.4.2	Mapeamento SLAM .....	23
3.5	Testes e coleta de dados.....	23
4	RESULTADOS.....	25
4.1	Mapeamento de custo.....	25
4.2	Testes de odometria, localização e mapeamento SLAM.....	26
4.2.1	Odometria .....	26
4.2.2	Localização.....	29
4.2.3	Mapeamento SLAM .....	30
5	DISCUSSÃO .....	33
6	CONCLUSÕES .....	35
	REFERÊNCIAS .....	36

# 1 INTRODUÇÃO

A robótica deve seu maior sucesso à indústria de manufatura, onde são utilizados principalmente robôs manipuladores (SIEGWART; NOURBAKHSH; SCARAMUZZA, 2011). Porém, esses robôs têm como limitação sua mobilidade, incapazes de se movimentar pela planta, restringindo suas tarefas a um espaço fixo. Um robô móvel, por outro lado, é capaz de se mover pelo seu ambiente de trabalho, aumentando a gama de tarefas que podem ser realizadas.

O mercado desta categoria de robô está em crescimento, como mostra a Figura 1. Estes robôs podem ser utilizados em ambientes internos, como hospitais, fábricas ou em centros de distribuição, como o Proteus, o primeiro robô autônomo da Amazon (AMAZON, 2022). Eles têm como desafio a navegação em espaços dinâmicos, muitas vezes compartilhados com humanos. Portanto, é necessária a capacidade de perceber seu ambiente e replanejar sua trajetória em tempo real, de modo a evitar colisões.

**Figura 1:** Mercado global de robôs autônomos de 2016 a 2021, com projeção até 2028.



Fonte: Statista (2023).

É neste contexto que se insere o projeto atual. Utilizando o robô Twil, representado na Figura 2, propõe-se um sistema de navegação autônomo com sensores que mapeiam o ambiente em conjunto com algoritmos de planejamento de trajetórias, permitindo a navegação sem colisões em ambientes previamente desconhecidos ou dinâmicos.

**Figura 2:** Modelo do robô utilizado.



Este robô já foi utilizado em trabalhos de conclusão de curso anteriores, como em Petry (2019) e Athayde (2021). Porém, devido ao avanço do campo da robótica, ferramentas utilizadas nesses trabalhos foram substituídas por versões atualizadas. É o caso do ROS 2, sucessor do ROS 1, que é uma coleção de bibliotecas e ferramentas para desenvolvimento de robôs, e do *Navigation 2*, um pacote para implementar navegação autônoma em robôs móveis, que substitui o *Navigation Stack* do ROS 1. Estas novas versões aprimoram certos aspectos, como a segurança no caso do ROS, e utilizam técnicas mais modernas, como o uso de árvores de comportamento no *Navigation 2*.

Neste trabalho, é dado seguimento ao desenvolvimento anterior no Twil, a partir da adição e configuração de novos sensores, como a câmera RGB-D Intel RealSense D435 e uma unidade de medida inercial (IMU). A câmera é responsável pela percepção do ambiente, que cria um mapa do ambiente, usado tanto na estimativa de posição do robô, quanto no planejamento de trajetórias. A IMU, por sua vez, é utilizado como um sensor auxiliar, que aprimora a odometria do robô através de dados de orientação e velocidade angular do robô.

Como o mapeamento está presente em dois sistemas, cada um com diferentes requisitos, foram criados dois mapas diferentes, um para o planejamento de trajetórias, em forma de mapa de custo, e outro para a localização do robô.

## 2 REVISÃO DA LITERATURA

Neste capítulo são apresentadas as ferramentas e os conceitos empregados na navegação autônoma de um robô móvel, com foco nas técnicas presentes neste trabalho.

### 2.1 ROBOT OPERATING SYSTEM 2 (ROS 2)

O ROS 2 é a segunda geração do Robot Operating System, um *framework* para desenvolvimento de robôs. Ele foi desenvolvido a partir do zero para atender as necessidades de robôs modernos, com suporte para customização extensiva. O seu *middleware*, *Data Distribution Service* (DDS), também é usado em sistemas de infraestrutura crítica, como aplicações militares, aeroespaciais e financeiras. Este padrão confere ao ROS segurança e suporte para comunicação em tempo real (MACENSKI; FOOTE et al., 2022).

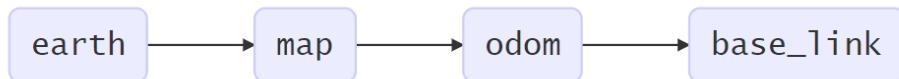
Um assunto relevante a este trabalho são os padrões de comunicação do ROS 2. Existem três tipos de comunicação no ROS 2: *topics*, *services* e *actions*. *Topics* são canais de comunicação unidirecionais, em que um nó, chamado de *publisher*, publica uma mensagem e outros nós, os *subscribers*, podem se inscrever no tópico publicado para receber essa mensagem. *Services* são um mecanismo do tipo *remote procedure call* (RPC), em que um nó faz uma chamada a outro nó, que executa uma computação e retorna um resultado, funcionando como um cliente e um servidor.

*Actions*, são utilizados para tarefas de longa duração, com possibilidade de cancelamento prematuro. O cliente começa a execução enviando uma requisição para o servidor, que retorna periodicamente o estado atual da tarefa. No término, é enviado o resultado, podendo ser sucesso ou falha. Um exemplo de uso é uma tarefa de navegação em que um *action client* envia uma requisição com um ponto de destino para um *action server* que responde com realimentação contínua da posição atual do robô e com o resultado ao finalizar a tarefa. Eles também são apropriados para utilização em árvores de comportamento.

### 2.2 ESTIMATIVA DE POSIÇÃO DE UM ROBÔ MÓVEL

A terminologia e convenção dos sistemas de coordenadas de um robô móvel utilizado neste trabalho é definido pela norma REP 105 (MEEUSSE, 2010). Essa especificação define quatro sistemas de coordenadas: `earth`, `map`, `odom` e `base_link`. Logo, a posição da base do robô, chamada de `base_link`, é definida através das transformações entre esses sistemas de coordenadas, como mostra a Figura 3.

**Figura 3:** Relação entre os sistemas de coordenadas utilizados na localização de um robô móvel.



Fonte: Meeusse (2010).

O sistema de coordenadas chamado de `earth` serve como referência global em aplicações com diversos mapas com interação de robôs entre eles. Neste trabalho, o foco é em um único robô em um único mapa, portanto, esse sistema de coordenadas não é utilizado.

Assim, a posição do robô móvel é definida através das transformadas entre os sistemas de coordenadas `map`, `odom` e `base_link`. A transformada entre `map` e `odom` é calculada pelo sistema de localização, enquanto que a transformada entre `odom` e `base_link` pelo sistema de odometria.

### 2.2.1 Odometria

A odometria publica a pose do robô em relação ao sistema de coordenadas `odom`. Essa posição pode divergir ao longo do tempo, sem limites, porém deve ser contínua. Devido à essas características, ela é útil para referência local, mas deve ser complementada com outros sistemas para referência global.

Essa odometria é geralmente obtida através de sensores incrementais, que estimam a posição e orientação do robô através da integração de dados de velocidade ou aceleração e, portanto, acumulam erros ao longo da distância percorrida, como é o caso de encoders das rodas.

Outros exemplos de fontes de odometria são Unidades de Medição Inercial (IMUs), que contém acelerômetros e giroscópios, e odometria visual, que utilizam sensores ópticos para estimar a velocidade do robô através da diferença de dados sucessivos.

### 2.2.2 Localização

O sistema de localização é responsável por publicar a transformada entre os sistemas de coordenadas `map` e `odom`, que serve como um ajuste do erro da odometria. A estimativa de posição do robô em relação ao `map` não deve divergir ao longo do tempo significativamente, porém, essa coordenada não é contínua e a posição do robô pode mudar de forma abrupta em relação a ela. Portanto, essa estimativa não precisa ser publicada a uma frequência tão alta quanto a odometria.

Logo, sensores absolutos são ideais para esses sistemas, já que não sofrem com erros de acumulação. Porém, sensores desse tipo geralmente exigem processamento sofisticado, ocasionando um longo tempo entre estimativas. Percebe-se que essas duas características são semelhantes ao requisitos da publicação da transformada entre `map` e `odom`.

Uma técnica comumente usada é a comparação de um mapa, obtido previamente, com dados de sensores ópticos. A localização por Monte Carlo é uma das aplicações mais

comuns desse sistema. Nela, os estados possíveis do robô são representados por partículas, criando uma distribuição de probabilidade. Com os dados da odometria e do sensor ótico, o estado possível do robô é atualizado, sendo a partícula com maior probabilidade escolhida como a estimativa de posição do robô.

Porém, em situações em que não há conhecimento prévio do ambiente, é necessário usar técnicas que realizam o mapeamento durante o deslocamento do robô.

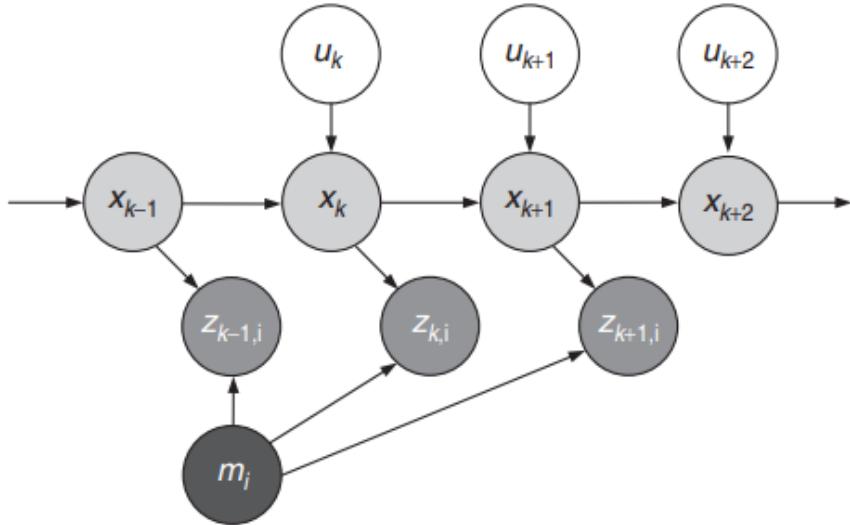
### 2.3 LOCALIZAÇÃO E MAPEAMENTO SIMULTÂNEO (SLAM)

Quando não há um mapa do ambiente obtido a priori, é preciso construir o mapa durante a navegação. Nesse caso, técnicas de localização e mapeamento simultâneo (SLAM) são usadas.

Resolver o problema de SLAM envolve a estimativa da trajetória do robô e do mapa do ambiente enquanto o robô se desloca. Ao longo da movimentação do robô em poses sequenciais desconhecidas  $x_{1:t} = x_1, x_2, \dots, x_t$  em uma mapa desconhecido  $m$ , são obtidas uma sequência de odometrias  $u_{1:t} = u_1, u_2, \dots, u_t$  e observações do ambiente  $z_{1:t} = z_1, z_2, \dots, z_t$ . O problema SLAM requer a obtenção da distribuição de probabilidades dada pela Equação 1, considerando que  $x_0$  é a posição inicial do robô (DURRANT-WHYTE; BAILEY, 2006). Um modelo gráfico deste problema, em forma de uma rede de Bayes dinâmica, está representado na Figura 4.

$$p(x_{1:t}, m | z_{1:t}, u_{1:t}, x_0) \quad (1)$$

**Figura 4:** Modelo gráfico do problema do SLAM.



Fonte: Durrant-Whyte e Bailey (2006).

Na literatura, uma variedade de soluções para esse problema estão disponíveis. Dentre eles, destaca-se o método baseado em um grafo de poses. O estado da arte atual, em relação a precisão e a velocidade da estimativa, de sistemas SLAM utilizam técnicas com base nesse método (GRISSETTI et al., 2010).

### **2.3.1 SLAM baseado em grafo de poses**

A aplicação do grafo de poses para mapeamento e localização geralmente consiste em quatro etapas: extração dos dados dos sensores; construção do grafo de poses; procura de candidatos a fechamento de laço; e otimização do grafo.

A construção do grafo de poses é feita utilizando os dados extraídos dos sensores de odometria e percepção. Cada nó deste grafo consiste na posição estimada do robô, dada pela odometria, com as informações obtidas pelo sensor ótico, comumente em forma de *features* da imagem.

Após a adição de um nó, é feita a procura de candidatos para o fechamento de laço, comparando os dados deste nó com de nós desconectados, ou seja, nós que não foram criados recentemente na mesma rota. Desta forma, é possível detectar quando o robô retorna a uma posição já visitada anteriormente.

Finalmente, levando em contas as restrições obtidas na construção do grafo, é realizada a otimização do caminho obtido, de forma a diminuir o erro da estimativa de posição.

### **2.3.2 Aplicações do SLAM no ROS**

Entre pacotes que implementam o SLAM baseado em grafos se destacam o SLAM Toolbox e o Cartographer (MACENSKI; JAMBRECIC, 2021). Porém, o Cartographer foi descontinuado pelo Google e não é mais atualizado. O SLAM Toolbox, por outro lado, foi escolhido como o pacote padrão de SLAM para o ROS 2 (MACENSKI, 2020), e oferece diversas melhorias em relação a sistemas de SLAM anteriores, como o GMapping, Karto, Hector e o próprio Cartographer.

Estes pacotes utilizam representações bidimensionais do ambiente, sendo sensores baseados em *scan* LASER a fonte de dado mais comum (CHONG et al., 2015). Porém, com o avanço da tecnologia, sensores Red Green Blue - Depth (RGB-D) e câmeras estéreo estão se tornando mais acessíveis, influenciando o desenvolvimento de sistemas de Visual SLAM (VSLAM). Dentre sistemas de VSLAM, destacam-se o ORB-SLAM3, OpenVSLAM e RTABMap, que possuem suporte a câmeras RGB-D e permitem modos de localização pura.

Neste modo, também presente no SLAM Toolbox, a localização é feita com um mapa base, obtido anteriormente, e com o estado atual do ambiente. Divergências entre o ambiente e o mapa base são tratados como obstáculos temporários, e não modificam permanentemente o mapa original.

Em Merzlyakov e Macenski (2021), é feita uma comparação entre sistemas VSLAM, mostrando que o OpenVSLAM é o pacote mais adequado para maioria dos casos. Dos pacotes citados neste artigo, apenas o RTABMap continua sendo desenvolvido ativamente, com suporte às versões mais recentes do ROS 2. Porém, novos sistemas de VSLAM promissores estão sendo desenvolvidos, como o NVidia Isaac ROS VSLAM e o StellaROS, que é um sucessor do OpenVSLAM.

## **2.4 FUSÃO DE DADOS DE SENSORES**

Robôs modernos estão cada vez equipados com mais sensores, ocasionando em mais de uma fonte de dado para a mesma informação. Um caso relevante neste trabalho, é

a odometria do robô, que pode ser estimada por diversos tipos de sensores, como IMU ou encoders de roda, cada um com características diferentes. Ao invés de utilizar apenas um sensor para estimar a odometria, os dados desses sensores, munidos da informação de covariância do mesmo, podem ser fundidos obtendo uma estimativa mais precisa. O filtro de Kalman, introduzido em Kalman (1960), é um estimador linear recursivo, comumente utilizado para aplicações deste tipo.

Um exemplo de aplicação deste filtro para estimativa de posição no ROS 2 é o pacote `robot_localization` (MOORE; STOUCH, 2014). Ele permite um grande número de entrada de dados de sensores, e tem suporte para diversos sensores, através da utilização de diferentes tipos de mensagens do ROS. Além disso, é possível configurar os dados que serão utilizados de cada sensor, podendo excluir os que não são relevantes ou confiáveis do resultado final.

## 2.5 MAPAS DE CUSTO

Um mapa de custo é uma representação de ambiente composta por uma grade de células que contém um valor, variando de desconhecido, livre, ocupado ou custo inflado.

Tradicionalmente, esses dados são armazenadas em mapas monolíticos, para utilização em planejamento de trajetórias. Essa implementação é aplicada com sucesso para caminhos curtos, mas pode apresentar dificuldade em lidar com ambientes dinâmicos maiores (LATOMBE, 2012).

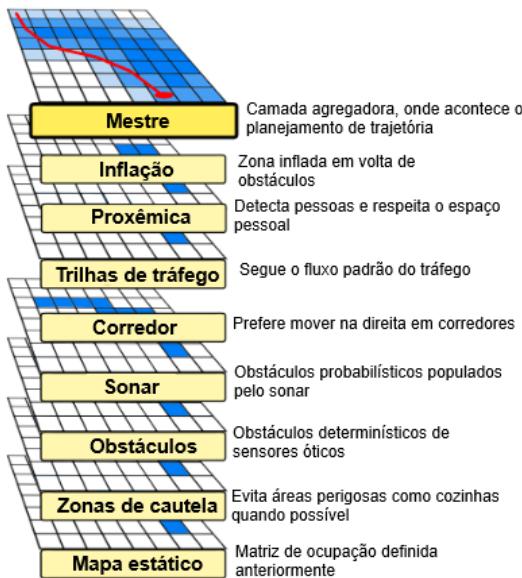
Uma solução para este problema são mapas de custo com camadas, que separam o processamento de dados em camadas semanticamente distintas (LU; HERSHBERGER; SMART, 2014). Esta separação permite a utilização de diferentes fontes de dados, possibilitando, por exemplo, atualizar o mapa com dados de sensores, sem modificar o original. Outras camadas podem ser configuradas para evitar certas áreas, manter uma distância segura de obstáculos ou para respeitar regras culturais, como manter-se a direita em corredores. A Figura 5 mostra uma configuração possível de camadas de mapas de custo.

Entre essas camadas, três são essenciais para mapear ambientes dinâmicos: a estática, a de obstáculos e a de inflação. A camada estática tem o mapa base, que deve ter apenas obstáculos fixos, dessa maneira, a trajetória planejada não será alterada em razão de objetos inexistentes.

A camada de obstáculos é responsável por atualizar o mapa de custo com obstáculos dinâmicos, através de dados de sensores óticos. Uma opção dessa camada, é a camada *voxel*, que utiliza uma representação em três dimensões dos dados dos sensores, introduzida em Marder-Eppstein et al. (2010). Nesse caso, é utilizado um espaço tridimensional, composto por blocos chamados de *voxels*, que indicam se o bloco está ocupado ou livre. Cada coluna deste campo é então projetada em um mapa bidimensional, com o valor dependendo da presença de *voxels* ocupados na mesma. Isto permite ao robô detectar obstáculos em diferentes alturas, atualizando o mapa apenas para obstáculos que podem colidir com o robô.

A camada de inflação utiliza os dados das camadas abaixo dela para criar uma zona de segurança entre os obstáculos. Essa zona tem valores intermediários entre os valores de obstáculo e livre, criando trajetórias seguras, mas que permitem que o robô passe próximo a obstáculos, caso necessário.

**Figura 5:** Exemplo de configuração de camadas de um mapa de custo.



Fonte: Adaptado de Lu, Hershberger e Smart (2014).

## 2.6 NAVIGATION2

O *Navigation2* (Nav2) é o sucessor do ROS *navigation stack*, permitindo a realização de tarefas complexas em diversos ambientes e classes de robôs cinemáticos. Baseando-se no legado do *navigation stack* do ROS 1, o Nav2 foi construído em cima do ROS2, implementando técnicas mais modernas para ter um sistema modular propício para ambientes dinâmicos, e com suporte a uma maior variedade de sensores (MACENSKI; MARTIN et al., 2020).

Uma árvore de comportamento é utilizada para orquestrar as tarefas de navegação, ativando os servidores de controle, planejamento e recuperação para navegação. Para executar nós de *actions*, são normalmente utilizados *Action servers* do ROS 2. Esta árvore de comportamento pode ser configurada pelo usuário através de um arquivo em XML, permitindo a descrição de comportamentos de navegação únicos sem necessidade de programação.

Na arquitetura, pode-se notar a utilização de dois mapas de custo, um local e outro global. O local, utilizado no servidor do controlador, realiza o planejamento a curto prazo e prevenção de colisão, enquanto o global, aplicado no servidor de planejamento, é usado principalmente para planejamento a longo prazo.

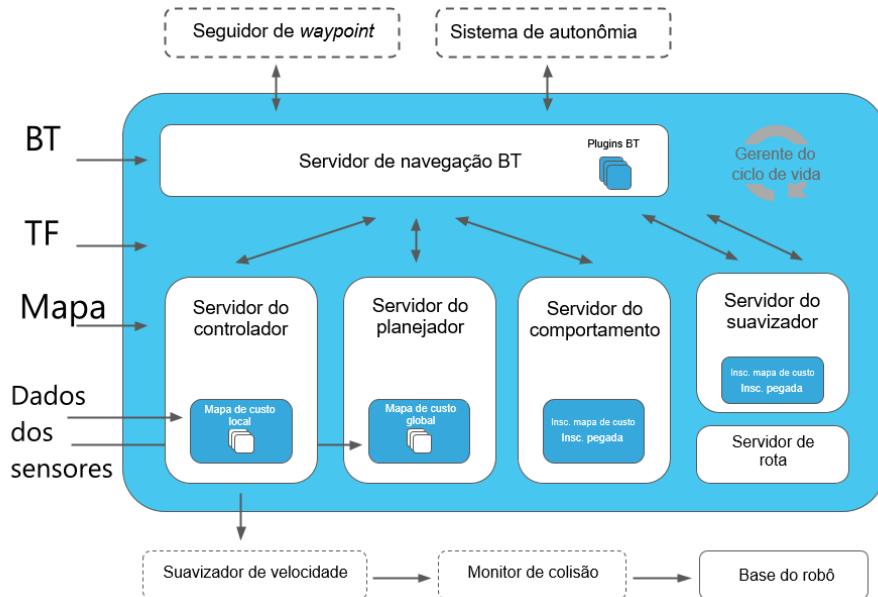
### 3 METODOLOGIA

O sistema de navegação do Twil foi desenvolvido utilizando a versão Humble do ROS 2 e com auxílio de ferramentas suportadas por esta versão. Dentre as ferramentas utilizadas, destacam-se o Gazebo, utilizado para simulação do robô, e o *Navigation2*, que fornece diversas ferramentas para implementar e orquestrar um sistema de navegação autônoma.

Os variados componentes do robô, como os sensores empregados neste trabalho, foram implementados através de *plugins* do Gazebo, que comunicam o estado do robô durante a simulação através de tópicos do ROS 2. A câmera RGB-D Intel RealSense D435, utilizada para o mapeamento do ambiente, foco deste trabalho, foi implementada dessa forma.

O *Navigation2* é encarregado de enviar comandos de velocidade para o robô, para que ele chegue ao destino desejado de modo seguro, evitando colisões. Sua arquitetura está representada na Figura 6. As setas à esquerda da imagem representam as entradas necessárias do sistema, como a estimativa de posição do robô, representadas pela seta TF, em forma de transformadas sistemas de coordenadas conforme o REP-105 (MEEUSSE, 2010), além do mapa e dos dados dos sensores, que são utilizados para o planejamento de trajetórias.

**Figura 6:** Arquitetura do *Navigation2*.



Fonte: Adaptado de *Navigation2* (2020).

O servidor do planejador é responsável pela criação de trajetórias usando a representação do ambiente em forma de um mapa de custo. Utilizando essa trajetória como

referência, e comparando a estimativa de posição do robô, o servidor do controlador gera comandos de velocidade para o robô, que é a saída do *Nav2*.

Antes do início deste trabalho, o Twil já estava configurado para utilizar o *Nav2*, porém sem utilizar dados da câmera e IMU. Portanto, as trajetórias planejadas não eram atualizadas a fim de evitar obstáculos, além de não haver sistema de localização, ou seja, a transformada entre `map` e `odom` era estática, sem ajuste da odometria implementada.

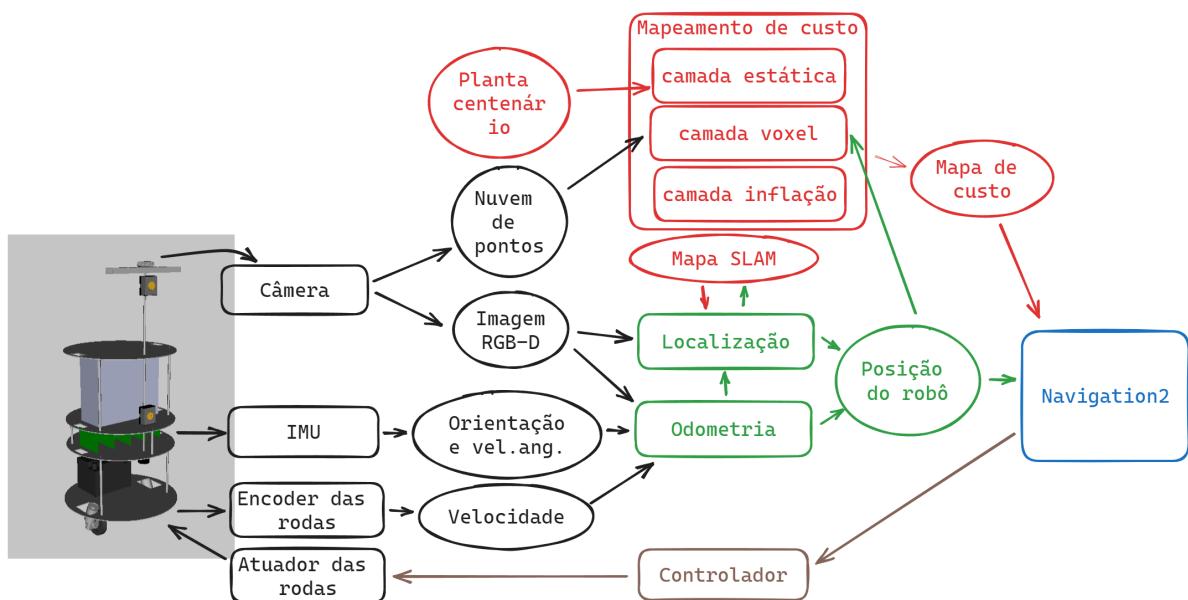
Em razão disso, foi adicionada a câmera RGB-D Intel RealSense D435 e um IMU ao robô, e o sistema de navegação foi aprimorado com a utilização desses sensores nos componentes de odometria, localização e mapeamento.

Um diagrama simplificado da arquitetura do sistema desenvolvido neste trabalho é mostrado na Figura 7. O desenvolvimento atual foi focado nos blocos em vermelho, que realizam o mapeamento do ambiente, e blocos em verde, que estimam a posição do robô. O bloco em azul, que representa as outras ferramentas do *Nav2*, e o controlador em marrom, que transforma os comandos de velocidade em comandos para as rodas do robô, já estavam configurados.

Os componentes de mapeamento criam duas representações do ambiente. Um mapa criado é o utilizado no planejamento de trajetórias, em forma de mapa de custo, onde foi adicionada uma camada para percepção de obstáculos utilizando a câmera RGB-D. Além disso, foi realizada a calibração da camada de inflação, para produzir caminhos mais seguros. O segundo mapa é criado e utilizado pelos sistemas de localização e mapeamento simultâneo (SLAM), adicionados ao Twil.

Em verde, estão os componentes referentes à estimativa de posição do robô, onde foram adicionados pacotes que realizam a odometria visual e pacotes de SLAM. Também, aprimorou-se a odometria das rodas, através da fusão de dados com o sensor IMU. Ademais, para realização de testes, se manteve a opção de utilizar a transformada estática entre `map` e `odom`, e foi adicionado um sistema de odometria utilizando a posição exata do robô no Gazebo.

**Figura 7:** Arquitetura do simplificada do sistema desenvolvido.



Nas seções seguintes, é apresentado o robô e o ambiente de testes utilizado neste trabalho. Em seguida, é detalhado o desenvolvimento dos sistemas necessários para a

navegação autônoma do robô neste ambiente. Finalmente, é descrita a coleta de dados para a análise dos resultados, que é apresentada no próximo capítulo.

### 3.1 CONFIGURAÇÃO DO ROBÔ

O modelo do robô está presente no pacote `twil_description`, que contém os arquivos de descrição do robô no formato XACRO, que é expandido para o formato URDF, utilizado pelo nodo `robot_state_publisher`, que publica essa descrição em um tópico. Esse modelo é utilizado no rviz para visualização do robô, como representado na Figura 2, e no Gazebo, para simulação. Os componentes físicos do robô, como sensores e atuadores, são criados a partir de *plugins* do Gazebo, também presentes no arquivo de descrição.

Durante este trabalho, foram configurados dois novos componentes ao robô, a câmera RGB-D Intel RealSense D435 e um IMU. A câmera está presente nos sistemas de odometria, localização e mapeamento, enquanto o IMU foi usado como suplemento ao sistemas de odometria, fornecendo dados de orientação do robô para fusão de dados.

Para utilização da câmera, é necessário o pacote `realsense-ros` (INTEL, s.d.), que contém o modelo da câmera. O *plugin* do Gazebo `camera_plugin`, é responsável pela publicação dos dados da câmera simulada. Cinco tópicos são responsáveis por estes dados: dois tópicos publicam as imagens não comprimidas, uma RGB e outra de profundidade; dois tópicos com informações suplementares destas imagens; e um tópico com a nuvem de pontos produzida pela câmera.

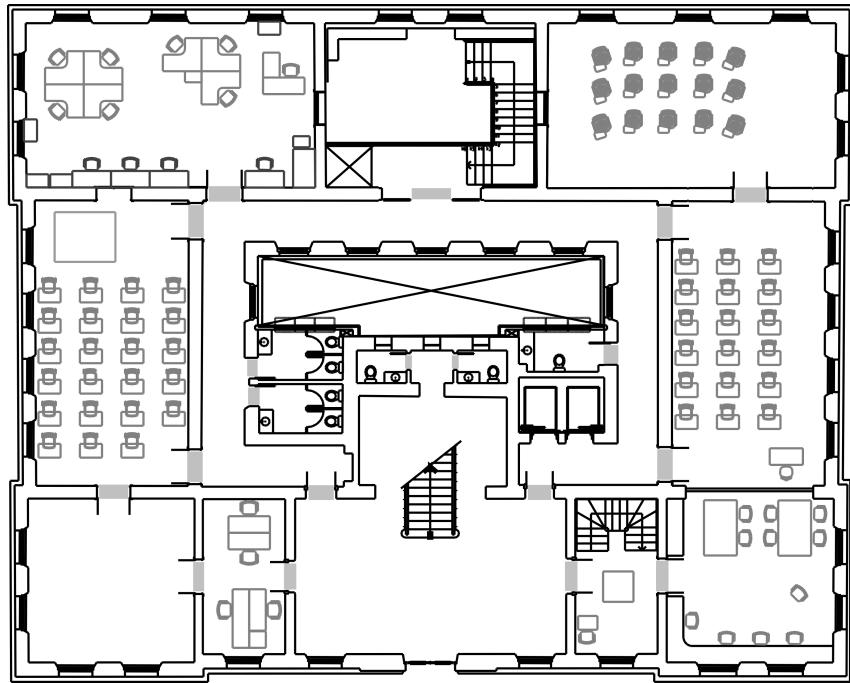
A adição do IMU foi feita utilizando o *plugin* `GazeboRosImuSensor`, que publica mensagens do tipo `sensor_msgs/Imu`. Essas mensagens contêm a orientação, velocidade angular e aceleração linear do robô. Por si só, o IMU não é confiável para estimar a posição do robô porque necessita integrar a aceleração linear duas vezes para calcular a posição e, portanto, é muito suscetível a erros de acumulação. Porém, os dados referentes a orientação e velocidade angular, podem complementar as outras fontes de odometria, através da fusão de dados. Para representar fisicamente o IMU no robô, foi reutilizada a descrição de uma placa de circuito impresso *Eurocard*, que também representa outros componentes do robô.

A atuação das rodas do robô, a partir dos comandos de velocidade produzidos pelo *Nav2*, é feita por controladores configurados no pacote `twil_bringup`. Diversos deles estão disponíveis para uso com o Twil, porém o controlador `twist_mrac_controller`, do pacote `linearizing_controllers` foi o escolhido e integrado ao *Nav2* em trabalhos anteriores.

### 3.2 AMBIENTE DE SIMULAÇÃO

O sistema foi testado em uma simulação do prédio Centenário da Escola de Engenharia da UFRGS, utilizando o Gazebo. No pacote `ufrgs_map` está incluído uma representação em formato PGM da planta do prédio, mostrada na Figura 8, desenvolvida em Petry (2019). Além da imagem da planta, também está presente um arquivo de configuração em formato YAML, que contém a resolução e a origem do mapa. Este arquivo ainda contém parâmetros para utilização como mapa de custo, indicando a faixa de valores para considerar uma célula como livre ou ocupada.

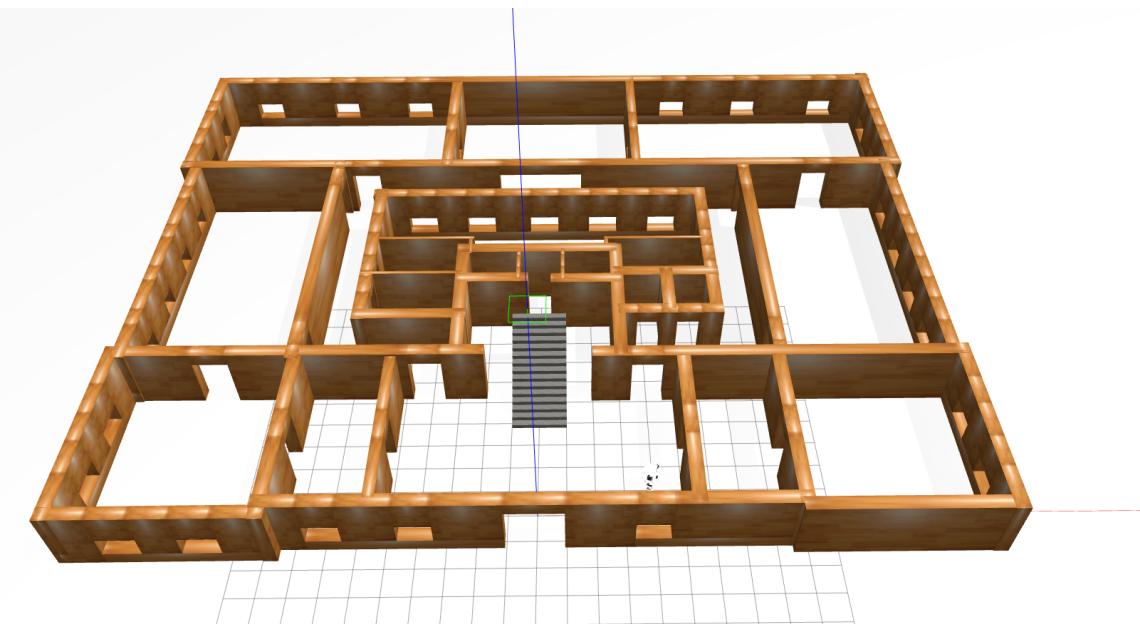
**Figura 8:** Planta do 1º andar do prédio Centenário da EE-UFRGS.



Fonte: Petry (2019).

O pacote `ufrgs_gazebo` contém arquivos de mundos do Gazebo para simulação. Ele, porém, estava desatualizado e não continha uma representação completa do prédio Centenário. Utilizando o *Building Editor* do Gazebo, foi criado um modelo simplificado do prédio em 3D, com base na sua planta, mostrado na Figura 9.

**Figura 9:** Ambiente Gazebo com modelo do prédio Centenário da EE-UFRGS.



Não foram adicionados os obstáculos representados em cinza claro da planta da Figura 8, que não são obstáculos fixos. Para teste com obstáculos dinâmicos ou temporários são adicionados novos objetos a este mundo durante a execução da simulação no Gazebo.

### 3.3 ESTIMATIVA DE POSIÇÃO DO ROBÔ

A estimativa de posição é feita utilizando o sistemas de coordenadas definido pela norma REP 105, conforme descrito na Seção 3.3. Portanto, a pose do robô é estimada por dois sistemas, o de odometria e de localização.

#### 3.3.1 Odometria

O sistema de odometria é responsável pela publicação da transformada entre o sistema de coordenadas `odom` e `base_link`, utilizando sensores incrementais.

Neste trabalho, foram utilizados e comparados três sistemas de odometria, que além da transformada, também publicam mensagens do tipo `nav_msgs/Odometry` no tópico `/odom`. Antes deste trabalho, a odometria era publicada apenas pelo controlador, através do pacote `twist_mrac_controller`, utilizando dados do encoder nas rodas para estimar a posição do robô. O cálculo da posição é feito pelo pacote `arc_odometry`, que utiliza o modelo cinemático de um robô móvel de acionamento diferencial para estimar a posição e orientação do robô a partir das rotação das rodas. Para aprimorar a qualidade desta odometria, foi adicionado o pacote `robot_localization`, que realiza a fusão de dados da odometria das rodas com os dados do IMU.

A fusão é realizada através de um filtro de Kalman estendido, onde devem ser escolhidos quais dados dos sensores devem ser considerados. Os criadores do pacote `robot_localization` recomendam não utilizar dados de posição da odometria das rodas. Dessa forma, a posição é calculada pelo mesmo pacote a partir dos dados obtidos pela fusão das velocidades fornecidas ao pacote. Quando são fornecidas estimativas de posição, não é realizado o cálculo a partir da velocidade, sendo usado apenas a fusão dos dados de posição.

Apesar disso, decidiu-se utilizar tanto a posição quanto a velocidade na fusão de dados, que faz com que a pose publicada pelo pacote seja igual a estimada pelo `arc_odometry`, já que o IMU não fornece dados de posição. Isso foi feito porque a velocidade obtida pelo `arc_odometry` é dada em referência ao sistema de coordenadas global, ou seja, o `odom`, enquanto que o `robot_localization` espera que a velocidade seja em referência a base do robô. Portanto, o `robot_localization` não é capaz de estimar corretamente a posição do robô com base na velocidade publicada pelo `arc_odometry` sem um nodo adicional para conversão.

A velocidade angular e orientação, por outro lado, foram fundidas com os dados do IMU, que fornece uma estimativa mais precisa destes campos.

Como alternativa a este sistema, foi implementado um sistema de odometria visual, publicado pelo executável `rgbd_odometry` do pacote `rtabmap_odom`, que utiliza imagens da câmera RGB-D com auxílio do sensor IMU para estimava da posição. Com a informação de profundidade, são comparadas *features* visuais entre imagens consecutivas, a velocidade do robô é estimada com um algoritmo baseado em *Random Sample Consensus* (RANSAC) (LABBE, 2023).

É possível fundir os dados destes dois sistemas de odometria, porém, optou-se primeiramente por mantê-los separados para realizar a comparação do desempenho entre eles. Caso ambos sistemas tenham desempenhos satisfatórios, a fusão de dados pode fornecer um resultado mais preciso.

Finalmente, para realização de testes, foi implementado um sistema de odometria com a posição real do robô no Gazebo. O *plugin* P3D, publica uma mensagem do tipo

`nav_msgs/Odometry` com a posição do robô em relação ao sistema de coordenadas `odom`. Quando este sistema de odometria é selecionado, as informações do P3D são retransmitidas no tópico `/odom` e o pacote `odom_to_tf_ros2` publica a transformada entre `odom` e `base_link` a partir destes dados.

Os sistemas de odometria utilizados estão resumidos na Tabela 1, onde é indicado o pacote, a fonte de dados e o tipo de mensagem recebido por cada um.

**Tabela 1:** Sistemas de odometria

Pacote utilizado	Fonte de dados	Tópico	Tipo da mensagem
<code>robot_localization</code>	Rodas IMU	<code>/controller/odom</code> <code>/sensor imu</code>	<code>nav_msgs/Odometry</code> <code>sensor_msgs/Imu</code>
<code>rtabmap_odom</code>	Câmera	<code>/depth/image_raw</code>	<code>sensor_msgs/Image</code>
		<code>/depth/camera_info</code>	<code>sensor_msgs/CameraInfo</code>
<code>odom_to_tf_ros2</code>	Gazebo	<code>/color/image_raw</code>	<code>sensor_msgs/Image</code>
		<code>/color/camera_info</code>	<code>sensor_msgs/CameraInfo</code>
<code>odom_to_tf_ros2</code>	Gazebo	<code>/sensor imu</code>	<code>sensor_msgs/Imu</code>
		<code>/fake_odom</code>	<code>nav_msgs/Odometry</code>

### 3.3.2 Localização

O sistema de localização, por outro lado, publica esporadicamente a transformada entre o sistema de coordenadas `map` e `odom`, para ajustar os erros de divergência da odometria, através de sensores absolutos.

Anteriormente, não era utilizado nenhum sensor para localização do robô, e a transformada entre `map` e `odom` era estática, causando divergência na estimativa de posição durante a navegação. Este sistema foi mantido para testes, porém foram implementados dois sistemas de localização baseados em localização e mapeamento simultâneos (SLAM), utilizando a câmera RGB-D Intel RealSense D435, para melhorar a estimativa de posição do robô publicada pela odometria.

Uma alternativa à sistemas SLAM é o `nav2_amcl`, que utiliza localização por Monte Carlo em um mapa construído previamente, que poderia ser a planta do prédio, para estimar a posição do robô. Porém, essa opção foi descartada, pois o robô deve ser capaz de navegar em ambiente dinâmicos ou pouco conhecidos, onde um mapa estático não seria adequado. Além disso, esse pacote só é compatível com representações bidimensionais do ambiente, que causaria problemas na detecção de objetos tridimensionais, como as escadas, agravando as divergências que já existem entre o ambiente do Gazebo e a planta.

Os dois sistemas de SLAM utilizados foram o `slam_toolbox` e o `rtabmap_slam`, resumidos na Tabela 2.

**Tabela 2:** Sistemas de localização

Pacote utilizado	Fonte de dados	Tópico	Tipo de mensagem recebida
<code>slam_toolbox</code>	Câmera	<code>/depth/image_scan</code>	<code>sensor_msgs/LaserScan</code>
<code>rtabmap_slam</code>	Câmera	<code>/depth/image_raw</code>	<code>sensor_msgs/Image</code>
		<code>/depth/camera_info</code>	<code>sensor_msgs/CameraInfo</code>
<code>rtabmap_slam</code>	Câmera	<code>/color/image_raw</code>	<code>sensor_msgs/Image</code>
		<code>/color/camera_info</code>	<code>sensor_msgs/CameraInfo</code>

O *slam\_toolbox*, realiza o SLAM em 2D e foi criado para utilizar sensores a laser. Portanto, são esperadas mensagens do tipo `sensor_msgs/LaserScan` para percepção do ambiente neste sistema. Como a câmera RGB-D não publica mensagens desse tipo, o pacote `depthimage_to_laserscan` foi usado para converter a imagem de profundidade da câmera na mensagem esperada, em forma de um feixe de luz em duas dimensões localizado na altura da câmera. O *rtabmap*, por outro lado, realiza o SLAM em 3D e aproveita todos os tópicos publicados pela câmera RGB-D, não necessitando de conversão de mensagens.

Ambos sistemas publicam a transformada entre `map` e `odom`, além da posição global do robô no tópico `/pose`, em mensagens do tipo `geometry_msgs/PoseWithCovarianceStamped`, que são utilizadas para comparação com a posição real e a estimada pela odometria.

Apesar de realizar o mapeamento do ambiente em tempo-real, esses sistemas podem ser iniciados com informações de uma sessão prévia de mapeamento, que pode aumentar a precisão da localização, caso o mapa seja mais fiel que o construído. Porém, nos testes realizados, a sessão de SLAM foi iniciada do zero, para simular um ambiente desconhecido.

### 3.4 MAPEAMENTO

O mapeamento é um aspecto essencial para a navegação autônoma. Ele é utilizado tanto no planejador de trajetórias, para perceber e evitar obstáculos dinâmicos, quanto no sistema de localização do robô, para estimar a posição do robô no ambiente.

Apesar do sistema de SLAM fornecer um mapa do ambiente, este mapa não é utilizado no mapa de custo. Como este mapa não está completo desde o início, não é possível planejar trajetórias para ambientes não visitados previamente. Além disso, é preferível que o mapa da camada estática tenha apenas obstáculos fixos, como paredes e escadas, e utilizar a camada de obstáculos para obstáculos dinâmicos. Esta separação permite que o mapa estático seja utilizado em diversas sessões, porém com uma camada de obstáculos nova a cada sessão. Ademais a camada de obstáculos pode ser configurada de forma independente que o SLAM, podendo utilizar outros sensores ou considerar uma altura de obstáculos diferente.

Portanto, foram criadas duas representações do ambiente, uma obtida pelo SLAM, que é utilizada apenas para localização, e outra realizada utilizando os *plugins* do mapa de custo do Nav2, que é usado no planejamento de trajetória.

#### 3.4.1 Mapeamento de custo

O mapa de custo empregado no planejamento de trajetórias é composto por três camadas: a camada estática, que possui uma representação simples do ambiente, preferencialmente sem obstáculos temporários; a camada de obstáculos ou *voxel*, onde são utilizados dados dos sensores ópticos para atualizar o mapa com obstáculos percebidos; e a camada de inflação, que cria um campo de custo ao redor dos obstáculos.

A planta do prédio Centenário, representada na Figura 8, foi utilizada como mapa estático. Ele foi configurado para considerar apenas as cédulas em preto como ocupadas, que equivalem a obstáculos fixos, como paredes e a escada. As cédulas em cinza, que representam a obstáculos prováveis como cadeiras e mesas, são consideradas livres, já que esses objetos devem ser captados pelos dados dos sensores do robô.

A camada de obstáculos utiliza os dados da camera RGB-D para atualizar o mapa. Existem dois *plugins* que podem ser aplicados para este fim, o `obstacle_layer` e

o `voxel_layer`. Ambos suportam mensagens do tipo `PointCloud2`, que são publicadas pela câmera RGB-D, porém empregam técnicas diferentes para atualizar o mapa. O `obstacle_layer` utiliza *ray casting* em 2D para determinar a posição dos obstáculos. O `voxel_layer`, por outro lado, cria um grade tri-dimensional, divido em blocos chamados de *voxels*, que podem estar ocupados ou livres.

Apesar do mapa de custo ser em 2D, a simulação e o mundo real podem ter obstáculos de diferentes alturas, como mesas e cadeiras. Portanto, é necessária a percepção em 3D do ambiente para evitar colisões com estes obstáculos. Logo, foi utilizado o `voxel_layer` para atualizar o mapa de custo.

Para a configuração do `voxel_layer`, é preciso definir a resolução e o número de *voxels* no eixo Z utilizados. A câmera RGB-D Intel RealSense D435 do Twil está localizada a uma altura de aproximadamente 1,37 metro do chão, que é a altura mínima do gride de *voxels*. O número máximo de *voxels* permitido pelo *plugin* é 16. Tendo em vista estes dados, foi definido um gride de *voxels* com 15 *voxels* de resolução 0,1 metro, criando um gride de 1,5 metro de altura. Portanto, são captados apenas obstáculos dentro deste campo, e objetos em uma altura superior a 1,5 metro não são considerados. Isso é importante para evitar a detecção de obstáculos que não são relevantes para a navegação como, no caso do ambiente de simulação utilizado, a moldura das portas.

A última camada, `inflation_layer`, cria uma zona de segurança ao redor dos objetos captados nas outras camadas, para garantir uma distância segura ao planejar a trajetória. Em Zheng (2019), recomenda-se que a camada de inflação cubra um raio grande em volta de obstáculos com uma curva de decaimento de inclinação baixa, para criar um campo que cubra a maior parte do mapa de custo. Desta forma, são criadas trajetórias que passam no meio de obstáculos, mantendo uma distância segura, diminuindo o risco de possíveis colisões e realizando movimentos suaves.

### **3.4.2 Mapeamento SLAM**

O mapeamento realizado pelo sistemas SLAM é utilizado apenas para a localização do robô pelo mesmo pacote que o criou. Foram criados dois mapas, um por cada pacote de localização implementado, `slam_toolbox` e `rtabmap_slam`, descritos na Seção 3.3.2.

Devido aos dados dos sensores à sua disposição, o `slam_toolbox` só é capaz de criar mapas em duas dimensões. Porém, o `rtabmap_slam` pode criar mapas em duas ou três dimensões. Como o robô Twil não é capaz de se movimentar no eixo Z, neste trabalho só serão comparados os mapas bidimensionais criados por ambos pacotes.

## **3.5 TESTES E COLETA DE DADOS**

Devido à independência entre o mapeamento de custos e os demais sistemas desenvolvidos neste trabalho, foram realizados dois tipos de testes diferentes. Para testar o mapeamento de custo, foram criadas trajetórias com e sem obstáculos para testar o desempenho das camadas de obstáculos e inflação. Já para conferir o funcionamento dos sistemas de odometria, localização e mapeamento SLAM, utilizaram-se arquivos de gravação de tópicos do ROS 2, chamados de *rosbags*.

Estes arquivos permitem a execução de diferentes ensaios usando os mesmos dados de entrada. Outro benefício foi o alívio na exigência de processamento do computador

utilizado, já que não foi necessário realizar a simulação do mundo real no Gazebo e a execução dos nodos de localização simultaneamente.

Para a criação das *rosbags*, foi feito um *script* em *bash* que contém todos os tópicos necessários para a execução dos sistemas de odometria e localização, além de tópicos auxiliares para visualização do experimento e para análise dos resultados, como a posição real do robô publicada no tópico **/ground\_truth**, pelo *plugin* P3D do Gazebo. O tópico utilizado no sistema de odometria perfeita criado por esse mesmo *plugin* difere do tópico **ground\_truth** porque a posição publicada na odometria é em relação ao sistema de coordenadas **odom** e não a referência global, ou seja, **map**, como é no *ground truth*.

Os tópicos necessários para a execução dos sistemas de estimativa de posição estão nas Tabelas 1 e 2, apresentadas anteriormente. Porém, além dos tópicos que são inscritos explicitamente na configuração do pacote, os sistemas de odometria e localização, que incorporam a câmera RGB-D, também necessitam da árvore de transformações da base do robô à câmera, contida no tópico **/tf**. Portanto, esta informação deve estar presente durante a execução dos testes.

Uma solução para disponibilizar esta transformada seria gravar o tópico **/tf** durante a execução da simulação. Porém, isso não é possível, porque esse tópico já possui as transformadas entre **map**, **odom** e **base\_link**, o que causaria conflitos com as transformadas publicadas pelos sistemas de odometria e localização sendo testados. Logo, deverão ser executados novamente os mesmos nodos que disponibilizam a posição da câmera em relação a base do robô.

A transformação das juntas do robô são publicadas pelo **robot\_state\_publisher**. Existem dois tipos de juntas entre a base do robô e a câmera: estáticas, que têm sua informação definida no arquivo de descrição do robô, e dinâmicas, que são publicadas pelo **joint\_state\_broadcaster** no tópico **/joint\_states**, conforme as configurações definidas no **twi\_l\_bringup**. O executável **robot\_state\_publisher** lê as informações desse tópico e do arquivo de descrição para publicar as transformadas entre os componentes do robô no tópico **/tf**. Portanto, para evitar a execução do Gazebo, esse tópico foi gravado no banco de dados, e foi executado o **robot\_state\_publisher** com o arquivo de descrição do robô, obtendo assim, o mesmo estado do robô durante a execução da simulação utilizada para a gravação da *rosbag*.

Para facilitar a execução deste publicador de transformadas do robô e dos sistemas de odometria e localização, foi criado um arquivo de inicialização com os nodos necessários para realizar os testes com as entradas provenientes da *rosbag*.

Além disso, optou-se por criar outro *script* em *bash* para gravar os dados dos testes realizados para análise posterior. A criação dos gráficos foi realizado em *Python*, com a biblioteca Matplotlib a partir dos dados das *rosbags* convertidos o formato *csv*.

Os gráficos foram construídos utilizando os dados de posição x e y das mensagens do tipo **nav\_msgs/Odometry**, publicadas pelos sistemas de odometria e pelo Gazebo com a posição estimada e real do robô. Para o sistemas de localização, foram utilizados os dados de posição x e y das mensagens do tipo **geometry\_msgs/PoseWithCovarianceStamped**, que são referentes a posição global do robô calculada por esses pacotes, ou seja, a transformada entre **map**, **odom** e **base\_link**.

O erro da estimativa de posição foi calculado através da diferença entre a posição real e a posição estimada, conforme a Equação 2. Utilizando os dados dos testes, foi criado um gráfico com esse erro longo da distância real percorrida pelo robô.

$$d_{erro} = \sqrt{(x_{real} - x_{estimado})^2 + (y_{real} - y_{estimado})^2} \quad (2)$$

## 4 RESULTADOS

Neste capítulo, são apresentados os resultados dos testes realizados, separados em seções. Primeiramente é apresentado o efeito das camadas do mapa de custo no planejamento de trajetórias. Em seguida, as estimativas de posição do robô, produzidas pelos sistemas de odometria, localização são comparadas com a posição real. Finalmente, são mostrados os mapas criados pelos sistemas SLAM.

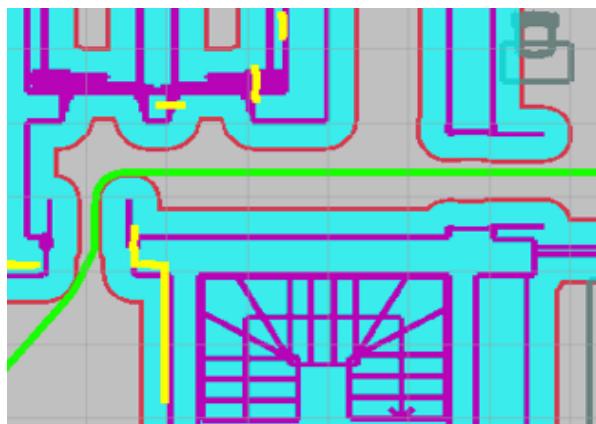
### 4.1 MAPEAMENTO DE CUSTO

Este trabalho realizou modificações em duas das três camadas do mapa de custo utilizadas, a camada de obstáculos e a camada de inflação. A camada estática foi mantida, utilizando a planta do prédio Centenário da Escola de Engenharia da UFRGS, como em trabalhos antigos.

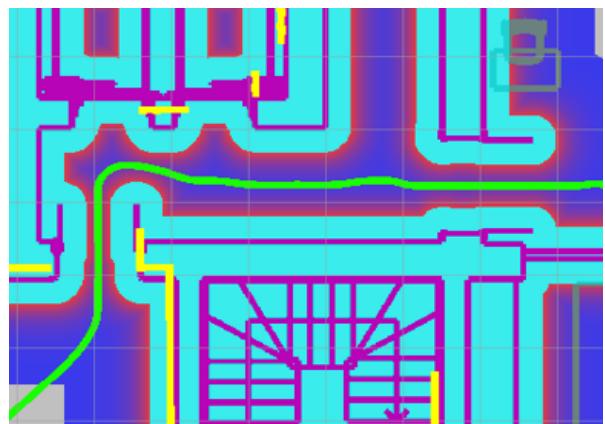
A camada de inflação porém, foi ajustada para melhorar a qualidade dos caminhos criados, de acordo com a recomendação de Zheng (2019). A camada de inflação original criava uma pequena zona de segurança ao redor dos obstáculos, com uma borda de 0,35 m e com fator de decaimento de 3,0. Representada na Figura 10a. Para criar um campo de segurança maior, capaz de cobrir corredores inteiros, foi ajustado a borda para 2,0 m. Em razão do aumento da borda, o fator de decaimento precisou ser ajustado para 4,0, porque a inclinação original fazia com que certos corredores fossem evitados.

**Figura 10:** Trajetórias criadas com diferentes configurações da camada de inflação

(a) Mapa de custo com inflação baixa.



(b) Mapa de custo com inflação alta.

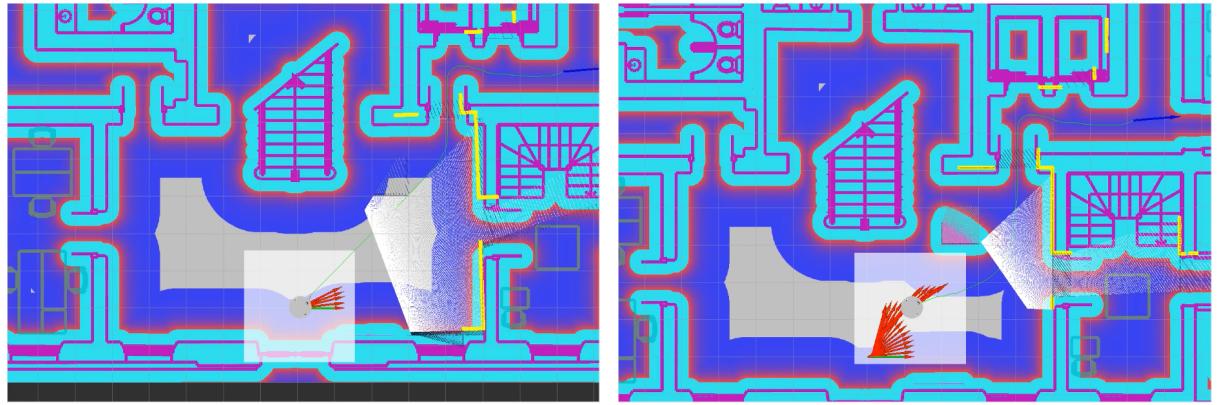


Finalmente, foi testada a camada de obstáculos adicionada neste trabalho, que utiliza dados da câmera RGB-D para adicionar novos obstáculos no mapa, mostrado na

Figura 11. Para realizar esse teste, foi adicionada uma caixa ao mundo do Gazebo, durante a simulação e então foi criada uma rota que passe por esse obstáculo, como na Figura 11a. Ao se aproximar desta caixa, ela é detectada pela nuvem de pontos da câmera, representada como pontos brancos, e é adicionada ao mapa de custo, como é visto na Figura 11b.

**Figura 11:** Mapeamento de obstáculos da camada voxel

(a) Mapa de custo antes da detecção do obstáculo. (b) Mapa de custo após a detecção do obstáculo.



Percebe-se nestas duas imagens que, caso fosse utilizada uma representação plana do ambiente, através de um feixe bidimensional na altura da câmera, que pode ser visto em amarelo nas Figuras 11a e 11b, o obstáculo não seria percebido, e o robô colidiria com a caixa.

## 4.2 TESTES DE ODOMETRIA, LOCALIZAÇÃO E MAPEAMENTO SLAM

O testes de odometria, localização e mapeamento SLAM foram realizados na mesma *rosbag*, como descrito na Seção 3.5, permitindo dessa forma, a mesma entrada em cada sistema.

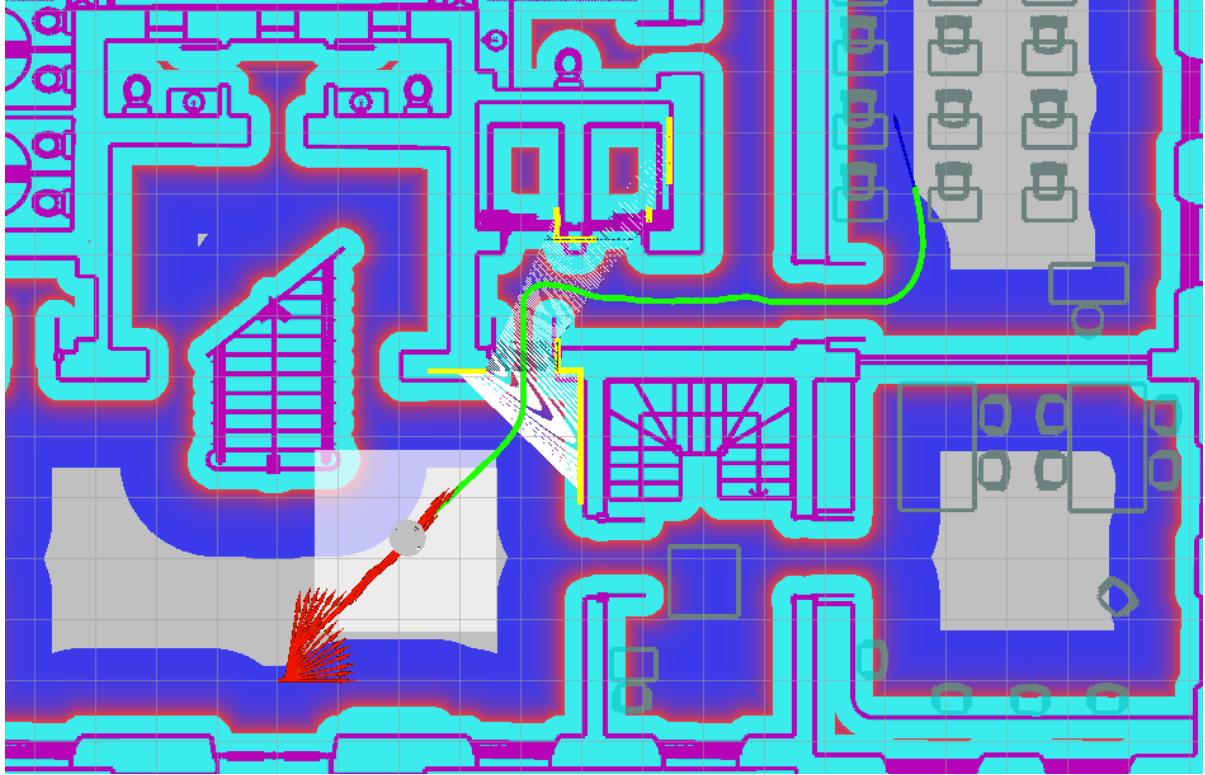
Na Figura 12, é mostrado o robô realizando a trajetória de testes. As setas em vermelho representam a mensagem de odometria do mesmo, onde é indicada a pose atual dele. A linha verde indica a trajetória planejada do robô. O caminho realizado nesse teste começa na origem, segue em direção à porta no canto superior direto da sala, percorre o corredor e entra na primeira sala. É possível comparar a percepção de diferentes ambientes nesse teste, como em corredores estreitos e salas abertas. A distância total percorrida foi de 13,08 m.

Devido ao tamanho do banco de dados em razão da captura das imagens da câmera, não foi possível realizar uma simulação longa, que percorresse uma grande área do mapa e retornasse ao ponto inicial, testando assim o fechamento de laço, que é um ponto importante no mapeamento de sistemas SLAM.

### 4.2.1 Odometria

Após o fim da simulação e da captura de dados, foram executados os sistemas de odometria na gravação gerada. Neste teste, utilizou-se a transformada estática entre os sistemas de coordenadas `map` e `odom`, para comparar a odometria das rodas e visual.

**Figura 12:** Robô durante o teste realizado.



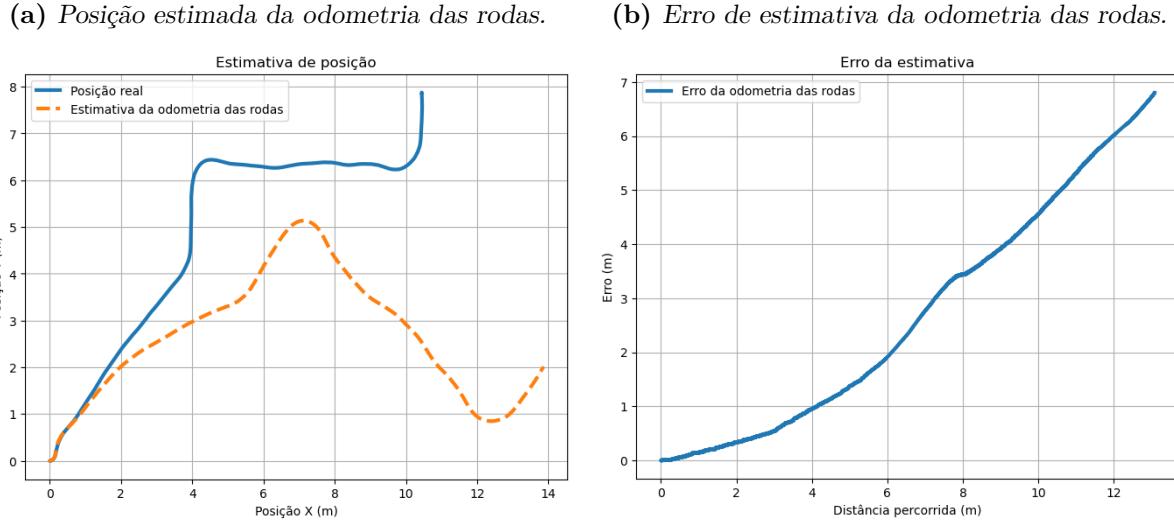
A partir dos dados obtidos, foram criados gráficos com as estimativas da posição do robô nos eixos X e Y em relação a origem do sistema de coordenadas `map` dos sistemas de odometria, comparados com a posição real do robô, e gráficos com o erro de estimativa de posição ao longo da distância percorrida.

Importante notar que os gráficos de erro de distância apresentam um ruído aparente. Isso ocorre devido à diferença de frequência de publicação das mensagens de posição real e estimativa de posição. Entre a geração de uma estimativa e outra, o erro de distância aumenta enquanto o robô move e a estimativa não é atualizada.

A odometria das rodas, mostrada na Figura 13, apresentou uma grande divergência em relação a posição real do robô, com a curva da posição estimada inclinada para a direita, em relação a curva da posição real, como mostra a Figura 13a, indicando um erro na estimativa da orientação do robô.

O erro de orientação é especialmente indesejado porque, quando é causado, se transforma em erro de posição lateral e aumenta sem limite (BORENSTEIN; FENG, 1996). Na primeira parte do trajeto, da origem até a primeira porta, a orientação estimada divergiu constantemente, causando uma curva exponencial do erro, como é visto na Figura 13b. Este erro continuou crescendo até o final do trajeto, causando uma grande divergência na posição final, com uma distância de 6,8 m entre a posição real e a estimada.

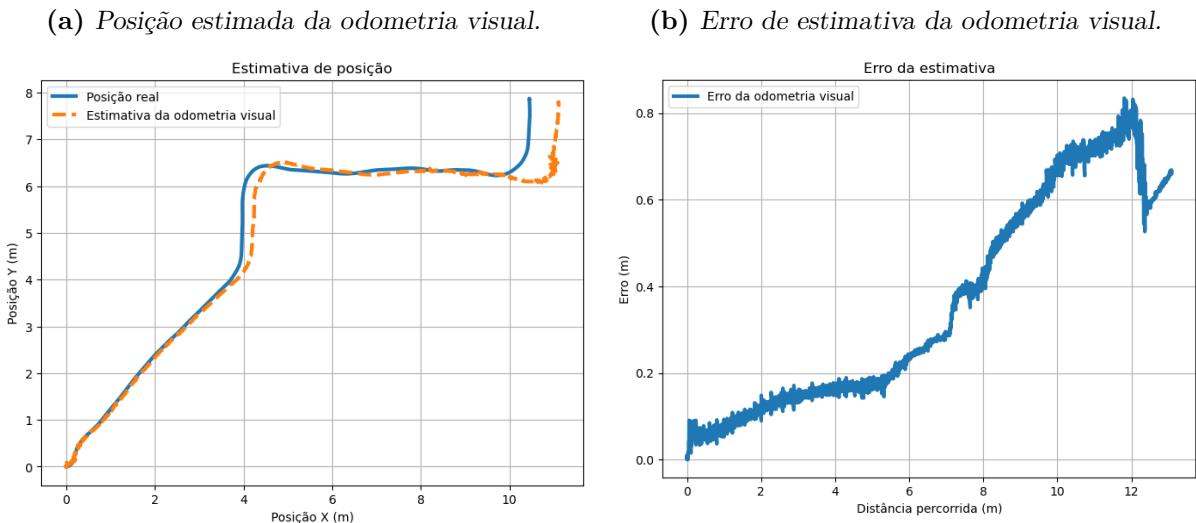
**Figura 13:** Resultado da odometria das rodas.



Por outro lado, a odometria visual, mostrada na Figura 14, apresentou um resultado satisfatório. No gráfico da posição estimada, da Figura 14a, mostra que ocorreu a mesma inclinação para a direita que a odometria das rodas, porém menos acentuada. Em alguns pontos da trajetória no corredor, a posição estimada coincide com a posição real. Porém, a estimativa de posição nesses pontos pode ter ocorrido em momentos diferentes ao da posição real, já que este gráfico não apresenta informação do tempo. Deve ser então analisado o gráfico do erro de estimativa, representado na Figura 14b, que mostra aumento do erro de estimativa ao longo do trajeto do corredor, expondo que os pontos de coincidência entre a posição real e a estimada não foram publicados no mesmo instante.

O erro de estimativa dessa odometria atingiu um pico de 0,8 m no final do corredor, diminuiu enquanto o robô percorria a curva final, e finalmente aumentou novamente atingindo um erro de 0,66 m no final do trajeto.

**Figura 14:** Resultado da odometria visual.

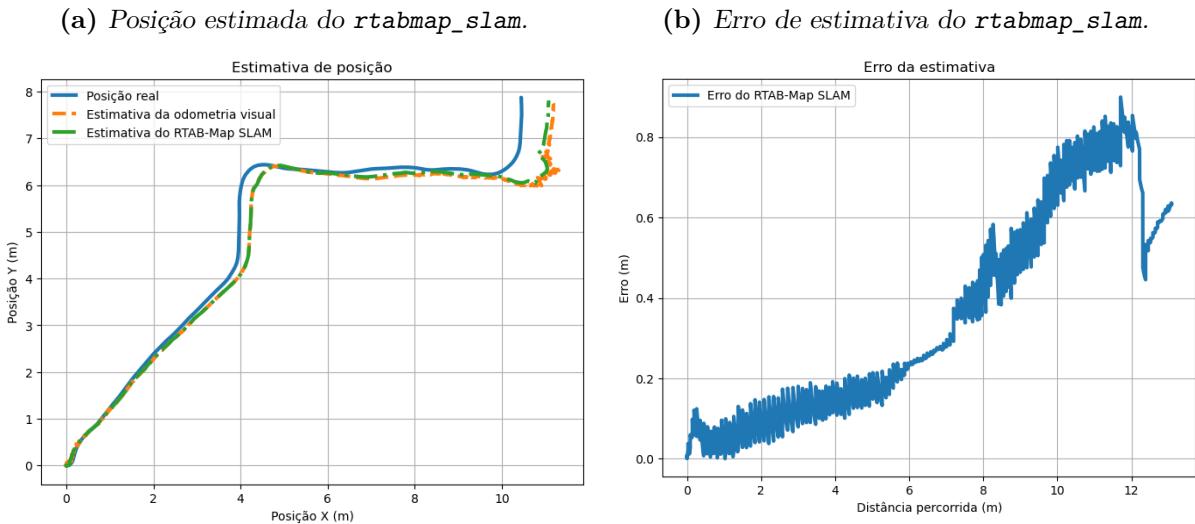


#### 4.2.2 Localização

Como a odometria visual apresentou um resultado satisfatório, ela foi utilizada em conjunto com os sistemas de localização, onde é testado o ajuste dos erros de odometria pelo sistema de localização. Portanto, foram criados novos gráficos, adicionando a posição global estimada pelos sistemas de localização. Essa posição é obtida através da transformada entre `map`, `odom` e `base_link`.

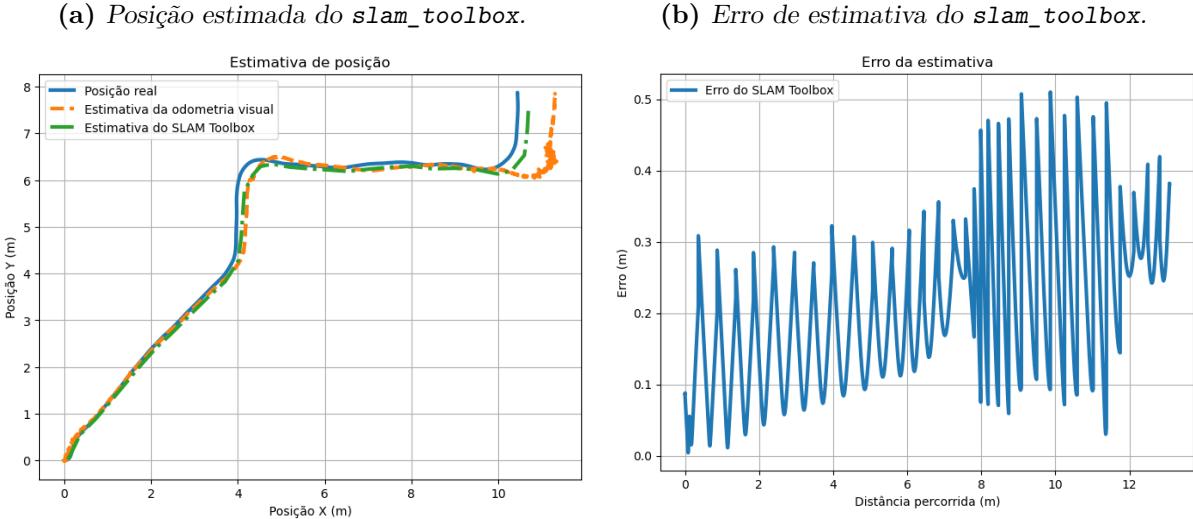
A Figura 15 mostra o resultado do sistema de localização do pacote `rtabmap_slam`. A localização neste caso aprimorou levemente a estimativa de posição, diminuindo o erro para 0,63 m na final do trajeto. É possível perceber pelo gráfico da Figura 15b que a curva do erro manteve a mesma forma da curva do erro da odometria visual. Porém, houve menos ruído na estimativa de posição durante a curva, como mostra a Figura 15a.

**Figura 15:** Resultado do pacote `rtabmap_slam`.



O `slam_toolbox`, que tem seu resultado representado na Figura 16, por outro lado, apresentou um resultado mais preciso, diminuindo o erro de estimativa para aproximadamente 0,37 m no final da trajetória. Isto aconteceu, em grande parte, porque não houve o acúmulo de erro durante o trajeto do corredor, como mostra a Figura 16b, que havia acontecido na odometria visual. A Figura 16a mostra que o ruído na estimativa de posição durante a curva foi atenuado da mesma forma que o `rtabmap_slam`.

**Figura 16:** Resultado do pacote `slam_toolbox`.



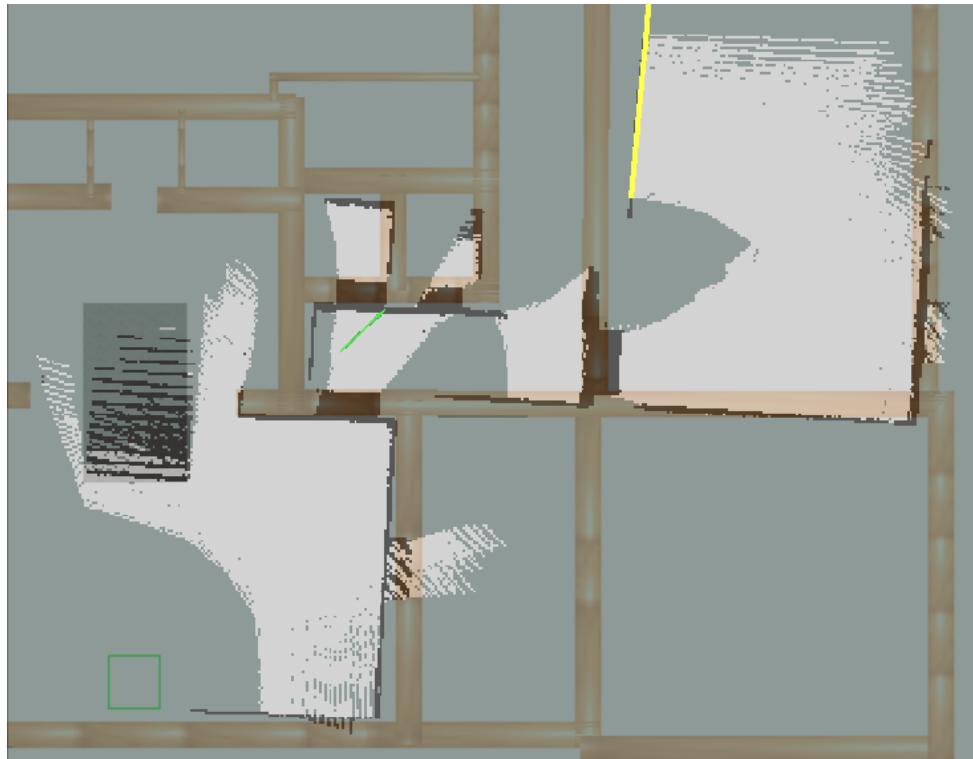
Nota-se que, apesar de utilizar a mesma entrada em todas execuções da simulação, a odometria visual apresentou resultados ligeiramente diferentes em cada execução. Isso acontece porque a máquina utilizada para a execução não possui capacidade de processamento suficiente para realizar os cálculos necessários por estes pacotes na frequência configurada. Na execução com o `textttslam_toolbox`, que exige alta capacidade de processamento, esse problema foi mais evidente. Os algoritmos de SLAM também não puderam ser executados na frequência desejada. Portanto, em máquinas mais potentes, a estimativa de posição pode ser mais precisa.

#### 4.2.3 Mapeamento SLAM

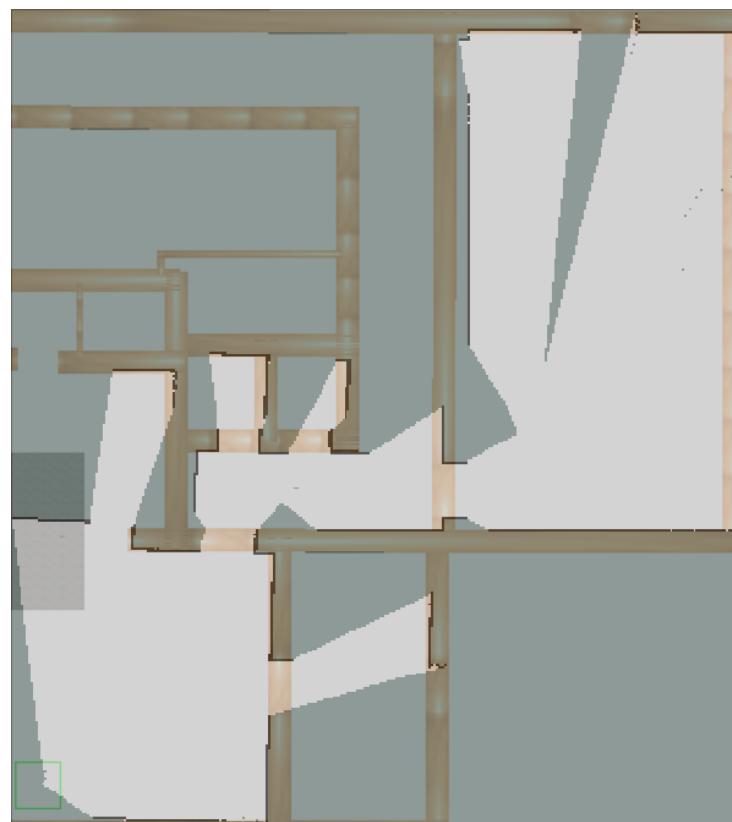
Como mencionado anteriormente, os dois sistemas de localização utilizados realizam o mapeamento simultaneamente com a localização. Portanto, tanto o `slam_toolbox`, quanto o `rtabmap_slam` geraram um mapa do ambiente durante a execução da simulação. Estes mapas podem ser utilizados para novas sessões de localização, ou em mapas de custo para planejamento de trajetória.

Na Figura 17, é mostrado o mapa gerado pelo `rtabmap_slam`, que foi criado utilizando os dados tridimensionais da câmera RGB-D, enquanto que na Figura 18 é mostrado o mapa produzido pelo `slam_toolbox`, que usa um feixe de luz plano, obtido através da conversão da imagem de profundidade da câmera. A representação do ambiente no Gazebo foi sobreposta aos mapas para facilitar a comparação. Nestes mapas, as áreas em branco são áreas livres, e as áreas em preto são consideradas ocupadas, enquanto que as áreas em cinza são desconhecidas.

**Figura 17:** Mapa construído pelo *rtabmap\_slam*.



**Figura 18:** Mapa construído pelo *slam\_toolbox*.



É possível perceber diferenças na geração destes mapas devido ao tipo de mensagem utilizado. O mapa gerado pelo *rtabmap\_slam* elabora um representação mais detalhada da

escada, percebendo-a desde sua base, enquanto que o `slam_toolbox` só mapeia a escada na mesma altura da câmera, que é a altura da mensagem passada a ela. Além disso, o `rtabmap_slam` mapeou as molduras das portas, que estão acima da altura da câmera. Este comportamento, porém, pode ser modificado, já que não são obstáculos relevantes para a navegação.

Outro diferença é no mapeamento do chão. Devido ao campo de visão da câmera, não é captado o chão em frente ao robô, que portanto não é mapeado pelo `rtabmap_slam`. Por outro lado, como o `slam_toolbox` utiliza um feixe de luz em duas dimensões, é considerado que todo espaço entre o robô e o primeiro obstáculo detectado é livre, mapeando assim o chão não captado pela câmera. Em situações em que não é detectado nenhum obstáculo dentro do limite da câmera, como a área entre o final da trajetória do robô e a porta da sala no canto superior direito da Figura 18, o chão não é mapeado.

Percebe-se, também, que o mapa gerado pelo `slam_toolbox` é mais fiel quanto a posição das paredes do ambiente real, já que o mapa gerado pelo `rtabmap_slam` possui uma inclinação que não corresponde ao ambiente real. Isso é devido a estimativa da posição do robô, apresentadas anteriormente, em que o `slam_toolbox` apresentou uma estimativa mais precisa.

## 5 DISCUSSÃO

Os resultados obtidos mostram que a utilização da câmera no robô Twil permite o mapeamento do ambiente em tempo-real, possibilitando a navegação autônoma do robô em locais desconhecidos ou com obstáculos dinâmicos. Isso foi realizado através da atualização do mapa de custos usado pelo planejador de trajetórias. Com uma camada *voxel*, responsável por detectar obstáculos tridimensionais no ambiente, e uma camada de inflação, que cria uma zona de segurança ao redor dos obstáculos, o robô é capaz de evitar colisões. Este sistema pode ser aprimorado com a utilização de outras camadas do mapa de custo, como a camada *Spatio Temporal Voxel Layer* (MACENSKI; TSAI; FEINBERG, 2020), que adiciona um decaimento temporal ao obstáculos detectados, já que eles podem ser temporários.

Além disso, a câmera aprimorou a estimativa de posição do robô, que permite o seguimento correto da trajetória planejada, recomendando-se a combinação de odometria visual com o sistema de localização `slam_toolbox` para esta aplicação.

A escolha da odometria é feita em razão do desempenho superior da odometria visual em relação a odometria das rodas. Porém, a odometria das rodas apresentou um resultado pior do que o esperado, podendo ser causado por erros na descrição do robô, o que pode ser ajustado. Além disso, a odometria visual pode apresentar resultados inferiores em ambientes reais, já que o teste em simulação é feito em condições ideais, sem a presença de ruído.

Logo, em trabalhos futuros, recomenda-se ajustes no sistema de odometria das rodas, para garantir uma estimativa mais precisa da posição do robô. O erro de orientação apresentado neste estimativo é grave, já que cria uma divergência sem limite. Para mitigar este erro, é possível utilizar as informações de velocidade angular e orientação do IMU presente no robô para calcular a estimativa de posição.

Isto pode ser feito com o pacote `robot_localization`, desfazendo-se dos dados da posição dada pela mensagem publicada pela odometria das rodas. Desta forma, a posição do robô seria calculada utilizando a velocidade linear das rodas e a orientação do IMU, possibilitando a obtenção de uma estimativa com menor erro. Porém, a velocidade publicada pela odometria das rodas deve ser modificada para ser em relação a base do robô, como é esperado pelo `robot_localization`, e não em relação ao sistema de coordenadas `odom`, como é publicada atualmente.

Com a odometria das rodas ajustada, ambos sistemas de odometria devem ser comparados novamente. Em caso de desempenho semelhante, recomenda-se a utilização da odometria das rodas por ser mais simples, desta forma, exigindo menor complexidade de processamento. Porém, caso a exigência computacional não seja um problema, ambas estimativas podem ser utilizadas em conjunto, fundidas através de um filtro de Kalman para obter uma pose mais precisa.

Em relação ao mapeamento, o pacote `rtabmap_slam` criou um mapa mais detalhado do ambiente, devido a utilização de dados tridimensionais. Porém, isto não afetou o

desempenho da navegação, com o `slam_toolbox` apresentando uma melhor estimativa de posição. Portanto, caso seja desejado um mapa do ambiente mais detalhado, recomenda-se a utilização do `rtabmap_slam`, em uma gravação, e não durante a navegação em tempo-real, da mesma forma que foi feito neste trabalho.

Além disso, devem ser realizados testes de SLAM que percorram um caminho que revisite áreas já vistas, para testar o fechamento de laço. Não foi possível testar isto neste trabalho, porque a execução da simulação em conjunto com os sistemas de odometria e localização exigiu muita capacidade de processamento, e os arquivos de dados, que permitem separar a execução destes sistemas, não são praticáveis em trajetórias longas, devido ao armazenamento necessário.

## 6 CONCLUSÕES

Este trabalho atingiu o objetivo proposto de mapear o ambiente para navegação autônoma utilizando a câmera RGB-D do robô Twil, para evitar colisões em ambientes dinâmicos ou pouco conhecidos. Isso foi feito através da atualização do mapa de custo utilizado pelo planejador de trajetórias, com a adição de uma camada *voxel* ao mapa de custo, responsável por detectar obstáculos tridimensionais no ambiente. Além disso, a câmera aprimorou a estimativa de posição do robô, que permite o seguimento correto da trajetória planejada, recomendando-se a combinação de odometria visual com o sistema de localização `slam_toolbox`. Porém, a navegação foi testada apenas em ambiente simulado, já que o sistema não foi implementado no robô real.

Em trabalhos futuros, os componentes aqui desenvolvidos podem ser aplicados no robô real. Para isso, o Gazebo deverá ser dispensado e os dados deverão todos ser obtidos através do mundo real. Porém, primeiramente devem ser realizados ajustes na odometria das rodas, como detalhado anteriormente.

Ao implementar o sistema no robô real, é possível que seja necessária uma recalibração dos sistemas de odometria e localização. Neste caso, recomenda-se a utilização dos *scripts* de gravação de *rosbags* produzidos neste trabalho para coleta e análise dos dados para calibração, já que os tópicos publicados pelo robô real e simulado devem ser os mesmos.

## REFERÊNCIAS

- AMAZON. *10 years of Amazon robotics: how robots help sort packages, move product, and improve safety.* 2022. Disponível em: <<https://www.aboutamazon.com/news/operations/10-years-of-amazon-robotics-how-robots-help-sort-packages-move-product-and-improve-safety>>. Acesso em: 4 ago. 2023.
- ATHAYDE, R. C. *Localização de robôs móveis no ROS.* 2021. Trabalho de Conclusão de Curso (Graduação em Engenharia de Controle e Automação) – Universidade Federal do Rio Grande do Sul, Porto Alegre.
- BORENSTEIN, J.; FENG, L. Measurement and correction of systematic odometry errors in mobile robots. *IEEE Transactions on Robotics and Automation*, v. 12, n. 6, p. 869–880, 1996.
- CHONG, T. et al. Sensor Technologies and Simultaneous Localization and Mapping (SLAM). *Procedia Computer Science*, v. 76, p. 174–179, 2015. 2015 IEEE International Symposium on Robotics and Intelligent Sensors (IEEE IRIS2015). ISSN 1877-0509.
- DURRANT-WHYTE, H.; BAILEY, T. Simultaneous localization and mapping: part I. *IEEE Robotics & Automation Magazine*, v. 13, n. 2, p. 99–110, 2006.
- GRISSETTI, G. et al. A Tutorial on Graph-Based SLAM. *IEEE Intelligent Transportation Systems Magazine*, v. 2, n. 4, p. 31–43, 2010. DOI: 10.1109/MITS.2010.939925.
- INTEL. *realsense-ros.* Santa Clara, CA, EUA: Intel. Disponível em: <<https://github.com/IntelRealSense/realsense-ros/>>. Acesso em: 05 de ago. de 2023.
- KALMAN, R. E. A New Approach to Linear Filtering and Prediction Problems. *Journal of Basic Engineering*, v. 82, n. 1, p. 35–45, mar. 1960. ISSN 0021-9223.
- LABBE, M. *Nav2 Overview.* 2023. Disponível em: <[https://wiki.ros.org/rtabmap\\_odom#rgbd\\_odometry](https://wiki.ros.org/rtabmap_odom#rgbd_odometry)>. Acesso em: 20 jan. 2024.
- LATOMBE, J. *Robot Motion Planning.* [S.l.]: Springer US, 2012. (The Springer International Series in Engineering and Computer Science). ISBN 9781461540229. Disponível em: <<https://books.google.com.br/books?id=nQ7aBwAAQBAJ>>.
- LU, D. V.; HERSHBERGER, D.; SMART, W. D. Layered costmaps for context-sensitive navigation. In: 2014 IEEE/RSJ International Conference on Intelligent Robots and Systems. [S.l.: s.n.], 2014. P. 709–715.
- MACENSKI, S. *SLAM Toolbox: The New Default SLAM Implementation for ROS2.* 2020. Disponível em: <<https://discourse.ros.org/t/slam-toolbox-the-new-default-slam-implementation-for-ros2/12369>>. Acesso em: 28 jan. 2024.
- MACENSKI, S.; JAMBRECIC, I. *SLAM Toolbox: SLAM for the dynamic world.* [S.l.: s.n.], jul. 2021. Disponível em: <<https://doi.org/10.21105/joss.02783>>.

- MACENSKI, S.; MARTIN, F. et al. The Marathon 2: A Navigation System. In: 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). [S.l.]: IEEE, out. 2020.
- MACENSKI, S.; TSAI, D.; FEINBERG, M. Spatio-temporal voxel layer: A view on robot perception for the dynamic world. *International Journal of Advanced Robotic Systems*, v. 17, n. 2, p. 1729881420910530, 2020.
- MACENSKI, S.; FOOTE, T. et al. Robot Operating System 2: Design, architecture, and uses in the wild. *Science Robotics*, v. 7, n. 66, eabm6074, 2022.
- MARDER-EPPSTEIN, E. et al. The Office Marathon: Robust navigation in an indoor office environment. In: 2010 IEEE International Conference on Robotics and Automation. [S.l.: s.n.], 2010. P. 300–307.
- MEEUSSE, W. REP 105. 2010. Disponível em: <<https://www.ros.org/reps/rep-0105.html>>. Acesso em: 29 jan. 2024.
- MERZLYAKOV, A.; MACENSKI, S. *A Comparison of Modern General-Purpose Visual SLAM Approaches*. [S.l.: s.n.], 2021. arXiv: 2107.07589 [cs.RO].
- MOORE, T.; STOUCH, D. A Generalized Extended Kalman Filter Implementation for the Robot Operating System. In: PROCEEDINGS of the 13th International Conference on Intelligent Autonomous Systems (IAS-13). [S.l.]: Springer, jul. 2014.
- NAVIGATION2. Nav2 Overview. 2020. Disponível em: <<https://navigation.ros.org/index.html>>. Acesso em: 20 jan. 2024.
- PETRY, G. R. *Navegação de um robô móvel em ambiente semi-estruturado*. 2019. Trabalho de Conclusão de Curso (Graduação em Engenharia de Controle e Automação) – Universidade Federal do Rio Grande do Sul, Porto Alegre.
- SIEGWART, R.; NOURBAKHSH, I. R.; SCARAMUZZA, D. *Introduction to Autonomous Mobile Robots*. 2. ed. Cambridge, MA, EUA: The MIT Press, fev. 2011. ISBN 9780262015356.
- STATISTA. *Size of the global market for autonomous mobile robots (AMR) from 2016 to 2021, with a forecast through 2028*. 2023. Disponível em: <<https://www.statista.com/statistics/1285835/worldwide-autonomous-robots-market-size/>>. Acesso em: 4 ago. 2023.
- ZHENG, K. *ROS Navigation Tuning Guide*. [S.l.: s.n.], 2019. arXiv: 1706.09068 [cs.RO].