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 Universidade Federal do Rio Grande do Sul como parte dos requisitos para a obtenção do título de *Bacharel 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

Palavras-chave: Automação e Controle, Robótica.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE LISTAGENS

LISTA DE ABREVIATURAS

BT	Behavior Tree
RGB-D	Red Green Blue - Depth (vermelho verde azul - profundidade)
ROS	Robot Operating System
SLAM	Simultaneous Localization and Mapping
UFRGS	Universidade Federal do Rio Grande do Sul

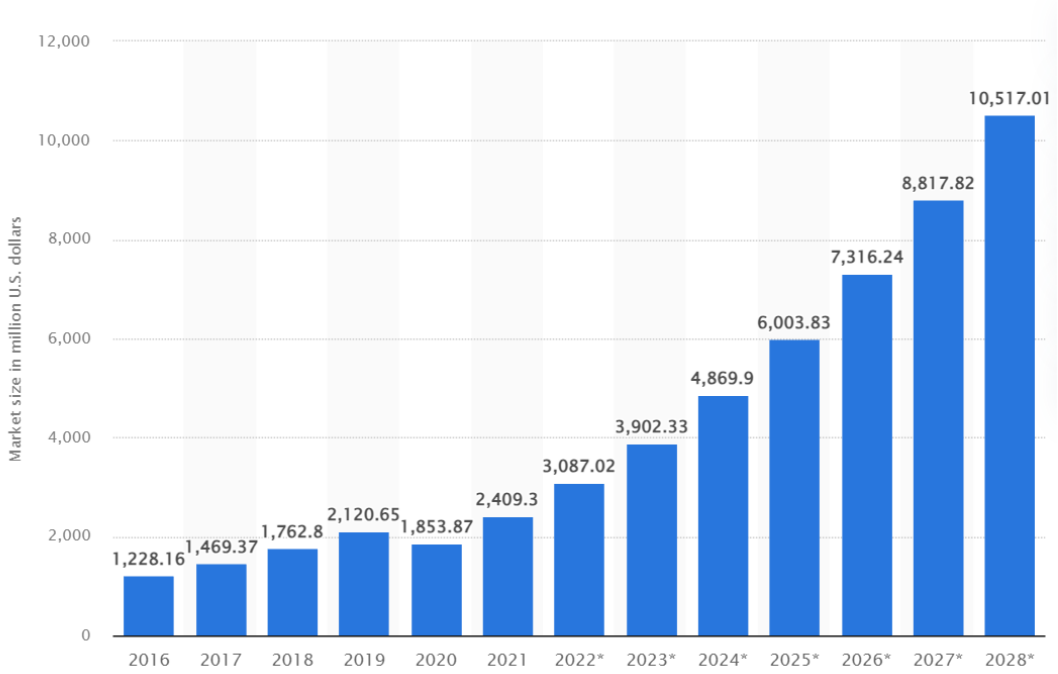
SUMÁRIO

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 movimentar pelo seu ambiente de trabalho, aumentando a gama de tarefas que podem ser realizadas.

Por esta razão, o mercado de robôs móveis 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). Esta categoria de robôs, porém, tem como desafio a navegação em ambientes dinâmicos, muitas vezes compartilhados com humanos. Portanto, é necessário um robô capaz 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 de percepção e um sistema de planejamento de trajetórias para navegar em ambientes dinâmicos ou pouco conhecidos.

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 de robótica móvel, novas ferramentas foram desenvolvidas, como o ROS 2 e o *Navigation2*, que utilizam técnicas modernas que serão abordadas ao longo deste trabalho.

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. É baseado no padrão Data Distribution Service (DDS), que é um padrão de comunicação utilizado em sistemas de infraestrutura crítica, como sistemas militares e financeiros (macenski; foote et al., 2022).

Uma mudança relevante a este trabalho do ROS 2 é nos padrões de comunicação. 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. *Topics* e *services* já existiam no ROS 1.

Actions, por outro lado, são únicos ao ROS 2. Este padrão de comunicação é utilizado para tarefas de longa duração, em que o cliente envia uma requisição e o servidor responde com um *feedback* periódico durante a execução, além do resultado da tarefa ao seu término, podendo ser falha ou sucesso. Durante a execução, é possível cancelar a tarefa.

Actions podem ser usados, por exemplo, em tarefas de navegação, em que um *action client* envia uma requisição com um ponto de destino para um *action server* que responde com um *feedback* contínuo da posição atual do robô e com o resultado final. Sua definição a torna apropriada na 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
Fallback	?
Sequence	→
Parallel	⇉
Action	texto
Condition	texto
Decorator	◊

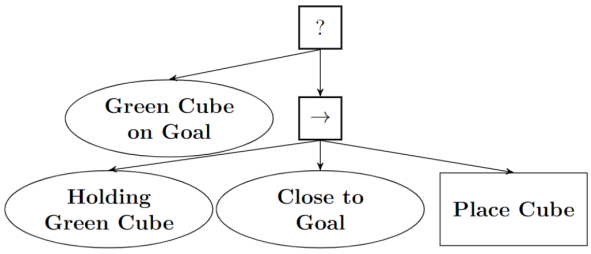
Fonte: Elaborado pelo autor

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* caso a condição seja verdadeira ou falsa, respectivamente.

Estes nós podem ser combinados para facilmente criar comportamentos empregues em robôs, como mostra a Figura 2. Esta BT 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 são utilizadas em projetos de robótica, como o robô JIBO, 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 utilizado 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.

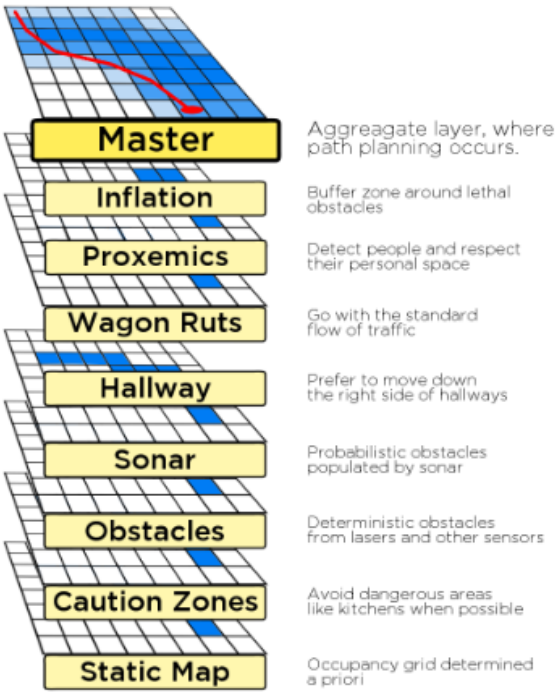
2.3.2 Sensores e SLAM

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 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

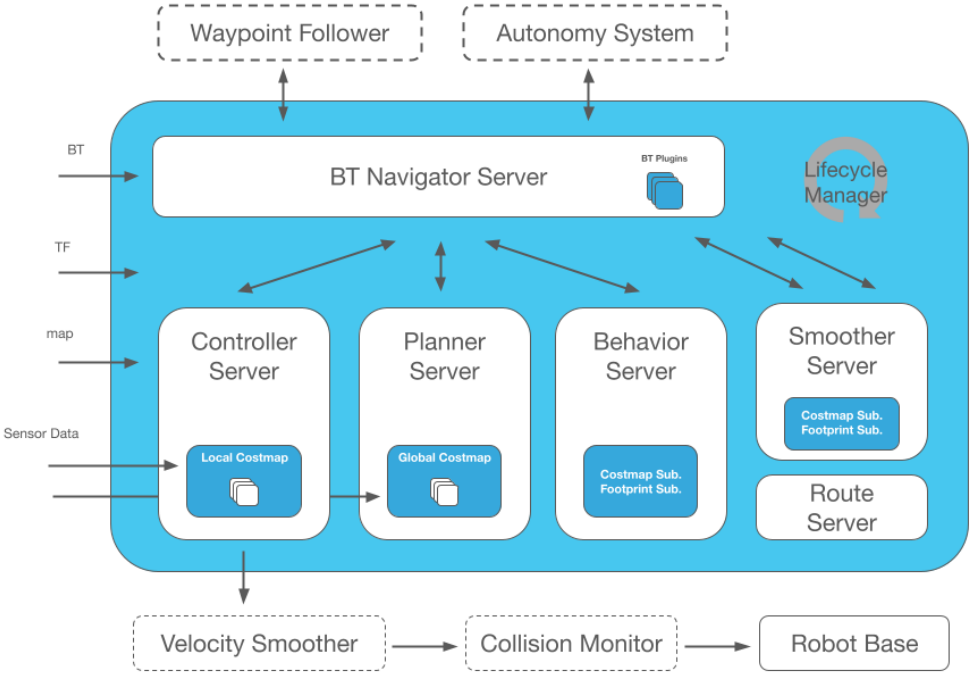
O *Navigation2* (Nav2) é o sucessor do ROS *navigation stack*, permitindo a realização de tarefas complexas em diversos ambientes e classes de robôs cinemáticos. Baseando-se no legado do *navigation stack* do ROS 1, o Nav2 foi construído em cima do ROS2, implementando técnicas mais modernas para ter 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, é utilizado para planejamento a curto prazo e prevenção de colisão, enquanto o mapa global, é aplicado no servidor de planejamento, é utilizado 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 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 da UFRGS, onde foi criada *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 do robô, nestes arquivos são incluídos *plugins* do Gazebo que simulam sensores, como a câmera do robô.

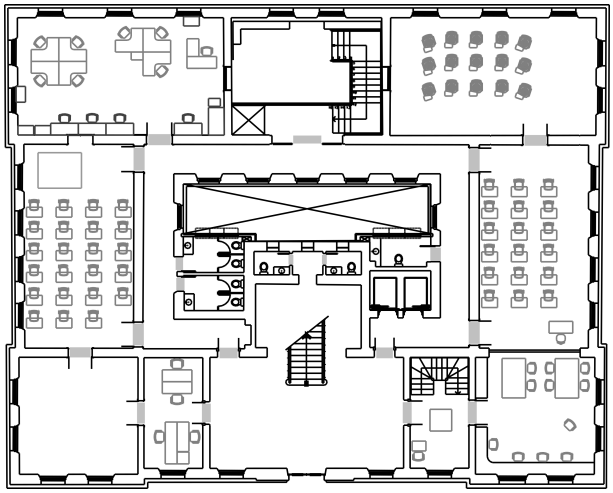
O sensor de percepção escolhido para o robô Twil é a câmera Intel RealSense D435. Para sua utilização é necessária a instalação do pacote *realsense-ros*(INTEL, s.d.), que contém os arquivos de descrição da câmera, além da definição das mensagens publicadas pela mesma. Para a simulação, foi adicionado 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.

A movimentação é implementada no pacote *twil_bringup*, que contém arquivos de configuração dos controladores, além de arquivos de *launch* para teste. Os controladores estão presentes nos pacotes *linearing_controllers* e *non_smooth_backstep_controller*, disponíveis no repositório da UFRGS. Estes controladores também são responsáveis pela publicação de informações de odometria no tópico *odom*, que serão utilizadas para localização do robô.

3.2 Ambiente de simulação

A verificação do funcionamento do sistema de navegação será feita em uma simulação em 3D do prédio Centenário da EE-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: *Planta do 1º andar do prédio Centenário da EE-UFRGS*

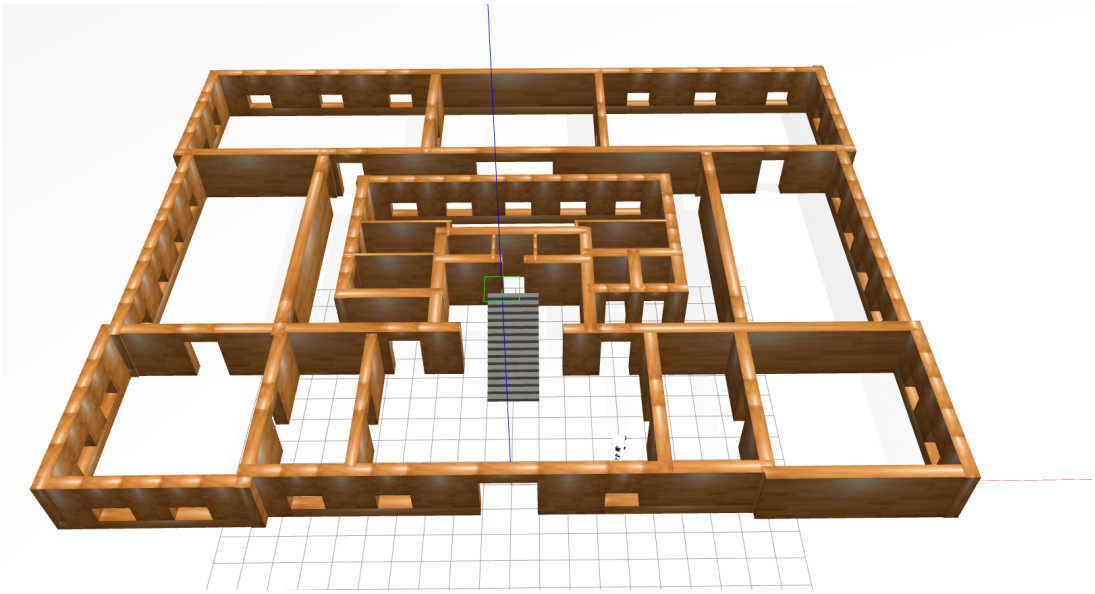


Fonte: Petry (2019)

Figura 5. No pacote *ufrgs_map* está incluído uma representação em formato DWG desta imagem, para utilização na camada estática do mapa de custo durante a navegação.

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 mapeamento e localização. 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

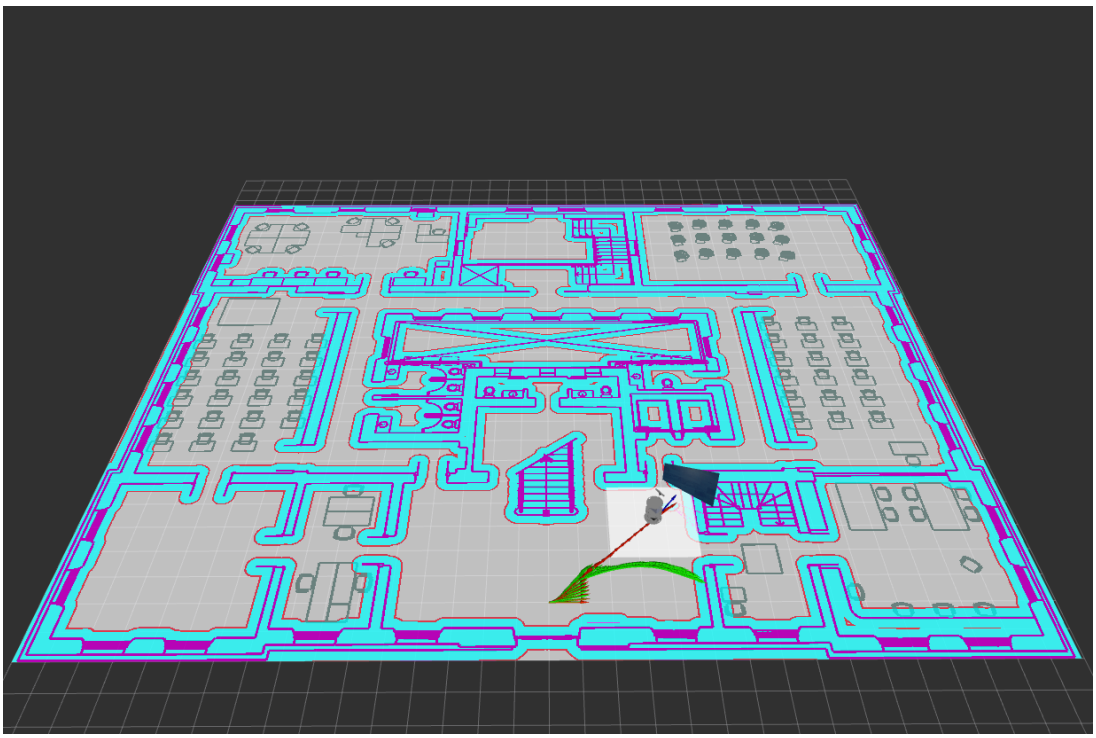
A navegação será realizada utilizando o *Navigation 2*. A movimentação é iniciada com uma mensagem do tipo *NavigateToPose*, enviada através do RViz. O *Nav2* recebe esta

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, o 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 trajetória diverge mais que o esperado da trajetória real, portanto, devem ser realizados ajustes no 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. Desta forma, as desvantagens de cada sensor podem ser compensadas pelos outros sensores.

Uma opção é utilizar sensores inercias, simulados no Gazebo com o *plugin* IMU. Para realizar a fusão dos dados, utiliza-se o pacote *robot_localization*, que combina os dados através de um filtro de Kalman.

Deve ser implementada também a odometria visual, utilizando a 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.

Os pacotes *ORB-SLAM3* e *OpenVSLAM*, por outro lado, foram criados para utilizar câmeras RGB-D, porém não estão integrados ao *Nav2* e, devido ao caráter recente das técnicas de VSLAM, estes não são tão robustos quanto os pacotes anteriores.

A implementação da odometria visual com a tradicional, deve ser realizada para atender os requisitos de transformações necessárias para o funcionamento do *Nav2*, descritos pelo padrão REP 105 (MEEUSSE, 2010). Resumidamente, é essencial a existência de transformadas de *map* para *odom* e *odom* para *base_link*.

3.3.2 Mapeamento

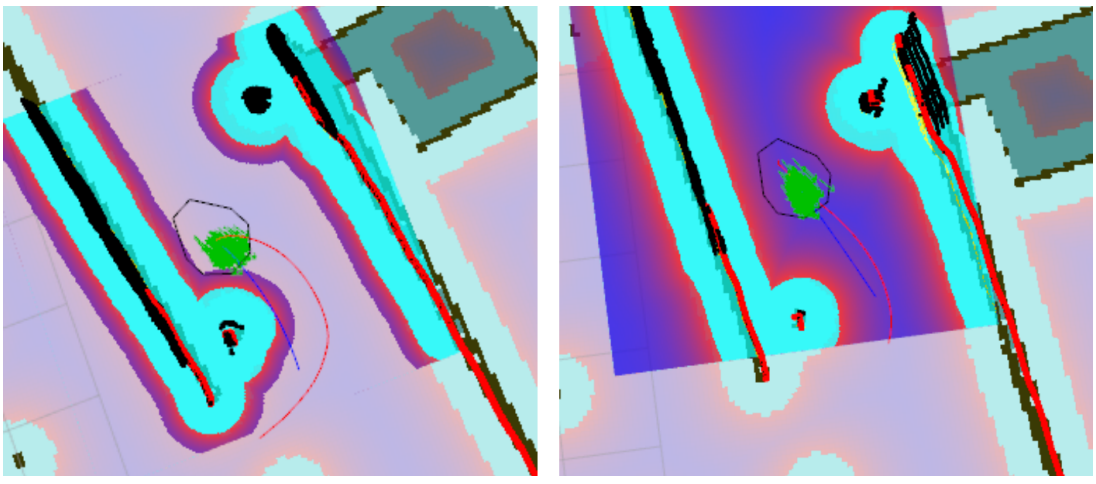
A representação do ambiente no *Nav2* é feita através de um mapa de custo. As camadas do mapa são definidas através de *plugins* que permitem a diversas fontes de dados para a construção do mapa. Neste trabalho, serão utilizados os *plugins*: *static_layer*, *voxel_layer*, *obstacle_layer* e *inflation_layer*.

A camada estática, do *plugin static_layer*, é utilizada para representar o mapa estático 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 novos mapas para utilização nesta camada.

As camadas *voxel_layer* e *obstacle_layer* utilizam dados dos sensores de percepção 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*, porém ambas podem utilizar mensagens do tipo *PointCloud2* ou *LaserScan*. A inclusão destas camadas é fundamental para evitar colisões com obstáculos inesperados.

A última camada, *inflation_layer*, cria uma zona de segurança ao redor dos obstáculos, 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.

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



Fonte: Zheng (2019)

3.3.3 Planejamento de trajetórias

Diversos *plugins* de planejamento de trajetórias estão disponíveis no *Nav2*, como o *NavFnPlanner*, *Smac Planner 2D*, *Theta Star Planner*. Neste trabalho, será utilizado o *NavFnPlanner*, que pode utilizar os algoritmos *Dijkstra* ou *A**. Este planejador foi escolhido porquê é o planejador recomendado para robôs circulares.

4 CONCLUSÃO

O planejamento realizado neste trabalho evidenciou um ponto essencial na utilizou do ROS 2, a sua modularidade. Isto permitiu a utilização de diversos pacotes de terceiros, aproveitando o trabalho de outros desenvolvedores. Além disso, o desenvolvimento do projeto pode ser efetuado gradualmente, isolando cada funcionalidade em tópicos distintos, facilitando a comparação de resultados de técnicas diferentes, como será feito 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. Além disso, para facilitar a reprodução 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%2F9780429489105>.

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/reps/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.RO].

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.RO].