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 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
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

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

Porto Alegre, Janeiro 2024

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 autonôma, Mapeamento de ambientes.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS

BT *Behavior Tree* (árvore de Comportamento)

RGB-D *Red Green Blue - Depth* (vermelho verde azul - profundidade)

ROS *Robot Operating System* (sistema operacional de robôs)

SLAM *Simultaneous Localization and Mapping* (localização e mapeamento simultâneos)

UFRGS Universidade Federal do Rio Grande do Sul

SUMÁRIO

1 INTRODUÇÃO

eu poderia separar a introducao em secoes, como motivacao, objetivo, organizacao do trabalho
tambem poderia mostrar o robo e dizer o que ja estava feito antes do trabalho

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, 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.

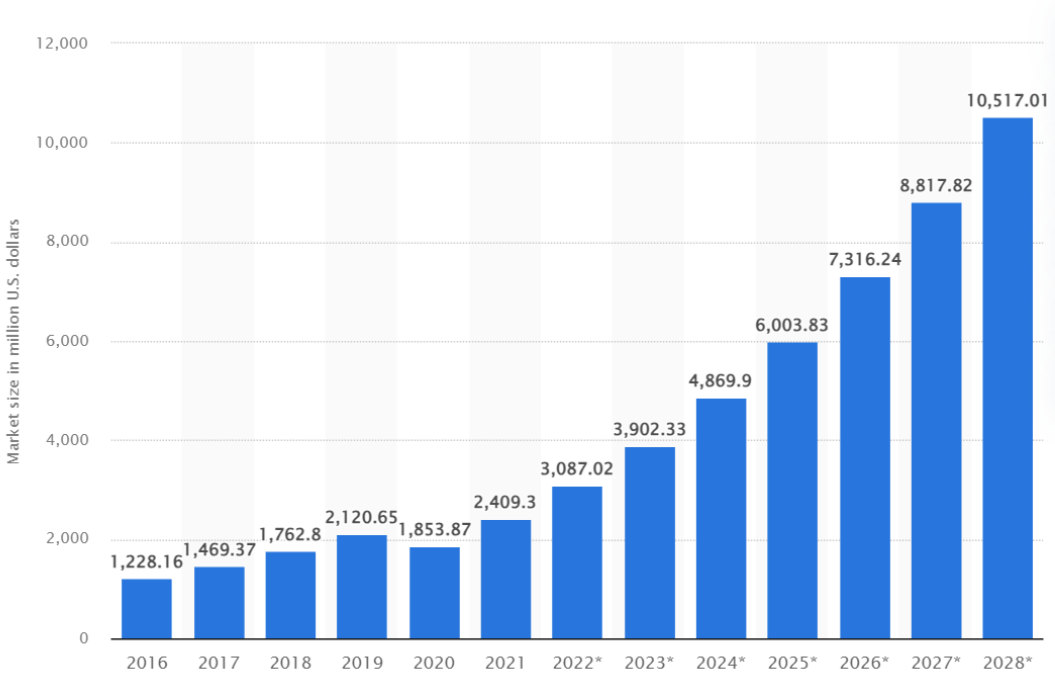
Laranja disse que esta parte ficou meio enrolada

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 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 novas versões, que implementam técnicas modernas que serão abordadas ao longo do 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.

Neste trabalho, será dado seguimento ao desenvolvimento anterior no Twil, com ajustes na odometria, além da adição de localização utilizando uma câmera de profundidade e mapeamento do ambiente para permitir a navegação autônoma em ambientes dinâmicos. Nos capítulos seguintes, é apresentado o embasamento teórico necessário para este desenvolvimento, além do planejamento da implementação deste sistema.

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



Fonte: Statista (2023).

2 REVISÃO DA LITERATURA

Neste capítulo são apresentados os conceitos necessários para o desenvolvimento do trabalho, como o Robot Operating System (ROS) 2, árvores de comportamento, mapeamento de ambientes e o *Navigation 2*.

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 *Data Distribution Service* (DDS) é utilizado como o *middleware*, que também é utilizado em sistemas de infraestrutura crítica, como aplicações militares, aeroespaciais e financeiras. Este padrão confere ao ROS 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, 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 response periodicamente com 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 Árvores de comportamento

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

Eu comentei essa seção porque achei desnecessária, já que eu não mexi nas ar-

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

Isso deveria ser uma seção e não uma subseção. Nas subseções poderiam ter os plugins/camadas

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 2 mostra uma configuração possível de camadas de mapas de custo.

colocar que a figura é traduzida/adaptada

colocar mais referencias(Laranja recomendou)

2.3.2 Sensores e SLAM

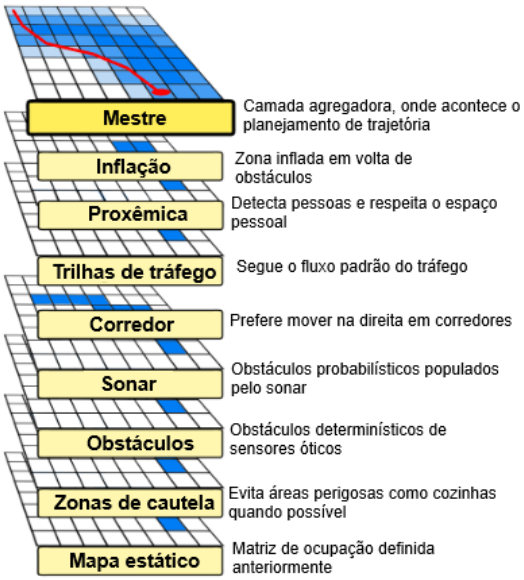
A escolha do sensor é importante para o mapeamento, pois afeta a qualidade e quantidade de informações obtidas pelo robô, além de determinar a escolha das ferramentas utilizadas para o mapeamento do ambiente (CHONG et al., 2015).

Sensores acústicos, como sonares e sensor de óticos, 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*.

Falar mais do slam/vslam eu acho

Figura 2: *Exemplo de configuração de camadas de um mapa de custo.*



Fonte: Adaptado de 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).

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.

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.

3 METODOLOGIA

Utilizando ferramentas da versão Humble do ROS 2, foi desenvolvido um sistema de navegação autônoma para o robô Twil, que utiliza uma câmera RGB-D Intel RealSense D435 para mapeamento do ambiente. 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 utilizados neste trabalho, foram implementados através de *plugins* do Gazebo, que comunicam o estado do robô durante a simulação através de tópicos do ROS 2.

explicar melhor a arquitetura do nav2 aqui
a imagem da arquitetura não é muito boa, entao precisa de uma explicacao geral melhor
é uma boa explicar o que faz o servidor de controlador(define a velocidade do robo pra seguir a trajetoria), já que é utiliza outro contrador para transformar a velocidade em comandos para as rodas do robo

O *Navigation2*, que tem sua arquitetura mostrada na Figura 3, foi configurado para utilizar estes componentes. O *Nav2* é responsável pelo envio comandos de velocidade para o robô, para que ele chegue ao destino desejado de modo seguro, evitando colisões. Para isso, são necessários: uma representação do ambiente, a posição do robô neste ambiente e um controlador para transformar os comandos de velocidade em comandos para as rodas do robô.

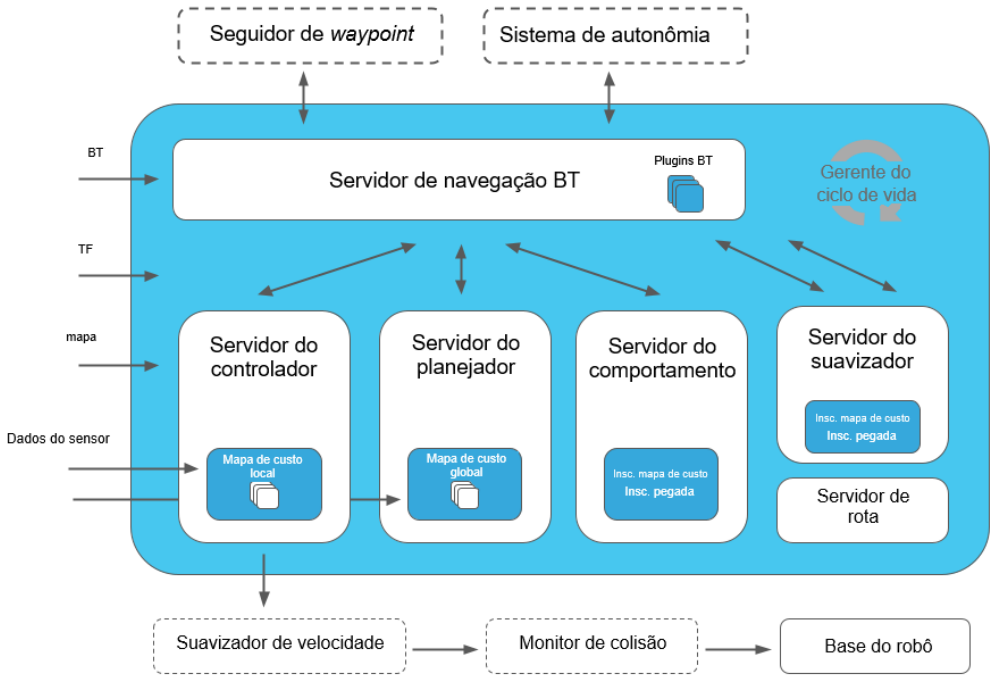
A representação do ambiente é feita através de um mapa de custo, e é onde o servidor do planejador se baseia para construção de trajetórias seguras do robô. A posição do robô é obtida através de transformações entre os sistemas de coordenadas `map`, `odom` e `base_link` do robô, conforme a especificação REP 105 (MEEUSSE, 2010). Esta posição é utilizada pelo servidor de controlador para gerar comandos de velocidade conforme a trajetória planejada.

Antes do início deste trabalho, o Twil já estava configurada para utilizar o *Nav2*, porém sem utilizar dados da câmera e IMU. Portanto, as trajetórias planejadas não eram atualizadas para evitar obstáculos e não havia sistema de localização, ou seja, a transformada entre `map` e `odom` era estática, portanto não havia 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 destes sensores nos de odometria, localização e mapeamento.

no diagrama simplificado, optei por nao colocar uma seta da localizacao para camada voxel, mas é claro que precisa da posicao do robo, só que ele usa a localizacao do nav2, entao isso foi configurado "implicitamente"(eu também achei que essa seta ia poluir muito o diagrama)

Figura 3: *Arquitetura do Navigation2.*



Fonte: Adaptado de Navigation2 (2020).

Um diagrama simplificado da arquitetura do trabalho é mostrado na Figura 4. Neste trabalho, todos sistemas conectados a câmera e o IMU foram modificados. O desenvolvimento deste trabalho foi focado nos componentes em vermelho e verde, que utilizam os dados da câmera. Em vermelho, estão os componentes referentes ao mapeamento do ambiente, onde são criados dois mapas. O primeiro mapa é um mapa de custo utilizado no planejamento de trajetórias, onde foi adicionada uma camada para percepção de obstáculos utilizando a câmera RGB-D, além da calibração da camada de inflação. O segundo mapa é criado e utilizado pelos sistemas de localização do robô. Em verde, estão os componentes referentes a estimativa de posição do robô, onde foram adicionados pacotes que realizam a odometria visual e pacotes de SLAM. Também foi realizado ajustes na odometria das rodas, através da fusão de dados com o sensor IMU. Para testes, foi mantida 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.

apesar de eu ter adicionado a fusao, no final nao mudou nada na odometria, talvez seja melhor nem mencionar

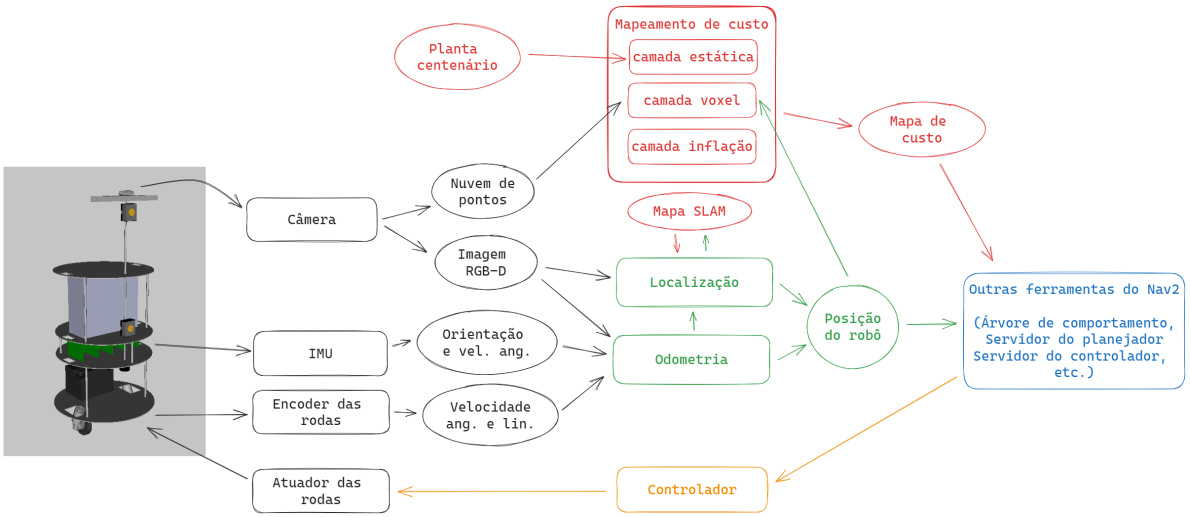
Nas seções seguintes, será apresentado o robô e o ambiente de testes utilizado neste trabalho. Em seguida, será detalhado o desenvolvimento dos sistemas necessários para a navegação autônoma do robô neste ambiente. Finalmente, será descrita a coleta de dados para a análise dos resultados, que é apresentada no próximo capítulo.

3.1 Configuração do robô

foi retirada uma roda e a bateria foi movida para tras, mas nao mudou nada, entao nao sei se coloco

O modelo do robô está presente 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. Este modelo é utilizado no rviz para visualização do robô, além de

Figura 4: *Arquitetura do simplificada do trabalho.*



ser utilizado no Gazebo para simulação. Para a simulação dos componentes físicos do robô, como sensores e atuadores, são utilizados *plugins* do Gazebo.

Durante este trabalho, foram configurados dois novos componentes ao robô, a câmera RGB-D Intel RealSense D435 e um IMU. A câmera foi utilizada nos sistemas de odometria, localização e mapeamento, enquanto o IMU foi utilizado para melhorar a qualidade da odometria, fornecendo dados de orientaçao do robô.

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. São utilizados cinco tópicos para publicar estes dados: `/camera/color/image_raw`, `/camera/color/camera_info`, `/camera/depth/image_raw`, `/camera/depth/camera_info` e `/camera/infra1/image_raw`.

talvez faça mais sentido colocar o tipo de msg, ou só explicar os tópicos

A adição do IMU foi feita utilizando o *plugin* `GazeboRosImuSensor`, que publica mensagens do tipo `sensor_msgs/Imu`. Estas mensagens contém a orientação, velocidade angular e aceleração linear do robô. Por si só, o IMU não é confiável para estimar a posição do robô, porém, estes dados podem ser fundidos com outras fontes de odometria para melhorar a acurácia da estimativa de posição. Como o modelo físico do IMU, foi reutilizada a descrição da placa de circuito impresso *Eurocard*, que já era utilizada em outros componentes do robô.

A conversão dos comandos de velocidade em movimento de rodas do robô é feita pelo controlador, executado pelo `twil_bringup`, que utiliza o controlador *twist_mrac_controller*, presente no pacote `linearizing_controllers`. Este controlador foi configurado em trabalhos anteriores para ser utilizado com o *Navigation2*. O pacote `arc_odometry` é executado junto por este controlador para transformar os dados de posição das juntas da roda em mensagens do tipo `nav_msgs/Odometry`, que publicam a posição, orientação e velocidades do robô em relação ao sistema de coordenadas `odom`.

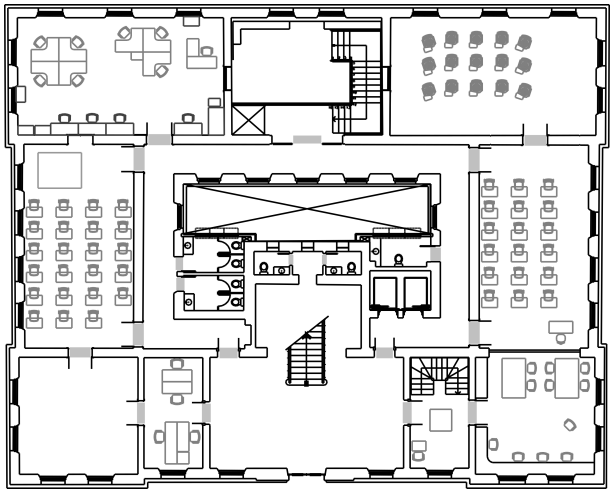
3.2 Ambiente de simulação

O sistema será simulado em uma simulação do prédio Centenário da Escola de Engenharia da UFRGS, utilizando o Gazebo, como em trabalhos anteriores. No pacote `ufrgs_map` está incluído uma representação em formato PGM da planta do prédio, mostrada

na Figura 5, desenvolvida em trabalhos anteriores. Além da imagem da planta, também está presente um arquivo de configuração em formato YAML, que contém a resolução, origem do mapa. Como este mapa é utilizado como mapa de custo, este arquivo de configuração também contém parâmetros de configuração que definem os valores que indicam se uma célula está livre ou ocupada.

essa ultima frase ta terrível

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

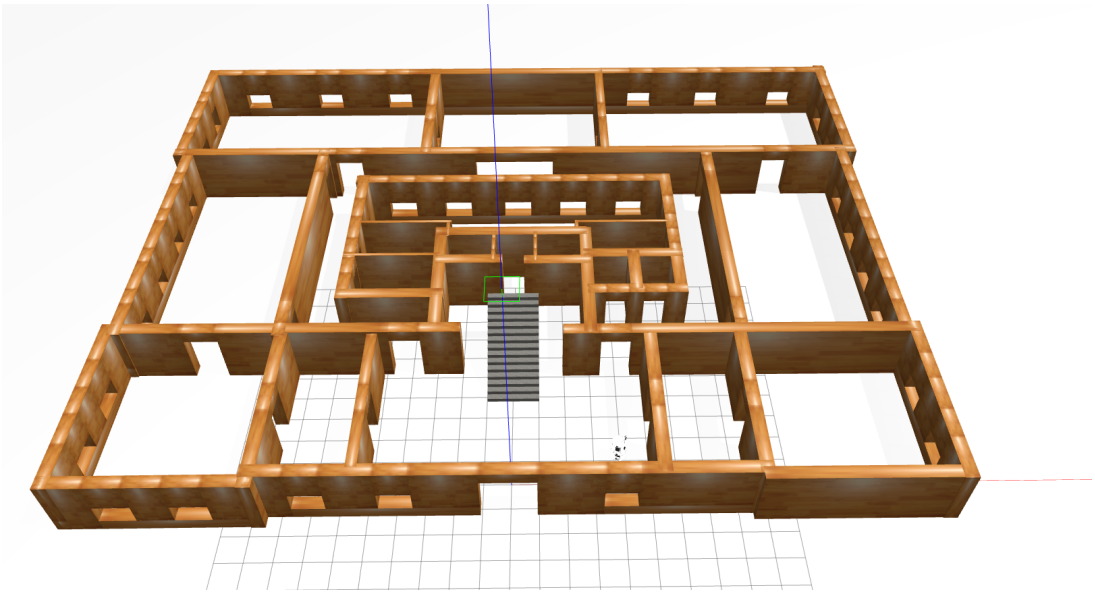


Fonte: Petry (2019).

O pacote `ufrgs_gazebo` contém arquivos de mundos do Gazebo para simulação. Este pacote, porém, estava desatualizado e não continha uma representação do prédio Centenário. Utilizando o *Building Editor* do Gazebo, foi criado um modelo do prédio em 3D, com base na planta do prédio, mostrado na Figura 6.

será que escrevo sobre a textura das paredes?

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



Para testar a prevenção de colisões com obstáculos dinâmicos, são adicionados objetos padrões do Gazebo, como caixas e cilindros, durante a simulação neste ambiente.

3.3 Odometria e localização

A estimativa de posição é feita utilizando o sistemas de coordenadas definido pela norma REP 105 (meeusse, 2010), que define quatro sistemas de coordenadas: `earth`, `map`, `odom` e `base_link`. Como só é utilizado um mapa, o sistema de coordenadas `earth` não será utilizado. Todas transformadas entre estes sistemas de coordenadas são publicadas no tópico `/tf`.

O sistema de odometria é responsável pela publicação da transformada entre o sistema de coordenadas `odom` e `base_link`. A posição do robô em relação ao `odom` deve ser contínua, porém pode divergir gradualmente. Por esta razão, são geralmente utilizados sensores incrementais na odometria, que estimam a posição e orientação do robô por integração e, portanto, são suscetíveis a erros de acumulação.

Neste trabalho, serão 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 publicado 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. Para aprimorar a qualidade desta odometria, foi adicionado o pacote `robot_localization`, que realiza a fusão de 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. Apesar da recomendação de configuração dos criadores do pacote, onde é sugerido utilizar apenas dados de velocidade e não de posição, para os dados calculados pelo `arc_odometry`, decidiu-se utilizar tanto a posição quanto a velocidade na fusão de dados. Isso foi feito porque a velocidade calculada pelo `arc_odometry` é calculada em referência ao sistema de coordenadas global, neste caso, o `odom`, enquanto que o `robot_localization` espera que a velocidade seja em referência a base do robô. Desta forma, a posição publicada pelo `robot_localization` é a mesma que a publicada pelo `arc_odometry`, já que não foram utilizados outros dados de posição. Porém, a orientação e velocidade angular da odometria das rodas são fundidas com os dados do IMU, melhorando a precisão apenas destes dados.

Como alternativa a este sistema, foi implementado um sistema de odometria visual, publicado pelo executável `rgbd_odometry` do pacote `rtabmap_odom`, que utiliza dados da câmera RGB-D com o sensor IMU para estimava da posição. Utilizando características das imagens RGB, com a informação de profundidade da imagem de profundidade, é utilizado o método RANSAC, acrônimo de *Random Sample Consensus*, para comparar imagens consecutivas e estimar a velocidade do robô.

eu traduzi "features"para caracteristicas, mas talvez seja um termo especifico de visao computacional

referenciar

explicaçao de odometria visual usando ransac que encontrei:
doi: 10.1109/IMTIC.2018.8467263

será que

É possível, também, fundir os dados destes dois sistemas de odometria, porém, optou-se por mantê-los separados para facilitar a comparação do desempenho entre eles.

Finalmente, para realização de testes, foi implementado um sistema de odometria "perfeito". Utilizando o *plugin* P3D do Gazebo, é publicada a posição do robô em relação ao sistema de coordenadas `odom` como uma mensagem do tipo `nav_msgs/Odometry` em um tópico chamado `fake_odom`. Esta mensagem é retransmitida para o tópico `/odom` e utilizada pelo pacote `odom_to_tf_ros2` para publicar a transformada entre `odom` e `base_link`.

tenho que explicar em algum lugar que esse fake odom é diferente do ground truth porque o ground truth é em relacao ao map e o fake odom é em relacao ao odom

Tabela 1: *Sistemas de odometria*

Pacote utilizado	Fonte de dados	Tipo de mensagem recebida
`robot_localization`	Encoder das rodas	`nav_msgs/Odometry`
	IMU	`sensor_msgs/Imu`
`rtabmap_odom`	Câmera	`sensor_msgs/CameraInfo`
		`sensor_msgs/Image`(RGB e profundidade)
	IMU	`sensor_msgs/Imu`
`odom_to_tf_ros2`	Posição no Gazebo	`nav_msgs/Odometry`

a camera no rgbd odom usa dois topicos do do tipo image, por isso que colocou (rgb e profundidade), mas achei que ficou estranho
talvez faça sentido colocar o tipo de mensagem image duas vezes e colocar rgb numa e profundidade na outra

Anteriormente, nenhum sistema de localização era utilizado, e a transformada entre `map` e `odom` era estática. Esta transformada foi mantida para testes, e foram implementados sistemas de localização para ajustar a odometria do robô utilizando sensores absolutos, que não acumulam erros. Esta transformada é publicada em uma frequência menor que a da odometria, permitindo a utilização de sistemas de localização com alta complexidade de processamento.

nao achei bom o prox paragrafo

Uma opção de sistema de localização é o `nav2_amcl`, que utiliza *Adaptive Monte Carlo Localization (AMCL)* e um mapa estático para estimar a posição do robô. Porém, optou-se por utilizar sistemas de SLAM, que realizam o mapeamento simultaneamente com a localização, já que o objetivo é a utilização do robô em ambiente dinâmicos ou pouco conhecidos, onde um mapa estático não seria adequado. Além disso, a planta do prédio e o mapa do Gazebo não são idênticos, que afeta o desempenho do sistema mesmo em situações ideais. Portanto, dois sistemas de SLAM foram implementados, o *slam_toolbox* e o *rtabmap*.

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, foi utilizado o pacote `depthimage_to_laserscan` 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 utiliza todos os tópicos publicados pela câmera RGB-D, e não necessita de conversão de mensagens.

Ambos sistemas 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, estes sistemas podem ser iniciadoscom 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.

talvez seja bom falar sobre o loop-closure no paragrafo anterior

Tabela 2: *Sistemas de localização*

Pacote utilizado	Fonte de dados	Tipo de mensagem recebida
`slam_toolbox`	Câmera	`sensor_msgs/LaserScan`
`rtabmap_slam`	Câmera	`sensor_msgs/CameraInfo` `sensor_msgs/Image`(RGB e profundidade)
`tf2_ros`		

3.4 Mapeamento

Um aspecto essencial para a navegação autônoma é a representação do ambiente. O planejamento de trajetória necessita de um mapa do ambiente, na forma de um mapa de custo, em que cada célula contém um valor indicando o custo de atravessa-la. Um algoritmo de pesquisa utiliza este mapa para encontrar a trajetória com menor custo para o destino desejado.

Apesar do sistema de SLAM fornecer um mapa do ambiente, este mapa não será 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, o mapa pode ser criado em estados não desejados como, por exemplo, com trajetórias bloqueadas por obstáculos temporários.

Portanto, haverão duas representações do ambiente, uma criada pelo SLAM, que é utilizada pelo mesmo sistema para localização, e outra criada utilizando os *plugins* do mapa de custo do Nav2, que é utilizada para planejamento de trajetória.

O mapa de custo utilizado é composto por três camadas:

1. a camada estática, que possui uma representação simples do ambiente, sem obstáculos.

2. a camada de obstáculos, onde são utilizados dados dos sensores óticos para atualizar o mapa com obstáculos dinâmicos.

3. a camada de inflação, que cria um campo de custo ao redor dos obstáculos.

A planta do prédio Centenário, representada na Figura 5, foi utilizada como mapa estático. O mapa foi configurado para considerar apenas as cédulas em preto escuro como ocupados, que equivalem a obstáculos fixos, como paredes e a escada. As cédulas em cinza claro são consideradas livres, que representam a posição provável de objetos móveis, como cadeiras e mesas.

A camada de obstáculos utiliza os dados da camera RGB-D para atualizar o mapa. Existem dois *plugins* que podem ser utilizados para este fim, o `obstacle_layer` e o `voxel_layer`. Ambos podem utilizar mensagens do tipo `PointCloud2`, que são publicadas pela câmera RGB-D, porém utilizam 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 gride tri-dimensional, divido em blocos chamados de *voxels*, que podem estar ocupados ou livres.

nao achei traducao para ray casting

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 necessário 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 metros 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, serão captados obstáculos dentro deste campo, porém obstáculos com altura superior a 1,5 metro não serão detectados. 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.

os artigos sobre o Nav2 utilizam o Spatio Temporal Voxel Layer, que introduz um "decay"a camada voxel
queria usar ele mas o parece que o pacote nao foi migrado pro Humble, tenho que escrever isso em algum lugar

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, criando um campo em grande parte do mapa de custo. Desta forma, são criadas trajetórias que passam no meio de obstáculos, mantendo a maior distância possível entre o robô e possíveis colisões. As as Figuras 7a e 7b, mostram uma comparação entre o mapa de custo com inflação baixa e alta.

nao sei se coloco aqui ou nos resultados a comparação entre as duas inflações

3.5 Testes e coleta de dados

Para testar e comparar os sistemas de odometria, localização e mapeamento, foram utilizados arquivos de gravação de tópicos do ROS 2, chamados de *bags*. A utilização desta funcionalidade permite a comparação dos sistemas utilizando os mesmos dados de entrada. Além disso, foi possível realizar a simulação no Gazebo e executar os nodos de localização separadamente, o que aliviou a exigência de processamento do computador utilizado.

Para a gravação dos *bags*, foi criado um programa em *bash* que executa o comando de gravação com os tópicos desejados e cria uma pasta com o nome de acordo com o argumento passado ao programa. Os tópicos gravados foram:

- `/camera/color/image_raw`
- `/camera/color/camera_info`
- `/camera/depth/image_raw`
- `/camera/depth/camera_info`
- `/camera/infra1/image_raw`
- `/joint_states`

- `/ground_truth`
- `/sensor/imu`
- `/twist_mrac_controller/odom`

ao inves de criar essa lista aqui, acho seria melhor referenciar as tabelas do sistema de odometria e localizacao(que tem que ter os tópicos neste caso), e explicar os tópicos que sobraram, como o ground truth e o joint states

fiquei com a impressao que ficou enrolada o paragrafo a seguir, entao acho que seria melhor reescrever depois

Para a execução dos *bags*, não é necessário executar o Gazebo e o *Navigation2* portanto, foi criado um novo arquivo de lançamento, que executa apenas os apenas os nodos necessários para o teste.

Os executáveis que utilizam os dados da câmera necessitam da árvore de transformadas de coordenadas do base do robô até a câmera. Na simulação, as transformadas estáticas são definidas no arquivo de descrição do robô e as transformadas dinâmicas são publicadas pelo Gazebo. Portanto, no arquivo de lançamento, foi adicionado o pacote de descrição do robô e no *bag* foi gravado o tópico `/joint_state`, que contém a posição das juntas dinâmicas do robô. Isso permitiu que não fosse necessário nem executar o Gazebo, nem gravar o tópico `/tf` de transformadas do robô, já que deseja-se republicar estas transformadas no teste.

Os dados dos testes também foram gravados no formato *bag*. Isso permitiu a utilização do programa `Plotjuggler`, para a análise dos dados e criação de gráficos após a execução dos testes.

referenciar o plot-juggler

4 RESULTADOS E DISCUSSÃO

Neste capítulo, serão apresentados os resultados dos testes realizados. Estes resultados estão organizados em três seções, começando pela análise do efeito do mapeamento de custo na criação de trajetórias. Em seguida, será mostrado os resultados da odometria, localização e mapeamento SLAM dos diferentes pacotes utilizados para a mesma sacola de dados.

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.

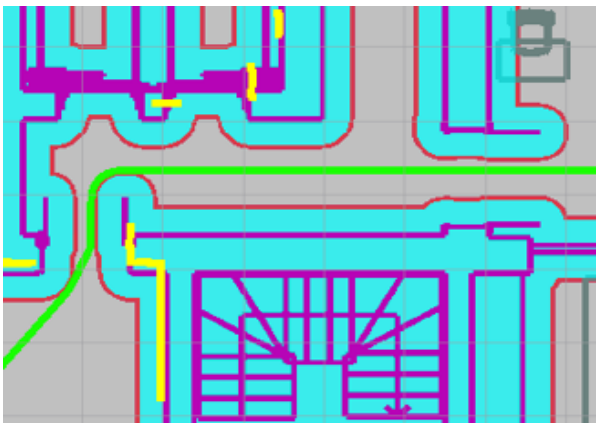
A camada de inflação porém, foi ajustada para melhorar a qualidade dos caminhos criados, de acordo com a recomendação de Zheng (2019). A camada de inflação original criava uma pequena zona de segurança ao redor dos obstáculos, com uma borda de $0,35$ m e inclinação de $3,0$. Representada na Figura 7a. 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, a inclinação precisou ser ajustada para $4,0$, porque a inclinação original fazia com que certos corredores fossem evitados.

eu ja referenciei em outro lugar esse guia, referencio de novo??

talvez isso deveria estar na metodologia

Figura 7: *Trajetórias criadas com diferentes configurações da camada de inflação*

(a) *Mapa de custo com inflação baixa.*



(b) *Mapa de custo com inflação alta.*

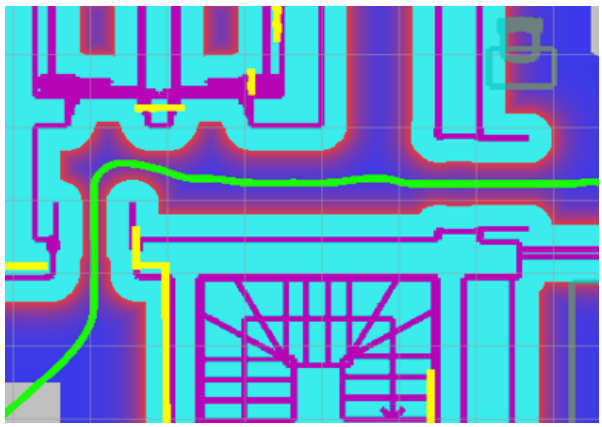
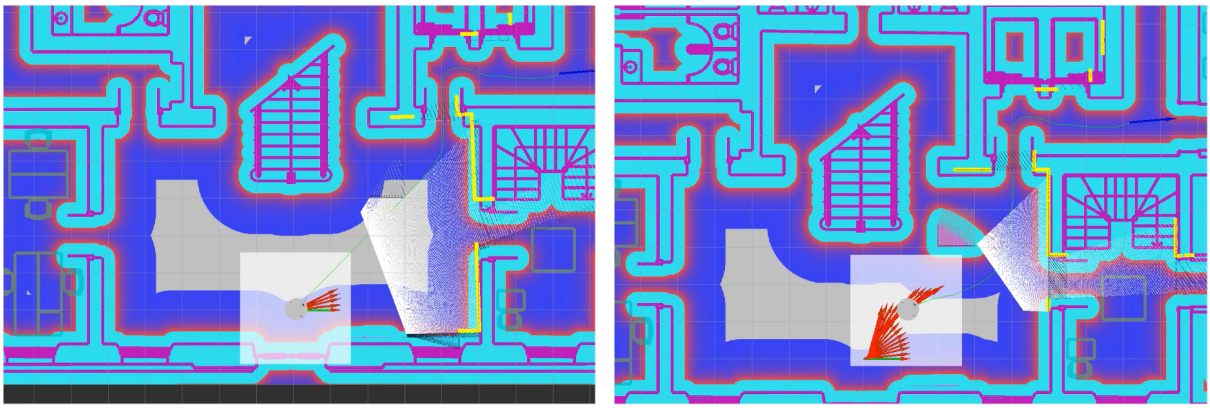


Figura 8: *Mapeamento de obstáculos da camada voxel*

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



4.2 Odometria, localização e mapeamento SLAM

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

Na Figura 9, é mostrado o robô realizando a trajetória de testes. As setas em vermelho representam a mensagem de odometria do robô, onde é indicado a posição e orientação do robô. A linha verde indica a trajetória planejada do robô. O caminho realizado neste teste começa na origem, segue em direção a porta no canto superior direto da sala, percorre o corredor e entra na primeira sala. É possível comparar a percepção de diferentes ambientes neste teste, como em corredores estreitos e salas abertas.

Devido ao tamanho da sacola 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.

Após o fim da simulação e captura de dados, foram executados os sistemas de odometria na sacola gerada. Neste teste, utilizou-se a transformada estática entre os sistemas de coordenadas `map` e `odom`, para comparar a odometria das rodas e visual. Na Figura 10, são mostrados as estimativas da posição do robô nos eixos X e Y em relação a origem do sistema de coordenadas `map` dos dois sistemas de odometria em verde, comparados com a posição real do robô em vermelho.

A odometria das rodas, mostrada na Figura 10a, mostra grande divergência em relação a posição real do robô, com essa divergência enviesada para a direita. Este resultado não é esperado, e indica alguma falha no sistema de odometria das rodas, podendo ser causada pela descrição do robô.

acho que seria uma boa referenciar um artigo que diga a divergencia esperada da odometria das ro-

Por outro lado, a odometria visual, mostrada na Figura 10b, apresentou um resultado satisfatório. Existe uma divergência ao longo do tempo, além de um erro significativo na última curva da trajetória, estimando um caminho com alto ruído. Porém, esta odometria apresentou desempenho satisfatório para utilização em conjuntos com os testes dos sistemas de SLAM. Porém, este desempenho pode não ser replicado em ambientes reais, devido ao ruído presente fora da simulação.

colocar que, devido ao processamento e meu note nao ser tao bom a odometria visual nao ficou igual em todas as execucoes mesmo com a mesma entrada

Figura 9: *Robô durante o teste realizado.*

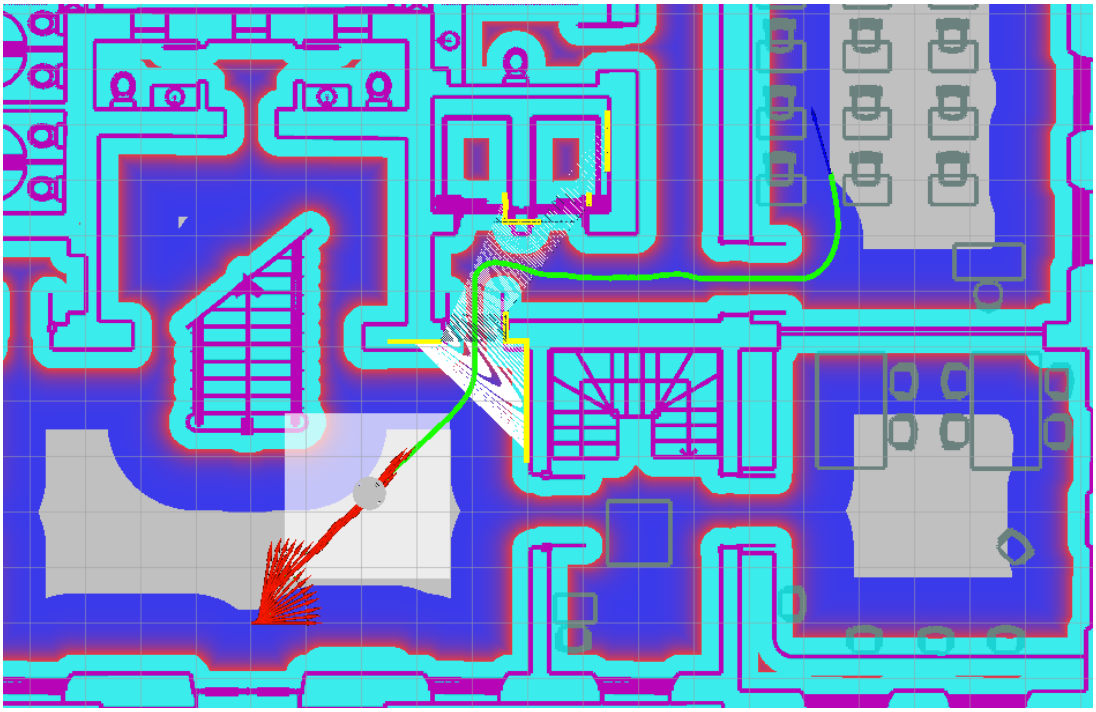
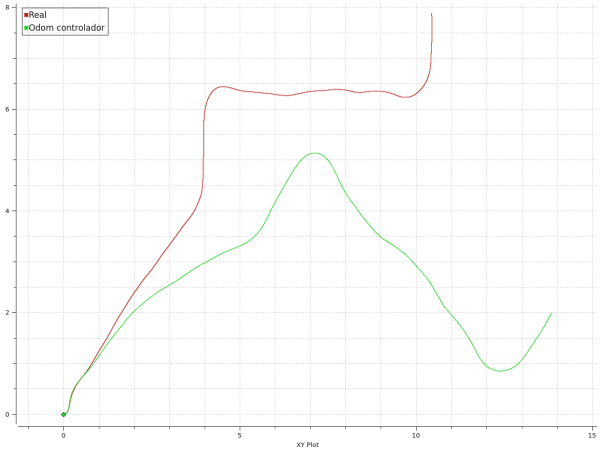


Figura 10: *Sistemas de odometria*

(a) *Odometria das rodas.*



(b) *Odometria visual.*

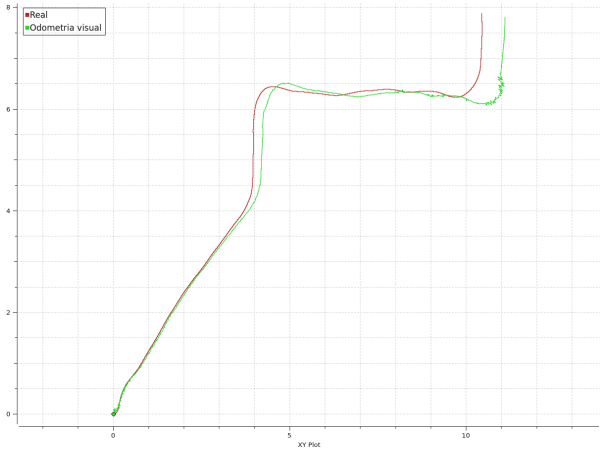
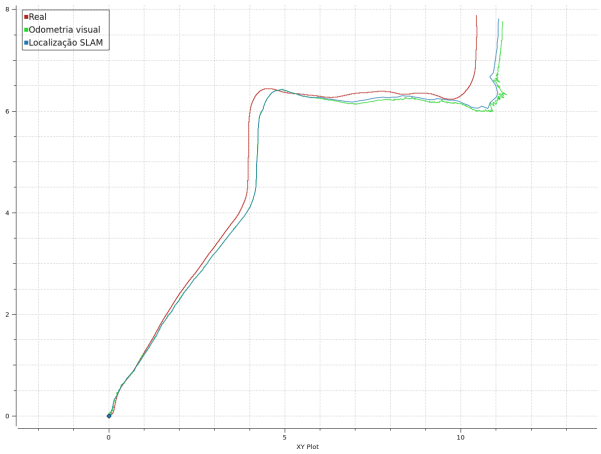


Figura 11: *Sistemas de localização*

(a) *Pacote* `rtabmap_slam`.



(b) *Pacote* `slam_toolbox`.

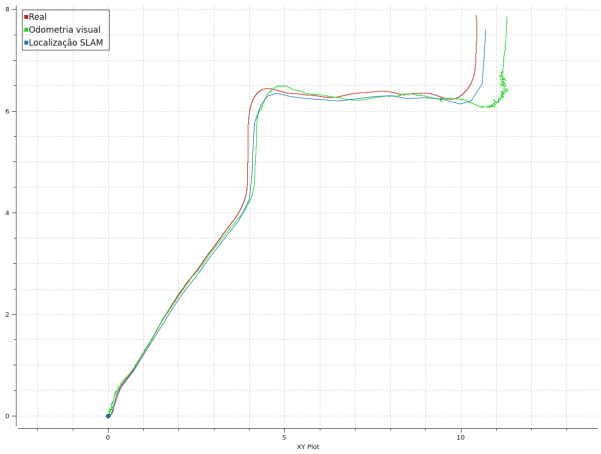


Figura 12: *Pacote* `rtabmap_slam`.

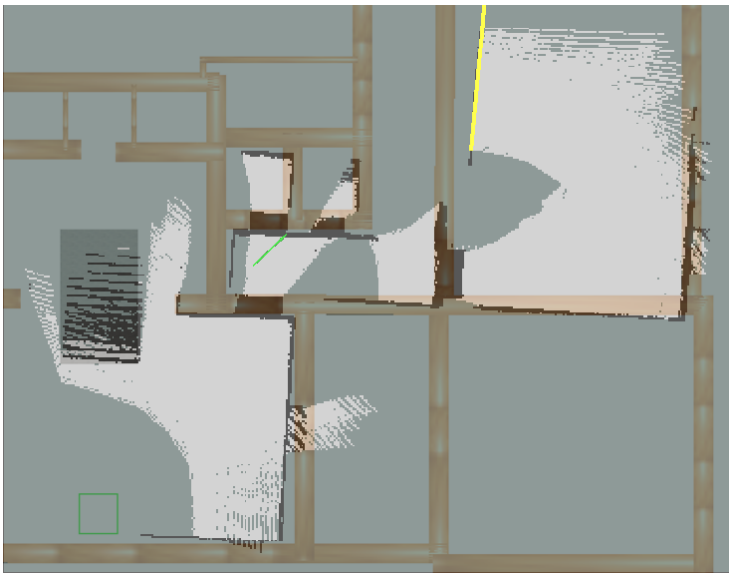
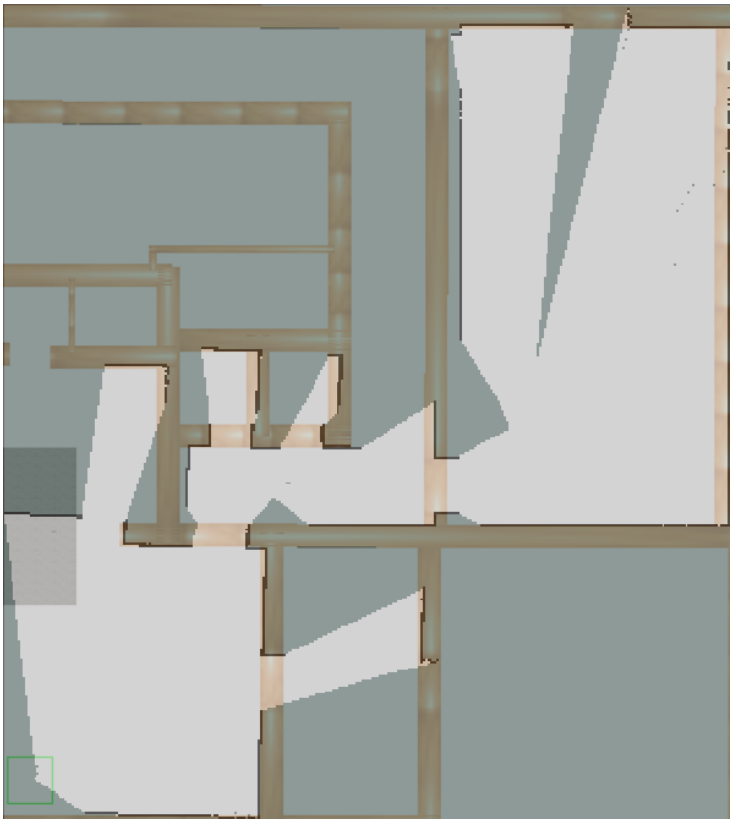


Figura 13: *Pacote* `slam_toolbox`*.*



5 CONCLUSÃO

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>.

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].