Projeto de Graduação

DEE DEPARTAMENTO DE ENGENHARIA ELÉTRICA

16 de Dezembro de 2023

SLAM com Odometria Visual e LiDAR 2D em um Robô Móvel Compacto

Lucas Simões de Almeida

PUC RIO

Projeto de Graduação

DEPARTAMENTO
DE ENGENHARIA
ELÉTRICA

SLAM com Odometria Visual e LiDAR 2D em um Robô Móvel Compacto

Estudante: Lucas Simões de Almeida

Orientador: Marco Antonio Meggiolaro

Co-Orientador: João Carlos Virgolino Soares

Trabalho apresentado como requisito parcial para a conclusão do curso de Engenharia de Controle e Automação na Pontifícia Universidade do Rio de Janeiro, Rio de Janeiro, Brasil.

Agradecimentos

Agradeço aos meus pais, Daniel Simões de Almeida e Carmen Lúcia Pereira da Silva de Almeida por me proporcionarem educação e diversas oportunidades.

Aos meus avós, Ana Clementina Vieira de Almeida e Armando Simões de Almeida por terem-me acolhido em sua própria casa durante toda minha graduação.

À toda minha família por todo o amor e carinho.

À minha companheira Amanda Ribeiro de Almeida por todo seu apoio, sempre me motivando a seguir em frente até mesmo nos dias mais difíceis.

À todos os meus amigos, pelas risadas e boa companhia.

À equipe de competição de robótica da PUC-Rio Riobotz por toda a experiência e aprendizado.

Aos membros do Laboratório de Robótica da PUC-Rio, que acompanharam o desenvolvimento deste projeto e instigaram a busca pelo conhecimento na área da robótica.

Ao meu Coordenador Eduardo Costa da Silva por todo seu auxílio durante minha graduação.

Ao meu Orientador Marco Antonio Meggiolaro pela oportunidade de realizar este projeto.

Ao meu Co-orientador João Carlos Virgolino Soares por todo o apoio com o desenvolvimento deste projeto e por me instruir no fascinante tema de SLAM.

Ao meu tutor Gabriel Fischer Abati por me auxiliar durante o projeto, sanando dúvidas pontuais e oferecendo possíveis soluções para diferentes problemas que surgiram.

Ao criador do projeto Nanosaur, Raffaello Bonghi pela inspiração e uso do seu material de código aberto.

E finalmente, agradeço ao Grande Arquiteto do Universo, pela benção da vida e existência.

Resumo

Este projeto de graduação traz como objetivo geral apresentar um robô compacto que utiliza de algoritmos probabilísticos e de mapeamento de código aberto para processar informações obtidas de um ambiente fechado através da fusão de dois sensores: Uma câmera de rastreamento Intel Realsense T265 capaz de realizar odometria visual e um LiDAR 2D Hokuyo URG-04-LX-UG01 capaz de identificar a distância de objetos com alta precisão. Como objetivos específicos pretende: Estudar e compreender o problema de SLAM e as metodologias consideradas estado da arte; Projetar e construir um robô móvel compacto com tração diferencial; Integrar o sistema mecânico, eletrônico e *software* do robô; Combinar a informação sensorial da odometria visual inercial da câmera e as distâncias obtidas pelo LiDAR para realizar a localização e mapeamentos simultâneos em ambientes internos; Demonstrar que o robô compacto desenvolvido é capaz de gerar um *occupancy grid map* de um ambiente enquanto estima sua própria localização, solucionando o problema de SLAM em tempo real e validar os resultados através de comparações entre a trajetória estimada pelo robô com um *ground truth* desenvolvido. Utilizou-se do *framework* ROS para auxiliar na distribuição de tarefas entre *hardware* e *software*. O robô foi inteiramente impresso em 3D seguindo o projeto *open-source* desenvolvido por Raffaelo Bonghi. O sistema foi testado com experiências em ambientes internos, demonstrando a capacidade do robô de efetuar *loop closures*, reconhecer as extremidades de um nó e criar um *occupancy grid map* em tempo real. O robô percorreu um ambiente fechado, tendo seu movimento controlado por teleoperação, obtendo o mapa 2D do ambiente, bem como a sua trajetória completa estimada. Apesar de ser um robô de pequeno porte com limitações de *hardware*, o sistema demonstrou ser capaz de solucionar o problema de SLAM em tempo real, e realizar a tarefa continuamente por longos períodos de tempo. A qualidade do mapeamento em ambientes internos e estáticos demonstrou condizer com a topologia do ambiente.

Palavras-chave: Robôs Móveis, Compacto, Código Aberto, SLAM

SLAM with Visual Odometry and 2D LiDAR in a Compact Mobile Robot

Abstract

The overall aim of this graduation project is to present a compact robot that uses open-source probabilistic algorithms and mapping to process information obtained from a closed environment through the fusion of two sensors: An Intel Realsense T265 tracking camera capable of performing visual odometry and a Hokuyo URG-04-LX-UG01 2D LiDAR capable of identifying the distance of objects with high precision. The specific objectives are: To study the SLAM problem and the methodologies considered state of the art; To design and build a compact mobile robot with differential drive; To integrate the robot's mechanical, electronic and software; To combine the sensory information from the camera's visual inertial odometry and the distances obtained by the LiDAR to perform simultaneous localization and mapping in indoor environments; To demonstrate that the compact robot developed is capable of generating a *occupancy grid map* of an environment while estimating its own location, solving the SLAM problem in real time and to validate the results by comparing the estimated trajectory by the robot with a developed *ground truth*. The ROS *framework* was used to help distribute tasks between *hardware* and *software*. The robot was entirely 3D printed following the *open-source* project developed by Raffaelo Bonghi. The system was tested with indoor experiments, demonstrating the robot's ability to perform *loop closures*, recognize the edges of a node and create a *occupancy grid map* in real time. The robot traveled through a closed environment, with its movement controlled by teleoperation, obtaining a 2D map of the environment, as well as its complete estimated trajectory. Despite being a small robot with *hardware* limitations, the system proved capable of solving the SLAM problem in real time and performing the task continuously for long periods of time. The quality of the mapping in indoor and static environments proved to match the topology of the environment.

Keywords: Mobile Robots, Compact, Open-Source, SLAM

Sumário

Lista de Figuras

Lista de Tabelas

1 Introdução

Manipuladores robóticos são amplamente utilizados em processos industriais automatizados, geralmente desenvolvidos para a automatização de tarefas repetitivas. No entanto, estes robôs são normalmente fixos e só podem trabalhar em um espaço confinado. Em contrapartida, robôs móveis autônomos são capazes de se deslocar, o que permite um espaço maior de configuração nas mesmas aplicações dos robôs fixos e também abrindo possibilidade para novas tarefas [1].

As aplicações para robôs móveis são extensas e variadas, abrangendo desde a exploração de ambientes desconhecidos e perigosos para seres humanos até a execução de tarefas logísticas e de manutenção autônoma, também sendo eficientes em automatizar tarefas repetitivas. Um exemplo é o HyQReal, apresentado na Fig. 1, desenvolvido pelo Dynamic Legged Systems lab do Instituto Italiano de Tecnologia. Além de ter capacidades autonômas, o robô possui um sistema de locomoção hidráulico, demonstrando ter torque o suficiente para rebocar um avião de 3 toneladas.

Figura 1: Robô quadrúpede HyQReal rebocando um avião de 3 toneladas [2]

Para ser totalmente autônomo, um robô móvel precisa ter uma estimativa de sua pose (posição e orientação) e de um mapa de seu ambiente de operação, para determinar corretamente os seus comandos de movimento e planejar a sua trajetória. No entanto, robôs normalmente não dispõe desta informação *a priori*. Portanto, tem de estimar simultaneamente o mapa e a pose utilizando apenas da informação de seus sensores a bordo. Esta situação é comumente conhecida na literatura como o problema da Localização e Mapeamento Simultâneos (SLAM) [3] [4].

O SLAM é considerado um problema fundamental na robótica móvel. Solucionar este problema normalmente significa que o robô móvel dispõe de informação geométrica sobre os obstáculos no ambiente e sobre a sua posição em relação a esses obstáculos, o que lhe permite efetuar uma navegação autônoma. Um exemplo se encontra na Fig. 2, um carro completamente autônomo desenvolvido pela empresa Google.

Um problema recorrente é que a informação sensorial não é totalmente confiável, devido à presença de erros de ruído e quantização, logo o robô necessita de uma forma robusta de tratar estes dados sensoriais. Ambientes não estruturados e o movimento do robô também proporcionam alta incerteza e não linearidade. Estes problemas podem ser resolvidos com algoritmos probabilísticos, como os filtros de Kalman, a estimação *máxima a posteriori* (MAP) e os filtros de partículas [3] [6]. Os métodos de estimativa MAP trazem algumas vantagens em comparação com os outros métodos de filtros, em termos de precisão e custo computacional [7] [8].

Os sensores mais utilizados para aplicações de SLAM 2D são um LiDAR ou uma câmera [9]. Os LiDARs são mais apropriados para a construção de mapas 2D porque têm medições de profundidade com alta

Figura 2: Veículo autônomo desenvolvido pela Google [5]

precisão. A principal desvantagem deste tipo de sensor é seu elevado custo. As câmeras estéreo, por outro lado, têm maior riqueza de informação e são mais baratas, mas as suas estimativas de profundidade são menos precisas do que medir diretamente. As câmeras RGB-D também são uma alternativa de baixo custo capaz de obter diretamente medições de profundidade.

Além dos problemas mencionados anteriormente, robôs móveis especializados em localização e mapeamento simultâneos frequentemente não são compactos [10]. Esta característica restringe o acesso do veículo em espaços confinados, como tubulações, destroços, túneis, entre outros locais estreitos. Aplicações de SLAM em robôs móveis compactos podem ser desafiadoras, além de ser necessário que o robô projetado tenha um *hardware* capaz de processar os complexos algoritmos probabilísticos e de mapeamento, é também necessário que o robô tenha locomoção e sensores a bordo integrados em seu sistema. Na Fig. 3 é apresentando um robô autônomo com esteiras capaz de operar em ambientes irregulares, entretanto não compacto.

Figura 3: Veículo autônomo com esteiras em terreno irregular [11]

Para solucionar este problema, este projeto de graduação inclui o desenvolvimento de um robô móvel compacto capaz de realizar mapeamento e localização 2D de ambientes fechados, combinando informações obtidas por uma câmera de rastreamento Intel Realsense T265 que realiza odometria visual e um sensor LiDAR Hokuyo URG-04-LX-UG01 para identificar a distância de objetos. Essas informações são processadas em uma placa NVIDIA Jetson Nano capaz de suportar processamento digital de imagens.

Utilizando de algoritmos probabilísticos de estimação de estado de código aberto, é proposta uma estrutura baseada no *framework Robotic Operational System* (ROS) para integrar os dados de ambos os sensores e executar um mapeamento e localização simultânea 2D, baseado na estimativa MAP. A proposta é combinar a odometria visual da câmera e as distâncias obtidas pelo LiDAR para criar um *occupancy grid map*, ao mesmo tempo que se estima a pose do robô em tempo real. Durante os testes o robô percorrerá um ambiente fechado, tendo seu movimento controlado por teleoperação, obtendo o mapa 2D do ambiente, bem como a sua trajetória completa estimada.

2 Objetivos

Objetivo geral

Desenvolver um robô compacto que utiliza de algoritmos probabilísticos e de mapeamento de código aberto para processar informações obtidas de um ambiente fechado através da fusão de dois sensores: uma câmera de rastreamento Intel Realsense T265 capaz de realizar odometria visual inercial e um LiDAR 2D Hokuyo URG-04-LX-UG01 capaz de identificar a distância de objetos com alta precisão.

Objetivos específicos

- Estudar o problema de SLAM e as metodologias consideradas estado da arte;
- Projetar e construir um robô móvel compacto com tração diferencial;
- Integrar o sistema mecânico, eletrônico e *software* do robô;
- Combinar a informação sensorial da odometria visual da câmera e as distâncias obtidas pelo LiDAR para realizar a localização e mapeamentos simultâneo em ambientes internos;
- Demonstrar que o robô compacto desenvolvido é capaz de gerar um occupancy grid map de um ambiente, enquanto estima sua própria *pose* neste mapa, solucionando o problema de SLAM em tempo real;
- Validar os resultados através de comparações entre a trajetória estimada pelo robô com um *ground truth* desenvolvido.

3 Revisão Bibliográfica

O objetivo deste capítulo é apresentar uma revisão bibliográfica dos temas relacionados à esse trabalho. A revisão foi dividia em três seções. A primeira refere-se ao conceito de SLAM e sua importância na robótica contemporânea. O segundo aborda a categoria de robôs móveis, correlacionando os desafios únicos envolvidos no desenvolvimento destes robôs e suas aplicações para SLAM. Finalmente, abordaremos a robótica probabilística, um tema fundamental para o desenvolvimento de robôs móveis.

A Introdução ao SLAM

O problema de SLAM trata-se de colocar um robô móvel em um ambiente desconhecido, para que o robô construa incrementalmente um mapa consistente deste ambiente enquanto determina simultaneamente sua localização neste mapa. Solucionar o problema de SLAM é crucial para diversas aplicações, incluindo robótica móvel, veículos autônomos e operações de logística. Segundo Durrant-Whyte e Bailey [3], solucionar o problema de SLAM, máquinas ganham a habilidade de interpretar e navegar em territórios desconhecidos, representando um passo essencial para a verdadeira autonomia robótica.

Historicamente, o problema de SLAM foi formulado e abordado de múltiplas maneiras, com implementações em variados domínios, desde robôs em ambientes ao ar livre e internos até contextos subaquáticos e aéreos. Pensando em encontrar soluções para o problema de SLAM, foram pensandos na utilização de uma variedade de sensores, incluindo: LiDARs, radares, câmeras, codificadores, GPS e IMUs [3].

SLAM envolve a determinação simultânea do estado de um robô equipado com sensores e a criação de um modelo cartográfico do ambiente percebido por esses sensores. O estado do robô é definido por sua *pose*, mas pode incluir outras variáveis como velocidade, atrito e parâmetros de calibração. Cadena et al. [4] descreve que o mapa é uma representação dos elementos relevantes no ambiente, como a localização de pontos de referência e obstáculos. Acrescenta Grisetti et al. [8] que a escolha de uma representação particular do mapa depende dos sensores utilizados, das características do ambiente e do algoritmo de estimativa.

Muitos autores discutem a representação de ambientes através de mapas, geralmente destacando dois objetivos principais. O primeiro é a utilidade desses mapas em tarefas secundárias, como ajudar no planejamento de rotas ou proporcionar uma visão clara aos operadores humanos. O segundo objetivo é minimizar os erros das estimativas de estado do robô. Sem um mapa, a navegação puramente baseada em estimação acumularia erros significativos ao longo do tempo. Contudo, com um mapa que inclua pontos de referência distintos, o robô pode corrigir seus erros de localização ao revisitar áreas já conhecidas, um processo conhecido pela literatura como *loop closure*. Portanto, solucionar o problema de SLAM é vital em cenários onde não existe informação do ambiente a priori [3] [4] [6] [8] [9] [12].

A crescente popularidade do SLAM está ligada à expansão das aplicações de robótica móvel em ambientes internos, a qual elimina a possibilidade de usar GPS para determinar a localização. O SLAM surge como uma alternativa viável aos mapas criados manualmente, possiblitando a operação de robôs sem a necessidade de uma estrutura de localização externa ou guias para o robô [4] [13].

A estrutura do *pose-graph* SLAM simplifica o problema de estimativa ao abstrair os dados brutos das medições. Grisetti et al., apresentam esse método em detalhe em [8]. Os autores mencionam que as recentes descobertas para a época, sobre a estrutura do problema SLAM e os avanços nos domínios de álgebra linear de matrizes esparsas resultaram em abordagens eficientes para o problema de otimização de grafos, permitindo que o método alcançasse o estado da arte.

Em sua dissertação de mestrado, Luknanto [7] avalia os diferentes tipos de SLAM 2D mais empregados em sistemas robóticos atualmente, enfatizando a importância da otimização e do custo computacional envolvido em aplicações de SLAM. Dos quatro algoritmos avaliados pelo autor, dois possuem uma estrutura de grafos e representação cartográfica de *occupancy grid maps*: o Google Cartographer e a SLAM toolbox. Debeunne e Vivet [14] avaliam as diferentes metodologias de SLAM focando em soluções que envolvem fusão sensorial com LiDARs.

Identificando a necessidade de um *framework* de SLAM robusto, em 2021 Macenski e Jambrecic se dedicaram ao desenvolvimento da SLAM Toolbox, um *framework* que se destaca por sua versatilidade e facilidade de integração. Este *framework*, completamente *open-source*, permitiu um fácil acesso ao problema de SLAM, contribuindo significativamente para a pesquisa e desenvolvimento de possíveis soluções do problema. A SLAM Toolbox foi utilizada neste projeto e é descrita em detalhes na seção de metodologia [13].

B Robôs móveis

Robôs móveis requerem multidisciplinaridade para serem propriamente elaborados. Roland [1] em seu livro *Introduction to autonomous mobile robots* enfatiza que apesar de que os robôs móveis tenham um vasto mercado, necessitam integrar diferentes áreas do conhecimento, como cinemática, visão computacional, teoria da informação, inteligência artificial, teoria da probabilidade, dentre outras.

Existem dois tipos de robôs móveis com rodas: robôs móveis omnidirecionais que independem de sua orientação para se mover e robôs móveis não holonômicos que dependem de sua orientação para se mover. No capítulo 13 do livro *Modern Robotics Mechanics,Planning, and Control*, escrito por Lynch e Park [15], é detalhado a modelagem e controlabilidade, bem como o planejamento de movimento, planejamento de trajetória e o controle de *feedback* de ambos tipos de robôs móveis com rodas. Interessante destacar que Lynch e Park finalizam cada capítulo com um resumo dos principais aspectos abordados, sugerem referências para leitura e trazem uma série de exercícios, configurando-se como um excelente livro didático para quem se propõem a ampliar o conhecimento sobre a robótica.

A odometria na robótica móvel é o processo de estimar a *pose* do chassi a partir da medição do movimento das rodas e de cálculos da cinemática do robô. Como já indica o nome, estimativas não são representações absolutas da realidade e possuem erros como a derrapagem das rodas ou erros de integração numérica. Lynch e Park [15] denotam a necessidade de completar a informação da odometria com outros sensores como: o GPS, o reconhecimento visual de pontos de referência, sensores de ultrassom, dentre outros. Enfatizam também, que a odometria fornece resultados melhores para curtos períodos de tempo, sendo necessário uma correção das estimativas odométricas periodicamente com outras modalidades, por exemplo o uso de filtros digitais para corrigir as estimativas, como o filtro Kalman, o filtro de partículas ou similares.

A modelagem de veículos com esteiras é complexa, especialmente em casos onde o sistema percorre uma superfície irregular, ou seja, uma superfície que possui constantes variações de atrito e linearidade. O trabalho de Santos [16] investiga soluções para o controle de robôs móveis tracionados por esteiras, comparando o modelo cinemático de um veículo tracionado por esteiras com a de um robô de tração diferencial. No movimento das esteiras existe um ponto de contato médio, que pode ser considerando um ponto equivalente de contato do robô. O Autor enfatiza que robôs móveis com esteiras possuem uma dinâmica de rotação similar ao de um veículo de tração diferencial, permitindo considerar este ponto médio de contato como ponto de centro de rotação instantâneo. Essa definição aproxima o modelo cinemático de pequenos robôs tracionados por esteiras para um modelo de tração diferencial, pois o movimento linear e angular resultante desses dois tipos de robôs é muito semelhante. As divergências entre estes modelos surgem em casos de derrapagem e deslizamento, tornando necessário considerações especiais quando um veículo com esteiras é modelado para aplicações de terreno irregular.

A ausência de um sinal de GPS e a baixa luminosidade impossibilitam a utilização de múltiplos sensores para a estimação da *pose* de um robô em um ambiente confinado ou interno. Além disso, solos escorregadios e irregulares acarretam acúmulos expressivos de erros durante o cálculo da odometria através do giro das rodas [10].

Com relação aos problemas citados, Cruz Junior et al. [10] correlaciona as necessidades de um sistema projetado para ambientes confinados com as aplicações de SLAM, enfatizando a importância da escolha de sensores robustos para as operações neste tipo de ambiente. O autor afirma que o sensor de percepção mais comumente utilizado para aplicações de localização e mapeamento em ambientes internos é o LiDAR, que é capaz de fornecer posicionamentos mais precisos quando na presença de obstáculos dinâmicos e em ambientes irregulares. O LiDAR independe de um sinal de comunicação externo como o GPS e é um dos sensores mais robustos para ambientes de baixa luminosidade, além disso existem metodologias capazes de estimar a odometria pelo uso do LiDAR, o que configura este sensor como a melhor escolha para aplicações de SLAM em ambientes internos e confinados.

C Robótica Probabilística

Publicado em 2005 por Thrun, Burgard e Fox, o livro *Probabilistic Robotics* é uma obra fundamental no campo da robótica probabilística. Pela primeira vez, foram apresentados de maneira formal e detalhada os métodos para minimizar e gerenciar as incertezas estatísticas na percepção e no controle de sistemas robóticos. Este livro é especialmente relevante para o tema de SLAM, descrevendo métodos probabilísticos que continuam sendo referências na área até os dias atuais [17].

As técnicas probabilísticas na robótica se beneficiaram de décadas de pesquisa, integrando diferentes campos como teoria da probabilidade, estatística e busca e otimização computacional. A robótica probabilística

tem paralelos com vários subcampos da inteligência artificial, incluindo visão computacional e linguagem e fala. Ressalta-se que as técnicas probabilísticas resolveram diversos problemas pendentes da robótica e proporcionaram novos *insights* teóricos sobre a estrutura dos problemas e suas soluções [18].

As técnicas da robótica probabilística permitiram lidar com as incertezas inerente às percepções e ações dos robôs [19]. Esta abordagem ofereceu um grande avanço para aplicações de robótica, considerando que em situações reais é quase impossível obter informações completas e precisas, devido as imperfeições dos sensores, ruído nas leituras, limitações durante o armazenamento de dados, incertezas na dinâmica do ambiente e diversos outras características que acumulam erro nas medições. De maneira a contornar estes desafios, a robótica probabilística recorre à métodos estatísticos e probabilísticos para melhor entender e representar a incerteza inerente dos sensores.

Um componente fundamental da Robótica Probabilística é a modelagem da incerteza por meio de funções de distribuição de probabilidade (PDF) [19]. Exemplificando: "um robô que busca determinar sua própria localização em um ambiente desconhecido pode usar uma distribuição de probabilidade para representar todas as possíveis posições onde ele pode estar, dadas as informações coletadas até o momento". À medida que o robô vai obtendo mais dados, ele consegue melhorar e refinar sua posição reduzindo a incerteza quanto a percepção de sua posição. Neste sentido, a abordagem probabilística permite a utilização de robôs em diferentes aplicações, como em sistemas de navegação autônoma com planejamento de trajetória, por exemplo, utilizando da probabilidade das medições coletadas do ambiente serem verdadeiras, criando assim uma trajetória que evita possíveis colisões.

4 Fundamentos Teóricos

Neste capítulo, abordaremos os conceitos teóricos essenciais para a elaboração das metodologias empregadas. Iniciamos com a apresentação do modelo de robôs com tração diferencial, e em seguida os conceitos probabilísticos necessários. Posteriormente, discutiremos os princípios relacionados aos sensores, incluindo odometria visual e as aplicações de LiDARs na robótica. Logo após, será apresentada a formulação e estrutura do problema de SLAM. Por último, a teoria envolvida na representação cartográfica de *occupancy grid maps*.

A Robôs móveis com tração diferencial

O robô de tração diferencial é uma das arquiteturas mais básicas e eficientes para robôs móveis com rodas. Essa configuração inclui duas rodas independentes, que giram em torno de um mesmo eixo e possuem um raio r. Para garantir estabilidade e manter o robô nivelado, ele também pode ser equipado por rodas passivas deslizantes de baixo atrito, também conhecidas como *casters*. Considere a Fig. 4. O robô é posicionado em um eixo de coordenadas global (x', y'). O ângulo de esterçamento do robô é ϕ. O ponto de referência local do robô é identificado pelo ponto $P = (x, y)$. Este ponto é normalmente obtido na posição central dos atuadores, entretanto, considerando uma aproximação do modelo cinemático de esteiras, o ponto P é posicionado no ponto médio de contato das esteiras. A cinemática do veículo de apresentado é descrita pelas Eqs. 1 e 2: [15] [16]

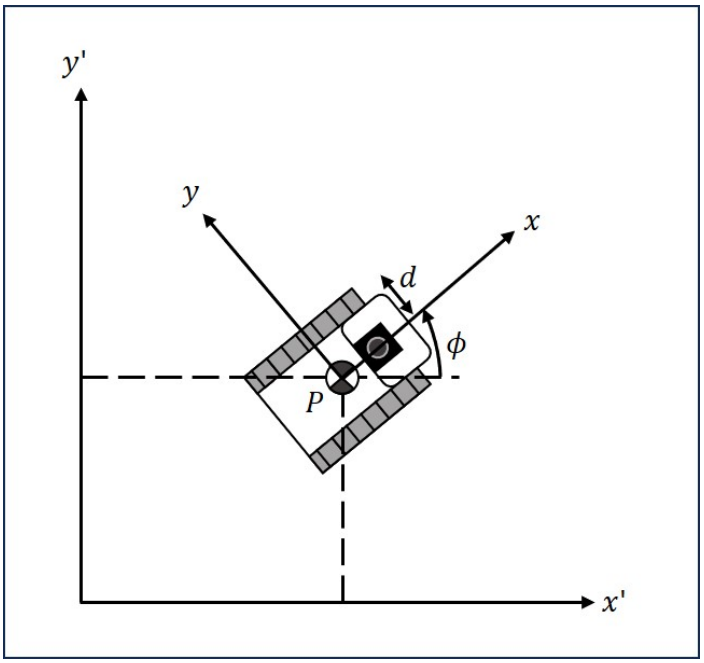

Figura 4: Modelo cinemático do robô de esteiras aproximado para o robô de tração diferencial

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\phi} \end{bmatrix} = \begin{bmatrix} \cos\phi & 0 \\ \sin\phi & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix} \tag{1}$$

Para melhor representar o modelo cinemático de esteiras, consideramos que $\dot{\phi}_D$ e $\dot{\phi}_E$ são as velocidades tangeciais lineares das esteiras direita e esquerda, respectivamente, e que é possível controlar estas velocidades de forma independente. Uma velocidade tangencial linear positiva em qualquer uma das esteiras indica que a respectiva esteira está se movendo para frente.

$$\begin{bmatrix} v \\ \omega \end{bmatrix} = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & \frac{-1}{2} \end{bmatrix} \begin{bmatrix} \dot{\phi}_D \\ \dot{\phi}_E \end{bmatrix} \tag{2}$$

Ao combinar as Eq. 1 e 2, torna-se possível descrever um vetor de estados $q = (x, y, \phi)$ em relação à

velocidade das esteiras. Tomando em base essa configuração, é possível representar a cinemática do robô pela Eq. 3.

$$\dot{q} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\phi} \end{bmatrix} = \begin{bmatrix} \frac{1}{2}\cos\phi & \frac{1}{2}\cos\phi \\ \frac{1}{2}\sin\phi & \frac{1}{2}\sin\phi \\ 1/2 & -1/2 \end{bmatrix} \begin{bmatrix} \dot{\phi}_D \\ \dot{\phi}_E \end{bmatrix} \tag{3}$$

Um robô equipado com tração diferencial possui duas vantagens notáveis: sua simplicidade de construção e sua alta capacidade de manobra. Em termos de simplicidade, normalmente cada roda é acionada diretamente por um motor ligado ao seu eixo, o que facilita a montagem e a manutenção do sistema. Quanto à capacidade de manobra, essa configuração permite que o robô execute rotações no próprio eixo, girando as rodas em direções opostas. Esta característica confere ao robô uma grande agilidade, especialmente útil em espaços restritos. Contudo, um ponto a ser considerado é que as rodas passivas, embora eficientes em ambientes internos, podem não ser a melhor escolha para uso externo. Apresentando dificuldade em deslocamento de terrenos irregulares ou em superfícies que demandam maior tração, limitando assim a aplicabilidade do robô em ambientes ao ar livre [1] [15].

B Conceitos Probabilísticos

B.1 Teoria da Probabilidade

Na robótica probabilística utiliza-se de variáveis aleatórias para representar o estado de um robô e suas medições. Considere X como uma variável aleatória e que $p(x)$ é a probabilidade de que X assuma um valor específico x. Esta relação é expressa na Eq. 4. [17]

$$p(x) = p(X = x) \tag{4}$$

Como o problema de SLAM pretende estimar o estado contínuo do robô e do ambiente, a equação acima não é aplicável, uma vez que esta considera variáveis aleatórias discretas. Para variáveis aleatórias contínuas, pode-se estimar a probabilidade de que a variável contínua se encontra dentro de um intervalo, possibilitando sua representação no formato de uma função de densidade de probabilidade não negativa.

B.2 Distribuição Gaussiana

A distribuição Gaussiana é muitas vezes usada para modelar a incerteza nos problemas de SLAM. Esta distribuição possui dois parâmetros principais: a média μ e a matriz de covariância Σ [17]. A Eq. 5 descreve a densidade de probabilidade de uma distribuição gaussiana multivariada, onde μ é um vetor e Σ é uma matriz simétrica positiva semidefinida.

$$p(x) = N(x;\mu,\Sigma) = det(2\pi\Sigma)^{1/2} exp\{\frac{-1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\} \tag{5}$$

Para facilitar o uso da Eq. 5 para aplicações de SLAM, pode-se substituir a matriz de covariância pela matriz de incertezas Ω, assim como a alteração do vetor de incertezas no lugar de μ. Resultando nas Eqs. 6 e 7 [17].

$$v = \Sigma^{-1}\mu \tag{6}$$

$$\Omega = \Sigma^{-1} \tag{7}$$

O que nos leva à Eq. 8.

$$p(x) = N(x;v,\Omega) = nexp\{\frac{-1}{2}x^T\Omega x + x^T v\} \tag{8}$$

B.3 Probabilidade Condicional

Considere a probabilidade de um evento aleatório ocorrendo por si só $p(x)$. Agora considere que a probabilidade do evento descrito seja influenciada devido à presença de outro evento aleatório $p(y)$ ter ou não ocorrido. Através da Eq. 9 é possível descrever a relação probabilísticas das variáveis aleatórias x e y descritas [17].

$$p(x|y) = \frac{p(x,y)}{p(y)} \tag{9}$$

onde $p(x,y)$ é designada por distribuição conjunta.

Utilizando da regra de Bayes que relaciona duas probabilidades condicionais, tem-se a Eq. 10.

$$p(x|y) = \frac{p(y|x)p(x)}{p(y)} \tag{10}$$

A regra de Bayes é fundamental para o problema de SLAM. Correlacionar duas probabilidades condicionais assegura que os dados obtidos são atualizadas devido a novas considerações probabilísticas. Considerando um robô abstraindo medições pelo ambiente, pode-se dizer que a crença inicial da posição dele é $p(x)$. Ao obter uma nova informação do ambiente $p(y)$, é possível utilizar da regra de bayes e atualizar as estimativas, obtendo uma probabilidade posterior $p(x|y)$.

O teorema da probabilidade total, enunciado na Eq. 11 para o caso contínuo, pode ser utilizado para calcular probabilidades com uma dependência condicional [17].

$$p(x) = \int p(x|y)p(y)dy \tag{11}$$

B.4 Independência Condicional

Este conceito probabilístico infere o contrário do descrito na seção anterior. Considere a probabilidade de um evento aleatório ocorrendo por si só $p(x)$. Agora considere que a probabilidade do evento descrito não seja influenciada devido à presença de outro evento aleatório $p(y)$ ter ou não ocorrido.

A Eq. 12 descreve esse fenômeno, indicando que a probabilidade condicional de x dado y é depende apenas da probabilidade de ter x ocorrido.

$$p(x|y) = p(x) \tag{12}$$

A independência probabilística simplifica problemas estatisticos ao abstrair a correlação dos dados. Para contextos de SLAM, essa condição facilita a representação dos modelos [17].

C Formulação e estrutura do problema de SLAM

No SLAM, tanto a trajetória do robô como a localização de todos os pontos de referência são estimadas em tempo real, sem a necessidade de qualquer conhecimento *a priori* da localização [3].

Considera-se um robô móvel movendo-se através de um ambiente fazendo observações relativas de uma série de pontos de referência desconhecidos, usando um sensor localizado no robô como demonstrado na Fig. 5. Em um instante de tempo k, as seguintes variáveis são definidas:

- x_k: O vetor de estado que descreve a localização e orientação do veículo.
- u_k: O vetor de controle, aplicado no tempo $k-1$ para acionar o veículo para um estado x_k no instante k.
- m_i: Um vetor que descreve a localização do i^{th} ponto de referência cuja verdadeira localização é assumida invariante no tempo.
- z_{ik}: Uma observação tirada do local veículo do i^{th} ponto de referência no tempo k. Quando há várias observações marcantes em qualquer momento ou quando o específico marco não é relevante para a discussão, a observação será simplesmente escrita como z_k.

Adicionalmente os seguintes conjuntos são definidos:

- $X_{0:k} = \{x_0, x_1, ...x_k\} = \{X_{0:k-1}, x_k\}$: O histórico de localizações do veículo.
- $U_{0:k} = \{u_0, u_1, ...u_k\} = \{U_{0:k-1}, u_k\}$: O histórico de controle utilizado
- $m = \{m_1, m_2, ...m_k\}$: Conjunto com todos os pontos de referência
- $Z_{0:k} = \{z_0, z_1, ...z_k\} = \{Z_{0:k-1}, z_k\}$: O histórico de todas as observações de pontos de referência

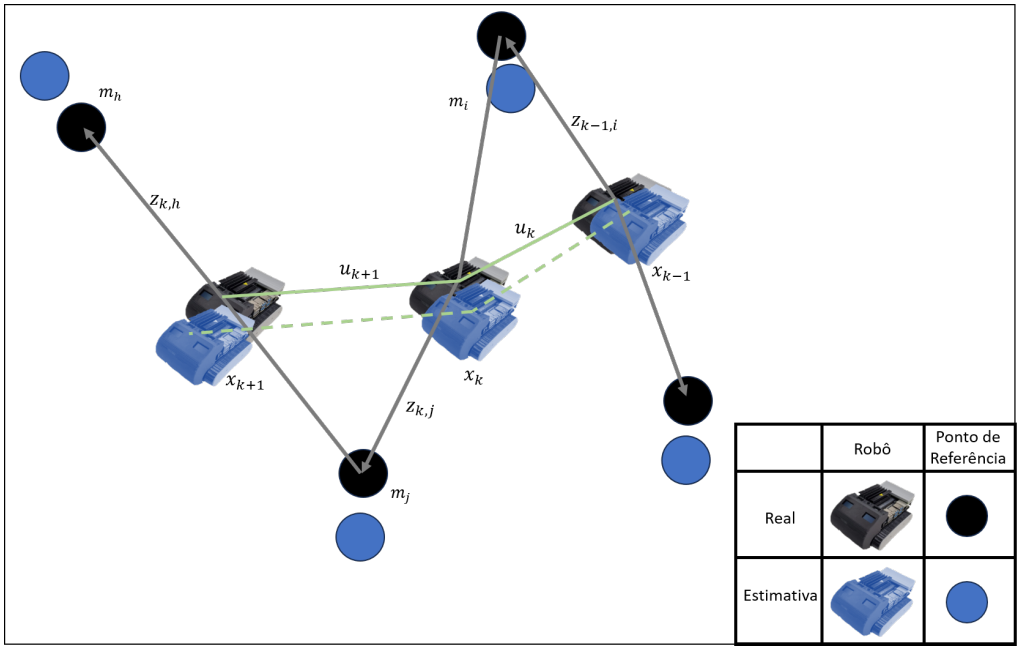

Figura 5: Estrutura do problema de SLAM representada com o Nanosaur

O Robô não tem conhecimento de sua posição e precisa utilizar das leituras sensoriais e do histórico de controle para estimar sua posição, sendo necessário calcular a Eq. 13 para todos os instantes k. Essa equação descreve a distribuição probabilística da posição do robô x, dado o histórico de medições sensoriais e controles utilizados até o instante k. [3]

$$P(x_k, m|Z_{0:k}, U_{0:k}, x_0) \tag{13}$$

Diversas metodologias podem ser empregadas para representar os pontos de referência z_k. Uma abordagem comum envolve o uso de *landmarks*, que são marcos com características específicas no ambiente as quais o robô foi instruido a detectar como pontos de referência. Esta técnica proporciona uma maior consistência na aquisição dos pontos de referência. No entanto, exige a identificação de objetos com características ou formatos peculiares, o que pode ser um desafio ou até mesmo impossível em ambientes totalmente desconhecidos. Outra estratégia é a utilização de *point-clouds*, que são conjuntos de pontos mapeados em um sistema de coordenadas comum, frequentemente empregados em SLAM 3D. Neste projeto, optou-se pela abordagem de *occupancy grids*, que será discutido em detalhe na próxima seção [3] [4] [8].

Para a Eq. 13, é imperativo assumir que o ambiente é estático e que todas as medições são independentes. Assumir essas características permite estruturar o problema de SLAM como uma *Dynamic Bayesian Network* (DBN) demonstrado na Fig. 6, onde cada nó representa uma das variáveis aleatórias do problema e as conexões entre os nós indicam que há uma dependência condicional entre elas [8].

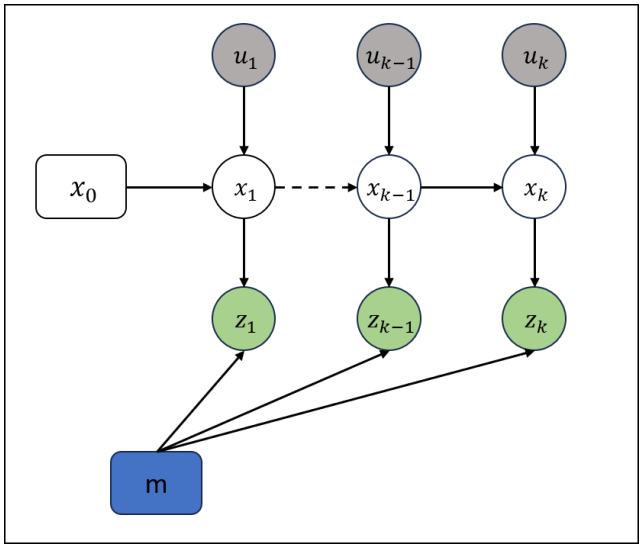

Figura 6: Estrutura da *Dynamic Bayesian Network*

C.1 *Pose-Graph* SLAM

A metodologia de *Pose-graph* consiste fundamentalmente de duas etapas: *front-end* e *back-end*. O *front-end* utiliza das informações sensoriais para construir um grafo, onde cada nó do grafo representa uma estimativa de *pose* do robô, uma rotina de curto período. Outra função do *front-end* é a correlação das *poses* estimadas. Ao associar *poses* anteriores com as novas, o algoritmo efetua *loop closure* reduzindo a incerteza das estimativas, uma rotina de longos períodos. O *back-end* pretende otimizar o grafo gerado selecionando *poses* e medições de menor incerteza na construção de uma trajetória ou mapa. A estrutura geral deste sistema é ilustrada na Fig. 7 [8].

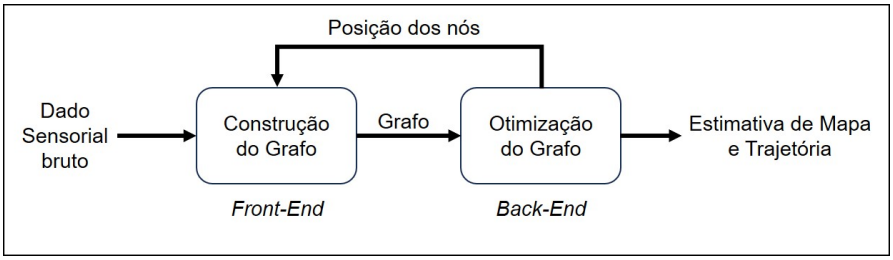

Figura 7: Diagrama de *Pose-Graph* SLAM

Uma das grandes vantagens de *Pose-Graph* SLAM é que o uso de grafos permite descrever o problema de estimativa abstraindo as medições brutas do sensor. Estas medições brutas são substituídas por "medições virtuais" nas arestas do grafo. Esta "medição virtual" da aresta entre dois nós é uma distribuição de probabilidade sobre as localizações relativas das duas posições, condicionada às suas medições mútuas.

A Fig. 8 demonstra visualmente o método descrito [8], onde uma aresta liga o vértice x_i e o vértice x_j. Esta aresta tem origem na medição z_{ij} . A partir da posição relativa dos dois nós, é possível calcular a medida esperada $\hat{z}_{ij}$ que representa x_j visto no quadro de x_i. O erro $e_{ij}(x_i, x_j)$ depende do deslocamento entre a medida esperada e a medida real. Uma aresta é totalmente caracterizada pela sua função de erro $e_{ij}(x_i, x_j)$ e pela matriz de informação Ω_{ij} da medida que representa a sua incerteza [8].

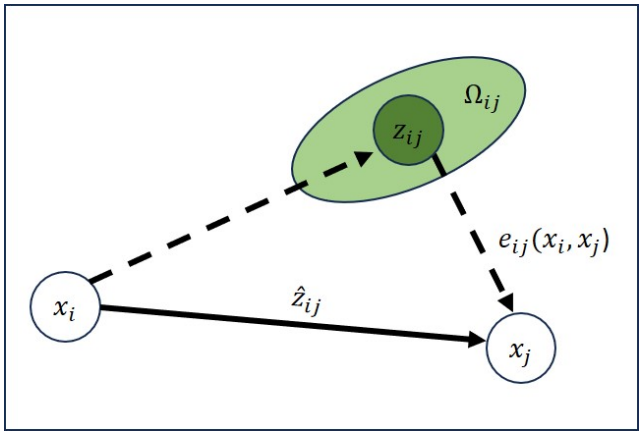

Figura 8: Representação de uma aresta em *Pose-Graph* SLAM

A estimativa MAP é uma abordagem estatística utilizada para calcular parâmetros desconhecidos ou um conjunto deles em contextos probabilísticos. Este método combina uma distribuição de probabilidade *a priori* com a função de verosimilhança, resultando na distribuição de probabilidade *a posteriori* dos parâmetros. O intuito da estimativa MAP é determinar o estado que otimiza essa distribuição *a posteriori*, levando em consideração as medições realizadas [8].

Para contextos de *Pose-Graph* SLAM, a estimativa MAP é crucial, permitindo o calculo da trajetória do robô e o mapeamento do ambiente. As medições realizadas pelo robô são essenciais para desenvolver um modelo probabilístico. Além disso, informações anteriores relacionadas ao movimento do robô e às características do ambiente são incorporadas para estimar a distribuição a posteriori da posição do robô e do mapa [8].

D LiDAR 2D em SLAM

O *Light Detection And Ranging* (LiDAR) 2D é um sensor óptico de alcance que mede a seção transversal da estrutura geométrica do ambiente. Quando acionado, o *laser* acoplado a um motor rotativo, gira em um movimento circular. A Fig. 9 retrata o princípio de funcionamento do LiDAR 2D. O bloco verde representa um robô móvel com esteiras equipado com um sensor LiDAR 2D. O retângulo cinza denotam os obstáculos no ambiente. As setas azuis indicam os feixes *laser* emitidos pelo LiDAR e os pontos azuis são as medições obtidas pela seção transversal entre os feixes [14] [20].

Ao obter as medições, podemos introduzi-las no algoritmo de SLAM e fundi-los com outros sensores para construir um mapa do ambiente. O sistema LiDAR SLAM inclui três módulos: o módulo de *front-end* que realiza *scan matching*, o módulo de deteção de *loop closure* e o módulo de *back-end* de otimização.

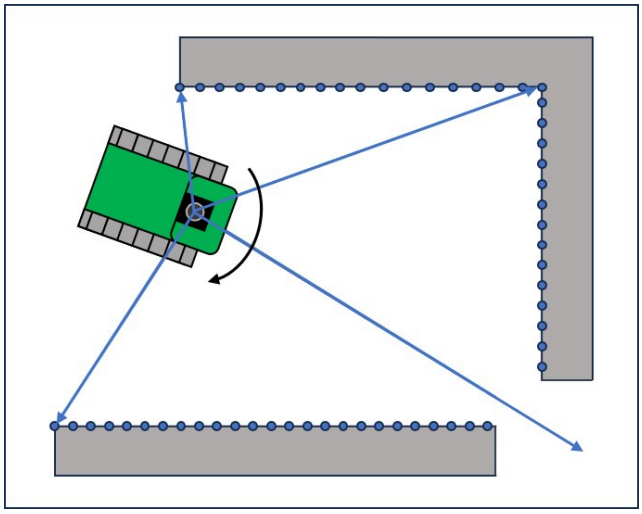

Figura 9: Princípio do funcionamento de um LiDAR

D.1 Medição das distâncias

O LiDAR funciona com base em dois tipos de medição: a *pose* do sensor e a distância dos objetos. Existem diferentes métodos para medir a distância, sendo um deles o uso de um *laser* pulsado. Neste método, o *laser* é emitido por um transmissor, alcança o ambiente e é refletido de volta para o receptor do sensor. A distância é calculado no tempo que a luz leva para atingir a superfície e retornar ao receptor. Este cálculo é demonstrado na Eq. 14. [1]

$$D = c\frac{\Delta T}{2} \tag{14}$$

em que D é a distância ao objeto, c é a velocidade da luz e ΔT é o tempo de resposta. Este sistema é limitado pela necessidade de um sinal de retorno, exigindo o uso de *lasers* potentes para medir longas distâncias.

Outra técnica para calcular distâncias se baseia na medição da fase, utilizando um *laser* de onda contínua com modulação de amplitude (AMCW). A diferença de fase entre a luz emitida e a refletida é usada para determinar a distância, conforme descrito na Eq. 15: [1]

$$D = \frac{c}{2}\frac{\Delta\phi}{(2\pi f_M)} \tag{15}$$

onde $\Delta\phi$ é o desvio de fase e f_M é a frequência de modulação da amplitude do sinal. A principal desvantagem desta abordagem é que o alcance que pode ser medido sem ambiguidade é curto, tipicamente cerca de $100m$.

D.2 *Scan Matching*

Para aplicações de SLAM, apenas a obtenção das distâncias não é o suficiente. Um robô ao se mover altera sua *pose* e a do sensor a todo instante. Logo, é imperativo a correlação das distâncias obtidas de diferentes *poses* para corrigir as estimativas de odometria e mapeamento.

Um método comum para reduzir a incerteza do SLAM com LiDAR é o *Scan Matching*. Através de correlações das varreduras e *poses* de diferentes instantes de tempo, é possível corrigir a estimativa de localização e mapeamento reduzindo as incertezas de curto período. *Scan Matching* é crucial para SLAM, sendo também responsável por efetuar o *Loop closure*, que reduz as incertezas das estimativas de longo período [14].

Considere que um robô móvel retorna a um lugar conhecido. O algoritmo detecta um *Loop closure* conectando o nó atual com o antigo, indicando uma aresta entre esses nós. A observação de ambas as poses é então comparada para calcular a transformação virtual que deve mapear uma medida de forma ótima para a outra. Seja $z_{i,j}$ a informação de odometria entre as poses i e j, e $\hat{z_{i,j}}$ o valor esperado do erro, $e_{i,j}$ pode ser calculado simplesmente subtraindo um do outro. Esse cálculo é demonstrado na Eq. 16. [8]

$$e_{i,j}(x_i, x_j) = z_{i,j} - \hat{z_{i,j}} \tag{16}$$

Sendo C o conjunto de pares de índices para os quais existe uma restrição observada z, o objetivo de uma abordagem de máxima verossimilhança é encontrar a configuração dos nós x^* que minimiza a verossimilhança negativa $F(x)$ de todas as observações [8]

$$F(x) = \sum_{\langle i,j\rangle \in C} {e_i}^T \Omega e_{i,j} \tag{17}$$

Assim, procura-se resolver a Eq. 18 [8]

$$x^* = argmin_x F(x) \tag{18}$$

E *Occupancy Grid Maps*

Occupancy Grids são uma representação cartográfica usada na robótica para mapear o ambiente ao redor de um sistema autônomo. Essa metodologia divide o espaço a ser mapeado em um conjunto de células dispostas em formato matricial 2D, onde cada célula representa uma pequena área do ambiente. Para cada célula é atribuído um valor atrelado à probabilidade de que estejam ocupadas, ou seja, a probabilidade que exista um obstáculo naquela área do ambiente. Em sua forma mais simples esses valores são binários. Por exemplo, 0 representaria uma célula livre e 1 uma célula ocupada. A representação de um *Occupancy Grid Map* pode ser vista na Fig. 10, onde o ambiente é o mesmo da Fig. 9. [6] [8]

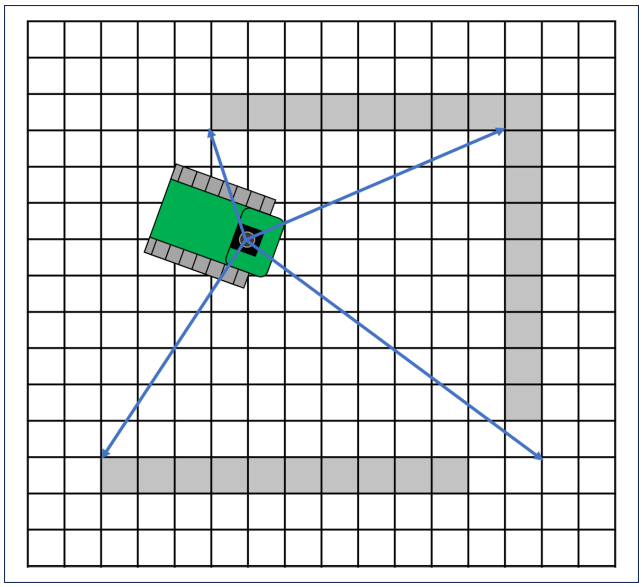

Figura 10: *Occupancy Grid Map* de um LiDAR

Essa representação é útil para algoritmos de localização e mapeamento simultâneo. Para o mapeamento, os sensores do robô como câmeras, *lasers* ou sonares, coletam informações sobre o ambiente. Com base nessas informações, o algoritmo de SLAM atualiza as células da grade de ocupação para refletir a probabilidade de cada célula estar ocupada ou livre. Isso permite criar um mapa detalhado do ambiente. Simultaneamente, o robô compara as informações das células de *Occupancy Grid* com as leituras dos sensores para estimar sua própria posição no ambiente. À medida que o robô se move e obtém mais dados sensoriais, a grade de ocupação é continuamente atualizada para refletir as mudanças no ambiente e na posição do robô. As *Occupancy Grid* são uma representação eficaz que auxilia algoritmos de SLAM com LiDAR a reduzir a incerteza inerente à percepção do ambiente e à localização do robô.

F Odometria Visual

Um problema comum na robótica móvel é a estimação da distância percorrida pelo robô. Normalmente, robôs móveis usam *encoders*, sensores que contam o número de revoluções das rodas e consequentemente permitem o cálculo da distância percorrida em função do número de revoluções lido. Essa abordagem é extremamente eficaz, entretanto apresenta erros acumulados devido a deslizamento e terrenos escorregadios onde a roda gira em falso [6].

Odometria visual é uma alternativa que processa a estimativa da trajetória usando apenas a entrada de uma única ou múltiplas câmaras ligadas ao agente. Câmeras são ricas em informações, devido à sua capacidade de capturar uma ampla gama de dados visuais sobre o ambiente. Para a aplicação da odometria visual é necessário estimar o movimento translacional e rotativo do robô através de uma sequência de imagens. Para utilizar estas informações de forma efetiva, é necessário transformar o espaço de coordenadas da imagem para o espaço de coordenadas da configuração do robô. Três técnicas de estimativa de movimento são comumente utilizadas na odometria visual: 3D para 3D, 3D para 2D, e 2D para 2D [12].

F.1 Transformada 3D para 3D

Na abordagem de estimativa de movimento 3D para 3D, o movimento é calculado através da triangulação de pontos 3D, observados em uma sequência de imagens. Esta transformação do espaço de imagem da câmera é estimada minimizando a distância euclidiana 3D entre pontos correspondentes, conforme demonstrado na seguinte Eq. 19. [12]

$$T = \underset{T}{argmin} \sum_{i} |X_i - X'_i|^2 \tag{19}$$

Na qual, T representa a transformação estimada entre duas imagens consecutivas. X é o ponto de característica 3D observado no instante F_k, equanto X' é o ponto 3D correspondente da característica no instante anterior F_{k-1}. O índice i denota o número mínimo de pares de características necessários para definir a transformação. O número mínimo de pontos requeridos varia conforme a quantidade de graus de liberdade do sistema e o tipo de modelo utilizado. Incrementar o número de pontos além do mínimo necessário pode melhorar a precisão da transformação, embora isso também aumente o custo computacional [12].

F.2 Transformada 3D para 2D

A estimativa de movimento 3D para 2D assemelha-se à abordagem 3D para 3D, mas diferencia-se ao minimizar o erro de re-projeção 2D para determinar a transformação necessária. A função de custo para este método é expressa pela seguinte Eq. 20. [12]

$$T = \underset{T}{argmin} \sum_{i} |z - f(T, X'_i)|^2 \tag{20}$$

Onde, z representa o ponto característico observado na imagem atual F_k, e $f(T, X'_i))$ é a função de re-projeção do ponto 3D correspondente na imagem anterior F_{k-1} após aplicar a transformação T.

F.3 Transformada 2D para 2D e Visão Estéreo

A estimativa de movimento 2D para 2D é útil quando não se dispõe de dados 3D. Um exemplo seria a estimativa da transformação relativa entre duas imagens monoculares, na qual os pontos ainda não foram triangulados. Nesse caso recorre-se à geometria da visão estéreo o mesmo tipo de câmera utilizado neste projeto. Seu modelo pode ser observado na Fig. 11. [12]

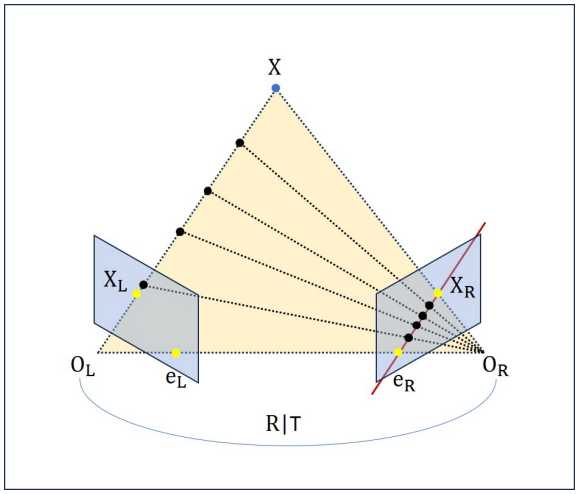

Figura 11: Modelo da câmera de visão estéreo

A figura representa duas câmeras pelos seus centros O_L e O_R e pelos planos de imagem em azul. O ponto 3D observado por ambas câmeras é marcado por X, enquanto as re-projeções das câmeras são indicadas por X_L e X_R. Os Epipolos das câmeras são representado por e_L e e_R. Os centros das câmeras, o ponto 3D e suas re-projeções nas imagens estão alinhados em um plano comum em amarelo. Um ponto de

imagem é retroprojetado no espaço 3D como uma semi-reta. Essa semi-reta é projetada como uma linha na segunda imagem, denominada linha epipolar, colorida em vermelho. O ponto 3D encontra-se sobre esta semi-reta, indicando que a imagem do ponto 3D na segunda vista deve estar localizada sobre a linha epipolar. A *Pose* entre O_L e O_R pode ser determinada utilizando a matriz essencial E, que representa algebricamente a geometria epipolar para uma calibração conhecida [12].

Cada câmera captura uma imagem 2D de um ambiente 3D, um processo conhecido como projeção em perspectiva. A restrição da visão estéreo aplicada nesta abordagem é expressa na Eq. 21. [12]

$$q'^{T} E q = 0 \tag{21}$$

Nesta equação, q e q' são pontos de imagem homogêneos correspondentes em duas imagens consecutivos, e E é a matriz essencial definida pela Eq. 22.

$$E = [t]_x R \tag{22}$$

na qual, R é a matriz de rotação, t é a matriz de translação dada por Eq. 23:

$$t = \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix} \tag{23}$$

e $[t]_x$ é a matriz simétrica oblíqua demonstrada na Eq. 24

$$[t]_x = \begin{bmatrix} 0 & -t_z & t_y \\ t_z & 0 & t_y \\ -t_y & t_x & 0 \end{bmatrix} \tag{24}$$

Na visão estéreo, a informação 3D é reconstruída por meio de triangulação em um único instante de tempo, observando características simultaneamente nas imagens esquerda e direita, que estão espacialmente separadas por uma distância de base conhecida.

F.4 Odometria Visual Inercial

Câmeras de visão estéreo oferecem uma odometria robusta mesmo durante longos períodos de uso. Porém o custo computacional do processamento de imagens é alto, impedindo a câmera de ter uma alta taxa de amostragem. A Câmera utilizada neste projeto, Intel Realsense T265, oferece uma solução inovadora que envolve combinar os dados de odometria da imagem e corrigir estes com os de uma IMU. Uma IMU pode possuir grandes ruídos, porém possui uma alta taxa de amostragem e alta precisão em curtos períodos. Já a câmera tem uma baixa taxa de amostragem mas possui pouco erro em longos períodos, portanto ambos sensores se complementam [6] [12].

A Fig. 12 demonstra os diferentes espaços de coordenadas utilizados na composição da odometria da câmera. Realizando odometria visual estéreo com as lentes *fisheye* e fazendo fusão de sensores com a IMU, a câmera é capaz de obter uma estimativa de *pose* corrigida e robusta para curtos e longos períodos de operação. Além disso, a fusão destes sensores permite reduzir a taxa de amostragem da câmera, o que consequentemente reduz o custo computacional da odometria visual e o consumo de potência (1.5W) da câmera.

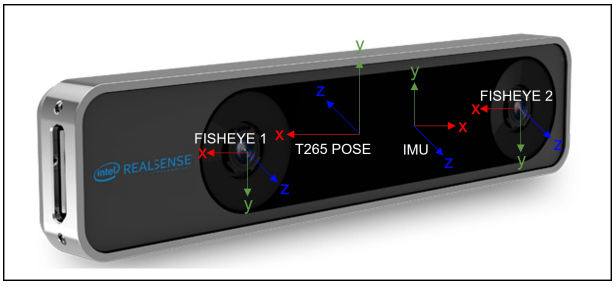

Figura 12: Transformadas feitas pela Câmera Intel Realsense T265

5 Metodologia

Neste capítulo será apresentada a metodologia abordada no projeto, explicando e justificado as técnicas e ferramentas que o compõe. Nesta seção será apresentado o *framework* ROS, a escolha e especificações dos sensores utilizados, o programa e metodologia de SLAM, e a estrutura de comunicação do sistema.

A ROS

ROS, da sigla em inglês *Robot Operating System* se trata de um sistema meta operacional completamente *open-source* que auxilia um sistema operacional existente com a distribuição de tarefas. Aplicações de robótica geralmente implicam a integração de diversos equipamentos para a obtenção de um resultado específico, por exemplo, atuadores, sensores, processadores com *softwares* especializados, dentre outros [21].

ROS possui um poderoso conjunto de bibliotecas, ferramentas e uma comunidade ativa, sendo amplamente utilizado em aplicações de robótica e é uma parte fundamental da integração do sistema. O *framework* atua como uma ponte entre *hardware* e *software*, utilizando estruturas bem definidas como por exemplo, os pacotes. Pacotes ROS são diretórios compostos de bibliotecas, arquivos de configuração, serviços, *nodes* e mensagens. *Nodes* contém a rotina operacional do pacote e interpretam o fluxo de entrada e saída das mensagens, as quais são chamadas de tópicos. Tópicos são mensagens do tipo classe que facilitam a comunicação entre diferentes pacotes por possuírem um formato padronizado [21].

ROS também possui arquivos do tipo de lançamento ou em inglês *launch*, que permitem a configuração e o acionamento de múltiplos pacotes com a chamada de um único arquivo. Este tipo de arquivo é especialmente útil, pois o sistema precisa mesclar múltiplos pacotes para alcançar seu objetivo, e convocar cada um deles individualmente entre testes seria ineficiente [21].

O diagrama da Fig. 13 descreve a metodologia utilizada. Cada bloco pode ser interpretado por um pacote ROS. Os dados obtidos pelo LiDAR são tratados por um pacote chamado e publicados como um tópico de *Laser Scan*. Os dados obtidos pela câmera T265 são tratados por outro pacote e publicados como um tópico de *Pose*. A SLAM Toolbox, que também é um pacote ROS, monitora os tópicos emitidos de *Laser Scan* e *Pose*, publicando um tópico de mapa como *Occupancy grid map*. Para versões mais recentes da SLAM Toolbox também é publicado um tópico de *Pose* estimada. Entretanto, devido a limitações do projeto, foi necessário utilizar de uma versão antiga do pacote que ainda não possuía essa função. Para solucionar este problema foi desenvolvido um pacote *Listener* que monitora as transformadas feitas pela SLAM Toolbox e as publica em um tópico de *Pose*.

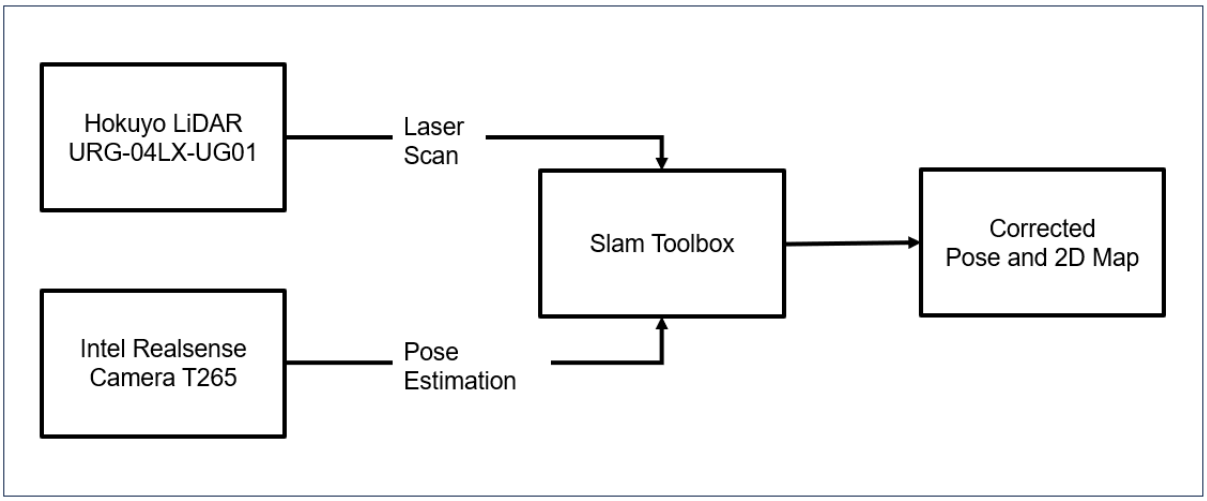

Figura 13: Diagrama de blocos da metodologia do sistema

B SLAM Toolbox

A SLAM toolbox foi criada para fornecer uma solução versátil, precisa e eficiente para mapeamento e localização em tempo real. Desenvolvida por Steve Macenski, este pacote inclui as funcionalidades SLAM

2D padrão esperadas na robótica móvel, como iniciar, mapear e guardar mapas no formato PGM, juntamente com utilitários incorporados úteis para guardar mapas. Além disso, permite o refinamento, o remapeamento ou a continuação do mapeamento de um *pose-graph* guardado (serializado) em qualquer ponto [13].

O projeto também inclui um modo de localização baseado na otimização, construído sobre o *pose-graph*, que pode ser executado opcionalmente num modo de "odometria LiDAR" sem um mapa prévio, facilitando os *loop closures* locais. Suporta modos de mapeamento síncronos e assíncronos, aumentando a sua versatilidade. A SLAM toolbox foi concebida com *solvers* de otimização baseados em *plugins*, apresentando um novo *plugin* otimizado baseado no *Google Ceres* [22].

A SLAM toolbox fornece dois tipos de *nodes* ROS: *node* síncrono e *node* assíncrono. No *node* síncrono, o sistema calcula todas as *poses* de varrimento que passaram pelo filtro de movimento, apesar de terem chegado novas medições enquanto o sistema não terminou de estimar as *poses* de exploração. Isto significa que o sistema está atrasado e não calcula a *pose* da última medição até terminar de calcular a *pose* das medições anteriores [7].

Por outro lado, o *node* assíncrono nunca se atrasará. Tentando sempre calcular a *pose* da última medição, mesmo que ainda tenha tarefas de cálculo de poses anteriores. Quando o sistema recebe um novo scan enquanto ainda tem tarefas de cálculo de poses de anteriores, ignorará algumas das poses anteriores para poder recuperar o atraso para calcular a última varredura. Além disso, na fase de filtragem do movimento, este nó não verifica os registos de data e hora dos exames. Verifica apenas os movimentos de translação e rotação translacionais e rotacionais [7].

Ambos os *node* se inscrevem em tópicos de *scan laser* e odometria, e são responsáveis por publicar o mapa junto com a transformação de odometria para o mapa. Quando recebe uma mensagem no tópico *laser*, uma chamada de retorno é ativada, resultando na geração de uma *pose* (derivada da odometria) e um *laser scan* ligado a esse *node*. Estes *PosedScans* acumulam-se em uma fila, aguardando processamento pelo algoritmo [22].

Essa fila de *PosedScans* é utilizada na construção de um *pose-graph*, onde a odometria é aprimorada através do *laser scan matching*. O *pose-graph* é aplicado para calcular a *pose* do robô e detectar *loop closures*. Se um *loop closure* for identificado, o grafo de *pose* passa por otimização e as estimativas de *pose* são atualizadas. Essas estimativas são cruciais para calcular e divulgar a transformação do mapa para a odometria do robô. Os *Laser scans* associados a cada *pose* no grafo são empregados na criação e publicação de um mapa. A Fig. 14 demonstra o funcionamento do pacote através de um diagrama de blocos [22].

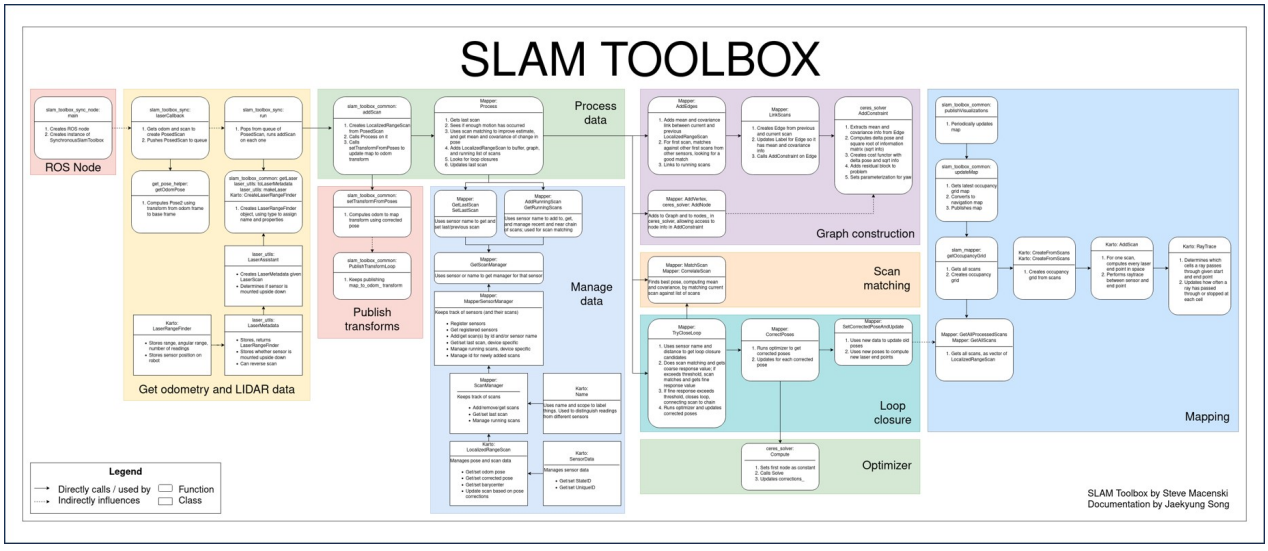

Figura 14: Diagrama de blocos da SLAM toolbox [22]

C Método de SLAM

A SLAM toolbox aborda o problema de SLAM 2D utilizando da representação cartográfica de *occupancy grid maps*, conforme descrito na sub-seção E da seção 4. O pacote utiliza da estrutura de *Pose-Graph*

SLAM e define o restringimento entre nós com a estimativa MAP. Este método é detalhado na sub-seção C da mesma seção [22].

Com uma estratégia semelhante à do Google Cartographer, a SLAM Toolbox divide o problema de SLAM em duas etapas. Na Etapa de *front-end*, a SLAM toolbox utiliza o processo de *scan-matching* baseado no Karto SLAM. O processo de *scan-matching* é explicado na sub-seção D da seção 4. Já para a etapa de *back-end*, o pacote adota o algoritmo de código aberto *CeresSolver*, o qual busca encontrar a configuração ótima da estrutura de grafos por meio da resolução do método de mínimos quadrados não-linear. O problema dos mínimos quadrados é um método matemático e estatístico usado para encontrar a função de ajuste para um conjunto de dados, minimizando a soma dos quadrados das diferenças entre os valores observados e os valores previstos [7] [22].

D Comunicação

Para acionar o robô remotamente foi adotado o protocolo de comunicação *Secure Shell* (SSH), o qual depende da existência de uma conexão de rede mútua entre os equipamentos para funcionar. No caso deste sistema os equipamentos são um computador operador e a Jetson Nano. Além disso, um receptor USB de um controle *joystick* é conectado ao robô para conduzir seu movimento por teleoperação.

Através da comunicação SSH é possível iniciar o sistema remotamente com um computador operador. Observa-se pelo diagrama da Fig. 15 os comandos feitos pelo computador operador, que estão representados por setas azuis.

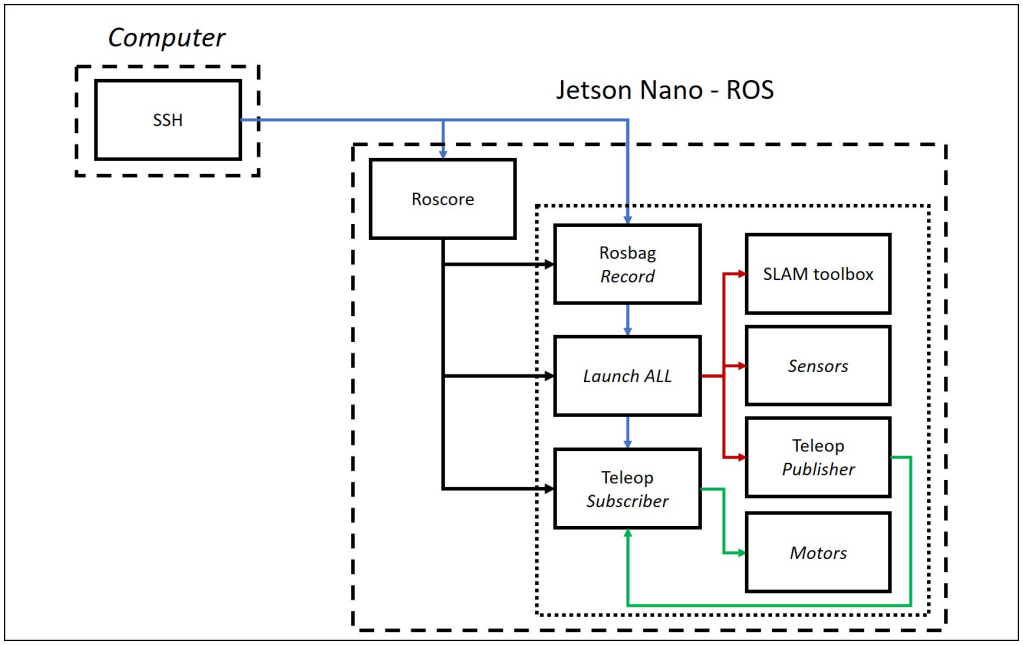

Figura 15: Diagrama de comunicação do sistema

Primeiramente inicia-se o *ROSCORE* que é um conjunto de bibliotecas fundamental para que qualquer sistema baseado em ROS funcione. No diagrama essa conexão é representada por setas pretas. A próxima etapa é iniciar o arquivo desenvolvido "Launch ALL" que convoca simultâneamente todos os *nodes* necessários para que o robô realize SLAM e o *node* que publica um tópico de "$/cmd\_vel$" em resposta ao movimento do controle *joystick*. Este acionamento simultâneo é representado por setas vermelhas. Para interpretar o tópico de "$/cmd\_vel$" é convocado outro *node* que interpreta o tópico e atua os motores, cuja interação é representada por setas verdes. Por último é dado o comando para gravar todos os tópicos em uma rosbag em tempo real.

6 Sistema

O sistema se trata de um robô móvel compacto com base em um projeto de código aberto conhecido como Nanosaur, criado por Raffaelo Bonghi [23]. Com exceção de seu *hardware*, toda a estrutura mecânica do robô pode ser impressa em 3D, facilitando modificar o projeto de acordo com as necessidades. Esta seção pretende explicar a estrutura geral do sistema, que pode ser observada no diagrama da Fig. 16. Duas modificações foram realizadas na estrutura mecânica do projeto original. A primeira modificação envolveu o desenvolvimento de um suporte adequado aos sensores utilizados neste projeto. A segunda alteração foi a implementação de um par de motores de maior torque, com o objetivo de compensar o peso adicional do suporte e dos sensores, além de estabilizar a leitura das medições diante da vibração causada pelo movimento. Essas mudanças são detalhadas nas subseções seguintes.

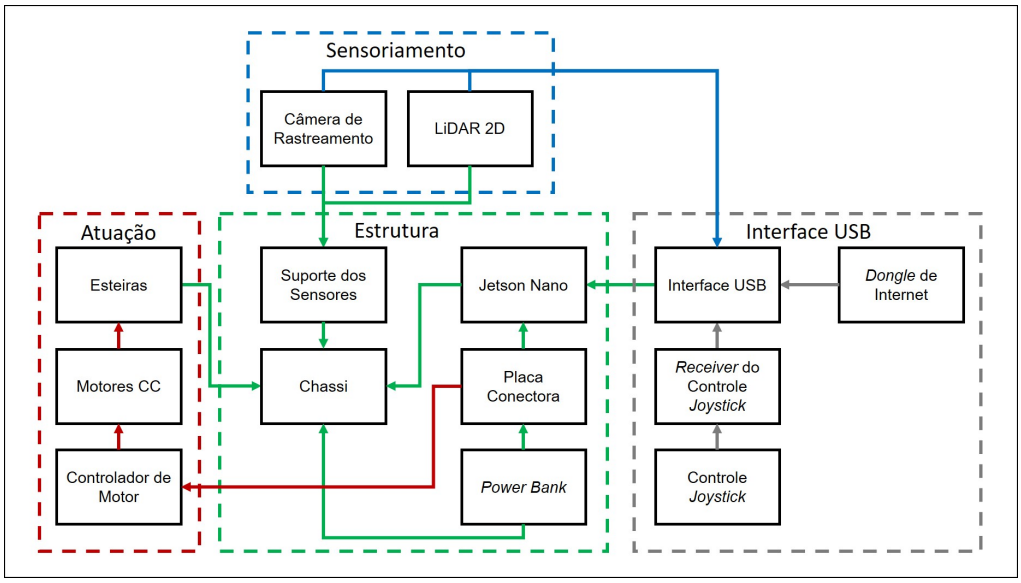

Figura 16: Diagrama geral do Sistema

A Placa Conectora e Alimentação

O sistema é alimentado por um *power bank* Anker de 10000mAH. Este dispositivo é suficientemente compacto para ser acoplado em um engaste debaixo do chassi do robô. Essa fixação é demonstrada na Fig. 17. A alimentação do sistema é distribuída por uma PCB customizada desenvolvida por Raffaelo [23]. A placa recebe a potência do *power bank* por meio de um conector USB, que passa por uma chave interruptora para ligar e desligar o robô com facilidade. A conexão de potência é então distribuída para as portas de alimentação da Jetson Nano e do controlador de motor, através de barramentos e fios *jumpers*, respectivamente. A placa conectora também integra as conexões I2C da Jetson Nano com as portas I2C do controlador de motor conectando-as com *jumpers*. A placa conectora com suas conexões e componentes instalados pode ser observada na Fig. 18.

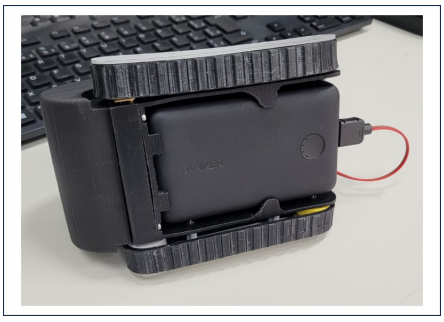

Figura 17: *Power Bank* de 10000mAH fixado em engaste debaixo do chassi

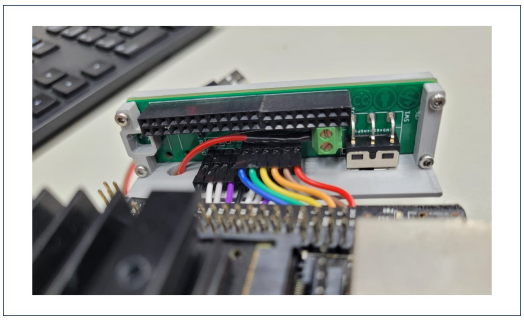

Figura 18: Placa conectora fixada em *Flap*

B Atuação

O robô utiliza dois *Micro Metal DC Gearmotors* de $6V$, que estão fixados em lados opostos na parte inferior frontal do chassi. No projeto original [23], esses motores possuem $150RPM$. Contudo, devido ao aumento significativo de peso causado pelo par de sensores adotados neste projeto, o centro de massa do sistema foi alterado, consequentemente causando acúmulos de inércia excessivos durante o movimento, dificultando manobras abruptas e causando instabilidade nas leituras devido a ruídos de vibração. A solução encontrada foi modificar o projeto para utilizar de um motor de maior torque, permitindo um movimento mais conciso, sem acúmulos expressivos de inércia. Além disso, a modificação reduziu os ruídos de vibração durante o movimento, aperfeiçoando as leituras dos sensores. Assim, o sistema foi modificado para usar um par de *Micro Metal DC Gearmotors* de $6V$ e $50RPM$, reduzindo a velocidade do robô em troca de estabilidade.

Para controlar a velocidade destes motores utiliza-se de uma placa "Adafruit DC Motor + Stepper FeatherWing", que pode ser vista na Fig. 19. A placa possui quatro chips de ponte H completa "TB6612", permitindo o controle de velocidade de até quatro motores de corrente contínua distintos. Esta placa é fixada na parte superior frontal do chassi, sendo conectada aos motores e até a placa conectora, que por sua vez conecta as portas I2C do controlador de motor às portas de comunicação I2C da Jetson Nano. A conexão descrita é apresentada na Fig. 20.

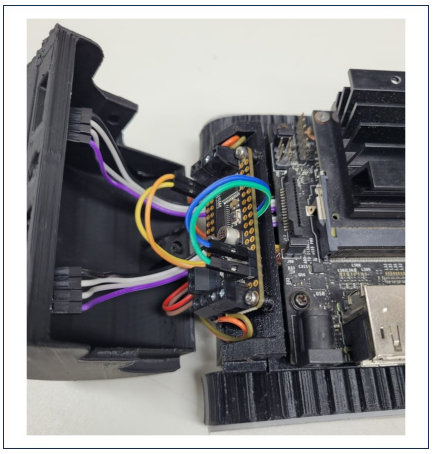

Figura 19: Controlador de Motor "Adafruit DC Motor + Stepper FeatherWing"

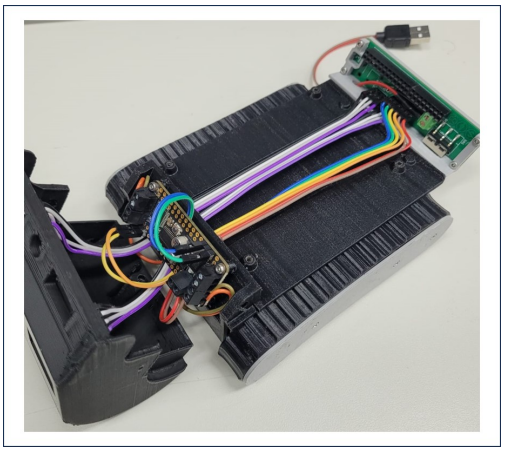

Figura 20: Controlador de Motor conectado à Placa conectora

Os motores são acoplados com esteiras impressas 3D para distribuir seu torque de forma uniforme por toda a extensão do robô, de forma semelhante ao uso de rodas passivas em robôs de tração diferencial [15]. Porém, o uso de esteiras aumenta complexidade envolvida da modelagem e odometria. De acordo com a tese de doutorado de Alexandro [16], pequenos robôs tracionados por esteiras com configuração semelhante de um robô de tração diferencial podem ser aproximados a um modelo de tração diferencial para superfícies regulares e de atrito constante. Considerando que o sistema é um robô compacto e que foi projetado para ambientes internos regulares, onde há pouco deslizamento e derrapagem até mesmo em movimentos curvilíneos, ele atende as características mínimas e pode ser aproximado a um modelo de tração diferencial.

C Sensoriamento

Cada sensor fornece um dado de entrada único ao sistema. O LiDAR 2D contribui com medições de distância precisas do ambiente, e a T265 realiza odometria visual inercial obtendo uma estimativa da *pose* do robô.

C.1 LiDAR 2D

O sensor utilizado para a obtenção das distâncias do ambiente foi o Hokuyo LiDAR URG-04LX-UG01 apresentado na Fig. 21.

Figura 21: Sensor Hokuyo LiDAR URG-04LX-UG01

Apesar deste sensor ser relativamente grande para um robô compacto, os *scan laser* do LiDAR 2D fornecem medidas de distâncias de alta precisão e alcance, evitando a necessidade do processamento de imagens de profundidade para obtenção das distâncias, o que permite a construção de um mapa com custos computacionais significativamente reduzidos. A Tab. 1 apresenta as especificações do LiDAR escolhido.

Tabela 1: Especificações Hokuyo LiDAR URG-04LX-UG01

Fonte de Alimentação	5VDC±5% (alimentação do barramento USB)
Fonte de Corrente	500mA durante inicialização
Potência	Aprox 2.5W
Peso	Aprox. 160g
Dimensões	Aprox. 50mm x 50mm x 70mm
Temperatura/Humidade Ambiente	-10 a +50 graus C, 85% ou menos (sem condensação, sem formação de gelo)
Fonte de luz	Fonte de luz Díodo laser semicondutor (λ=785nm), Classe de segurança *laser* 1
Área de medição	20 a 5600mm (papel branco com 70mm×70mm), 240°
Acurácia	Precisão 60 a 1.000mm : ±30mm, 1.000 a 4.095mm : ±3% da medição
Resolução Angular	Ângulo de passo: aprox. 0,36° (360°/1.024 passos)
Tempo de varredura	100ms/scan
Ruído	25dB ou inferior
Interface	USB2.0/1.1[Mini B](Velocidade máxima)
Sistema de comando	SCIP Ver.2.0
Iluminação ambiente	Lâmpada de halogéneo/mercúrio: 10.000Lux ou menos, Florescente: 6000Lux(Max)
Resistência à Vibração	10 a 55Hz, amplitude dupla 1,5mm a cada 2 horas nas direções X, Y e Z
Resistência à Impacto	$196m/s^2$, cada 10 vezes nas direções X, Y e Z

C.2 Câmera de Rastreamento

O sensor utilizado para a obtenção das estimativas de odometria foi a câmera Intel Realsense T265 apresentado na Fig. 22.

Figura 22: Câmera Intel Realsense T265

A câmera é equipada com um processador interno, duas lentes *Fisheye* e uma IMU de alta precisão. Combinando as respectivas informações visuais das lentes para criar uma imagem tridimensional e corrigindo esse dado com a informação da IMU, a câmera realiza SLAM Visual em seu processador embarcado, resultando em uma estimativa de sua própria *pose*, que pode ser usada como um dado de entrada para o sistema. A Tab. 2 apresenta as especificações da câmera.

Tabela 2: Especificações Intel Realsense T265

Geral	**Fonte de Alimentação**	5VDC±5% (alimentação do barramento USB)
	Fonte de Corrente	300mA
	Potência	Aprox. 2.3W
	Peso	Aprox. 60g
	Dimensões	108mm x 24,50mm x 12,50mm
	Temperatura/Humidade Ambiente	0 a +40 graus C, 90% ou menos (sem condensação, sem formação de gelo)
IMU	**Graus de Liberdade**	6
	Alcance de Aceleração	±4g
	Taxa de Amostragem do Acelerômetro	62.5Hz
	Alcance do Giroscópio	±2000º/s
	Taxa de Amostragem do Giroscópio	200Hz
Fisheye	**Pixels Ativos**	848x800
	Relação de Aspecto	1.06
	Formato	8bit,10bit*RAW*
	Tipo de Filtro	Filtro de corte Infravermelho
	Foco	Fixado
	Tipo de Obturador	Obturador Global
	Interface do Sinal	MIPI CSI-2, 2 X Vias
Imagem	**Base**	64±0.15
	Visores *Fisheye* esquerdo/direito	Sensor de imagem monocromático
	Formato	8bit, 10bit *RAW*
	Campo de visão	173º Profundidade

C.3 Suporte dos Sensores

Para fixar os sensores foi desenvolvido um suporte impresso em 3D que é acoplado na parte superior da peça frontal. O LiDAR é fixado por interferência no topo do suporte. A câmera é parafusada na frente do suporte. Essa configuração reduz os ruídos de vibração durante o deslocamento e permite que ambos os sensores sejam capazes de tirar medições do ambiente sem obstruir a leitura de um ao outro. O modelo CAD desenvolvido pode ser visto na Fig. 23.

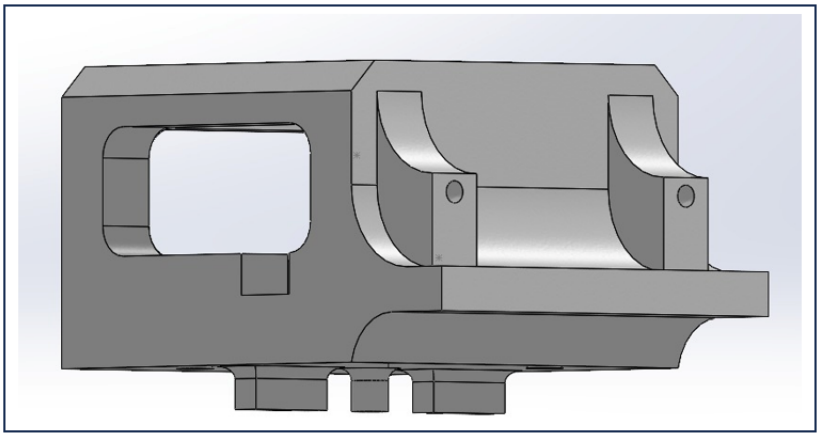

Figura 23: Modelo CAD desenvolvido do suporte dos sensores

D Jetson Nano e Integração do Sistema

A unidade de processamento e controle utilizada se encontra na Fig. 24, o *Kit* para desenvolvedor Jetson Nano da empresa NVIDIA.

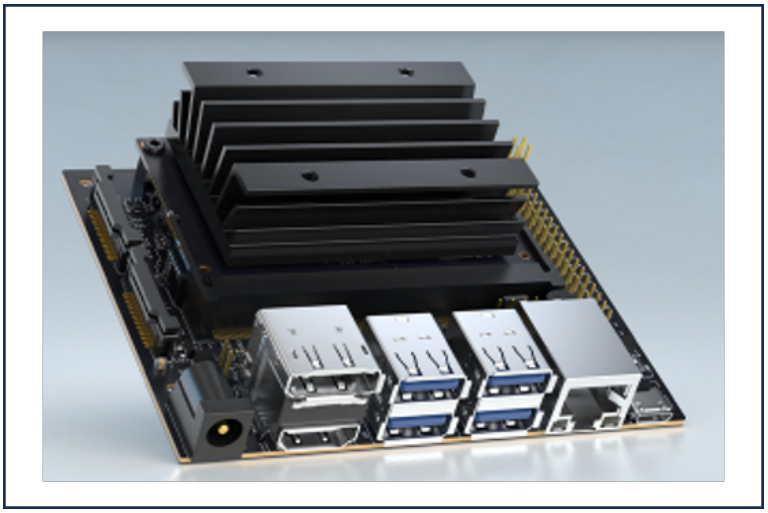

Figura 24: *Kit* para desenvolvedor Jetson Nano

Sendo efetivamente um computador compacto de 64 bits, este dispositivo é capaz de realizar processamento digital de imagens, incluindo classificação de imagens, detecção de objetos e segmentação. O *kit* pode ser operado em ambiente Linux, e possui uma comunidade ativa de desenvolvedores. A placa é otimizada para sistemas de baixa potência, consumindo de $5W$ à $10W$ dependendo de quantos processadores estiverem habilitados. A Tab. 3 apresenta as especificações da placa.

O *power bank* é única entrada de potência do sistema. Logo, a soma total das potências de consumo de todos os outros dispositivos não pode exceder a potência fornecida do *power bank*. Para solucionar este problema, foi necessário configurar a Jetson Nano para utilizar apenas dois de seus quatro processadores, mas também reduzindo o consumo para apenas $5W$. Essa modificação prejudicou a perfomance do Kit, mas foi necessária devido ao alto consumo de corrente do LiDAR 2D, especialmente ao ser ligado.

Para programação do equipamento, foi instalado na placa um sistema operacional Linux Ubuntu 18.04. Em seguida, o ambiente foi configurado para atender as necessidade do projeto, sendo também instalado ROS Melodic e múltiplos outros pacotes necessários para a operação do sistema.

Tabela 3: Especificações da Jetson Nano

GPU	GPU de arquitetura NVIDIA Maxwell™ de 128 núcleos
CPU	Processador Quad-Core ARM® Cortex®-A57 MPCore
Memória	LPDDR4 de 4GB e 64 bits
Armazenamento	Compatibilidade com microSD
Codificação de Vídeo	1x 4K30 | 2x 1080p60 | 4x 1080p30 | 9x 720p30 (H.264/H.265)
Decodificação de Vídeo	1x 4K60 | 2x 4K30 | 4x 1080p60 | 8x 1080p30 | 18x 720p30 (H.264/H.265)
Redes	Gigabit Ethernet, M.2 de chave E
Câmera	2x conectores de câmera de 15 pinos e 2 camadas MIPI CSI-2
Monitor	1x HDMI 2.0, 1x DP 1.2
USB	4x conectores USB 3.0 Tipo A e 1x conector Micro-B USB 2.0
Dimensões	100mm x 79mm x 30,21mm
Portas E/S	Cabeçote de 40 pinos (UART, SPI, I2S, I2C, PWM, GPIO)
	Cabeçote de automação de 12 pinos
	Cabeçote de ventoinha de 4 pinos
	Cabeçote POE de 4 pinos
	Fonte de alimentação DC
	Botões Liga/Desliga
	Forçar Recuperação e Redefinir

O robô construído pode ser observado na Fig. 25, sem a parte de sensoriamento. O sistema completamente integrado, incluindo o sensoriamento é apresentado na Fig. 26.

Figura 25: Sistema integrado

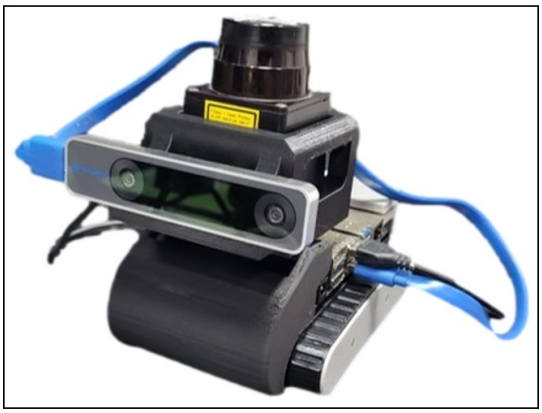

Figura 26: Sistema completo com sensoriamento

7 Resultados

Este capítulo contém os resultados de simulação e os resultados práticos do SLAM aplicado no robô. Para os testes práticos, o sistema foi testado múltiplas vezes com diferentes considerações, entretanto, sempre em ambientes internos e em superfícies planas. Todos os testes foram conduzidos no Laboratório de Robótica da PUC-Rio (LabRob), onde foi avaliada a capacidade do robô de efetuar *loop closures*, de reconhecer e determinar a real topologia do ambiente e simultaneamente estimar sua própria localização, criando um *occupancy grid map* com sua *pose* em tempo real. Todos os testes tiveram seus tópicos gravados em rosbags, permitindo a reprodução destes testes por terceiros.

A Simulações com *datasets*

Antes de implementar o algoritmo no robô, foram feitas simulações usando arquivos chamados *rosbags*. Esses arquivos, dentro do *framework* ROS, permitem a gravação em tempo real dos tópicos durante um experimento, sendo também gravado um tópico adicional chamado *clock*, que garante que a cronologia e sincronismo do teste seja preservada. Uma vez criada, a *rosbag* pode ser reproduzida em um ambiente ROS, simulando o fluxo de tópicos do teste original. Se necessário, existe também a flexibilidade de filtrar e selecionar tópicos específicos durante a reprodução da *rosbag* para adequar ao interesse da simulação. As *rosbags* escolhidas para a simulação são do dataset do robô *pioneer*, desenvolvidas e validadas em [24]. As Figs. 27 e 28 mostram testes da SLAM toolbox usando as *rosbags* mencionadas.

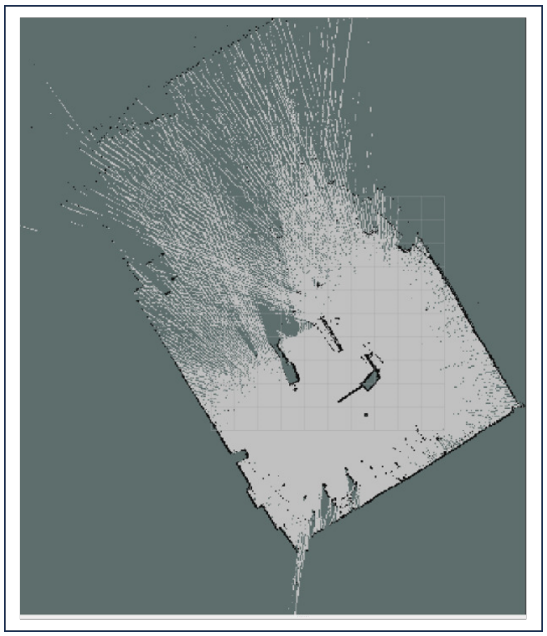

Figura 27: Teste da SLAM Toolbox usando dataset freiburg2 pioneer SLAM2

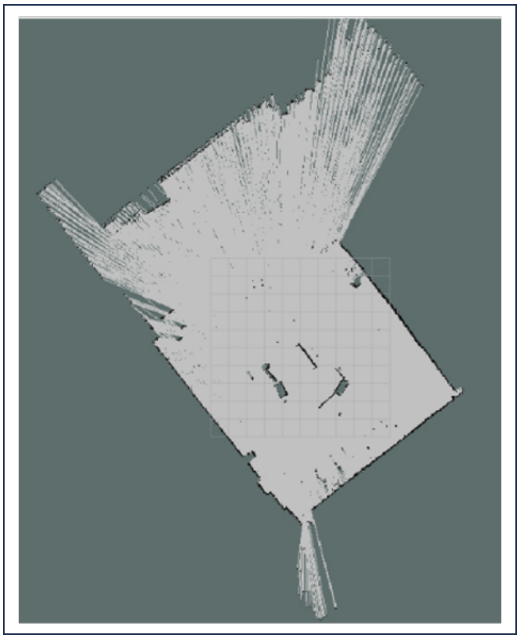

Figura 28: Teste da SLAM Toolbox usando dataset freiburg2 pioneer SLAM3

As simulações permitiram uma melhor compreensão do algoritmo e *framework*, oferecendo uma forma de visualizar seu comportamento, o que facilitou a seleção e configuração dos parâmetros para o sistema. O diagrama da Fig. 29 apresenta as conexões de tópicos entre os *nodes*. Observa-se pelo diagrama que a SLAM toolbox recebe apenas dois tópicos: a arvore de transformadas "/tf" que inclui a *pose* do robô, e o *scan laser* do LiDAR *"/scan"*. A SLAM toolbox atualiza as a árvore de transformadas com sua *pose* estimada e publica um tópico de *occupancy grid map* para atualizar o mapa. O tópico de mapa é um dado de entrada recursivo para a obtenção das estimativas do pacote.

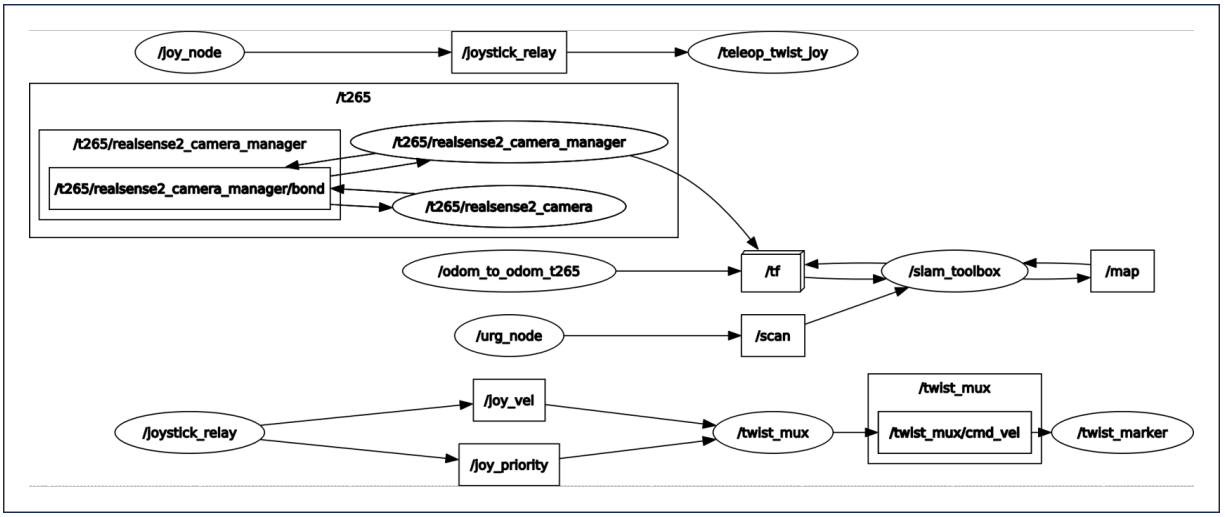

Figura 29: Diagrama de entradas e saídas entre *nodes*

B Experimentos

Os experimentos foram conduzidos no LabRob, onde o robô era posicionado em diferentes locais do ambiente, percorrendo salas distintas. O robô teve seu movimento controlado por teleoperação com um controle *joystick*. O sistema é acionado por um computador operador através do protocólo de comunicação SSH, o qual iniciava todos os *Nodes* necessários para realizar SLAM e teleoperação. Em geral, o ambiente consiste de múltiplos obstáculos bem definidos, com exceção da sala de mecânica do laboratório que possui uma geometria e obstáculos mais complexos. Para garantir que existe correlação entre a estimativa da localização do robô e de sua estimativa de mapa, durante o teste, o robô retornava a um ponto conhecido para efetuar *loop closure*. Estes experimentos focam em avaliar a qualidade da representação topológica do laboratório.

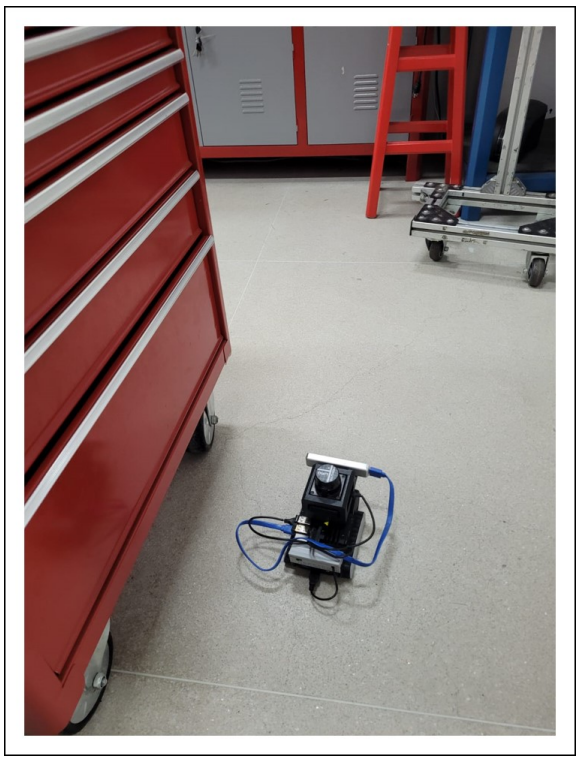

Figura 30: Robô adentrando a sala de mecânica

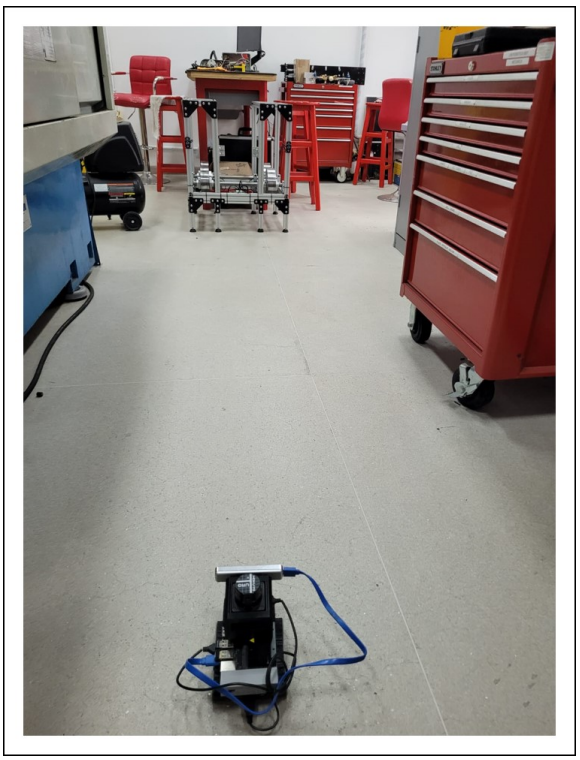

Figura 31: Robô percorrendo a sala de mecânica

O 1º teste inicializa a SLAM toolbox com o parâmetro síncrono. Este parâmetro faz com que todas as

informações dos sensores sejam levadas em consideração durante a construção do grafo, independente do atraso entre os dados. Isto implica em um aumento dos dados a serem processados e, consequentemente, um aumento do custo computacional do algoritmo. Portanto, esse parâmetro é indicado para mapeamento de pequenos ambientes, ou para sistemas que não precisam se preocupar com esse acréscimo no custo computacional.

O robô percorreu um total de três salas distintas do LabRob e demonstrou ser capaz de representar a topologia do ambiente com sucesso. O robô inicia seu trajeto no canto esquerdo inferior da sala de eletrônica. Após percorrer todas as outras salas, o robô retorna para a posição inicial do teste, demonstrando capacidade de *loop closure*. A Fig. 32 demonstra o resultado final do mapeamento deste teste, e a Fig. 33 apresenta este mesmo mapa com um vetor da trajetória estimada nos últimos instantes de mapeamento.

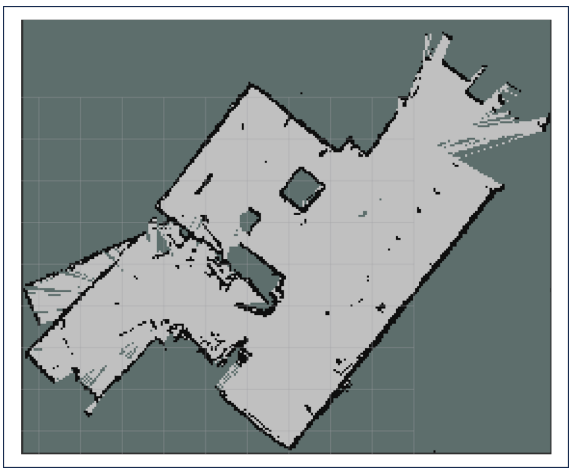

Figura 32: Mapa com parâmetro síncrono da SLAM toolbox

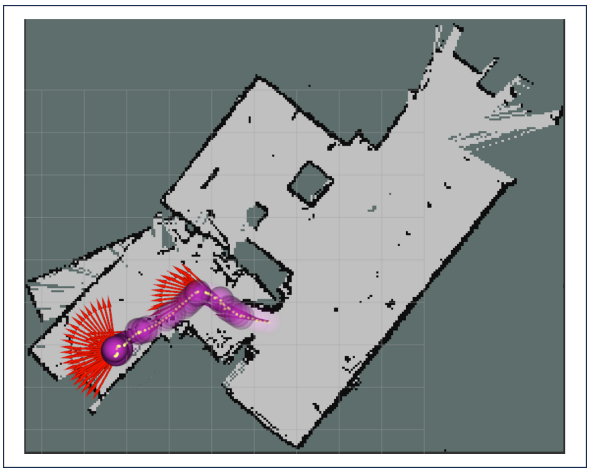

Figura 33: Mapa com parâmetro síncrono da SLAM toolbox com *buffer* da odometria

O 2º teste inicializa a SLAM toolbox com o parâmetro assíncrono, que filtra as informações a serem consideradas na construção do grafo, selecionando apenas os dados sem atraso. Essa configuração reduz significativamente o custo computacional do algoritmo.

O robô percorreu as mesmas três salas distintas do 1º teste e novamente foi capaz de representar a topologia do ambiente com sucesso. O robô inicia seu trajeto em cima do obstáculo que se assemelha um quadrado (uma pilastra). Após percorrer todas as outras salas o robô retorna para a posição inicial do teste, incetivando um *loop closure*. Apesar de realizar um trajeto diferente, e utilizar de um parâmetro com menor custo computacional, o mapa obtido demonstrou uma qualidade semelhante a do 1º teste. A Fig. 34 apresenta o resultado final do mapeamento, e a Fig. 35 apresenta o mapa com um vetor da trajetória estimada nos últimos instantes de mapeamento.

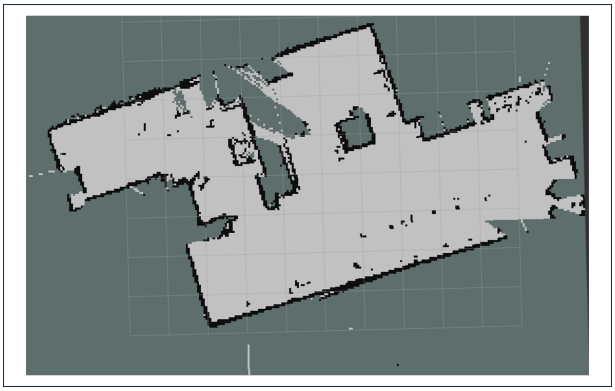

Figura 34: Mapa com parâmetro assíncrono da SLAM toolbox

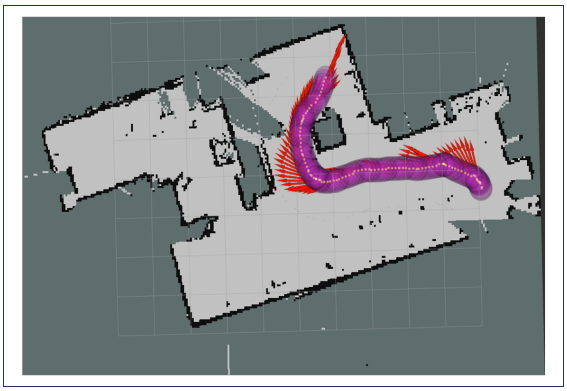

Figura 35: Mapa com parâmetro assíncrono da SLAM toolbox com *buffer* da odometria

O 3º teste mantém o parâmetro assíncrono para reduzir os custos computacionais do algoritmo e incrementa o número de salas a ser mapeado. A nova sala a ser mapeada é o setor de mecânica do laboratório,

sendo de longe a mais desafiadora para o sistema. A sala consiste de diversas bancadas, equipamentos e itens que são de difícil representação bi-dimensional.

O Robô percorreu as 4 salas distintas e sendo capaz de representar a topologia do ambiente com sucesso. Entretanto, durante a inicialização do sistema, o *node* rosbag *record* apresentou um erro e não foi ativado. Consequentemente, não houve uma gravação dos tópicos durante o percuso, apenas de seu último instante. O mapa da Fig. 36 apresenta o resultado final do mapa após o percurso do robô. Pela Fig. 37 se torna evidente que não foram gravados todos os dados de odometria, apenas de sua última *pose*. Apesar dessa falha, esse teste demonstrou que o tópico de mapa é atualizado, ou seja, uma vez construído independe dos dados *a priori*.

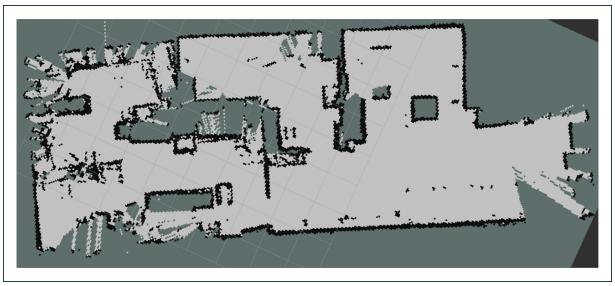

Figura 36: Mapa gravado após percurso

Figura 37: Mapa gravado após percurso com *buffer* de odometria

O 4º teste manteve todas as configurações utilizadas no 3º teste. A sala de mecânica, além de possuir uma geometria de representação complexa, também possui locais onde a conexão com a rede é de baixa qualidade. Como o sistema depende da comunicação SSH para ser teleoperado, perder a conexão à qualquer instante faz com que o teste seja encerrado.

O robô percorreu as quatro salas distintas e demonstrou ser capaz de representar a topologia do ambiente parcialmente com sucesso. As três primeiras salas foram mapeadas com qualidade, de forma semelhante aos testes anteriores. Apesar disso, durante o percurso na sala de mecânica, o robô perdeu conexão com a rede e o teste foi encerrado prematuramente. A Fig. 38 apresenta o mapa obtido. Analisando a Fig. 39, é possível identificar a útlima *pose* registrada antes do sistema perder conexão com a rede. Também se torna evidente o longo alcance do LiDAR utilizado através das medições da última *pose*.

Figura 38: Mapa encerrado prematuramente devido a uma queda de conexão

Figura 39: Mapa encerrado prematuramente devido a uma queda de conexão com *buffer* da odometria

O 5º teste também utilizou do modo assíncrono devido as considerações já menciondas. Este teste foi feito de forma extraordinária, onde duas sequências de mapeamento foram concatenadas. Para este teste, a última *pose* da primeira tomada de mapeamento foi a *pose* inicial da segunda tomada de mapeamento, ou seja, o robô iniciou a segunda tomada de mapeamento na mesma posição em que a primeira foi encerrada.

Para a primeira tomada de mapeamento, o robô percorreu um trajeto semelhante do 2º teste, mas dessa vez com a porta da sala de mecânica aberta. Após mapear as três salas distintas o robô retorna para sua posição inicial, forçando um *loop closure* e em seguida é desligado. Esta tomada de tempo foi gravada em uma rosbag.

Na segunda tomada de mapeamento, após a reinicialização do sistema, é feita a leitura dos tópicos gravadas da rosbag da primeira tomada de mapeamento. Com os dados do mapa anterior, o robô per-

correu brevemente as três salas iniciais enquanto era direcionado para a sala de mecânica. O teste é encerrado na extremidade superior esquerda da sala de mecânica.

O resultado da segunda tomada de mapeamento pode ser visto na Fig. 38. A Fig. 39 apresenta os últimos instantes de sua odometria, comprovando que o fim do teste foi feito sem um *loop closure*. Apesar de não haver um *loop closure* na segunda tomada, o algortimo demonstrou sua capacidade de retomar um mapa existente e atualizar com as novas informações obtidas.

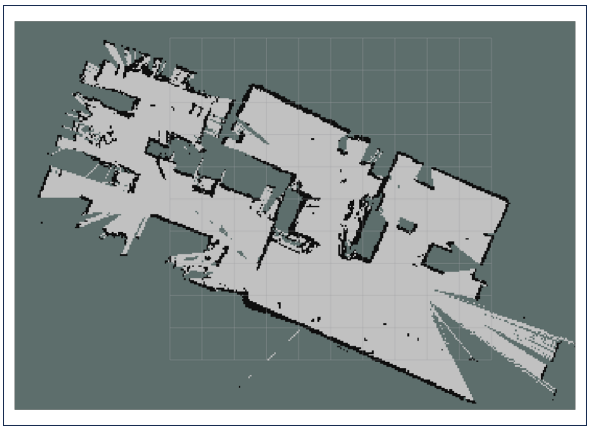

Figura 40: Mapeamento com atualização de mapa existente

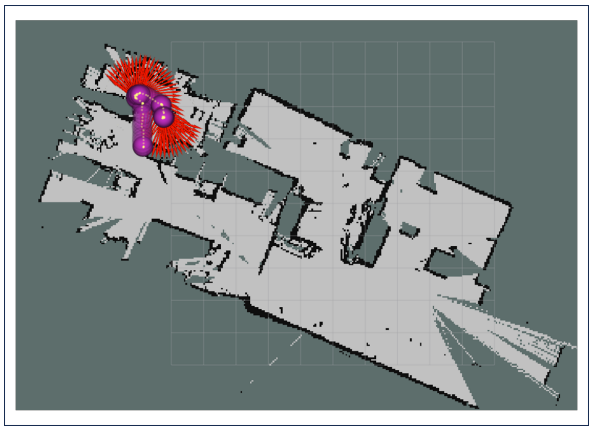

Figura 41: Mapeamento com atualização de mapa existente com *buffer* de odometria

8 Discussão dos Resultados

Neste capítulo será apresentado e discutido os resultados obtidos dos testes de validação. Estes testes foram conduzidos com o intuito de determinar a qualidade das estimativas de trajetória do sistema. A qualidade das estimativas é determinada pela magnitude absoluta do erro entre a *pose* estimada diretamente da câmera T265, a *pose* estimada pelo algoritmo da SLAM toolbox e a *pose* da trajetória *ground truth*. Deve-se lembrar que a versão utilizada da SLAM toolbox deste projeto não publicava sua estimativa de *pose*, portanto para o teste de validação foi necessário inicializar outro *node* personalizado, o SLAM *pose Listener/Subcriber*. Este *node* se inscreve nas transformadas feitas entre a odometria da SLAM toolbox e o mapa, publicando essa transformação como uma *pose*. O diagrama da Fig. 42 demonstra como o *node* é integrado com o resto da metodologia de comunicação apresentada na sub-seção D da seção 5.

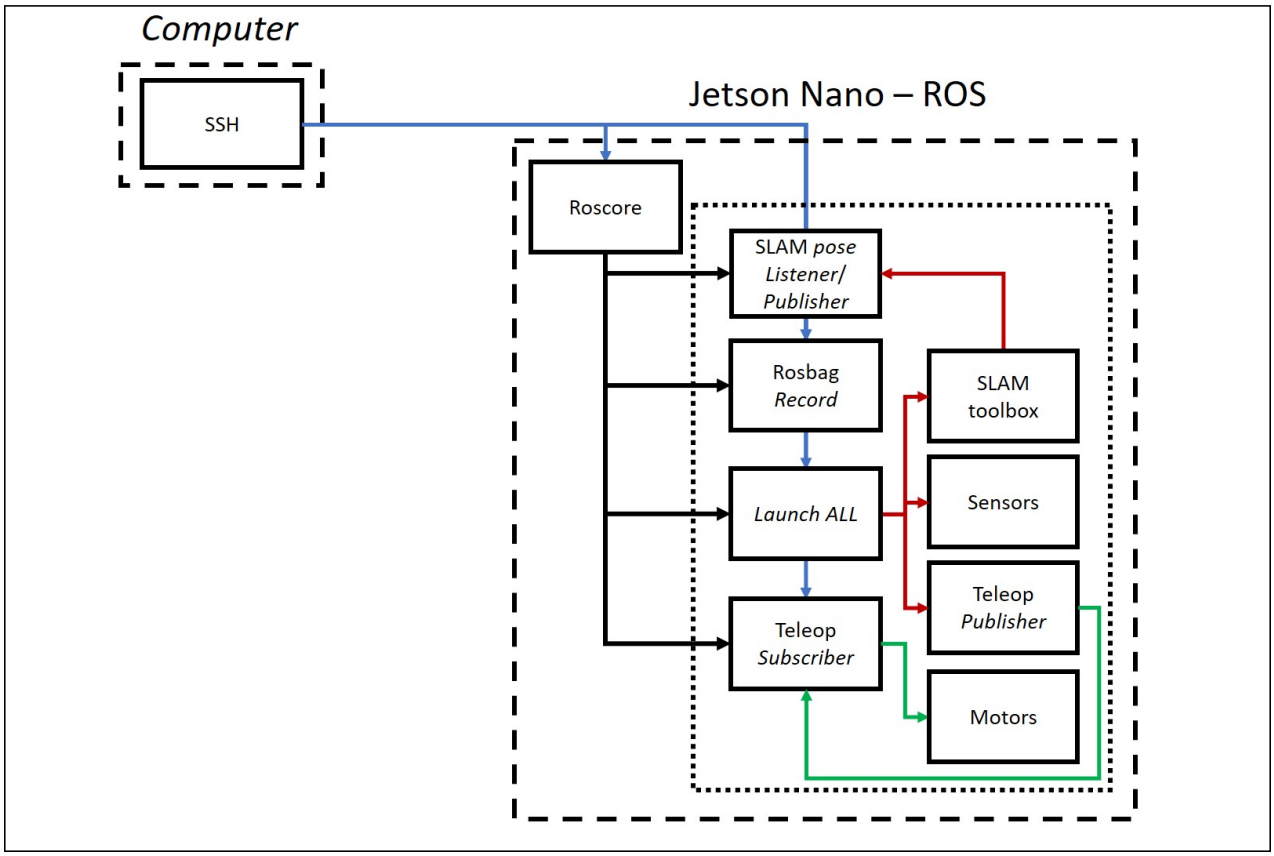

Figura 42: Diagrama de comunicação do sistema de Validação

A Métrica de validação

O Erro Absoluto da Trajetória (ATE) é utilizado para avaliar a consistência global da trajetória estimada, comparando a distância absoluta entre os componentes translacionais das trajetórias estimadas e do *ground truth*. Além disso é necessário alinhar ambas trajetórias, considerando que elas podem estar representadas em eixos de cordenadas diferentes. Utilizando do método de Horn, é possível achar a transformação de corpo rigído S correspondente. A Eq. 25 demonstra o cálculo computacional da ATE para cada passo de tempo i. [24] [25]

$$ATE_i = {Q_i}^{-1} S P_i \tag{25}$$

Onde P é a trajetória estimada, Q o *ground truth* e S é a transfomada que alinha as duas trajetórias. Para uma sequência de N *poses*, e considerando que $trans$ é a componente translacional de uma variável, o Erro Quadrático Médio (RMSE) da ATE é descrito pela Eq. 26. [24]

$$RMSE(ATE_{1:N}) = \sqrt{\frac{1}{N}\sum_{i=1}^{N} ||trans(ATE_i)||^2} \tag{26}$$

B Testes de Validação

O procedimento dos testes de validação envolve percorrer uma trajetória pré-definida, a qual o robô não possui nenhuma informação *a priori*, ou seja, o sistema deverá realizar SLAM para obter suas estimativas de trajetória. O sistema realiza a estimativa de duas trajetória distintas, uma providênciada diretamente pela câmera Intel Realsense T265 e outra providenciada pelo algoritmo de SLAM.

Uma câmera RGB-D é posicionada em paralelo ao chão, de forma que observe o movimento do robô através do rastreamento de uma *apriltag*,isto é, um *QR-code* com características específicas para representar uma *pose*. O trajeto das poses observadas irá compor o *ground truth* do teste. O Robô sempre inicia o teste na posição central de um retângulo demarcado no chão e então é direcionado para o ponto superior ou inferior mais próximo da demarcação da trajetória pré-definida. O robô percorre a trajetória até alcançar novamente a posição inicial, após o robô retornar ao centro do retângulo.

Os testes de validação foram salvos em duas *rosbags*, uma inclui os dados obtidos pela câmera RGB-D e a outra possui os dados do sistema em si. Nestes testes foram adicionados novos tópicos ao conjunto do sistema, que incluem a imagem de ambas as lentes *fisheye* e a *pose* estimada pelo algoritmo de SLAM, obtida pelo *node* descrito no ínicio desta seção.

A Fig. 43 apresenta a estrutura do teste de validação pela perspectiva da câmera observadora, e a Fig. 44 demonstra um teste de validação, no qual pode ser visto o robô com a *apriltag* percorrendo a trajetória demarcada.

Figura 43: Perspectiva da câmera observadora da estrutura de validação do sistema

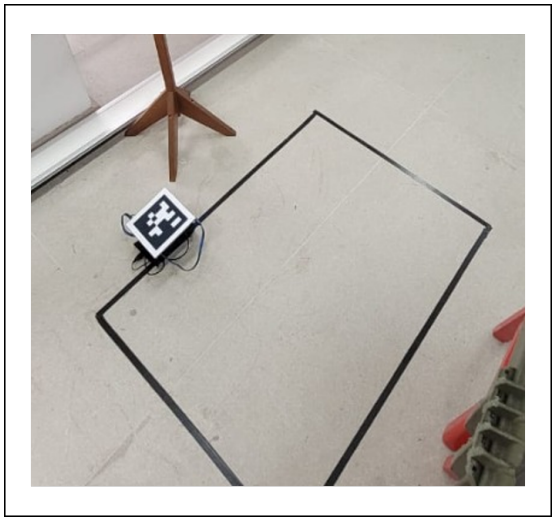

Figura 44: Robô com *apriltag* percorrendo o trajeto de validação

B.1 1º Teste de Validação

A Tab. 4 mostra os resultados do primeiro teste de validação. O trajeto estimado pela câmera T265 apresentou um erro quadrático médio de $98.7mm$ com desvio padrão de $41.8mm$. A média do erro foi $89.4mm$ e sua mediana $84.2mm$. O menor valor do erro foi de $6.24mm$ com máxima de $210mm$.

O trajeto estimado pelo algoritmo de SLAM apresentou um erro quadrático médio de $113mm$ com desvio padrão de $53.8mm$. A média do erro foi $99.9mm$ e sua mediana $91.0mm$. O menor valor de erro foi de $13.3mm$ com máxima de $246mm$.

Neste teste a estimativa de odometria da T265 melhor se aproximou do *ground truth*, obtendo um erro de sua estimativa menor do que a estimativa do algoritmo de SLAM, o que não deveria ser o caso. Este fenômeno pode ser consequência das limitações de *hardware* do projeto. Normalmente, o sistema já opera múltiplos ROS *nodes* e durante os teste de validação são necessários de ainda mais *nodes*. Além

disso, é também necessário o processamento de imagem observada pelas lentes *fisheye*, as quais são armazenadas como tópicos na *rosbag*, elevando significamente os custos computacionais do teste.

Outra possível explicação é que a SLAM toolbox na versão Melodic ainda não havia refinado o algoritmo de localização, uma vez que ele não publica a *pose*. Para a obtenção das estimativas do algoritmo foi necessário o uso de um *node* adicional para se inscrever e publicar as transformadas feitas entre a odometria e o mapa, assim obtendo a *pose*, podendo haver perdas nos dados. As Figs. 45 e 46 mostram as comparações das trajetórias estimadas e ground-truth com a câmera T265 e com SLAM, respectivamente.

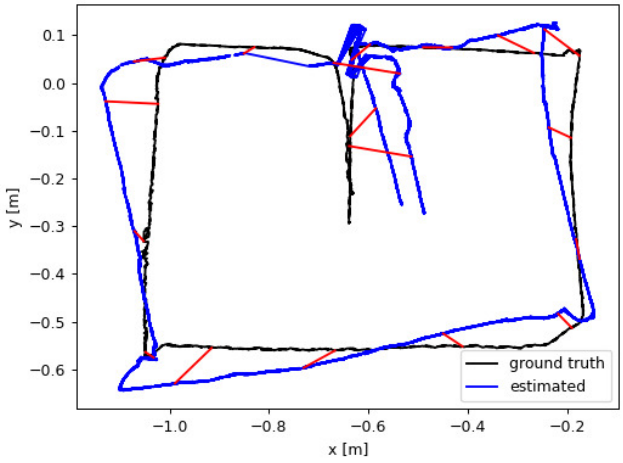

Figura 45: Resultado da primeira validação T265 X *Ground Truth*

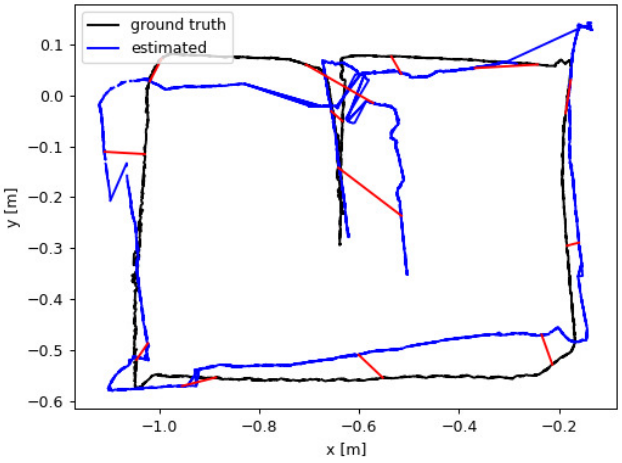

Figura 46: Resultado da primeira validação SLAM X *Ground Truth*

Tabela 4: Tabela de resultados da 1ª Validação

	Pares de Pose Comparados	**Erro Translacional Absoluto (ATE)**					
		RMSE	**Mean**	**Median**	**Std**	**Min**	**Max**
T265	2441	0.098691m	0.089408m	0.084164m	0.041787m	0.006243m	0.210397m
SLAM	1761	0.113499m	0.099955m	0.091020m	0.053767m	0.013283m	0.246347m

B.2 2º Teste de Validação

A Tab. 5 mostra os resultados numéricos do segundo teste de validação. O trajeto estimado pela câmera T265 apresentou um erro quadrático médio de $86.3mm$ com desvio padrão de $34.3mm$. A médio do erro foi $79.2mm$ e sua mediana $75.4mm$. O menor valor de erro foi de $10.9mm$ com máxima de $163.9mm$

O trajeto estimado pelo algoritmo de SLAM apresentou um erro quadrático médio de $88.0mm$ com desvio padrão de $40.8mm$. A média do erro foi $78.0mm$ e sua mediana $71.2mm$. O menor valor de erro foi de $9.28mm$ com máxima de $228mm$.

Neste teste a câmera e algoritmo apresentaram erros de estimativa próximos. Entretanto, pode-se observar pela Fig. 48, que há discontinuidade em suas estimativas, consequência da alta exigência dos processadores causando *stuttering*. O mesmo é verdade para o teste anterior, porém se torna mais evidente no segundo teste.

Estas discontinuidades impactam negativamente nas estimativas do algoritmo de SLAM, o que torna necessário uma re-localização para continuar a trajetória. Esse fenômeno esclaresce os valores altos apresentados nas estimatis de SLAM em seu erro máximo e desvio padrão para ambos testes. Além disso, percebe-se que existem menos *poses* para serem comparadas nas estimativa do algoritmo de SLAM.

Estes erros estão quantitativamente longe dos resultados de estimativa de localização do estado da arte. Os erros apresentados podem ser consequência de diversos fatores: a dificuldade da estimativa de localização de um robô compacto em um ambiente comparativamente muito maior, limitações de *hardware* que causaram *stuttering* durante o teste, erros de escala durante a comparação das estimativas e erros de *software* devido ao uso da versão Melodic da SLAM toolbox.

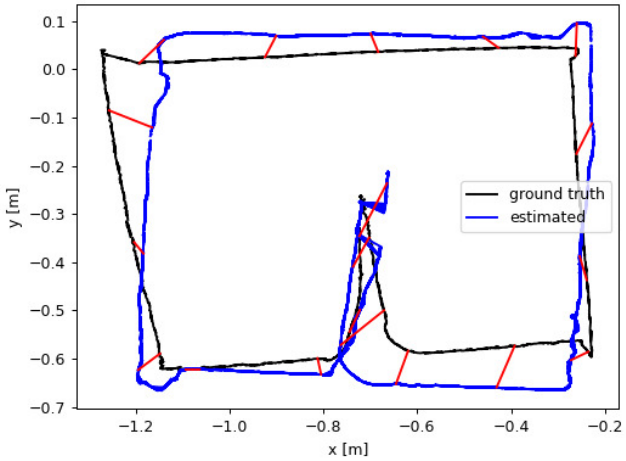

Figura 47: Resultado da segunda validação T265 X *Ground Truth*

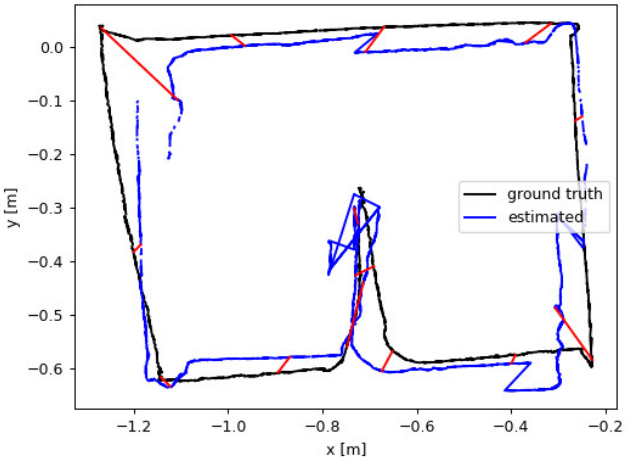

Figura 48: Resultado da segunda validação SLAM X *Ground Truth*

Tabela 5: Tabela de resultados da 2ª Validação

	Pares de Pose Comparados	**Erro Translacional Absoluto (ATE)**					
		RMSE	**Mean**	**Median**	**Std**	**Min**	**Max**
T265	2223	0.086283m	0.079184m	0.075377m	0.034272m	0.010904m	0.163948m
SLAM	1617	0.088045m	0.078039m	0.071201m	0.040765m	0.009285m	0.227953m

9 Conclusão e Trabalhos Futuros

Apesar de ser um robô de pequeno porte e com limitações de *hardware*, o sistema demonstrou ser capaz de solucionar o problema de SLAM em tempo real, efetuando a tarefa continuamente por longos períodos de tempo. Além disso, o mapeamento em ambientes internos e estáticos foi condizente com a topologia do ambiente. Entretanto, a qualidade dos resultados de localização apresentados nos teste de validação foram sub-ótimos, o que pode ser atribuído as limitações de *hardware* do sistema e o fato de que a versão utilizada da SLAM toolbox não publicava a *pose* em sua rotina, sendo necessário um *node* dedicado para abstrair esse dado. Concluindo, o projeto atendeu seus objetivos mas ainda há espaço para aperfeiçoamento.

Para trabalhos futuros é de interesse implementar uma rotina de navegação autônoma, repetindo os experimentos e avaliando a robustez do controle autônomo, incluindo o planejamentos de trajetórias e a otimização do movimento. Outro tópico de interesse para trabalhos futuros é a avaliação de outros tipos de algoritmos de SLAM e sua perfomance no sistema. Nestes testes serão avaliados a qualidade dos mapas e da estimativa de localização, comparando-os com os resultados desse projeto, sendo também considerado o custo computacional envolvido. Estes testes iriam incluir diferentes algoritmos de SLAM já inclusos no *framework ROS*, como por exemplo *gmapping*, EKF-SLAM, Hector-SLAM, dentre outros.

É também de interesse para trabalhos futuros migrar o sistema para ROS2, permitindo o uso de pacotes mais atualizados, o que poderia até melhorar o erro da estimativa de localização. Para isso será necessário a configuração de um *Docker*, um sistema operacional virtual, já que a Jetson Nano é incompatível com versões mais recentes de Ubuntu.

Referências

[1] R. Siegwart and I. R. Nourbakhsh, *"Introduction to Autonomous Mobile Robots"*. The MIT Press, 2004.

[2] I. I. D. T. Dynamic Legged Systems Lab, "Hyqreal," 2019. [Online]. Available: https://dls.iit.it/

[3] Durrant-Whyte and Bailey, "Simultaneous Localisation and Mapping (SLAM):Part I The Essential Algorithms," *"IEEE Robotics & Automation Magazine"*, 2006.

[4] C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. Reid, and J. Leonard, "Past, Present, and Future of Simultaneous Localization And Mapping: Towards the Robust-Perception Age," *IEEE Transactions on Robotics*, 2016.

[5] Waymo and Google, "Waymo firefly," 2015. [Online]. Available: https://waymo.com/

[6] P. Corke, W. Jachimczyk, and R. Pillat, *"Robotics, Vision and Control"*. Springer Tracts in Advanced Robotics, 2023.

[7] B. K. Luknanto, "A Review of 2D SLAM Algorithms on ROS," Master's thesis, POLITECNICO DI MILANO, 2018-2019.

[8] G. Grisetti, R. Kummerle, C. Stachniss, and W. Burgard, "A Tutorial on Graph-Based SLAM," *"IEEE Robotics & Automation Magazine"*, 2010.

[9] T. Chong, X. Tang, C. Leng, M. Yogeswaran, O. Ng, and Y. Chong, "Sensor Technologies and Simultaneous Localization and Mapping (SLAM)," *Procedia Computer Science*, 2015.

[10] G. P. da Cruz Junior, L. V. do Carmo Matos, H. Azpurua, G. Pessin, and G. M. Freitas, "Investigação de Técnicas LiDAR SLAM para um Dispositivo Robótico de Inspeção de Ambientes Confinados," *SBA - Sociedade Brasileira de Automática*, 2020.

[11] C. Robotics, "Warthog," 2016. [Online]. Available: https://clearpathrobotics.com/warthog-unmanned-ground-vehicle-robot/

[12] K. Yousif, A. Bab-Hadiashar, and R. Hoseinnezhad, "An Overview to Visual Odometry and Visual SLAM: Applications to Mobile Robotics," *Springer Tracts in Advanced Robotics*, 2020.

[13] S. Macenski and I. Jambrecic, "SLAM Toolbox: SLAM for the dynamic world," *The Journal of Open Source Software*, 2021.

[14] C. Debeunne and D. Vivet, "A Review of Visual-LiDAR Fusion based Simultaneous Localization and Mapping," *MDPI*, 2020.

[15] K. Lynch and F. Park, *"Modern Robotics: Mechanics, Planning, and Control"*. Cambridge Univeristy Press, 2017.

[16] A. J. V. dos Santos, "Analise e Controle de um Veículo Robótico Tracionado por Esteiras," Ph.D. dissertation, UFPB/CT, 2015.

[17] S. Thrun, D. Fox, and W. Burgard, *"Probabilistic Robotics"*. The MIT Press, 2005.

[18] S. Thrun, "Is robotics going statistics? the field of probabilistic robotics," *Communications of The ACM - CACM*, 2001.

[19] M. E. M. Mafalda, "Desenvolvimento de Sistema de Navegação Autônomo com Exploração de Fronteira," Master's thesis, Universidade Federal de Santa Catarina, Centro De Tecnologia, 2021.

[20] S. Huang, H.-Z. Huang, Q. Zeng, and P. Huang, "A Robust 2D Lidar SLAM Method in Complex Environment," *Photonic Sensors*, 2022.

[21] S. A. I. L. et al., "Robotic operating system." [Online]. Available: https://www.ros.org

[22] S. Macenski, "SLAM Toolbox," 2023, "SLAM Toolbox documentation". [Online]. Available: https://github.com/SteveMacenski/slam_toolbox

[23] R. Bonghi, "Nanosaur project," 2022, open-source robot documentation. [Online]. Available: https://nanosaur.ai/

[24] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers, "A Benchmark for the Evaluation of RGB-D SLAM Systems," *IEEE Transactions on Robotics*, 2012.

[25] R. Kümmerle, B. Steder, C. Dornhege, and et al, "On Measuring the Accuracy of SLAM Algorithms," *Springer Tracts in Advanced Robotics*, 2009.