

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

HENRIQUE SCHARLAU COELHO - 243627

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

Porto Alegre
2024

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

HENRIQUE SCHARLAU COELHO - 243627

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

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

ORIENTADOR:

Prof. Dr. Walter Fetter Lages

Porto Alegre
2024

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

HENRIQUE SCHARLAU COELHO - 243627

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

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

Orientador: _____

Prof. Dr. Walter Fetter Lages, UFRGS

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

Banca Examinadora:

Prof. Dr. Walter Fetter Lages, UFRGS

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

Prof. Dr. Rafael Antônio Comparsi Laranja, UFRGS

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

Prof. Dr. Eduardo André Perondi, UFRGS

Doutor pela Universidade Federal de Santa Catarina – Florianópolis, Brasil

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

Porto Alegre, Janeiro 2024

RESUMO

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

ABSTRACT

Keywords: KEYWORDS TRANSLATION PLACEHOLDER.

LISTA DE ILUSTRAÇÕES

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

LISTA DE TABELAS

1	Sistemas de odometria	20
2	Sistemas de localização.....	21

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	8
2	REVISÃO DA LITERATURA.....	10
2.1	Robot Operating System 2 (ROS 2).....	10
2.2	Estimativa de posição de um robô móvel.....	10
2.2.1	Odometria	11
2.2.2	Localização.....	11
2.3	Localização e Mapeamento Simultâneo (SLAM).....	12
2.3.1	Otimização de grafo de poses	12
2.3.2	Aplicações	12
2.4	Fusão de dados de sensores	12
2.5	Mapas de custo	12
2.6	Navigation2	13
3	METODOLOGIA	15
3.1	Configuração do robô	17
3.2	Ambiente de simulação.....	17
3.3	Estimativa de posição do robô.....	18
3.3.1	Odometria	18
3.3.2	Localização.....	20
3.4	Mapeamento	21
3.4.1	Mapeamento de custo.....	22
3.4.2	Mapeamento SLAM	22
3.5	Testes e coleta de dados.....	23
4	RESULTADOS.....	25
4.1	Mapeamento de custo.....	25
4.2	Testes de odometria, localização e mapeamento SLAM.....	26
4.2.1	Odometria	26
4.2.2	Localização.....	28
4.2.3	Mapeamento SLAM	29
5	DISCUSSÃO	32
6	CONCLUSÃO	34
	REFERÊNCIAS	35

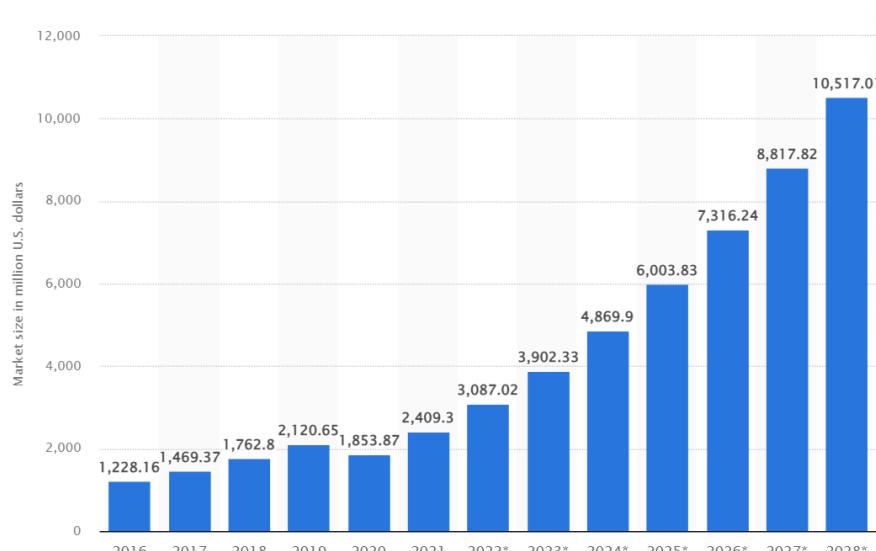
1 INTRODUÇÃO

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

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

Laranja disse que esta parte ficou meio enrolada

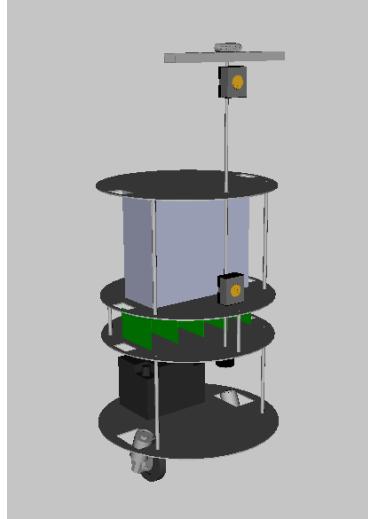
Figura 1: Mercado global de robôs autônomos de 2016 a 2021, com projeção até 2028.



Fonte: Statista (2023).

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

Figura 2: Modelo do robô utilizado.



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

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

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

2 REVISÃO DA LITERATURA

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

2.1 ROBOT OPERATING SYSTEM 2 (ROS 2)

O ROS 2 é a segunda geração do Robot Operating System, um *framework* para desenvolvimento de robôs. Ele foi desenvolvido a partir do zero para atender as necessidades de robôs modernos, com suporte para customização extensiva. O seu *middleware*, *Data Distribution Service* (DDS), também é usado em sistemas de infraestrutura crítica, como aplicações militares, aeroespaciais e financeiras. Este padrão confere ao ROS 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 retorna periodicamente o estado atual da tarefa. No término, é enviado o resultado, podendo ser sucesso ou falha. Um exemplo de uso é uma tarefa de navegação em que um *action client* envia uma requisição com um ponto de destino para um *action server* que responde com realimentação contínua da posição atual do robô e com o resultado ao finalizar a tarefa. Eles também são apropriados para utilização em árvores de comportamento.

2.2 ESTIMATIVA DE POSIÇÃO DE UM ROBÔ MÓVEL

A terminologia e convenção dos sistemas de coordenadas de um robô móvel utilizado neste trabalho é definido pela norma REP 105 (MEEUSSE, 2010). Esta especificação define quatro sistemas de coordenadas: `earth`, `map`, `odom` e `base_link`. Logo, a posição da base do robô, chamada de `base_link`, se da em relação a estes sistemas de coordenadas.

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

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

tenho que referenciar extensivamente isso nas duas seções seguintes
<https://www.ece.ufrgs.br/fetter/ele00070/mobrob/estimation.pdf>

2.2.1 Odometria

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

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

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

explicação de odometria visual usando ransac(que o rtabmap odom usa) que encontrei:
doi: 10.1109/IMTIC.2018.8467263

2.2.2 Localização

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

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

amcl

Uma técnica comumente usada é a comparação de um mapa, obtido previamente, com dados de sensores ópticos. A localização por Monte Carlo é uma das aplicações, mais comuns deste sistema. Nela, os estados possíveis do robô são representados por partículas, criando uma distribuição de probabilidade. Com os dados da odometria e do sensor ótico, o estado possível do robô é atualizado, sendo a partícula com maior probabilidade escolhida como a estimativa de posição do robô.

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

2.3 LOCALIZAÇÃO E MAPEAMENTO SIMULTÂNEO (SLAM)

refazendo...

2.3.1 *Otimização de grafo de poses*

2.3.2 *Aplicações*

2.4 FUSÃO DE DADOS DE SENSORES

Robôs modernos estão cada vez equipados com mais sensores, ocasionando em mais uma fonte de dado para a mesma informação. Um caso relevante neste trabalho, é a odometria do robô, que pode ser estimada por diversos tipos de sensores, com diferentes características. Ao invés de utilizar apenas um sensor para estimar a odometria, os dados destes sensores, munidos da informação de covariância do mesmo, podem ser fundidos obtendo uma estimativa mais precisa. O filtro de Kalman, introduzido em Kalman (1960), é um estimador linear recursivo, comumente utilizado para aplicações deste tipo.

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

2.5 MAPAS DE CUSTO

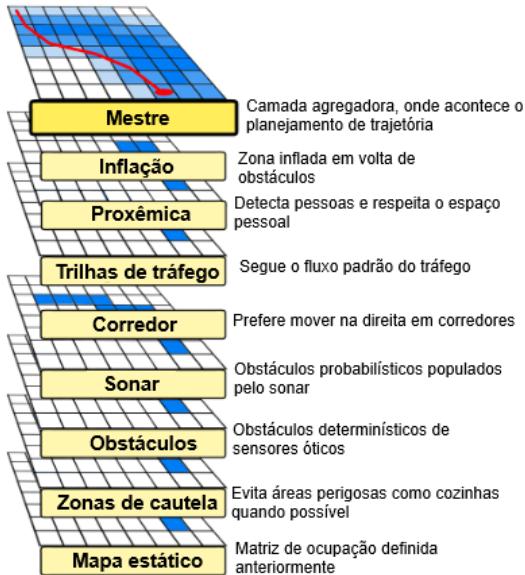
Um mapa de custo é uma representação de ambiente composta por uma grade de células que contém um valor, variando de desconhecido, livre, ocupado ou custo inflado.

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.

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 (LU; HERSHBERGER; SMART, 2014). Esta separação permite a utilização de diferentes fontes de dados, permitindo, por exemplo, adicionar dados de sensores para atualizar o mapa, sem modificar o mapa original. Outras camadas podem ser configuradas para evitar certas áreas, manter uma distância segura de obstáculos ou seguir para seguir regras culturais, como manter a direita em corredores. A Figura 3 mostra uma configuração possível de camadas de mapas de custo.

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

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



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

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

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

2.6 NAVIGATION2

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

não acho
que esse
para-
grafo ta
bom

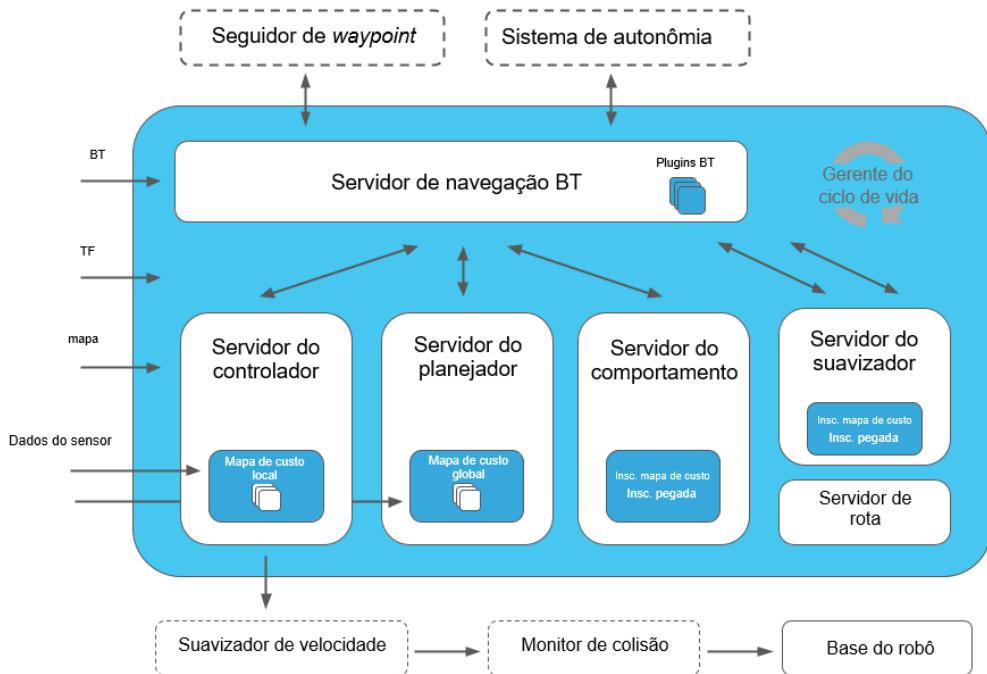
3 METODOLOGIA

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

Os variados componentes do robô, como os sensores 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. A câmera RGB-D Intel RealSense D435, utilizada para o mapeamento do ambiente, foco deste trabalho, foi implementada desta forma.

O *Navigation2*, que tem sua arquitetura mostrada na Figura 4, foi configurado para utilizar os componentes disponíveis do robô Twil. O *Nav2* é responsável pelo envio de comandos de velocidade para o robô, para que ele chegue ao destino desejado de modo seguro, evitando colisões. Como pré-requisito, devem ser fornecidos a este sistema a 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.

Figura 4: Arquitetura do *Navigation2*.



Fonte: Adaptado de *Navigation2* (2020).

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

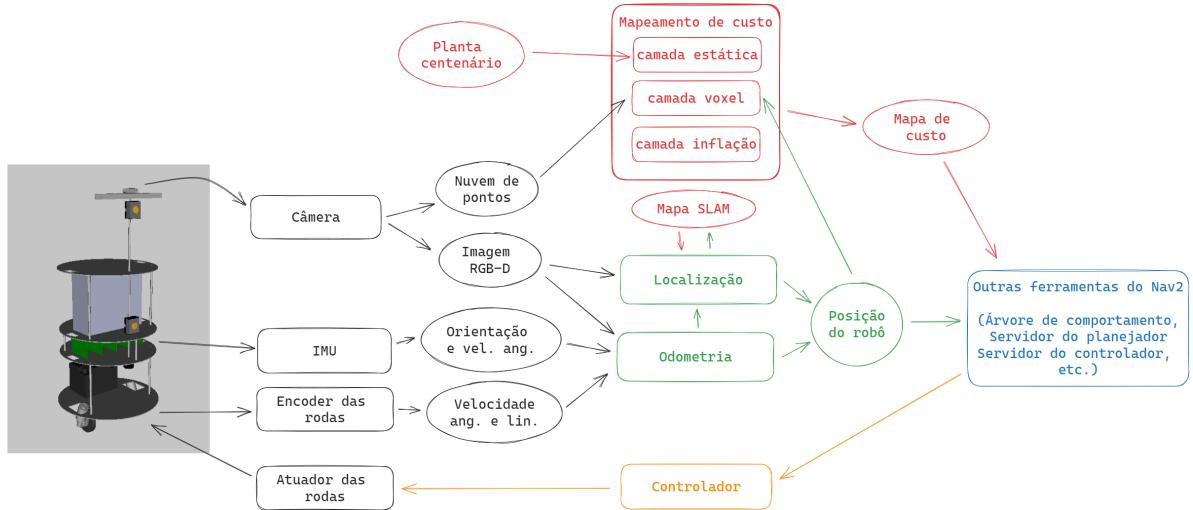
do robô é obtida através de transformações entre os sistemas de coordenadas `map`, `odom` e `base_link` do robô, conforme a especificação REP 105 (MEEUSSE, 2010). Esta posição é utilizada pelo servidor de controlador para gerar comandos de velocidade para seguir a trajetória planejada.

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 sistemas de odometria, localização e mapeamento.

Um diagrama simplificado da arquitetura do trabalho é mostrado na Figura 5. O desenvolvimento deste trabalho foi focado nos componentes em vermelho, que realizam o mapeamento do ambiente, e componentes em verde, que estimam a posição do robô. O bloco em azul, que representa as outras ferramentas do *Nav2*, e o controlador em amarelo, que transforma os comandos de velocidade em comandos para as rodas do robô, já estavam configurados.

Figura 5: Arquitetura do simplificada do trabalho.



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

Em verde, estão os componentes referentes 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. Ademais, para realização de 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.

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

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 nodo `robot_state_publisher`, que publica esta descrição em um tópico. Este modelo é utilizado no rviz para visualização do robô, como representado na Figura 2, e no Gazebo para simulação. Para a simulação dos componentes físicos do robô, como sensores e atuadores, são utilizados *plugins* do Gazebo, também presentes no arquivo de descrição.

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

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

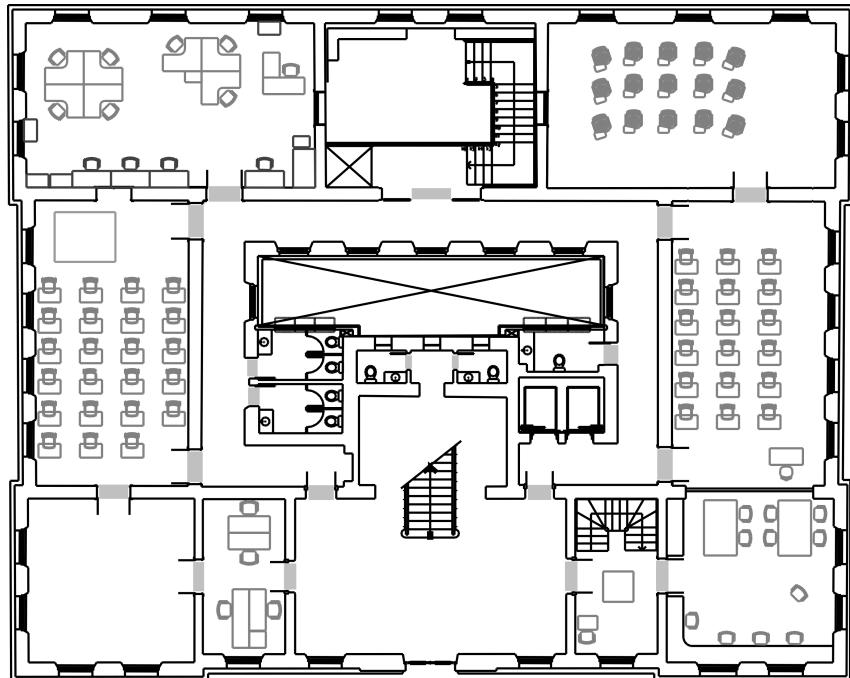
A adição do IMU foi feita utilizando o *plugin* `GazeboRosImuSensor`, que publica mensagens do tipo `sensor_msgs/Imu`. Estas mensagens contêm a orientação, velocidade angular e aceleração linear do robô. Por si só, o IMU não é confiável para estimar a posição do robô porque necessita integrar a aceleração linear duas vezes para estimar a posição, que pode introduzir erros significativos. Porém, os dados referentes a orientação e velocidade angular podem ser fundidos com outras fontes de odometria para complementar outras fontes de odometria. Para representar fisicamente o IMU no robô, foi reutilizada a descrição de uma placa de circuito impresso *Eurocard*, já utilizada em outros componentes do robô.

A conversão dos comandos de velocidade em movimento de rodas do robô é feita pelo controlador, configurado no pacote `twil_bringup`. Diversos controladores estão configurados e disponível para uso com o Twil, porém o controlador `twist_mrac_controller`, do pacote `linearizing_controllers` foi o escolhido e configurado para ser usado com o *Nav2* em trabalhos anteriores.

3.2 AMBIENTE DE SIMULAÇÃO

O sistema será simulado em uma simulação do prédio Centenário da Escola de Engenharia da UFRGS, utilizando o Gazebo. No pacote `ufrgs_map` está incluído uma representação em formato PGM da planta do prédio, mostrada na Figura 6, desenvolvida em Petry (2019). Além da imagem da planta, também está presente um arquivo de configuração em formato YAML, que contém a resolução, origem do mapa. Este arquivo de também contém parâmetros para utilização como mapa de custo, indicando a faixa de valores para considerar uma célula como livre ou ocupada.

Figura 6: 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 completa do prédio Centenário. Utilizando o *Building Editor* do Gazebo, foi criado um modelo simplificado do prédio em 3D, com base na sua planta, mostrado na Figura 7.

Não foram adicionados os obstáculos em cinza claro da planta da Figura 6, que não são obstáculos fixos. Para teste com obstáculos dinâmicos ou temporário são adicionados novos objetos a este mundo durante a execução da simulação no Gazebo.

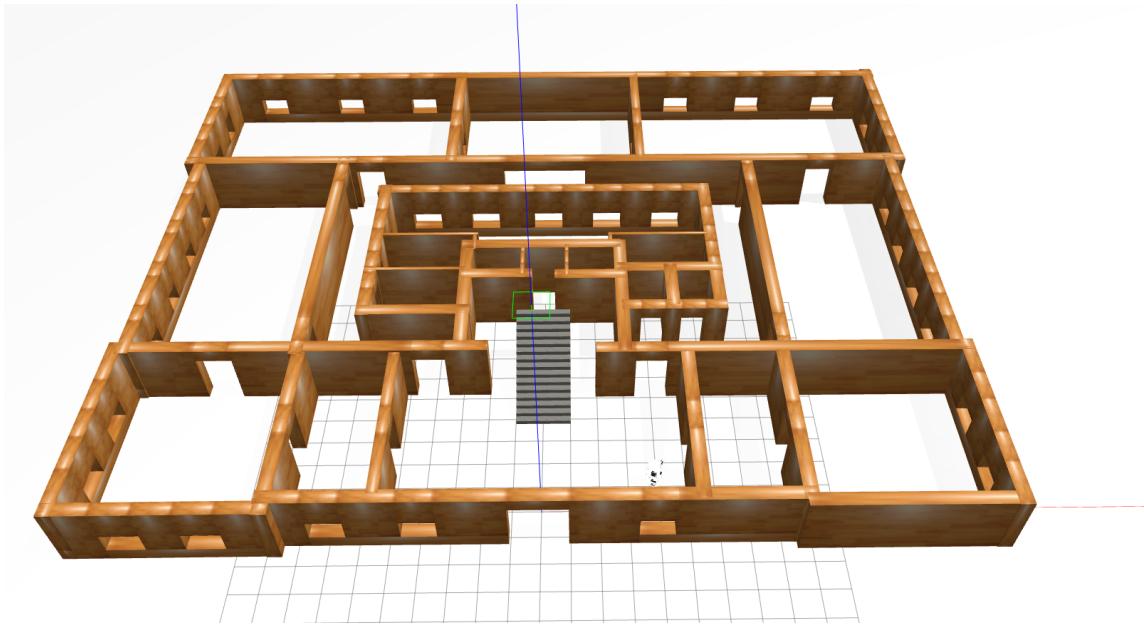
3.3 ESTIMATIVA DE POSIÇÃO DO ROBÔ

A estimativa de posição é feita utilizando o sistemas de coordenadas definido pela norma REP 105 (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 é utilizado. Todas transformadas entre estes sistemas de coordenadas são publicadas no tópico `tf`, por padrão do ROS 2. Estas transformadas são publicados por dois sistemas, o de odometria e de localização.

3.3.1 Odometria

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

Figura 7: Ambiente Gazebo com modelo do prédio Centenário da EE-UFRGS.



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 publicada apenas pelo controlador, através do pacote `twist_mux_controller`, utilizando dados do encoder nas rodas para estimar a posição do robô. O cálculo da posição é feito pelo pacote `arc_odometry`, que utiliza o modelo cinemático de um robô móvel de acionamento diferencial para estimar a posição e orientação do robô a partir das posições de junta das rodas. Para aprimorar a qualidade desta odometria, foi adicionado o pacote `robot_localization`, que realiza a fusão de dados da odometria das rodas com os dados do IMU.

reescrever e explicar melhor a parte da utilizacao da velocidade

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 (MOORE, 2016), 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. Portanto, a posição calculada pelo `robot_localization` com a velocidade publicado pelo `arc_odometry` não é correta.

Porém, utilizando os dados de posição da odometria de rodas, não é usado a velocidade para calcular a posição. Desta forma, a posição publicada pelo `robot_localization` é a mesma que a publicada pelo `arc_odometry`, já que esta é a única fonte de dados de posição. Porém, a orientação e velocidade angular da odometria das rodas são fundidas com os dados do IMU, melhorando a precisão destes campos na mensagem publicada.

Como alternativa a este sistema, foi implementado um sistema de odometria visual, publicado pelo executável `rgbd_odometry` do pacote `rtabmap_odom`, que utiliza imagens da câmera RGB-D com auxílio do sensor IMU para estimar a posição. Utilizando características das imagens RGB, com a informação de profundidade da imagem de profundidade,

eu tra-
duzi
"featu-
res" como
carac-
teristi-
cas, mas
talvez
seja um

é utilizado um método *Random Sample Consensus* (RANSAC), para comparar imagens consecutivas e estimar a velocidade do robô (LABBE, 2023).

É possível, 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 com a posição real do robô no Gazebo. Utilizando o *plugin P3D*, é publicada uma mensagem do tipo `nav_msgs/Odometry` com a posição do robô em relação ao sistema de coordenadas `odom` a um tópico auxiliar. Quando está sendo utilizado este sistema de odometria, o que é publicado neste tópico é retransmitido para o tópico `/odom` e o pacote `odom_to_tf_ros2` publica a transformada entre `odom` e `base_link` com base no dado de posição da mensagem publicada.

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

Tabela 1: Sistemas de odometria

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

a camera no rgbd odom usa dois topicos do tipo image, por isso que coloquei "(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

3.3.2 Localização

O sistema de localização, por outro lado, publica a transformada entre o sistema de coordenadas `map` e `odom`. Este transformada não deve acumular erros, porém não precisa ser contínua. Por esta razão, são utilizados principalmente sensores absolutos, que podem ter alta complexidade de processamento, já que não precisam ser atualizados em alta frequência, como a odometria.

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

Uma alternativa à sistemas SLAM é o `nav2_amcl`, que utiliza localização por Monte Carlo com um mapa construída previamente, que poderia ser a planta do prédio, para estimar a posição do robô. Porém, opção foi descartada, pois o objetivo é a utilização

do robô em ambiente dinâmicos ou pouco conhecidos, onde um mapa estático não seria adequado. Além disso, este pacote só é compatível com representações bidimensionais do ambiente, que causaria problemas na detecção de objetos tridimensionais como as escadas, agravando as diferenças já existem entre o ambiente do Gazebo e a planta.

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

Tabela 2: Sistemas de localização

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

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

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

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

3.4 MAPEAMENTO

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

Apesar do sistema de SLAM fornecer um mapa do ambiente, este mapa não 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, é preferível que o mapa da camada estática tenha apenas obstáculos fixos, como paredes e escadas, e utilizar a camada de obstáculos para obstáculos dinâmicos. Esta separação permite que o mapa estático seja utilizado em diversas sessões, porém com uma camada de obstáculos nova a cada sessão. Além disso, a camada de obstáculos pode ser configurada de forma independente que o SLAM, podendo utilizar outros sensores ou considerar uma altura de obstáculos diferente.

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

3.4.1 Mapeamento de custo

O mapa de custo utilizado é composto por três camadas: a camada estática, que possui uma representação simples do ambiente, preferencialmente sem obstáculos temporários; a camada de obstáculos ou *voxel*, onde são utilizados dados dos sensores óticos para atualizar o mapa com obstáculos percebidos; e a camada de inflação, que cria um campo de custo ao redor dos obstáculos.

A planta do prédio Centenário, representada na Figura 6, 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.

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.

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.

daria pra fazer uma tabela igual ao dos sistemas de odometria e localizacao, só que cada linha é uma camada do mapa de custo

3.4.2 Mapeamento SLAM

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

nao
achei
traducao
para ray
casting

Devido aos dados dos sensores utilizados o `slam_toolbox` é capaz de criar mapas apenas em duas dimensões. Porém, o `rtabmap_slam` pode criar mapas em duas ou três dimensões. Como o robô Twil não é capaz de se movimentar em três dimensões, neste trabalho só serão comparados os mapas em 2D criados por ambos pacotes.

poderia escrever mais nessa seção talvez

acho que podia falar um pouco sobre o loop closure aqui e porque nao testei em outro lugar(bag ficava muito grande porque o mapa é muito grande)

3.5 TESTES E COLETA DE DADOS

Devido a independência entre o mapeamento de custos e os demais sistemas desenvolvidos neste trabalho, foram realizados dois tipos de testes diferentes. Para testar o mapeamento de custo, foram executadas criadas trajetórias com e sem obstáculos para testar a performance das camadas de obstáculos e inflação. Para testar os sistemas de odometria, localização e mapeamento SLAM, foram utilizados arquivos de gravação de tópicos do ROS 2, chamados de sacolas, ou em inglês, *bags*.

Estas sacolas permitem a execução de diversas simulações usando os mesmos dados de entrada. Outro benefício foi o alívio na exigência de processamento do computador utilizado, já que foi realizar a simulação no Gazebo e executar os nodos de localização separadamente.

Para a gravação das sacolas, foi criado um programa em *bash* que contém todos os tópicos necessários para a execução dos sistemas de odometria e localização, além de tópicos auxiliares para comparação com a posição real do robô e visualização no rviz, como o tópico `/ground_truth`, que contém a posição real do robô, publicado pelo *plugin* P3D do Gazebo. Este tópico difere do tópico criado por este *plugin* utilizado no sistema odometria porque a posição deste é publicada em relação ao sistema de coordenadas `map` e não `odom`.

Os tópicos necessários para a execução dos sistemas de odometria estão nas Tabelas 1 e 2, descritos anteriormente. Porém, além destes que são inscritos explicitamente na configuração do pacote, os sistemas de odometria e localização que utilizam a câmera RGB-D também utilizam o tópico `/tf`, que contém a árvore de transformadas da base do robô até a câmera. Esta transformação é necessária para a utilização dos dados da câmera. Portanto, esta informação deve estar presente durante a execução dos testes.

Uma solução para disponibilizar esta transformada seria gravar o tópico `/tf` durante a execução da simulação. Porém, isto não é possível, porque este tópico já possui as transformadas entre `map`, `odom` e `base_link`, o que causaria conflitos com as transformadas publicadas pelos sistemas de odometria e localização sendo testados. Logo, durante a execução dos testes, devem ser publicadas as mesmas transformadas que foram publicadas durante a gravação da sacola.

A transformação das juntas do robô são publicadas pelo `robot_state_publisher`. Existem dois tipos de juntas entre a base do robô e a câmera: estáticas, que tem sua informação definida no arquivo de descrição do robô, e dinâmicas, que são publicadas pelo Gazebo, através das configurações definidas no `twil_bringup`, no tópico `joint_states`. O executável `robot_state_publisher` utiliza este tópico e o arquivo de descrição para publicar as transformadas entre os componentes do robô no tópico `/tf`. Portanto, para evitar a execução do Gazebo, este tópico foi gravado na sacola, e foi executado o `robot_state_publisher` com o arquivo de descrição do robô, obtendo assim, o mesmo estado do robô durante a execução da simulação utilizada para a gravação da sacola.

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

Os resultados dos testes foram analisados utilizando o programa *Plotjuggler* (FACTI, 2024), que permite criar gráficos de tópicos do ROS em tempo real ou a partir de uma sacola. Optou-se por criar outro programa em *bash* para gravar os tópicos do testes para criação dos gráficos no *Plotjuggler*. Desta forma, os dados dos testes ficam gravados e podem ser analisados posteriormente.

Os gráficos foram construídos utilizando os dados de posição x e y das mensagens do tipo `nav_msgs/Odometry` publicadas pelos sistemas de odometria e pelo Gazebo com a posição real do robô. Para os sistemas de localização, foram utilizados os dados de posição x, y das mensagens do tipo `geometry_msgs/PoseWithCovarianceStamped`, que são referentes a posição global do robô calculada por estes pacotes, ou seja, a transformada entre `map`, `odom` e `base_link`.

O *Plotjuggler* permite a utilização de programas escritos em Lua para manipulação dos dados. Aproveitando isto, é possível calcular o erro de posição entre a posição real e a posição estimada, conforme a Equação 1, e criar um gráfico com o erro de posição ao longo da distância real percorrida pelo robô.

$$d_{erro} = \sqrt{(x_{real} - x_{estimado})^2 + (y_{real} - y_{estimado})^2} \quad (1)$$

4 RESULTADOS

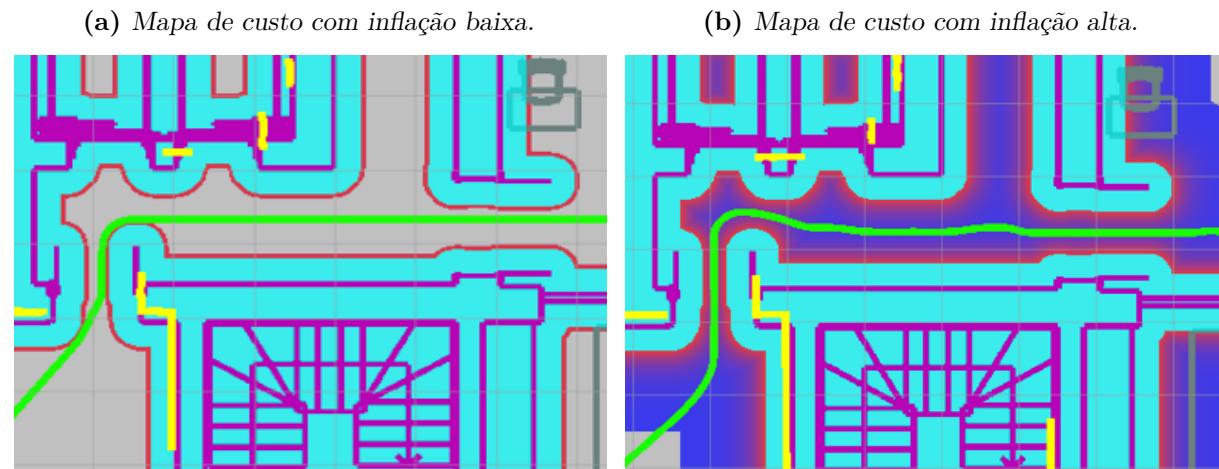
Neste capítulo, serão apresentados os resultados dos testes realizados, separados em seções. Primeiramente será apresentado o efeito das camadas do mapa de custo no planejamento de trajetórias. Em seguida, as estimativas de posição do robô, produzidas pelos sistemas de odometria, localização são comparadas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

4.1 MAPEAMENTO DE CUSTO

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

A camada de inflação porém, foi ajustada para melhorar a qualidade dos caminhos criados, de acordo com a recomendação de Zheng (2019). A camada de inflação original criava uma pequena zona de segurança ao redor dos obstáculos, com uma borda de 0,35 m e inclinação de 3,0. Representada na Figura 8a. 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.

Figura 8: Trajetórias criadas com diferentes configurações da camada de inflação



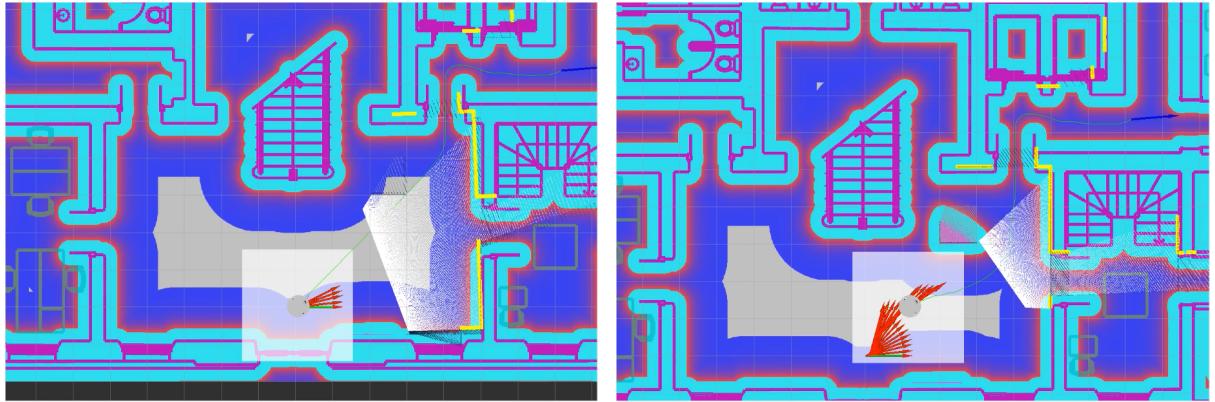
Finalmente, foi testada a camada de obstáculos adicionada neste trabalho, que utiliza dados da câmera RGB-D para adicionar novos obstáculos no mapa, mostrado na

tem que
ver quais
informa-
coes da
inflacao
eu coloco
aqui e
quais eu
coloco
na meto-
dologia

Figura 9. Para realizar este teste, foi adicionada uma caixa ao mundo do Gazebo, durante a simulação. Então, foi criada uma rota que passe por este obstáculo, como na Figura 9a. Ao se aproximar desta caixa, ela é detectada pela nuvem de pontos da câmera, representada como pontos brancos, e é adicionada ao mapa de custom, como é visto na Figura 9b.

Figura 9: Mapeamento de obstáculos da camada voxel

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



Percebe-se nestas duas imagens, que caso fosse utilizada uma representação plana do ambiente, na altura da câmera, que pode ser visto em amarelo nas Figuras 9a e 9b, o obstáculo não seria detectado, e o robô colidiria com a caixa.

4.2 TESTES DE ODOMETRIA, LOCALIZAÇÃO E MAPEAMENTO SLAM

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

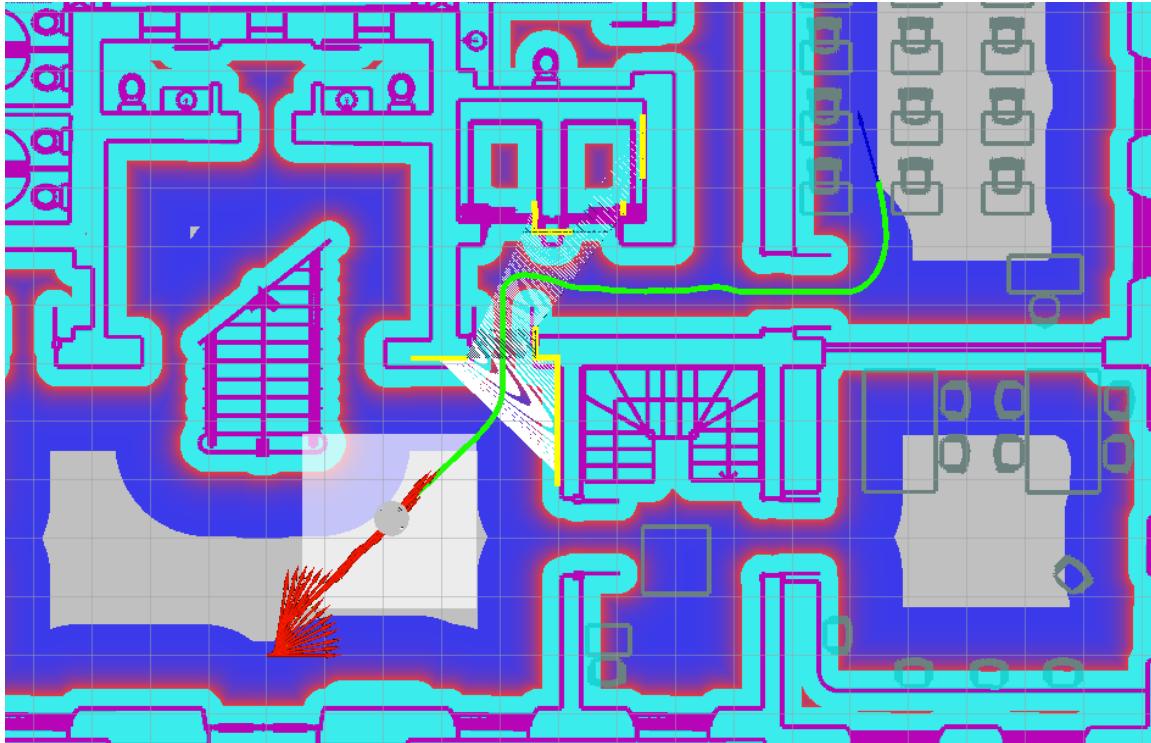
Na Figura 10, é mostrado o robô realizando a trajetória de testes. As setas em vermelho representam a mensagem de odometria do robô, onde é indicado a posição e orientação do robô. A linha verde indica a trajetória planejada do robô. O caminho realizado neste teste começa na origem, segue em direção 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. A distância total percorrida pelo robô foi de 13,08 m.

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.

4.2.1 Odometria

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. A

Figura 10: Robô durante o teste realizado.



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

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

A odometria das rodas, mostrada na Figura 11, apresentou uma grande divergência em relação a posição real do robô, com a curva da posição estimada inclinada para a direita, em relação a curva da posição real, como mostra a Figura 11a, indicando um erro na estimativa da orientação do robô.

O erro de orientação é especialmente indesejado porque, quando é causado, se transforma em erro de posição lateral e aumenta sem limite(BORENSTEIN; FENG, 1996). Na primeira parte do trajeto, da origem até a primeira porta, a orientação estimada divergiu constantemente, causando uma curva exponencial do erro, como é visto na Figura 11b. Este erro continuou crescendo até o final do trajeto, causando uma grande divergência na posição final, com uma distância de 6,8 m entre a posição real e a estimada.

Por outro lado, a odometria visual, mostrada na Figura 12, apresentou um resultado satisfatório. No gráfico da posição estimada, da Figura 12a, mostra que ocorreu a mesma inclinação para a direita que a odometria das rodas, porém menos acentuada. Como este gráfico não tem informação sobre o tempo, em certos da trajetória no corredor, a posição estimada parece estar igual a posição real, porém, na Figura 12b, é possível perceber que o erro de estimativa cresceu ao longo deste trajeto, atingindo um erro de 0,5 m no pico. Este erro diminuiu enquanto o robô percorria a curva final, porém logo aumentou novamente atingindo um erro de 0,66 m no final do trajeto.

Figura 11: Resultado da odometria das rodas.

(a) Posição estimada da odometria das rodas. (b) Erro de estimativa da odometria das rodas.

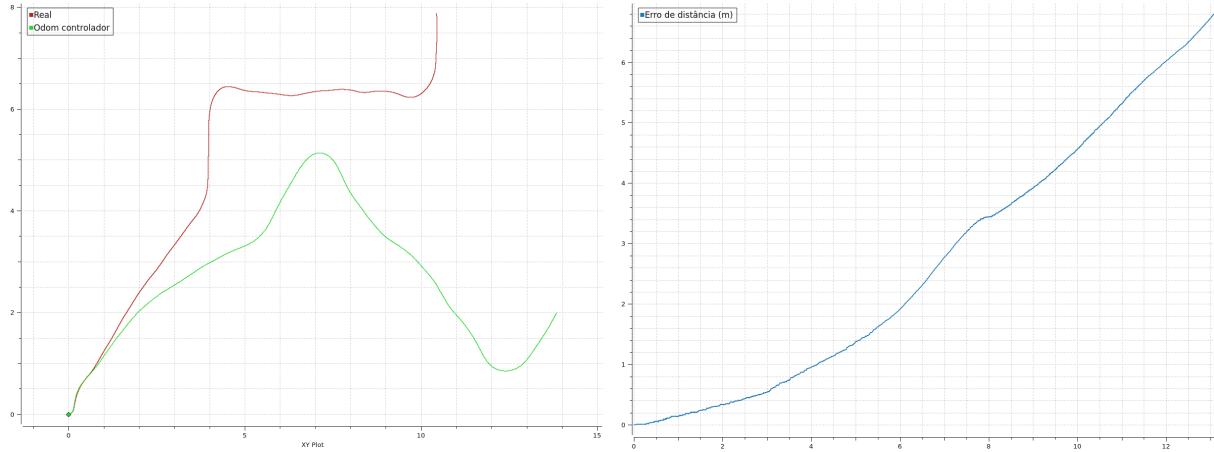
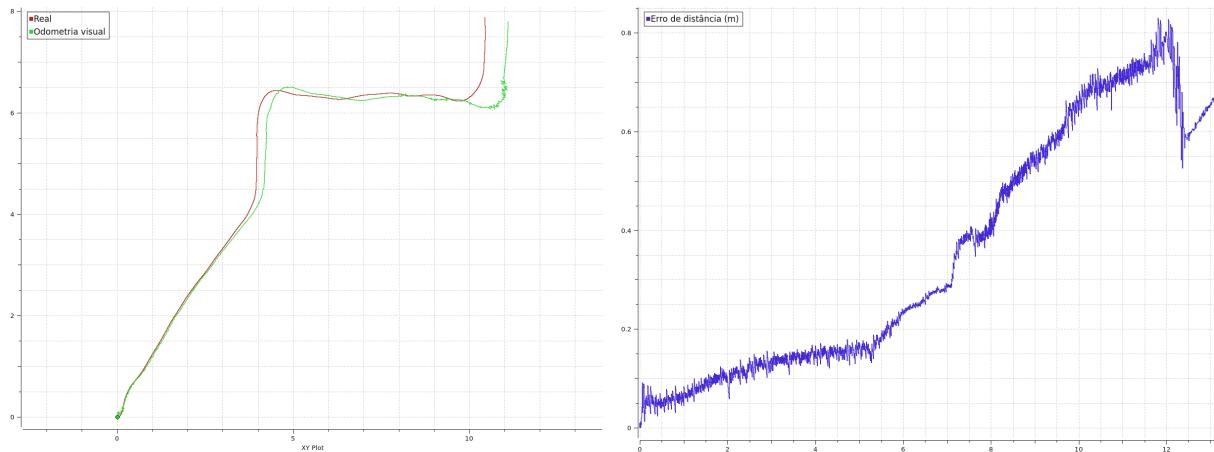


Figura 12: Resultado da odometria visual.

(a) Posição estimada da odometria visual. (b) Erro de estimativa da odometria visual.



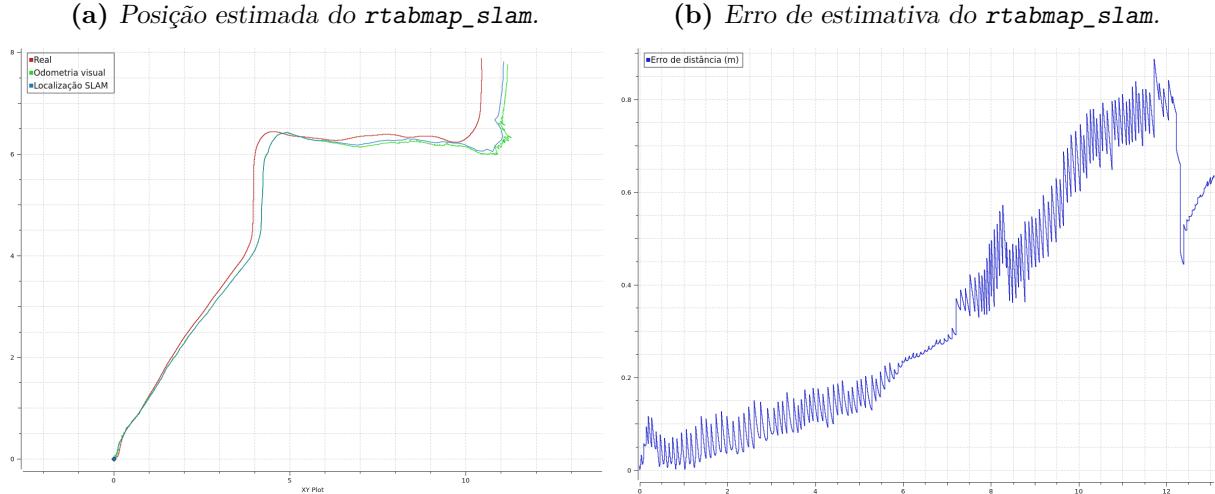
4.2.2 Localização

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

A Figura 13 mostra o resultado do sistema de localização do pacote `rtabmap_slam`. A localização neste caso teve um ajuste bem pequeno, diminuindo o erro para 0,63 m no pico na posição final. É possível perceber pelo gráfico da Figura 13b que a curva do erro manteve uma forma semelhante a curva do erro da odometria visual. Porém, houve menos ruído na estimativa de posição, como mostra a Figura 13a.

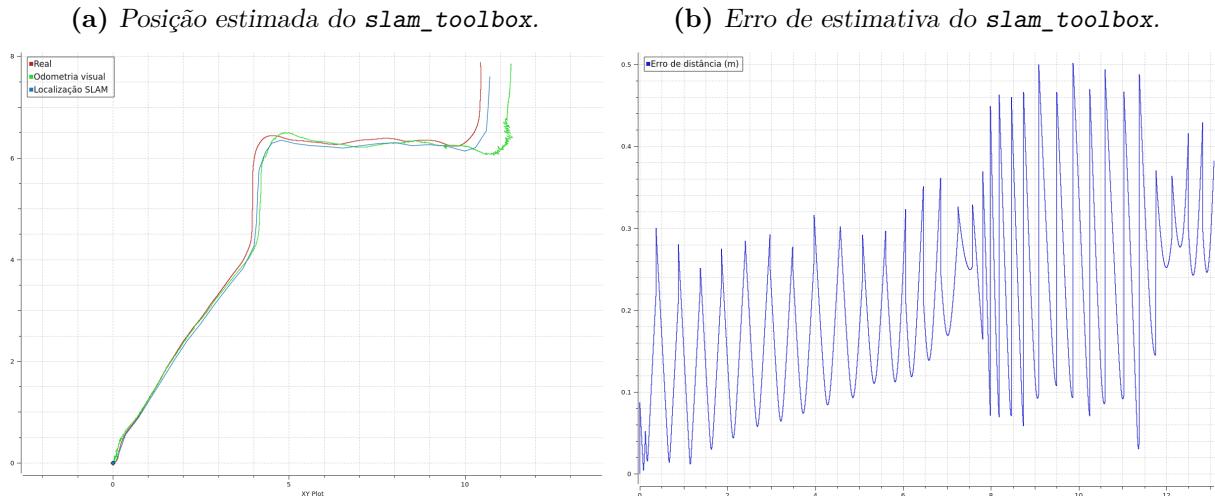
O `slam_toolbox`, que têm seu resultado representado na Figura 14, por outro lado, apresentou um resultado melhor, diminuindo o erro de estimativa para aproximadamente 0,37 m no final da trajetória. Isto aconteceu em grande parte porque não houve o acúmulo de erro durante o trajeto do corredor, como mostra a Figura 14b, que havia acontecido na odometria visual. A Figura 14a mostra que o ruído na estimativa de posição foi atenuado

Figura 13: Resultado do pacote `rtabmap_slam`.



da mesma forma que o `rtabmap_slam`.

Figura 14: Resultado do pacote `slam_toolbox`.



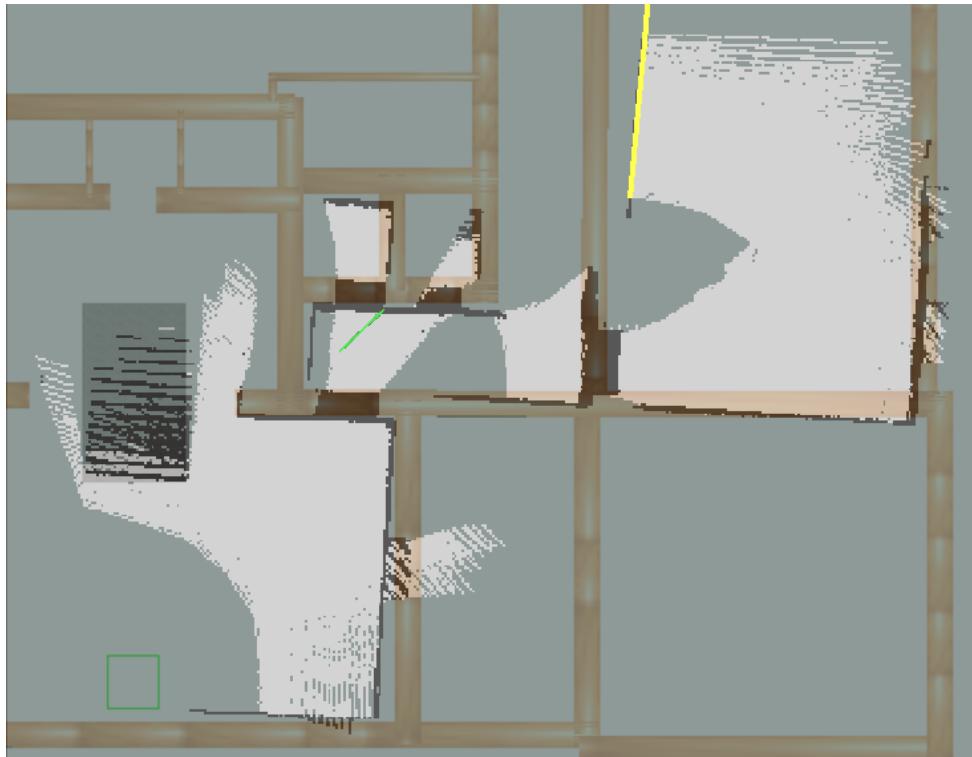
Nota-se que, apesar de utilizar a mesma entrada em todas execuções da simulação, a odometria visual apresentou resultados ligeiramente diferentes em cada execução. Isto acontece porque a máquina utilizada para a execução não possui capacidade de processamento suficiente para realizar os cálculos necessários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

4.2.3 Mapeamento SLAM

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

Na Figura 15, é mostrado o mapa gerado pelo `rtabmap_slam`, que cria um mapa utilizando os dados tridimensionais da câmera RGB-D, enquanto que na Figura 16 é mostrado o mapa gerado pelo `slam_toolbox`, que utiliza uma conversão da imagem de profundidade em um feixe de luz em duas dimensões. A representação do ambiente no Gazebo foi sobreposta aos mapas para facilitar a comparação. Nestes mapas, as áreas em branco claro são áreas livres, e as áreas em preto são consideradas ocupadas, enquanto que as áreas em cinza são desconhecidas.

Figura 15: Mapa construído pelo `rtabmap_slam`.

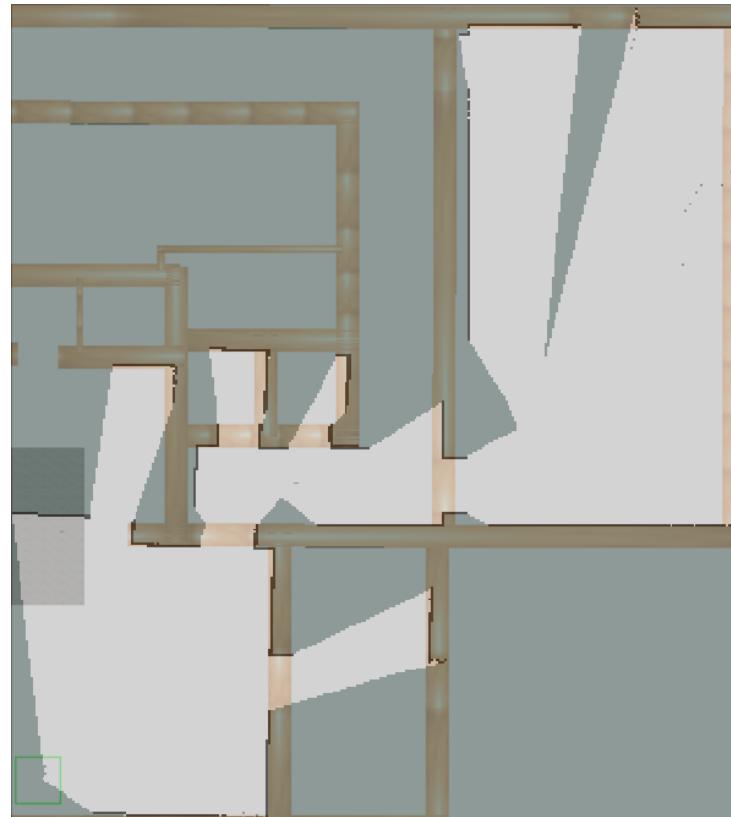


É possível perceber diferenças na geração destes mapas devido ao tipo de mensagem utilizado. O mapa gerado pelo `rtabmap_slam` gera um representação mais detalhada da escada, percebendo-a desde sua base, enquanto que o `slam_toolbox` só mapeia a escada na mesma altura da câmera, que é a altura da mensagem passada a ela. Além disso, o `rtabmap_slam` mapeou as molduras das portas, que estão acima da altura da câmera. Este comportamento, porém, pode ser modificado, já que não são obstáculos relevantes para a navegação.

Outro diferença é no mapeamento do chão dos mapas. Devido ao campo de visão da câmera, não é captado o chão em frente ao robô, que portanto não é mapeado pelo `rtabmap_slam`. Por outro lado, como o `slam_toolbox` utiliza um feixe de luz em duas dimensões, é considerado que todo espaço entre o robô e o primeiro obstáculo detectado é livre, mapeando assim o chão não captado pela câmera. Em situações em que não é detectado nenhum obstáculo dentro do limite da câmera, como a área entre o final da trajetória do robô e a porta da sala no canto superior direito da Figura 16, o chão não é mapeado.

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

Figura 16: Mapa construído pelo `slam_toolbox`.



do robô, apresentadas anteriormente, em que o `slam_toolbox` apresentou uma estimativa mais precisa.

5 DISCUSSÃO

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

Além disso, a câmera aprimorou a estimativa de posição do robô, que permite o seguimento correto da trajetória planejada. A combinação de odometria e localização que produziu a melhor estimativa foi a odometria visual, utilizando o pacote `rtabmap_odom` com a localização do pacote `slam_toolbox`.

A escolha da odometria se da pelo desempenho superior da odometria visual em relação a odometria das rodas. Porém, a odometria das rodas apresentou um resultado pior do que o esperado, podendo ser causado por erros na descrição do robô, o que pode ser ajustado. Além disso, a odometria visual pode apresentar resultados inferiores em ambientes reais, devido ao maior ruído presente fora da simulação, que foi feita em condições ideais.

Logo, em trabalhos futuros, recomenda-se ajustes no sistema de odometria das rodas, para garantir uma estimativa mais precisa da posição do robô. Como discutido anteriormente, a odometria das rodas apresentou um erro substantivo na estimada de odometria, que é o erro mais preocupante da odometria. Para mitigar este erro, é possível utilizar as informações de velocidade angular e orientação do IMU presente no robô para calcular a estimativa de posição.

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

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

Em relação ao mapeamento, o pacote `rtabmap_slam` criou um mapa mais detalhado do ambiente, devido a utilização de dados tridimensionais. Porém, isto não afetou o desempenho da navegação, com o `slam_toolbox` apresentando uma melhor estimativa de posição. Portanto, caso seja desejado um mapa do ambiente mais detalhado, recomenda-se a utilização do `rtabmap_slam`, em uma sacola de dados, e não durante a navegação em tempo-real, da mesma forma que foi feito neste trabalho.

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

6 CONCLUSÃO

Este trabalho atingiu o objetivo proposto de mapear o ambiente para navegação autônoma utilizando a câmera RGB-D do robô Twil, evitando assim colisões em ambientes dinâmicos ou pouco conhecidos. Porém, os testes foram realizados apenas em ambiente simulado, já que o sistema não foi implementado no robô real. Em trabalhos futuros, o desenvolvimento deste trabalho pode ser aplicado no robô real. Para isso, o Gazebo deverá ser dispensado e os dados deverão todos ser obtidos através do robô real. Porém, primeiramente deve ser realizado os ajustes na odometria das rodas, detalhado na discussão deste trabalho. Ao implementar o sistema no robô real, é possível que seja necessária uma recalibração dos sistemas de odometria e localização. Neste caso, recomenda-se a utilização dos programas de gravação de sacolas utilizados neste trabalho para análise dos resultados, já que não deve existir diferença entre os tópicos publicados pelo robô real e pela simulação.

REFERÊNCIAS

- AMAZON. *10 years of Amazon robotics: how robots help sort packages, move product, and improve safety.* 2022. Disponível em: <<https://www.aboutamazon.com/news/operations/10-years-of-amazon-robotics-how-robots-help-sort-packages-move-product-and-improve-safety>>. Acesso em: 4 ago. 2023.
- ATHAYDE, R. C. *Localização de robôs móveis no ROS.* 2021. Trabalho de Conclusão de Curso (Graduação em Engenharia de Controle e Automação) – Universidade Federal do Rio Grande do Sul, Porto Alegre.
- BORENSTEIN, J.; FENG, L. Measurement and correction of systematic odometry errors in mobile robots. *IEEE Transactions on Robotics and Automation*, v. 12, n. 6, p. 869–880, 1996. DOI: 10.1109/70.544770.
- FACONTI, D. *Plotjuggler.* [S.l.]: Davide Faconti, 2024. Disponível em: <<https://github.com/facontidavide/PlotJuggler>>. Acesso em: 21 jan. 2024.
- INTEL. *realsense-ros.* Santa Clara, CA, EUA: Intel. Disponível em: <<https://github.com/IntelRealSense/realsense-ros/>>. Acesso em: 05 de ago. de 2023.
- KALMAN, R. E. A New Approach to Linear Filtering and Prediction Problems. *Journal of Basic Engineering*, v. 82, n. 1, p. 35–45, mar. 1960. ISSN 0021-9223. DOI: 10.1115/1.3662552. eprint: https://asmedigitalcollection.asme.org/fluidseengineering/article-pdf/82/1/35/5518977/35_1.pdf. Disponível em: <<https://doi.org/10.1115/1.3662552>>.
- LABBE, M. *Nav2 Overview.* 2023. Disponível em: <https://wiki.ros.org/rtabmap_odom#rgbd_odometry>.
- 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.; TSAI, D.; FEINBERG, M. Spatio-temporal voxel layer: A view on robot perception for the dynamic world. *International Journal of Advanced Robotic Systems*, v. 17, n. 2, p. 1729881420910530, 2020. DOI: 10.1177/1729881420910530. eprint: <https://doi.org/10.1177/1729881420910530>. Disponível em: <<https://doi.org/10.1177/1729881420910530>>.

- 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>>.
- MARDER-EPPSTEIN, E. et al. The Office Marathon: Robust navigation in an indoor office environment. In: 2010 IEEE International Conference on Robotics and Automation. [S.l.: s.n.], 2010. P. 300–307. DOI: 10.1109/ROBOT.2010.5509725.
- 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.
- MOORE, T.; STOUCH, D. A Generalized Extended Kalman Filter Implementation for the Robot Operating System. In: PROCEEDINGS of the 13th International Conference on Intelligent Autonomous Systems (IAS-13). [S.l.]: Springer, jul. 2014.
- MOORE, T. *Configuring robot_localization*. 2016. Disponível em: <https://docs.ros.org/en/noetic/api/robot_localization/html/configuring_robot_localization.html>. Acesso em: 20 jan. 2024.
- NAVIGATION2. *Nav2 Overview*. 2020. Disponível em: <<https://navigation.ros.org/index.html>>. Acesso em: 30 jul. 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].