

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
2023

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
2023

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. Renato Ventura Bayan Henriques, UFRGS

Doutor pela Universidade Federal de Minas Gerais – Belo Horizonte, Brasil

Profa. Dra. Rafael Antônio Comparsi Laranja, UFRGS

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

Alceu Heinke Frigeri

Coordenador de Curso

Eng. de Controle e Automação

Porto Alegre, Agosto 2023

RESUMO

Neste trabalho é apresentado o embasamento teórico e o planejamento da implementação de mapeamento de ambientes a um sistema de navegação autônomo, com intuito de permitir o planejamento de trajetórias em um ambiente dinâmico ou pouco estruturado. Para obter este objetivo, será utilizado o Robot Operating System (ROS) 2 em conjunto com o *Navigation 2*, que será configurado para simular no Gazebo o robô móvel Twil, equipado com uma câmera de profundidade para mapeamento e localização. Após a verificação do sistema no ambiente simulado, será feita a adaptação para o robô real.

Palavras-chave: Robótica, Navegação autônoma, Mapeamento de ambientes.

LISTA DE ILUSTRAÇÕES

1	Mercado global de robôs autônomos de 2016 a 2021, com projeção até 2028.	7
2	Exemplo de árvore de comportamento de um robô manipulador móvel.	10
3	Exemplo de configuração de camadas de um mapa de custo.	12
4	Arquitetura do <i>Navigation2</i>	13
5	Planta do 1º andar do prédio Centenário da EE-UFRGS.	15
6	Ambiente Gazebo com modelo do prédio Centenário da EE-UFRGS. . .	15
7	RViz com o robô Twil no ambiente do prédio Centenário da EE-UFRGS.	16
8	Comparação entre curvas de inflação com inclinações baixa e alta.	18

LISTA DE TABELAS

1	Símbolos dos nós de uma árvore de comportamento.....	10
---	--	----

LISTA DE ABREVIATURAS

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

SUMÁRIO

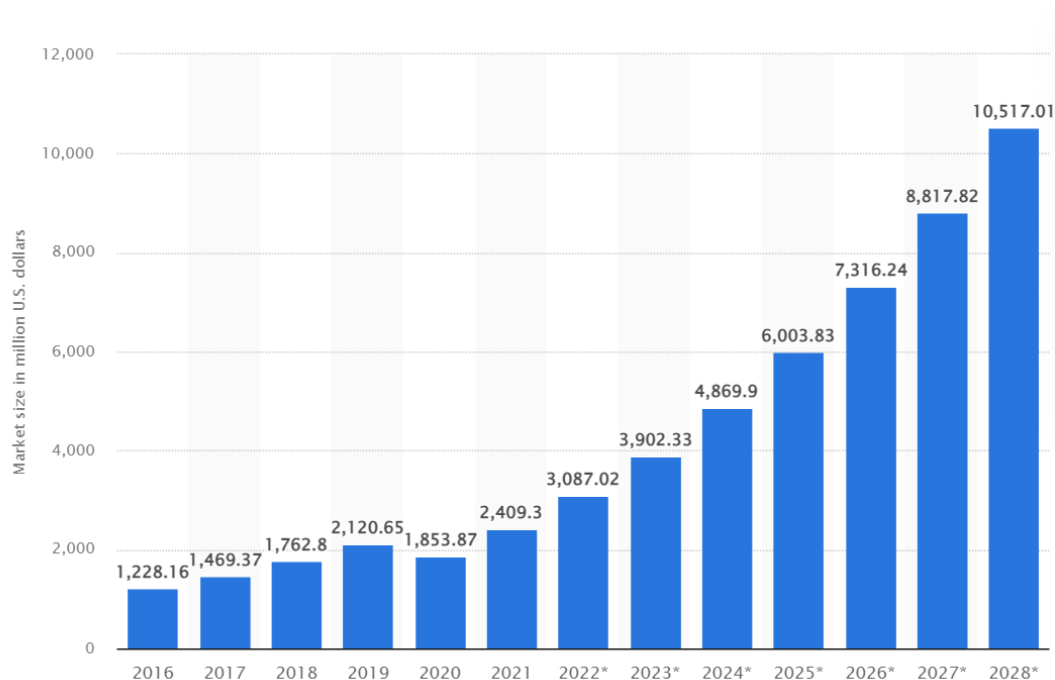
1	INTRODUÇÃO	7
2	REVISÃO DA LITERATURA.....	9
2.1	Robot Operating System 2 (ROS 2).....	9
2.2	Árvores de comportamento	9
2.3	Mapeamento	11
2.3.1	Mapas de custo	11
2.3.2	Sensores e SLAM.....	11
2.4	Navigation2	12
3	METODOLOGIA	14
3.1	Configuração do robô	14
3.2	Ambiente de simulação.....	14
3.3	Configuração do Nav2.....	16
3.3.1	Localização.....	16
3.3.2	Mapeamento	17
3.3.3	Planejamento de trajetórias.....	17
4	CONCLUSÃO	19
	REFERÊNCIAS	20

1 INTRODUÇÃO

Robótica deve seu maior sucesso a indústria de manufatura, onde são utilizados principalmente robôs manipuladores (SIEGWART; NOURBAKHSH; SCARAMUZZA, 2011). Porém, estes robôs têm como limitação sua mobilidade, incapazes de se movimentar pela planta, limitando 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 robô Proteus, da Amazon (AMAZON, 2022). Eles tem como desafio a navegação em ambientes 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, é proposto um sistema de navegação autônomo que utiliza sensores para mapear o ambiente em

conjunto com algoritmos de planejamento de trajetórias para permitir a navegação sem colisões em ambientes previamente desconhecidos ou dinâmicos.

Este robô já foi utilizado em trabalhos anteriores de conclusão de curso, como em Petry (2019) e Athayde (2021). Porém, devido ao avanço do campo da robótica, ferramentas utilizadas nestes trabalhos foram substituídas por novas versões, que implementam técnicas modernas que serão abordadas ao longo deste trabalho. 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 substituí o *Navigation Stack* do ROS 1.

2 REVISÃO DA LITERATURA

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. A implementação do *middleware Data Distribution Service* (DDS), que também é utilizado em sistemas de infraestrutura crítica, como aplicações militares, aeroespaciais e financeiras, lhe confere ótima 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 publica uma mensagem e outros nós podem se inscrever para receber essa mensagem. *Services* funcionam de forma cliente servidor, utilizando o padrão de requisição e resposta.

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 responde periodicamente com o estado atual da tarefa. No término, é enviado o resultado, podendo ser sucesso ou falha.

No ROS 1, *actions* eram implementados em uma biblioteca separada, mas, devido a sua importância na *navigation stack* do ROS 1 e no pacote *MoveIt!*, têm suporte nativo no ROS 2 (GEOFFREY BIGGS JACOB PERRON, 2020). 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 ÁRVORES DE COMPORTAMENTO

Árvores de comportamento, em inglês *behavior trees* (BT), foram desenvolvidas na indústria de jogos para aplicação em inteligência artificial de personagens não jogáveis, substituindo máquinas de estado. Elas se destacam por sua modularidade e reatividade, porém, mantendo as funcionalidades esperadas de uma máquina de estado (COLLEDANCHISE; ÖGREN, 2018).

O funcionamento de uma árvore de comportamento ocorre através de uma série de sinais enviados aos nós de uma árvore com uma frequência fixa. Este nó responde com o estado atual da execução, que pode ser *running*, se está em execução, *success*, se atingiu o objetivo, ou *failure* nos demais casos. Na formulação clássica, existem quatro categorias

de nós de controle (*Sequence*, *Fallback*, *Parallel* e *Decorator*) e duas categorias de nós de execução (*Action* e *Condition*). A Tabela 1 mostra os símbolos utilizados para representar os nós de uma árvore de comportamento.

Tabela 1: Símbolos dos nós de uma árvore de comportamento.

Tipo de nó	Símbolo
<i>Fallback</i>	$\boxed{?}$
<i>Sequence</i>	$\boxed{\rightarrow}$
<i>Parallel</i>	$\boxed{\Rightarrow}$
<i>Action</i>	$\boxed{\text{texto}}$
<i>Condition</i>	$\boxed{\text{texto}}$
<i>Decorator</i>	\diamond

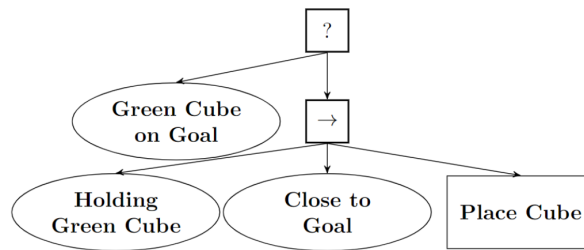
Fonte: Elaborado pelo autor

O resultado dos nós de controle dependem dos resultados de seus nós filhos. Por exemplo, o nó *Sequence* executa seus filhos em ordem até encontrar um nó que retorna *failure* ou *running*. Caso não encontre, retorna *success*. O nó *Fallback* funciona de forma semelhante porém procura filhos que retornem *success* ou *running*, só retornando *Failure*, caso contrário. O nó *Parallel*, executa todos os filhos em paralelo, com o resultado dependendo do estado de execução dos filhos. O nó *Decorator* modifica o resultado de um nó filho de acordo com uma regra definida pelo usuário.

O nó de execução *Action* executa um comando, e retorna o resultado final deste comando, como *sucess* caso o objetivo seja atingido, ou *failure*, caso contrário. Enquanto a tarefa está sendo executada, o nó retorna *running*. Finalmente, o nó *Condition* testa uma condição e retorna *success* ou *failure*, se o resultado for verdadeiro ou falso, respectivamente.

Estes nós podem ser combinados para facilmente criar comportamentos procurados em robôs, como mostra a Figura 2, que descreve a operação de *pick-and-place* de um robô manipulador móvel, utilizando os nós *Fallback*, *Condition*, *Sequence* e *Action*.

Figura 2: Exemplo de árvore de comportamento de um robô manipulador móvel.



Fonte: Colledanchise e Ögren (2018).

Como permitem a construção destes comportamentos de forma visual, sem a necessidade de programação, árvores de comportamento estão sendo cada vez mais utilizadas em projetos de robótica. Exemplos de produtos reais são o robô JIBO e o projeto iQmatic da Scania, que utiliza árvores de comportamento no sistema de navegação de caminhões autônomos (COLLEDANCHISE; ÖGREN, 2018).

2.3 MAPEAMENTO

Para navegação autônoma, o robô deve ter conhecimento prévio do ambiente para planejamento de trajetórias. Existem diversas formas de representação do ambiente, como mapas de gradientes, mapas de custo e vetores de espaços. Neste trabalho, o foco será no mapa de custo.

O mapeamento também auxilia na localização do robô, comparando o mapa construído com os dados dos sensores em tempo real. Além disso, os dados dos sensores podem ser utilizados para atualizar o mapa de custo, em casos de ambientes pouco conhecidos ou dinâmicos.

É possível utilizar os dados de localização e dos sensores para construir um novo mapa. Esta técnica é conhecida como *Simultaneous Localization and Mapping (SLAM)*, que permite a criação de mapas para ambientes pouco ou não conhecidos.

2.3.1 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.

Em mapas de custo tradicionais, seus dados são armazenadas em mapas monolíticos, para utilização em planejamento de trajetórias. Esta implementação é utilizada com sucesso para caminhos curtos, mas pode apresentar dificuldade em lidar com ambientes dinâmicos maiores (LU; HERSHBERGER; SMART, 2014).

Uma solução para este problema são mapas de custo com camadas, que separam o processamento dos dados dos mapas de custos em camadas semanticamente distintas. Por exemplo, os dados dos sensores e o mapa estático previamente conhecido são processados em camadas separadas e depois combinados em um único mapa de custo. A Figura 3 mostra uma configuração possível de camadas de mapas de custo.

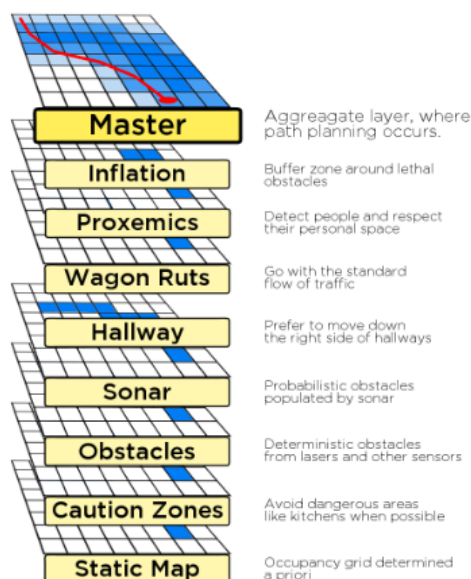
2.3.2 Sensores e SLAM

A escolha do sensor é importante para o mapeamento, pois afeta a qualidade e quantidade de informações obtidas pelo robô, além de determinar a escolha das ferramentas utilizadas para o mapeamento do ambiente (CHONG et al., 2015).

Sensores acústicos, como sonares e sensor de distância a *laser*, são utilizados em ferramentas SLAM 2D tradicionais. Estes sistemas são robustos e bem estabelecidos, com fácil integração ao sistema de navegação do ROS 2.

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 localização pura. Em Merzlyakov e Macenski (2021), é feita uma comparação entre estes sistemas, mostrando que o OpenVSLAM é a técnica mais adequada para maioria dos casos. Contudo, para ambientes internos com câmeras RGB-D, o RTABMap também teve um bom desempenho. Estes sistemas, porém, não são integrados nativamente ao *Navigation2*.

Figura 3: Exemplo de configuração de camadas de um mapa de custo.



Fonte: Lu, Hershberger e Smart (2014).

2.4 NAVIGATION2

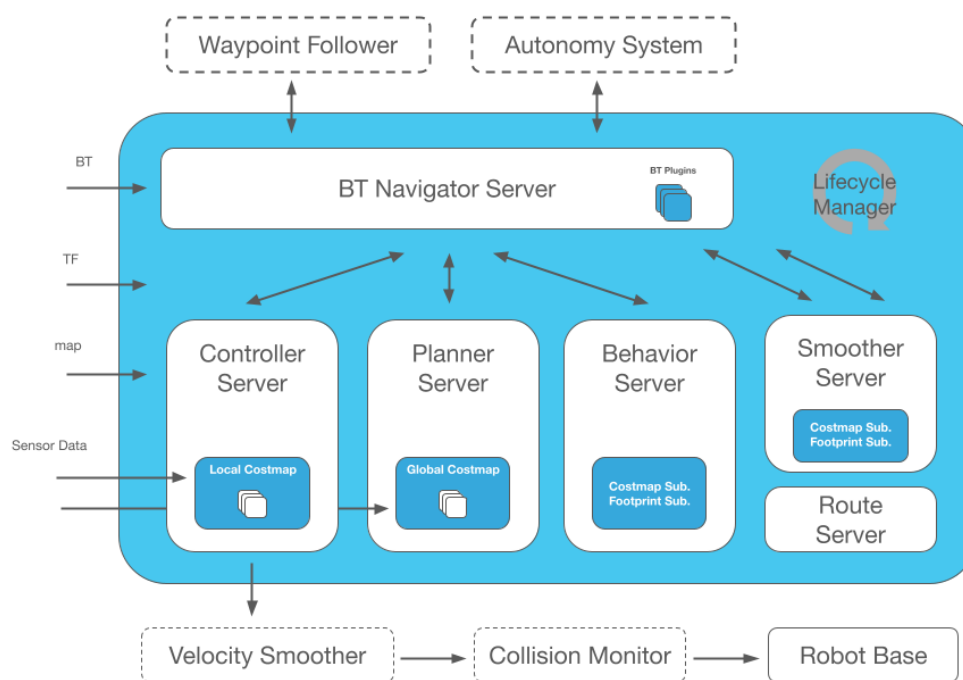
O *Navigation2* (Nav2) é o sucessor do ROS *navigation stack*, permitindo a realização de tarefas complexas em diversos ambientes e classes de robôs cinemáticos. Baseando-se no legado do *navigation stack* do ROS 1, o Nav2 foi construído em cima do ROS2, implementando técnicas mais modernas para ter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

Na Figura 4 é mostrada a arquitetura do Nav2. O *Behavior Tree (BT) Navigator Server* usa uma árvore de comportamento para orquestrar as tarefas de navegação, ativando os servidores de controle, planejamento e recuperação para navegar. 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.

Além disso, todos estes servidores utilizam o conceito de *Managed Nodes*, também conhecidos como *Lifecycle Nodes*. Estes nós utilizam máquinas de estados para gerenciar seu ciclo de vida, utilizando transições de estado desde sua criação a destruição. No caso de falha ou desligamento, o nó vai do estado ativo ao estado finalizado, seguindo a máquina de estados, permitindo que o sistema seja interrompido de forma segura.

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

Figura 4: *Arquitetura do Navigation2.*



Fonte: Navigation2 (2020).

3 METODOLOGIA

Neste capítulo será explicado o desenvolvimento do trabalho, descrevendo o estado atual do projeto, além de ajustes e adições planejadas para atingir os objetivos propostos. A maior parte do desenvolvimento do trabalho será realizada no pacote `twil`, composto por diversos pacotes menores, que definem as funcionalidades do robô Twil. Este pacote tem como dependências pacotes desenvolvidos dentro da Universidade Federal do Rio Grande do Sul (UFRGS), além de pacotes de terceiros, como ROS 2 e o Nav2. Todos os pacotes modificados neste trabalho estão disponíveis em um repositório git no servidor do Laboratório de Robótica da UFRGS, onde foi criada uma *branch* para o desenvolvimento deste projeto.

3.1 CONFIGURAÇÃO DO ROBÔ

O modelo do robô está descrito no pacote `twil_description`, que contém os arquivos de descrição do robô no formato XACRO, que é compilado para o formato URDF, utilizado pelo ROS 2. A utilização do XACRO permite a criação de parâmetros, que são utilizados nos arquivos de *launch* para habilitar e desabilitar partes do robô. Além da representação física, nestes arquivos são incluídos *plugins* do Gazebo que simulam os sensores do robô.

O Twil está equipado com a câmera RGB-D Intel RealSense D435. Para sua utilização é necessária a instalação do pacote `realsense-ros` (INTEL, s.d.), que contém os arquivos de descrição da câmera, além da definição das mensagens publicadas pela mesma. Para a simulação, foi adicionado o *plugin* do Gazebo `camera_plugin` ao arquivo de descrição do robô. Este *plugin* publica mensagens do tipo `sensor_msgs/PointCloud2`, que serão utilizadas para o mapeamento do ambiente.

Outros sensores como sonares, *encoders*, unidades de medição inercial (IMU, em inglês), além de outros, estão presentes no robô real e podem ser utilizados para auxiliar na navegação.

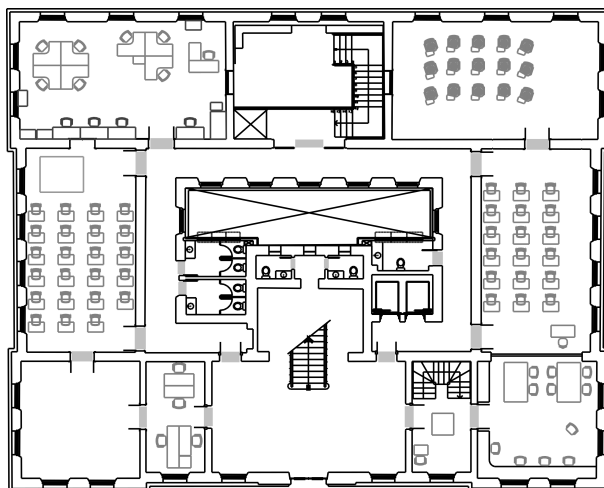
O pacote `twil_bringup`, contém arquivos de configuração dos controladores, além de arquivos de *launch* para teste de movimentação do robô. Os controladores estão presentes nos pacotes `linearing_controllers` e `non_smooth_backstep_controller`. Estes controladores também são responsáveis pela publicação de informações de odometria no tópico `/odom`, utilizadas para localização durante a navegação.

3.2 AMBIENTE DE SIMULAÇÃO

A verificação do funcionamento do sistema de navegação será feita em uma simulação em 3D do prédio Centenário da Escola de Engenharia da UFRGS. Utilizando o *Building*

Editor do Gazebo, foi criado um modelo em 3D do prédio, tendo como base a planta do 1º andar, mostrada na Figura 5. No pacote `ufrgs_map` está incluído uma representação em formato PGM desta imagem, para utilização na camada estática do mapa de custo.

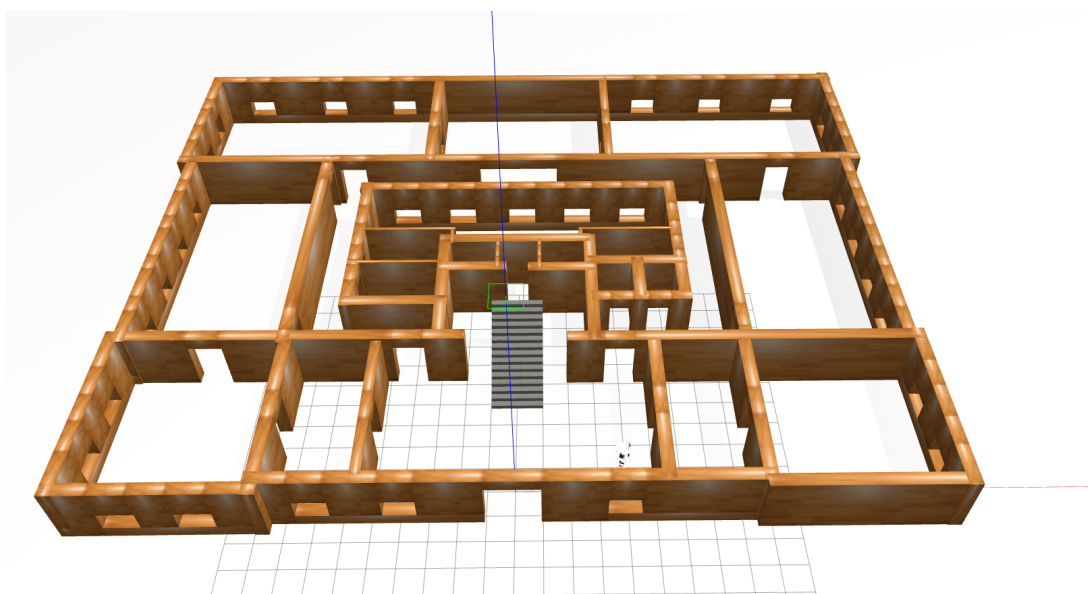
Figura 5: Planta do 1º andar do prédio Centenário da EE-UFRGS.



Fonte: Petry (2019).

O modelo em 3D, mostrado na Figura 6, faz parte do pacote `ufrgs_gazebo`, com um arquivo *world* do Gazebo, para utilizar como ambiente de testes de navegação. Obstáculos podem ser adicionados durante a simulação, como mesas e cadeiras, para testar a capacidade do robô de evitar colisões.

Figura 6: Ambiente Gazebo com modelo do prédio Centenário da EE-UFRGS.



Fonte: Autor.

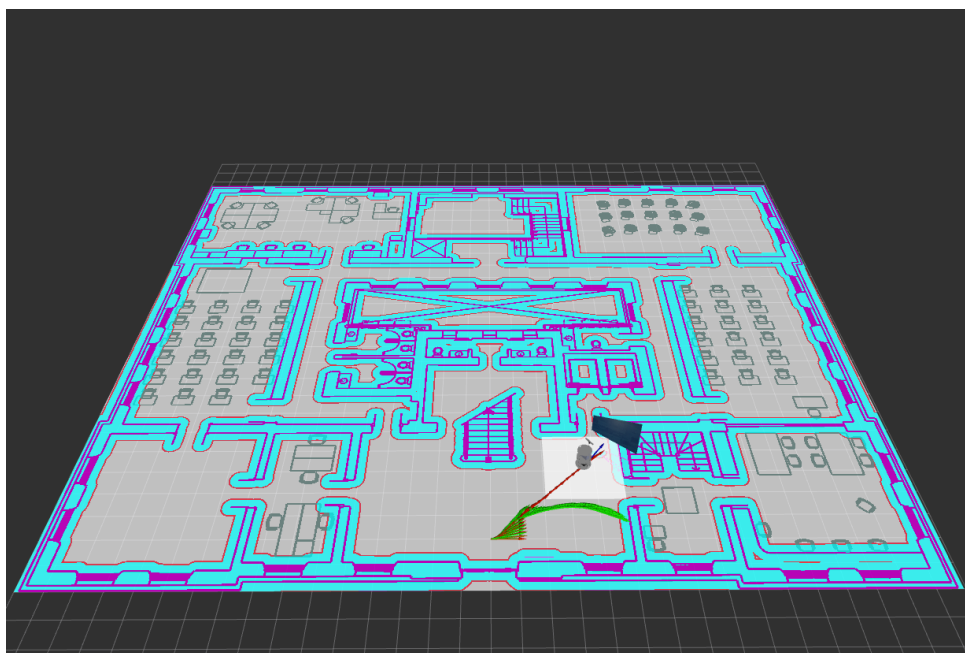
3.3 CONFIGURAÇÃO DO NAV2

A navegação será realizada utilizando o *Navigation 2*. Através do *software* de visualização de robôs do ROS, o RViz, é enviada uma mensagem do tipo `NavigateToPose`, com o destino desejado. O Nav2 recebe esta mensagem e, utilizando uma árvore de comportamento, orquestra as tarefas de navegação, ativando os servidores de controle, planejamento e recuperação para navegar até o ponto de destino. Porém, antes de utilizar este sistema, ele deve ser configurado para o robô Twil.

3.3.1 Localização

Primeiramente, deve ser configurado o sistema de localização do robô. Atualmente, está sendo utilizado os dados publicados pelo controlador no tópico `/odom` para estimar a posição do robô. Na Figura 7 é mostrado o robô Twil no RViz, com a representação gráfica da trajetória real, em verde, e a trajetória estimada pelo controlador, em vermelho. Nota-se que a posição esperada diverge da real mais do que o esperado, portanto, devem ser realizados ajustes nos parâmetros do controlador.

Figura 7: RViz com o robô Twil no ambiente do prédio Centenário da EE-UFRGS.



Fonte: Autor.

Para melhorar a acurácia da localização, é possível utilizar outras fontes de dados e, utilizando o pacote `robot_localization`, fundir os dados através de um filtro de Kalman. Desta forma, as desvantagens de cada sensor podem ser compensadas pelos outros sensores. Sensores inercias são comumente usados para esta finalidade. Utilizando o *plugin* IMU do Gazebo, ele pode ser adicionado à descrição do robô, publicando mensagens do tipo `sensor_msgs/Imu`.

Deve ser implementada, também, a odometria visual, utilizando uma câmera RGB-D. A odometria visual utiliza o mapa do ambiente e os dados dos sensores para estimar

a posição do robô. Os pacotes `nav2_amcl`, `slam_toolbox`, `ORB-SLAM3` e `OpenVSLAM` são capazes de realizar esta funcionalidade. Os dois primeiros estão integrados ao Nav2, porém foram criados para utilizar sensores LIDAR e, portanto, necessitam de mensagens do tipo `LaserScan`. A câmera RGB-D, por outro lado, publica mensagens do tipo `PointCloud2`, logo, é necessário converter estas mensagens para o tipo `LaserScan` para a utilização destes pacotes.

Em contrapartida, os pacotes `ORB-SLAM3` e `OpenVSLAM` foram criados para utilizar câmeras RGB-D. Entretanto, estes sistemas não estão integrados ao Nav2 e, devido ao caráter recente das técnicas de VSLAM, não são tão robustos quanto os pacotes anteriores. Desta forma, serão implementados ambos sistemas, para que seja possível comparar os resultados de cada um.

A localização seguirá a norma REP 105 (MEEUSSE, 2010), utilizada pelo Nav2. Ela define três sistemas de coordenadas: `map` e `odom`, fixos ao ambiente e o `base_link`, fixo a base do robô. A pose do robô em relação ao `odom`, deve ser contínua, porém pode divergir gradualmente, enquanto a pose do robô em relação ao `map` deve ser precisa, mas não é contínua. Logo, o `odom` é utilizado para referência local a curto prazo e o `map` é utilizado para referência global a longo prazo.

3.3.2 Mapeamento

A representação do ambiente no Nav2 é feita através de um mapa de custo. As camadas do mapa são definidas através de *plugins* que permitem diversas fontes de dados para a construção do mapa. Neste trabalho, serão utilizadas as camadas: `static_layer`, `voxel_layer`, `obstacle_layer` e `inflation_layer`, implementadas através de *plugins* do Nav2.

A camada estática, gerada pelo *plugin* `static_layer`, é utilizada para representar o mapa fixo do ambiente. Neste trabalho, será utilizada a planta do 1.º andar do prédio Centenário da EE-UFRGS, representada na Figura 5. Posteriormente, utilizando sistemas de SLAM, podem ser criados mapas atualizados para utilização nesta camada.

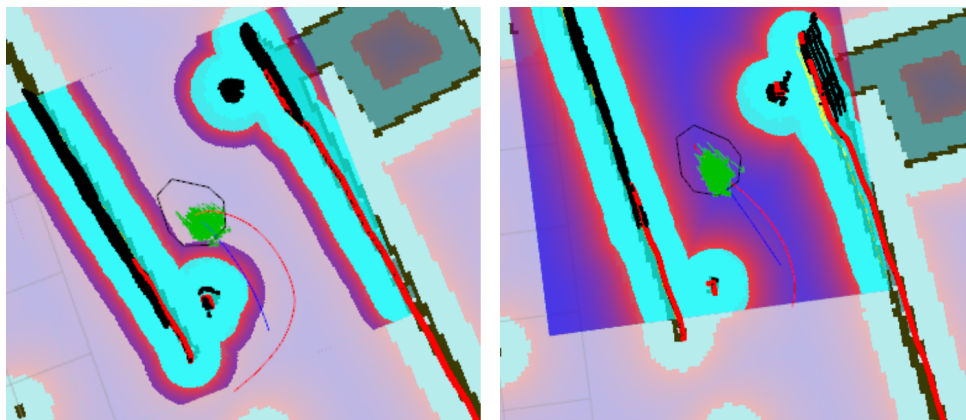
As camadas `voxel_layer` e `obstacle_layer` utilizam dados dos sensores para atualizar dinamicamente o mapa. A diferença entre estas camadas é que a `voxel_layer` utiliza dados 3D ao invés de 2D, como a `obstacle_layer`. Ambas podem utilizar mensagens do tipo `PointCloud2`, então não será necessária a conversão dos dados provenientes da câmera RGB-D. A inclusão destas camadas é fundamental para evitar colisões com obstáculos ausentes no mapa estático.

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 curva de inflação tenha uma inclinação baixa, para cobrir corredores inteiros. Desta forma, haveria preferência por caminhos que passem o mais longe possível dos obstáculos, como mostra a Figura 8.

3.3.3 Planejamento de trajetórias

Finalmente, é necessário configurar o planejamento de trajetórias. Diversos *plugins* estão disponíveis no Nav2, como o `NavFnPlanner`, `Smac Planner 2D` e `Theta Star Planner`. Neste trabalho, foi escolhido o `NavFnPlanner`, que pode utilizar os algoritmos

Figura 8: Comparação entre curvas de inflação com inclinações baixa e alta.



Fonte: Zheng (2019).

Dijkstra (DIJKSTRA, 1959) ou A* (HART; NILSSON; RAPHAEL, 1968). Este é o planejador recomendado pelos autores do *Navigation 2* para robôs circulares.

4 CONCLUSÃO

O planejamento realizado neste trabalho evidenciou a modularidade do ROS 2, que se tornou uma grande vantagem durante o projeto. Um exemplo disso é a utilização de diversos pacotes de terceiros, aproveitando o trabalho de outros desenvolvedores. Além disso, o desenvolvimento pôde ser realizado gradualmente, isolando cada funcionalidade, facilitando a comparação de resultados de técnicas diferentes, como será efetuado com os pacotes de SLAM e VSLAM. Apesar de não constar na metodologia, em caso de resultados satisfatórios, o sistema de navegação pode ser aplicado ao robô real, com pequenas modificações. Finalmente, para facilitar a reprodução dos resultados deste trabalho, pode ser criada uma imagem Docker com todos os pacotes necessários para a execução do projeto, simplificando a instalação e configuração do ambiente.

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: 8 abr. 2023. Acesso em: 04 de ago. de 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.
- 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. DOI: <https://doi.org/10.1016/j.procs.2015.12.336>. Disponível em: <<https://www.sciencedirect.com/science/article/pii/S1877050915038375>>.
- COLLEDANCHISE, M.; ÖGREN, P. *Behavior Trees in Robotics and AI*. Boca Raton, FL, EUA: CRC Press, jul. 2018. DOI: 10.1201/9780429489105. Disponível em: <<https://doi.org/10.1201/9780429489105>>.
- DIJKSTRA, E. W. A note on two problems in connexion with graphs. *Numerische mathematik*, Springer, v. 1, n. 1, p. 269–271, 1959.
- GEOFFREY BIGGS JACOB PERRON, S. L. *Actions*. 2020. Disponível em: <<https://design.ros2.org/articles/actions.html>>. Acesso em: 10 de ago. de 2023.
- HART, P.; NILSSON, N.; RAPHAEL, B. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics*, Institute of Electrical e Electronics Engineers (IEEE), v. 4, n. 2, p. 100–107, 1968. DOI: 10.1109/tssc.1968.300136. Disponível em: <<https://doi.org/10.1109/tssc.1968.300136>>.
- INTEL. *realsense-ros*. Santa Clara, CA, EUA: Intel. Disponível em: <<https://github.com/IntelRealSense/realsense-ros/>>. Acesso em: 05 de ago. de 2023.
- 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. DOI: 10.1109/IRoS.2014.6942636.
- 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. DOI: 10.1109/iros45743.2020.9341207. Disponível em: <<https://doi.org/10.1109/2Firos45743.2020.9341207>>.

- 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. DOI: 10.1126/scirobotics.abm6074. Disponível em: <<https://www.science.org/doi/abs/10.1126/scirobotics.abm6074>>.
- MEEUSSE, W. *REP 105*. 2010. Disponível em: <<https://www.ros.org/repos/rep-0105.html>>. Acesso em: 8 abr. 2023. Acesso em: 05 de ago. de 2023.
- MERZLYAKOV, A.; MACENSKI, S. *A Comparison of Modern General-Purpose Visual SLAM Approaches*. [S.l.: s.n.], 2021. arXiv: 2107.07589 [cs.R0].
- NAVIGATION2. *Nav2 Overview*. 2020. Disponível em: <<https://navigation.ros.org/index.html>>. Acesso em: 30 de jul. de 2023.
- 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: 8 abr. 2023. Acesso em: 04 de ago. de 2023.
- ZHENG, K. *ROS Navigation Tuning Guide*. [S.l.: s.n.], 2019. arXiv: 1706.09068 [cs.R0].