Universidade Federal
de Campina Grande

UNIVERSIDADE FEDERAL DE CAMPINA GRANDE
CENTRO DE ENGENHARIA ELÉTRICA E INFORMÁTICA
CURSO DE GRADUAÇÃO EM ENGENHARIA ELÉTRICA

Pedro José Gondim César

Estudo de técnicas de localização e mapeamento simultâneos aplicadas a robôs móveis com rodas

Campina Grande
2022

Pedro José Gondim César

Estudo de técnicas de localização e mapeamento simultâneos aplicadas a robôs móveis com rodas

Trabalho de Conclusão de Curso do Curso de Graduação em Engenharia Elétrica do Centro de Engenharia Elétrica e Informática da Universidade Federal de Campina Grande para a obtenção do título de Bacharel em Engenharia Elétrica.
Orientador: Prof. Antonio Marcus Nogueira Lima, Dr.

Campina Grande
2022

Pedro José Gondim César

Estudo de técnicas de localização e mapeamento simultâneos aplicadas a robôs móveis com rodas

Este Trabalho de Conclusão de Curso foi julgado adequado para obtenção do Título de "Bacharel em Engenharia Elétrica" e aprovado em sua forma final pelo Curso de Graduação em Engenharia Elétrica.

Campina Grande, 19 de fevereiro de 2023.

Prof. Antonio Marcus Nogueira Lima, Dr.
Orientador

Prof. Gutemberg Gonçalves dos Santos Júnior, Dr.
Avaliador(a)

Este trabalho é dedicado a minha mãe Eneide Gondim, que me apoiou financeira e emocionalmente e a meu avô Pedro Gondim, que me concedeu morada em Campina Grande.

AGRADECIMENTOS

Agradeço a minha família por desde sempre me guiar e ser a minha maior fonte de apoio.

Agradeço aos colegas de curso que contribuíram durante minha jornada, alguns destes que se tornaram meus amigos.

Agradeço as pessoas que compõe ou compuseram o e-Robótica por me ajudar a iniciar nessa área.

Agradeço ao professor Antonio Marcus pelo seu tempo, orientação e o imenso conhecimento que me foi passado.

Agradeço ao professores Marcos Morais e Gutemberg Gonçalves por junto com o professor Prof. Antonio Marcus Nogueira Lima, Dr.ministrar as aulas do projeto que serviram de alicerce para que eu pudesse elaborar este trabalho.

RESUMO

Técnicas de Localização e Mapeamento Simultâneo (LMS) são amplamente utilizadas na robótica móvel, por meio dessas técnicas o robô é capaz de criar um mapa de um ambiente desconhecido com a utilização de seus próprios sensores sem a necessidade do uso de uma fonte externa para fornecer o mapa ao robô. Um mapa do ambiente é necessário para realizar atividades da robótica móvel como a localização e navegação. Esse trabalho tem como objetivo fazer um estudo das técnicas existentes para localização e mapeamento simultâneos, bem como dos tipos possíveis de mapa a serem criados. Para realizar esse estudo será feita uma revisão da literatura existente e a técnica de LMS utilizando grafos será aplicada em ambiente simulado e em um robô real.

Palavras-chave: LMS. LMS Visual. Gazebo. Nanosaur.

ABSTRACT

Simultaneous Localization and Mapping (SLAM) techniques are widely used in mobile robotics. With the use of these techniques the robot is able to create the map of a unknown environment using it's own sensors, without the necessity of a external source to provide the map for the robot. A Environment map is needed in mobile robotics activities such as localization and navigation. This Bachelor's dissertation has the goal of researching the existing SLAM techniques and the types of maps that can be created. To achieve this goal the existing literature will be consulted and the Graph-Based SLAM technique is going to be applied on a virtual environment and on a real robot.

Keywords: SLAM. VISUAL SLAM. Gazebo. Nanosaur.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

GPIO	*General Purpose Input Output*
GPU	*Graphical Processing Unit*
I2C	*Inter-Integrated Circuit*
IMU	*Inertial Measurement Unit*
LMS	Localização e Mapeamento Simultâneo
ROS	*Robot Operating System*
RTOS	*Real Time Operating System*

LISTA DE SÍMBOLOS

x	Posição do robô no eixo x
y	Posição do robô no eixo y
θ	Deslocamento angular
v_r	velocidade da roda direita
v_l	velocidade da roda esquerda
b	Distância entre os eixos da roda
z	Leituras dos sensores exteroceptivos
u	Leituras dos sensores proprioceptivos
p	Variável de estado da posição do robô
a	evento a
c	evento c
d	evento d
r	raio

SUMÁRIO

1 INTRODUÇÃO

O LMS foi escolhido como o objeto de estudo deste trabalho devido a sua relevância no âmbito da robótica, portanto na seguinte deste capítulo onde são apresentadas as justificativas para o trabalho será feita uma breve introdução sobre o que é LMS e sobre sua importância para o desenvolvimento de robôs móveis. Este capítulo também tratará da metodologia adotada e dos objetivos gerais e específicos para o trabalho.

1.1 JUSTIFICATIVA

O processo de criação e configuração de um robô móvel geralmente tem como objetivo final torná-lo capaz de realizar uma tarefa útil à humanidade, como limpar o chão, cortar a grama, mover um objeto de um ponto a outro em uma fábrica e automatizar a produção, explorar os oceanos, explorar outros planetas, são algumas das milhares aplicações de um robô móvel.

Para tornar possível a realização dessas tarefas são necessárias várias etapas como o projeto dos controladores responsáveis pelo movimento, a configuração dos sensores exteroceptivos utilizados para perceber o ambiente ao redor do robô e dos sensores proprioceptivos, utilizados para fazer a leitura de variáveis do próprio robô, a criação de um mapa do ambiente dentre outras. Este trabalho faz o estudo da etapa de criação do mapa, mais especificamente das técnicas que podem ser utilizadas para que o próprio robô seja capaz de criar um mapa descritivo.

Um mapa descritivo é necessário pois a maioria das tarefas designadas para um robô móvel necessitará que este se movimente de um ponto ao outro no ambiente. Como no exemplo citado anteriormente de uma fábrica, é necessário que o robô saiba onde está para planejar o caminho a ser percorrido para chegar ao objeto que se deseja mover. A determinação do local em que se encontra o robô no ambiente corresponde a etapa da localização, o planejamento da trajetória a ser realizada até determinado local corresponde a etapa de navegação, ambas as etapas necessitam de um mapa para seu funcionamento.

Este mapa pode ser fornecido ao robô por uma fonte externa ou criado pelo próprio robô. No caso da criação do mapa pelo próprio robô, este irá percorrer todo o ambiente e realizará leituras com seus sensores de forma a determinar a posição dos objetos que estão ao seu redor com relação ao próprio robô, sendo necessário então transformar todas essas medições em um referencial global e fixo. Esta transformação somente é possível caso o robô saiba qual a sua posição no mapa enquanto faz a leitura. É preciso então resolver dois problemas simultaneamente, determinar a posição dos objetos com relação ao robô ao mesmo tempo em que determina a posição do robô com relação ao mapa que está sendo criado.

O objeto de estudo deste trabalho é justamente o conjunto de técnicas que resolvem esses problemas, são fundamentais para o desenvolvimento de um robô móvel pois com

sua utilização é possível eliminar a necessidade de um agente externo para a criação do mapa, como o ser humano, o que possibilita ao robô se tornar uma entidade com um maior nível de autonomia.

1.2 METODOLOGIA

A metodologia adotada para esse trabalho consiste na leitura sobre as técnicas existentes para o LMS e sobre os tipos de mapas que podem ser criados.

Dentre todas as técnicas existentes, a técnica de LMS baseada em grafos será aplicada em ambiente virtual com o simulador Gazebo e utilizando um robô real. A escolha desta técnica se deve a preexistência de um projeto open source que a utiliza para realizar a LMS. Este projeto sugere montar um robô chamado Nanosaur de baixo custo em comparação com outros robôs existentes no mercado o que torna possível sua construção.

Outras técnicas de LMS, como por exemplo as que se utilizam de sensores do tipo Lidar para a construção do mapa, necessitam de um investimento financeiro maior do que o realizado devido ao alto custo dos seus sensores, o que inviabiliza a construção de um robô que utiliza esse tipo de sensor. Por esse motivo, uma técnica de LMS que utiliza o sensor Lidar será implementada utilizando o pacote SLAM TOOLBOX em ambiente virtual com o uso do simulador Gazebo.

O robô Nanosaur (BONGHI, 2020) foi projetado por Raffaelo Bongui e utiliza para realizar o processamento de dados uma placa de desenvolvimento Nvidia Jetson Nano. O robô utiliza uma câmera como sensor sendo possível selecionar qual a câmera a ser utilizada dentre os diversos modelos para os quais existe suporte.

Devido a sua boa performance a câmera escolhida para realizar o processo de LMS Visual foi a Realsense. No quadro 1 estão descritos alguns dos itens utilizados. Devido a dificuldades encontradas para a construção do robô, descritas na 4.2, a base do robô teve que ser trocada sendo perceptíveis as diferenças entre o projeto original feito por Raffaello Bongui, ilustrado na Figura 1 , e o robô criado durante esse trabalho, ilustrado na Figura 2

Quadro 1 – Items utilizados.

Item	Função
MicroSD card 200 Gb	Armazenamento principal da placa
Adaptador Usb para Wifi	Conexão Wifi com a rede Local
Bateria	Fonte de energia
Placa de controle	Regular a velocidade dos motores com uma Ponte H
Chassi e motores	Base do robô com os motores para movimentação
Nvidia Jetson Nano	Processamento feito pelo robô
Ventoinha	Resfriamento da Placa
Peças 3D	Suporte para o Hardware mencionado
Intel Realsense Depth Câmera D435i	Sensor de visão
Computador	Comandar o robô pelo ROS

Fonte: Autoria Própria.

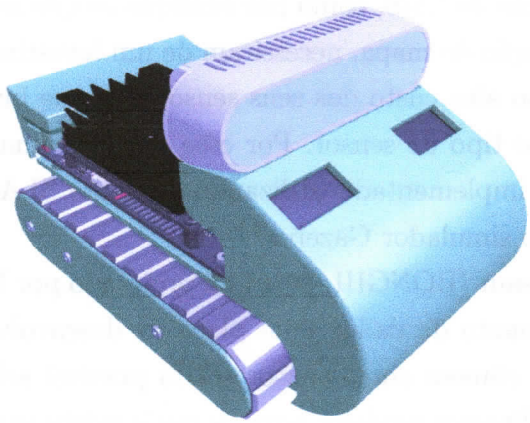


Figura 1 – Robô Nanosaur - Projeto Original.

Fonte: (BONGHI, 2020).

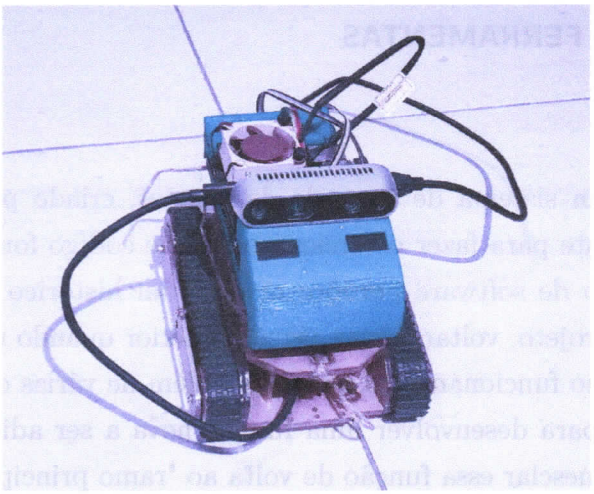


Figura 2 – Robô Nanosaur - Concepção.

Fonte: Autoria Própria.

1.3 OBJETIVOS

Nas seções abaixo estão descritos os objetivos gerais e os objetivos específicos deste TCC.

1.3.1 Objetivo Geral

O trabalho tem como objetivo geral obter conhecimento sobre as técnicas de resolução de LMS por meio e pesquisa na literatura existente e com a aplicação dessas técnicas de LMS em ambiente de simulação e em um robô real.

1.3.2 Objetivos Específicos

Para alcançar o objetivo geral é necessário:

- Preparar o ambiente de simulação para a implementação da técnica de LMS Visual
- Montar e testar o robô Nanosaur
- Estudar e implementar as técnicas de resolução do Problema de LMS
- Testar as técnicas de resolução do Problema de LMS

2 MÉTODOS E FERRAMENTAS

2.1 GIT

O Git é um sistema de controle de versões, criado por Linus Torvalds é utilizado principalmente para fazer o versionamento de código fonte. sua utilização durante o desenvolvimento de software permite manter um histórico de todas as modificações feitas durante o projeto, voltar a uma versão anterior quando uma modificação realizada causa problemas no funcionamento do código, além de várias outras possibilidades como criar um "ramo" para desenvolver uma função nova a ser adicionada e após concluir o desenvolvimento mesclar essa função de volta ao "ramo principal".

O uso do Git em conjunto com plataformas online que armazenam projetos como o GitHub permite que pessoas contribuam para o projeto independente de sua localização geográfica. Além de trazer mais segurança por armazenar uma cópia segura do código fonte do projeto que está sendo desenvolvido.

2.2 ROS

O *Robot Operating System* (ROS) é um projeto desenvolvido inicialmente por Eric Berguer e Keenann Wyrobek. A motivação para seu desenvolvimento foi a dificuldade que outros pesquisadores tinham para desenvolver software em robótica, por não existir até o momento uma solução que agregasse os projetos desenvolvidos por outros pesquisadores de forma que esse trabalho pudesse ser reaproveitado. Para testar suas ideias, um pesquisador precisava gastar a maior parte de seu tempo reescrevendo o código de outros de forma a adaptá-lo para seu ambiente de testes.

O ROS surge como uma forma de contornar essa dificuldade é um meta sistema operacional, com um conjunto de ferramentas e bibliotecas que permitem à qualquer um que deseje desenvolver em robótica reutilizar o trabalho feito por outros. Apesar da dificuldade inicial em conseguir financiamento e credibilidade com muitos possíveis investidores mencionando que era loucura tentar desenvolver um "Linux da Robótica", a ideia vingou e hoje o ROS é amplamente utilizado em robôs no mundo todo.

O ROS fornece serviços como abstração de hardware, controle de dispositivos em baixo nível, comunicação entre processos e também provê ferramentas para gerenciamento de pacotes. Um pesquisador, ao desenvolver uma solução, pode publicá-la na forma de um pacote que fica disponível para toda a comunidade que utiliza o ROS, outro pesquisador pode reutilizar esse mesmo pacote, implementar melhorias e desenvolver outra solução, fazendo a publicação do código fonte, geralmente com a utilização da ferramenta de versionamento de código git, o que permite que essa solução seja testada e reaproveitada por qualquer outro membro da comunidade.

Devido a algumas das limitações existentes no ROS1 como ser suportado somente

em sistemas operacionais que usam o kernel Linux, não haver suporte para *Real Time Operating System* (RTOS), dentre outras mencionadas em (GERKEY, 2014) foi criada uma nova versão do ROS o ROS2. Com o ROS1 é necessária a existência de processo mestre, denominado *roscore* que coordena a comunicação entre processos. Essa necessidade é eliminada no ROS2 com a utilização do DDS como camada intermediária.

Durante o desenvolvimento do projeto o ROS2 será utilizado, o que permitirá a utilização de soluções desenvolvidas por outros pesquisadores, de todas as ferramentas de desenvolvimento disponibilizadas pelo ROS2, bem como possibilitará o aprendizado sobre essa ferramenta que é amplamente utilizada na indústria.

2.3 GAZEBO

O Gazebo é uma das diversas ferramentas disponíveis para realizar simulação em robótica. Algumas outras ferramentas são CoppeliaSim e Isaac Sim. A capacidade de testar soluções em ambiente simulado, antes de utilizá-las em um robô real é fundamental para possibilitar o desenvolvimento em robótica. A maioria dos robôs disponíveis na indústria custa entre alguns milhares e algumas centenas de milhares de reais. Para realizar um investimento dessa magnitude, é necessário primeiramente saber se este é capaz de realizar a tarefa para a qual foi designado.

Para isso, são utilizadas ferramentas, onde são testados o software a ser utilizado no robô, se suas dimensões são compatíveis com o local onde este será instalado, a atuação de suas juntas, o funcionamento dos algoritmos, dispositivos de segurança além de vários outros testes que podem ser realizados.

O Gazebo foi escolhido por ter uma fácil integração com o ROS, com uma diversidade de plugins já disponíveis para realizar o teste de componentes do robô, ter uma grande variedade de modelos de ambientes e objetos disponíveis para simulação, o que facilita a criação de cenas para testar os algoritmos de LMS, ser amplamente usado na indústria e por ser *open source.* onde a física, a informação dos sensores, dentre outros são simulados e atualizados.

O *gzclient* tem a função de ler os dados gerados pelo *gzserver* e disponibilizá-los em uma interface gráfica para usuário final, além de ler os dados de entrada do usuário por meio de sua interface gráfica e disponibiliza-los para o *gzserver.* É possível simular sem executar o *gzclient* ,ou seja, sem uma interface gráfica, mas não é possível simular sem o *gzserver*

2.4 DOCKER

Durante o desenvolvimento de software na forma tradicional, o programador precisa configurar o ambiente com a instalação de bibliotecas, aplicativos e a realização de várias configurações para que o computador seja capaz de executar a aplicação que está sendo

desenvolvida. Tomando como exemplo o tema desse trabalho, no caso da criação de um script em Python que utilize o ambiente ROS2 para ler os dados dos sensores e implementar alguma técnica de SLAM, seria necessário antes de começar o desenvolvimento:

- Instalar o ROS2 na máquina.
- Instalar todas as dependências necessárias, nas versões corretas.
- Configurar corretamente o DDS
- Instalar o interpretador Python
- Instalar todas as bibliotecas necessárias em Python, nas versões corretas para executar o script.

Caso uma das etapas seja executada de forma errada a execução do programa não funcionará, o que traz problemas se o computador for trocado por um que não está configurado corretamente. Isso ocorre quando a solução é testada por outra pessoa para avaliar seu funcionamento. Caso existam problemas durante os testes, é díficil encontrar a causa pois o programador pode ter desenvolvido a solução errada ou o ambiente pode ter sido configurado incorretamente.

Outro problema que ocorre durante o desenvolvimento é a necessidade da utilização de um software que não é suportado pelo ambiente disponível. No trabalho a ser desenvolvido, será usado o pacote Isac ROS Visual Slam desenvolvido pela Nvidia. Esse pacote foi desenvolvido para funcionar com o ROS2 na versão Humble que é suportado no Ubuntu Jammy Jellyfish e em outros sistemas operacionais como Windows, porém o software base implementado para a Jetson Nano que será utilizada no robô o L4T (*Linux for Tegra*) que é baseado numa versão anterior do Ubuntu.

Portanto, para utilizar esse pacote na Jetson é necessário remover a dependência do sistema operacional e da configuração do ambiente na solução a ser implementada. O Docker é uma solução que é capaz de realizar essa tarefa com a utilização de containers.

Containers são uma forma de encapsular o código necessário para rodar uma aplicação bem como todas as dependências necessárias, de forma que a aplicação têm todo ambiente necessário a sua execução.

3 FUNDAMENTAÇÃO TEÓRICA

Esse capítulo foi escrito com o objetivo de fornecer os fundamentos necessários para entender o processo de Localização e Mapeamento Simultâneo (LMS). O entendimento sobre LMS não seria possível sem primeiro entender o que é o primeiro termo da sigla, a Localização. Por esse motivo na primeira seção deste capítulo será explicado o que é Localização e qual o problema que a Localização tenta resolver.

Em seguida a questão do Mapeamento será abordada com um descrição sobre os tipos de mapas existentes e qual o mapa que será criado pelo robô. Por fim, o problema da Localização e Mapeamento Simultâneo será abordado com uma descrição das técnicas de LMS que podem ser utilizadas.

3.1 O PROBLEMA DA LOCALIZAÇÃO

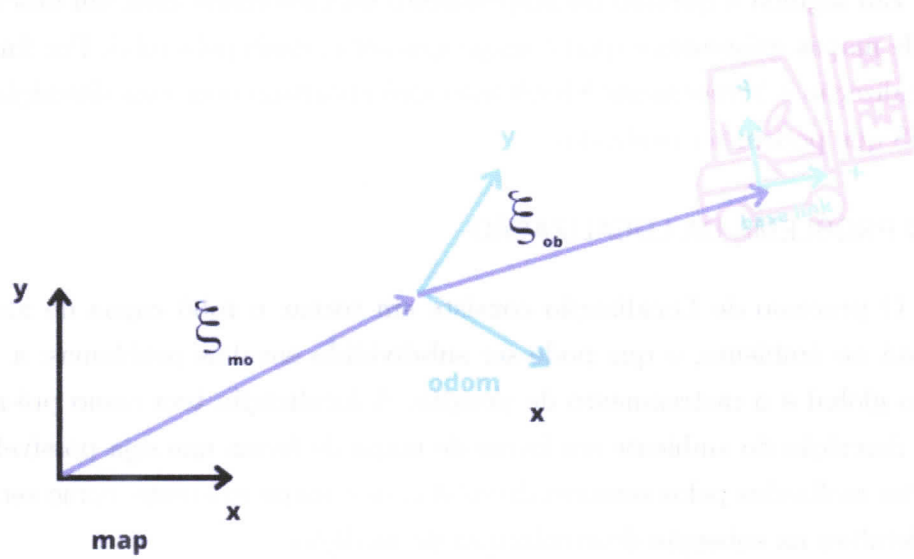
O processo de Localização consiste em tornar o robô capaz de identificar onde este está no ambiente, o que pode ser subdividido em dois problemas: a estimação da posição global e o rastreamento de posição. A localização tem como pré-requisito uma prévia descrição do ambiente em forma de mapa de forma que seja possível comparar as medições realizadas pelos sensores do robô com o mapa existente, como será descrito em mais detalhes na subseção de atualização de medição.

Dado esse mapa e um sistema de coordenadas de referência geralmente denominado por *map* (INFRASTRUCTURE, 2016) a localização tem como objetivo determinar a *pose* do robô, definida pela posição e orientação do mesmo $p = [x, y, \theta]$ com relação ao sistema de coordenadas de referência, dada uma posição inicial conhecida $p = [x_0, y_0, \theta_0]$ e uma série de medições de sensores exteroceptivos e proprioceptivos representadas respectivamente por $Z = [z_0, z_1, z_2, \ldots]$ e $U = [u_0, u_1, u_2, \ldots]$.

Na imagem 3 temos uma ilustração do conteúdo descrito com o sistema de coordenadas de referência denominado por *map*, um sistema de coordenadas intermediário denominado por *odom* e o sistema de coordenadas do robô denominado por *base link*.

Para determinar a *pose*do robô e necessário encontrar a transformação linear entre as coordenadas *base link* e *map* que pode ser obtida a partir da multiplicação entre as matrizes de transformação entre os sistemas de coordenadas *map* e *odom* ξ_{m_o} e os sistemas de coordenadas *odom* e *base link* ξ_{ob}.

Figura 3 – Sistemas de coordenadas - A posição do robô pode ser descrita como o deslocamento entre o referencial global e fixo denominado por *map* e o referencial do robô denominado por *base link*



Fonte: Própria.

Com esse intuito a metodologia geralmente adotada é a de desenvolver um modelo cinemático que relaciona as medições dos sensores proprioceptivos (encoders, acelerômetros, etc.) com o movimento realizado pelo mesmo e um modelo de medição, que geralmente consiste em realizar uma transformação de coordenadas das informações contidas no mapa e expressas em um referencial global para o referencial dos sensores exteroceptivos (Lidares, Câmera, etc) de forma comparar a leitura obtida pelos sensores com as leituras esperadas, caso o robô estivesse naquela posição como será visto na seção de atualização de medição, abaixo temos equações para o modelo cinemático de um robô de tração diferencial onde v_r e v_l são as velocidades das rodas direita e esquerda respectivamente e d a distância entre os eixos das rodas.

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} v\cos\theta \\ v\sin\theta \\ \frac{v_r - v_l}{2} \end{bmatrix} \tag{1}$$

$$v = \frac{v_r + v_l}{2} \tag{2}$$

$$\theta = \frac{v_r - v_l}{b} \tag{3}$$

Devido às simplificações necessárias a seu desenvolvimento, os modelos representarão características do robô de forma imprecisa, da mesma forma também haverá imprecisão na leitura dos sensores de forma que é desafiador representar a $posep = [x, y, \theta]$ de forma exata pois mesmo partindo de uma posição inicial $p = [x_0, y_0, \theta_0]$ conhecida há o acúmulo de erro devido as imperfeições dos modelos. Uma solução é representar a *pose* do robô em uma forma probabilística, utilizando métodos derivados do filtro de Bayes, com um conjunto de possíveis posições onde o robô possa estar associado a respectiva probabilidade de o robô ocupar aquela posição, conforme mostra a equação (4) para o caso discreto.

$$Bel(p(t)) = \sum_i P(p_i(t)|z_0, u_0, z_1, u_1, z_2, u_2, z_3, u_3 \ldots) \tag{4}$$

Na equação acima, o conjunto de posições $Bel(p_t)$ onde o robô acredita estar em certo instante t, é representado por uma probabilidade condicional de todas as medições anteriores. Não seria prático para uma aplicação em robótica gravar todas as informações dos sensores de forma que se essa fosse a abordagem utilizada, em algum momento haveria a extrapolação do poder computacional ou da memória disponível no Hardware existente. Portanto, a propriedade de Markov descrita abaixo será utilizada, outros teoremas fundamentais ao funcionamento do Filtro de Bayes também estão descritos, para mais informações consultar (THRUN; BURGARD; FOX, 2005)

3.1.0.1 Propriedade de Markov

A propriedade de Markov assume que o estado de um processo é função somente do estado anterior e de suas leituras atuais, considerando a variável de estado como a *pose* do robô a equação (4) será simplificada para a forma abaixo:

$$Bel(p(t)) = \sum_i P(p_i(t)|p_i(t-\tau), u_t, z_t) \tag{5}$$

3.1.0.2 Teorema da probabilidade total

O teorema da probabilidade total afirma que a probabilidade de um determinado evento a ocorrer é igual a soma das probabilidades de todos os condicionais possíveis, multiplicados pela probabilidade do condicional. Na equação (6) para um espaço discreto.

$$p(a) = \sum_i p(a|c_i)p(c_i) \tag{6}$$

onde a união de todos os condicionais é o espaço amostral $S = (c_1 \cup c_2 \cup c_3 \ldots)$

3.1.0.3 Teorema de Bayes

O teorema de Bayes relaciona a probabilidade de uma condicional $p(a|d)$ com sua inversa $p(d|a)$. de acordo com a equação (7).

$$p(a|d) = \frac{p(d|a)p(a)}{p(d)} \tag{7}$$

É fundamental para a robótica pois com ela podemos transformar a probabilidade do robô estar em um local a dado que observou uma leitura d dos sensores $p(x|y)$ na probabilidade de observar leitura d supondo que o robô estava no local a: $p(y|x)$.

3.1.1 Filtro de Bayes

Com o uso das propriedades acima e dos modelos cinemático e de medição é possível descrever o algoritmo do Filtro de Bayes que consiste em dois passos: a atualização de controle, associada ao modelo cinemático, e a atualização de medição.

3.1.1.1 Atualização de Controle

Supondo o conhecimento inicial da distribuição de probabilidade das possíveis posições ocupadas pelo robô $Bel(p(t-\tau))$, onde τ é o período de amostragem, a etapa de atualização de controle irá atualizar essa distribuição considerando que o robô realizou algum movimento detectado pelas leituras dos sensores proprioceptivos $u(t)$, movimento este provavelmente causado pelos controladores de movimento do robô ao interagir com os atuadores da roda. A nova distribuição de probabilidade é expressa por:

$$\overline{Bel(p(t))} = \sum_i P(p_i(t)|u_t, z_{t-\tau}, p_i(t-\tau)) * Bel(p_i(t-\tau)) \tag{8}$$

Devido ao erro de medição dos sensores proprioceptivos além das outras fontes de erro já citadas é considerado que a covariância da distribuição de probabilidade tende a aumentar com uma distribuição de probabilidade mais esparsa. Para corrigir esse erro existe a etapa da atualização de medição.

3.1.1.2 Atualização de Medição

Na etapa de atualização de medição os dados dos sensores são comparados com a representação obtida do mapa. O objetivo é obter uma representação no mesmo formato de (5) com a correção do erro adicionado na etapa anterior. A equação (5) será reescrita de forma a poder expressá-la em termos de (8) e da função de probabilidade obtida a partir das leituras dos sensores.

Com a aplicação da propriedade de Bayes à probabilidade obtida dentro da equação (5) temos:

$$
\begin{aligned}
Bel(p(t)) &= \sum_i P(p_i(t)|p_i(t-\tau), u_t, z_t) \\
&= \sum_i P(p_i(t)|z_0, u_0, z_1, u_1, z_2, u_2, z_3, u_3 \ldots) \\
&= \sum_i \frac{P(z_t|p_i(t)) * P(p_i(t)|z_{0:t-\tau}, u_{0:t-\tau})}{P(z_t|z_{0:t-\tau}, u_{0:t})} \\
&= n * \sum_i P(z_t|p_i(t)) * P(p_i(t)|z_{0:t-\tau}, u_{0:t}) \\
&= n * \sum_i P(z_t|p_i(t)) * P(p_i(t)|p_i(t-\tau), u_t, z_{t-\tau}) \\
&= n * \sum_i P(z_t|p_i(t)) * \overline{Bel(p_i(t))}
\end{aligned}
$$

Temos então que:

$$
Bel(p(t)) = n * \sum_i P(z_t|p_i(t)) * \overline{Bel(p_i(t))} \tag{9}
$$

3.1.2 Localização baseada em Filtro de Bayes

Dois dos principais métodos derivados do filtro de Bayes são:

- Localização baseada em Cadeias de Markov
- Localização baseada no Filtro de Kalman Estendido
- Localização baseada em Filtro de Partículas (Monte Carlo)

Para mais informações consultar (THRUN; BURGARD; FOX, 2005),(SIEGWART; NOURBAKHSH; SCARAMUZZA, 2011) e (HUANG; DISSANAYAKE, 2016), onde podem ser estudados outros métodos de Localização além dos citados.

3.2 TIPOS DE MAPAS

Na maioria das aplicações da robótica móvel o robô é configurado para realizar uma tarefa útil à humanidade. A execução dessa tarefa exige que o robô possa representar o ambiente em forma de mapa de forma a se localizar e navegar no ambiente, utilizando como base esse mapa. Neste capítulo serão descritos as duas classes principais de representações para um mapa, mapas métricos e mapas topológicos.

3.2.1 Mapas Métricos

Em um mapa métrico o ambiente é representado em termos de relações geométricas entre os objetos e um *frame* de referência fixo (PANZIERI *et al.*, 2022). Algumas ferramentas utilizadas para criar mapas métricos são o *Slam Toolbox* e o *Elbrus Slam*, implementando respectivamente mapas de grade de ocupação e mapas de características

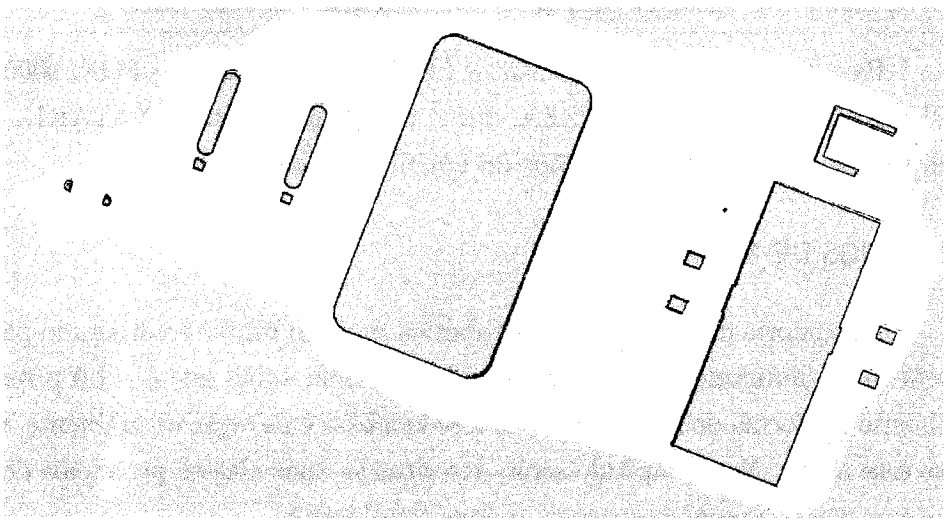
3.2.1.1 Mapa de Grade de Ocupação

É um mapa métrico onde o ambiente é representado por uma matriz de células de forma similar ao de uma foto digital em escala de cinza. Representa o ambiente por meio de pixels, onde cada pixel representa a intensidade luminosa do espaço representado pelo pixel. Porém, no caso de uma mapa de grade de ocupação, o valor de cada célula representa a probabilidade do espaço representado por esta célula estar ocupado ou não.

A principal vantagem da utilização desse tipo de mapa é a possibilidade de utilizá-lo para atividades de navegação onde a complexidade da tarefa de planejamento de trajetória é reduzida com a utilização do mesmo (YOUSIF; BAB-HADIASHAR; HOSEINNEZHAD, 2015), existindo inclusive bibliotecas prontas como a *Navigation 2* que utilizam esse tipo de mapa como base para realizar as atividades de planejamento.

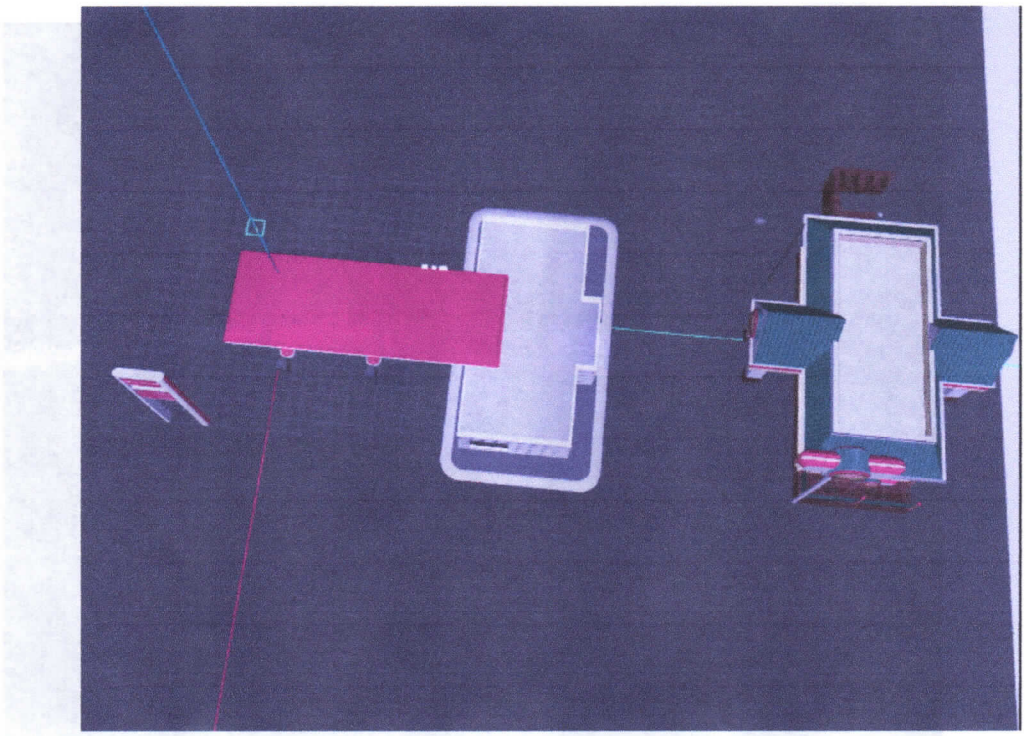
A principal desvantagem de sua utilização se deve ao fato de cada célula representar uma área específica do ambiente e consequentemente a memória utilizada para esse tipo de mapa cresce conforme o tamanho do ambiente, o que pode inviabilizar sua utilização em espaços amplos caso o robô não tenha um hardware robusto. Pacotes como SLAM Toolbox se utilizam de técnicas de LMS e sensores de distância como lidares para criar esse tipo de mapa. Na Figura 4 temos um mapa de grade de ocupação criado no ambiente virtual Gazebo com a utilização do pacote *SLAM Toolbox* e uma Figura do ambiente original utilizado para criar essa imagem.

Figura 4 – Mapa de Grade de Ocupação.



Fonte: Autoria Própria.

Figura 5 – Ambiente mapeado.



Fonte: Autoria Própria.

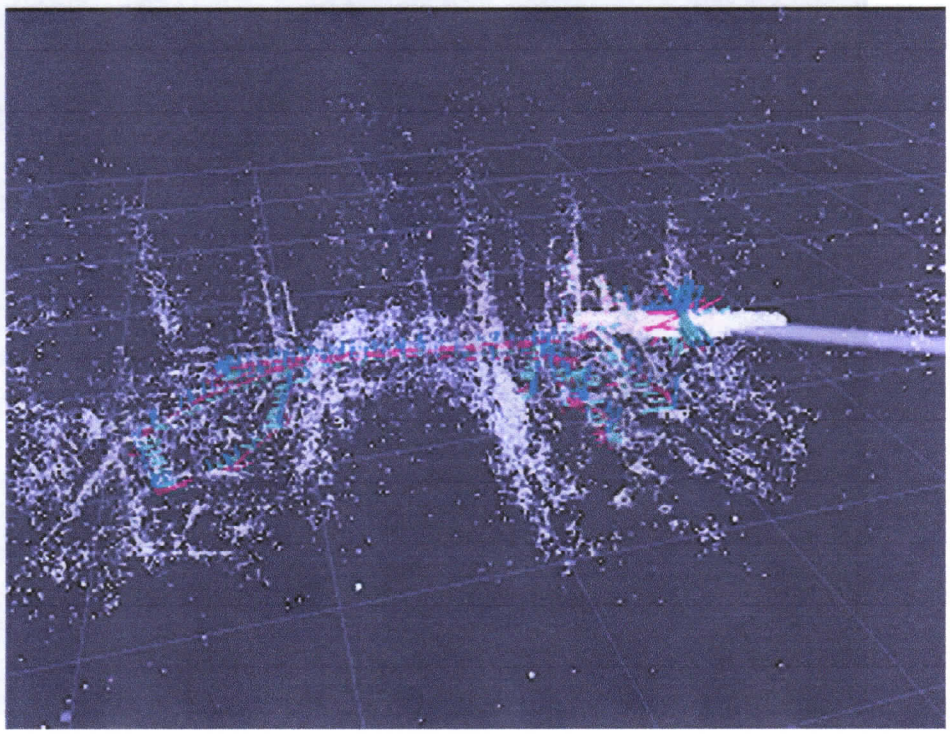
3.2.1.2 Mapa de Características

É um mapa métrico onde os elementos do ambiente são abstraídos e representados por elementos geométricos como pontos e linhas. Cada elemento é armazenado utilizando parâmetros que determinam sua posição no espaço e característica geométrica, como pode ser observado na equação (10) para uma linha no plano bidimensional.

$$l = [r, \theta] \tag{10}$$

Como essa abordagem necessita apenas dos parâmetros utilizados para representar os elementos no ambiente, tende a consumir menos memória do que a abordagem anterior. Na imagem 6 temos a visualização de uma mapa de características criado utilizando o pacote *Elbrus Slam* após o desenvolvimento do robô.

Figura 6 – Mapa de características.



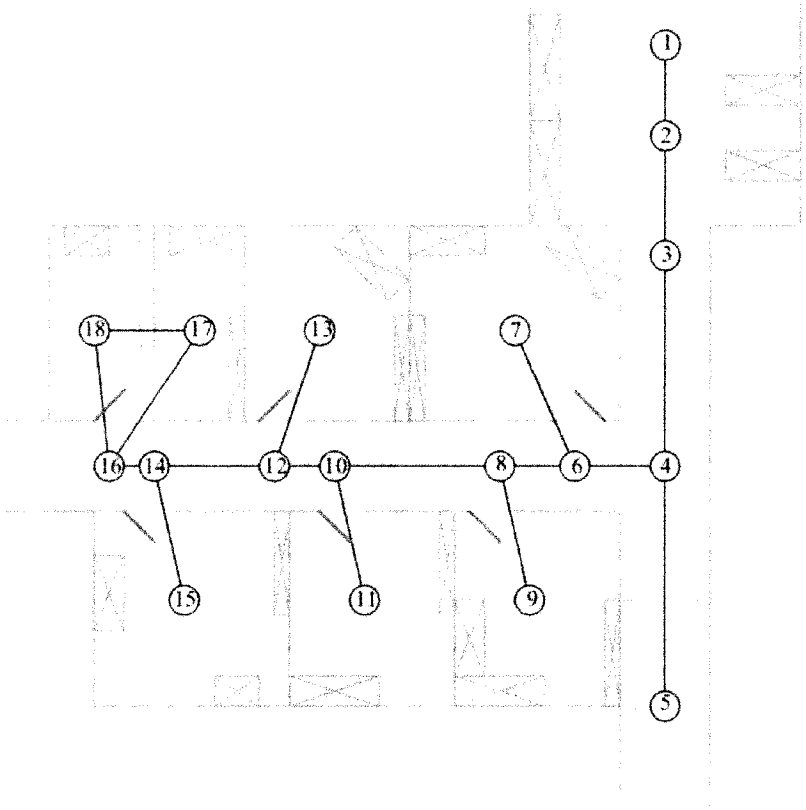
Fonte: Autoria Própria.

3.2.2 Mapas Topológicos

Mapas topológicos são representações do ambiente utilizando um par $\tau = (\mathcal{N}, \epsilon)$ onde $\mathcal{N}$ representa um conjunto de nós e ϵ representa a interconexão entre esses nós. Na imagem 7 temos um exemplo da representação de um ambiente utilizando um mapa topológico.

Mapas topológicos possuem a vantagem de ser uma representação mais compacta que os mapas métricos citados anteriormente, tendo menor consumo de espaço. Em (THRUN, 1998) existe descrição das vantagens e desvantagens de mapas topológicos.

Figura 7 – Mapa Topológico.



Fonte: Fonte: Autonomous Mobile Robots (SIEGWART; NOURBAKHSH; SCARAMUZZA, 2011)

3.3 O PROBLEMA DA LMS

Na subseção de atualização de medição foi visto como uma representação do ambiente em forma de mapa é importante para compará-lo com as leituras dos sensores e atualizar a localização do robô. Também já foram vistos na seção anterior quais são os principais tipos de mapas utilizados na robótica. Essa seção aborda outro problema o de como um mapa do ambiente pode ser criado.

Existem duas formas nas quais um robô pode obter um mapa do ambiente, ele pode receber um mapa de uma fonte externa ou ele pode criar o próprio mapa as técnicas de LMS resolvem o problema de criação do mapa.

As duas principais abordagens para a solução do problema de LMS são as de Filtragem (SAMAN; LOTFY, 2016) e de *Smoothing* (THRUN; MONTEMERLO, 2006), *GMapping* e *HectorSLAM* (KOHLBRECHER *et al.*, 2011) são exemplos de algoritmos populares de LMS que utilizam a abordagem do Filtro de Bayes, *Cartographer*, *KartoSLAM* e *SLAM Toolbox* e *Isaac ROS VISUAL SLAM* são exemplos de algoritmos populares que utilizam a técnica baseada em grafos.

3.3.1 Abordagem por Filtragem

Alguns algoritmos de LMS utilizam o Fitro de Bayes na seção de Localização. A diferença entre as equações 8 e 9 utilizadas respectivamente nas etapas de atualização de medição e atualização de controle para as equações implementadas pelos algoritmos de LMS que utilizam métodos derivados do filtro de Bayes é que o estado a ser estimado não será representado somente pela *pose* do robô p_t, mas pela *pose* do robô e o mapa atual a ser criado $x(t) = (p(t), m)$.

3.3.2 Abordagem por *Smoothing*

A abordagem por *Smoothing* é diferente da abordagem por Filtragem no sentido de que não é desejado estimar apenas o estado atual do robô $x(t) = (p(t), m)$, mas sim estimar a trajetória completa percorrida pelo robô durante o processo de mapeamento, utilizando um conjunto completo de medições realizadas ao longo dessa trajetória.

Uma solução para o processo de *Smoothing* proposta inicialmente por (LU; MILIOS, 1997), consiste em representar as medições e *poses* do robô em um conjunto de nós e restrições entre esses nós chamado *posegraph*, como mostrado nas figuras 8 e 9 na implementação feita no pacote *SLAM Toolboox*.

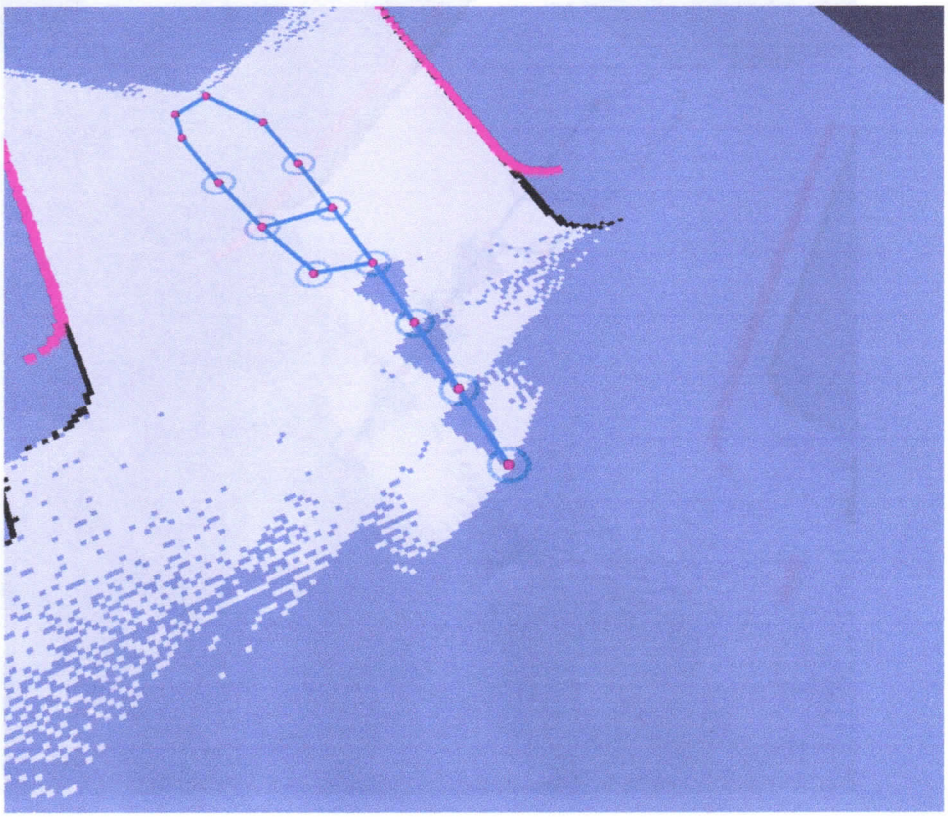
Nas figuras 8 e 9 as esferas vermelhas representam os nós, as retas em azul as restrições entre os nós e os pontos em vermelho as leituras do sensor lidar.

A medida que o robô percorre o ambiente, este utiliza leituras dos sensores que fornecem informação sobre a odometria, tipicamente encoders de roda, de forma a determinar seu deslocamento e calcular a posição estimada do próximo nó a ser adicionado ao grafo. Dessa forma, é possível determinar qual restrição será aplicada entre os dois nós.

Entretanto, devido a imprecisão dos sensores, essa restrição deve ter um certo nível de flexibilidade de forma que seja possível ajustar a distância entre os nós quando a malha é fechada. Durante a criação do mapa, o algoritmo irá comparar a leitura dos sensores com as anteriores e usará o processo de *scan matching* (HESS *et al.*, 2016) para tentar encontrar correspondências. Ao encontrar uma correspondência, irá recalcular todas as restrições anteriores de forma a aproximar a posição dos nós com a correspondência encontrada.

As imagens 10 e 11 foram geradas durante o processo de construção do mapa da Figura 4. A imagem 10 mostra o mapa com algumas imperfeições causadas pelo acúmulo de erro na localização do robô o que leva ao incorreto posicionamento do *posegraph*. A imagem 11 mostra o mapa momentos depois após o robô ter feito diversas outras leituras dos sensores e ter identificado associações entre essas leituras e leituras anteriores sendo feito o fechamento da malha para cada associação e consequentemente o recálculo do *posegraph*

Figura 8 – Conjunto de Nós e visualização da leitura dos sensores armazenada em um dos nós.



Fonte: Autoria Própria.

Figura 10 – Mapa criado com imprecisão.

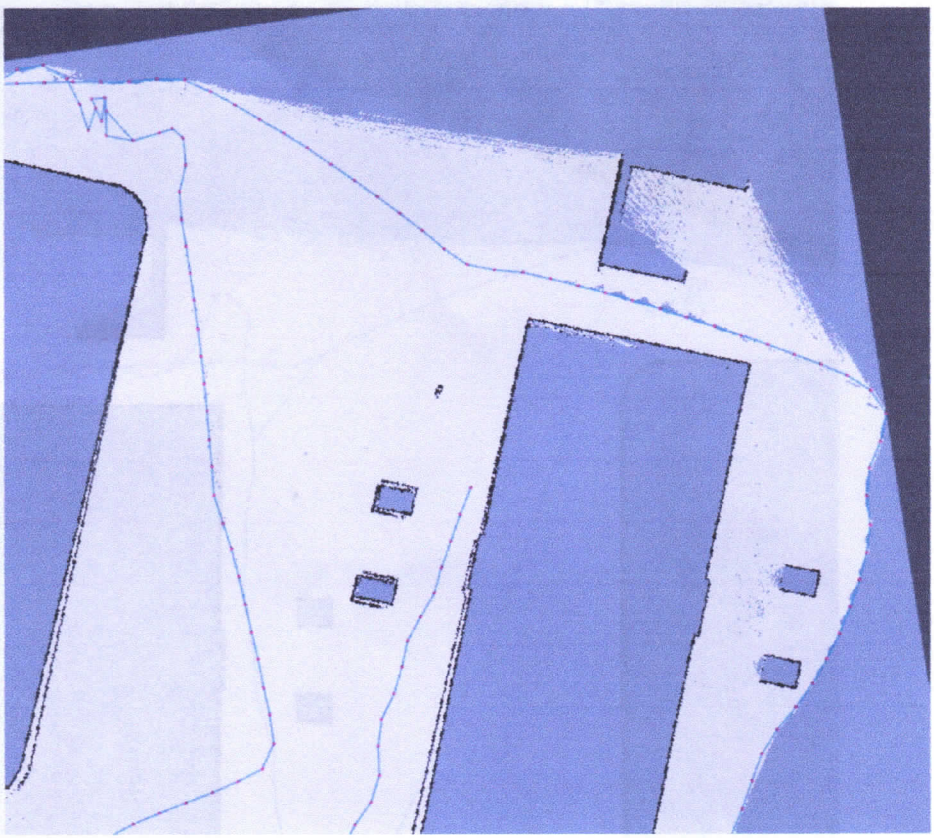
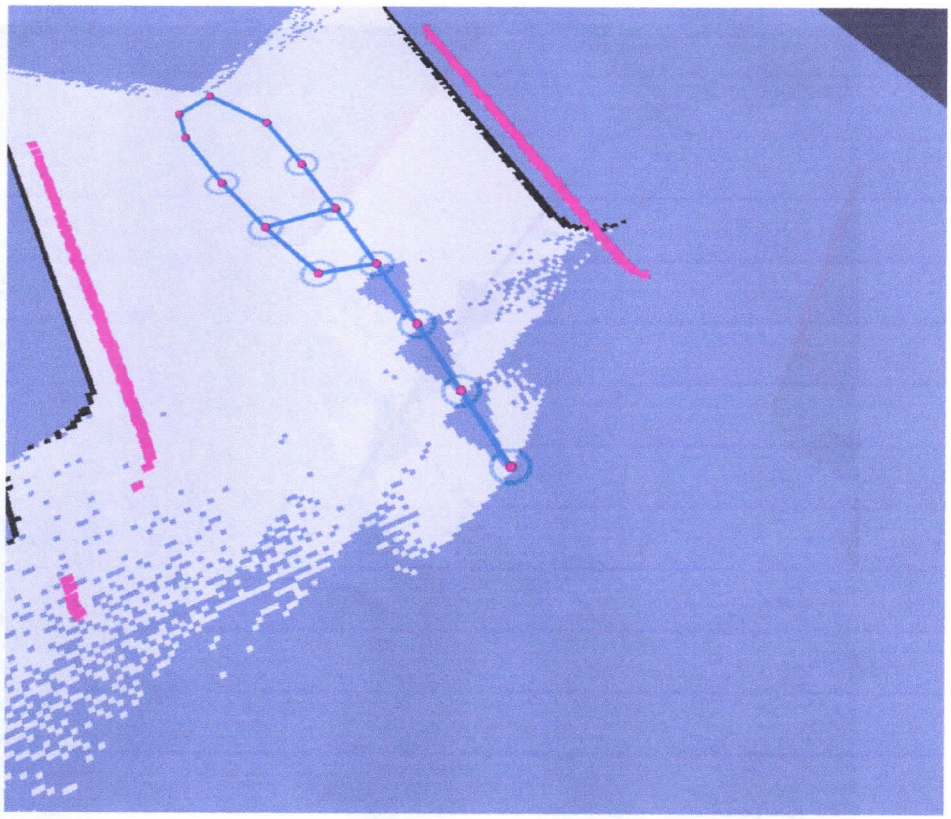
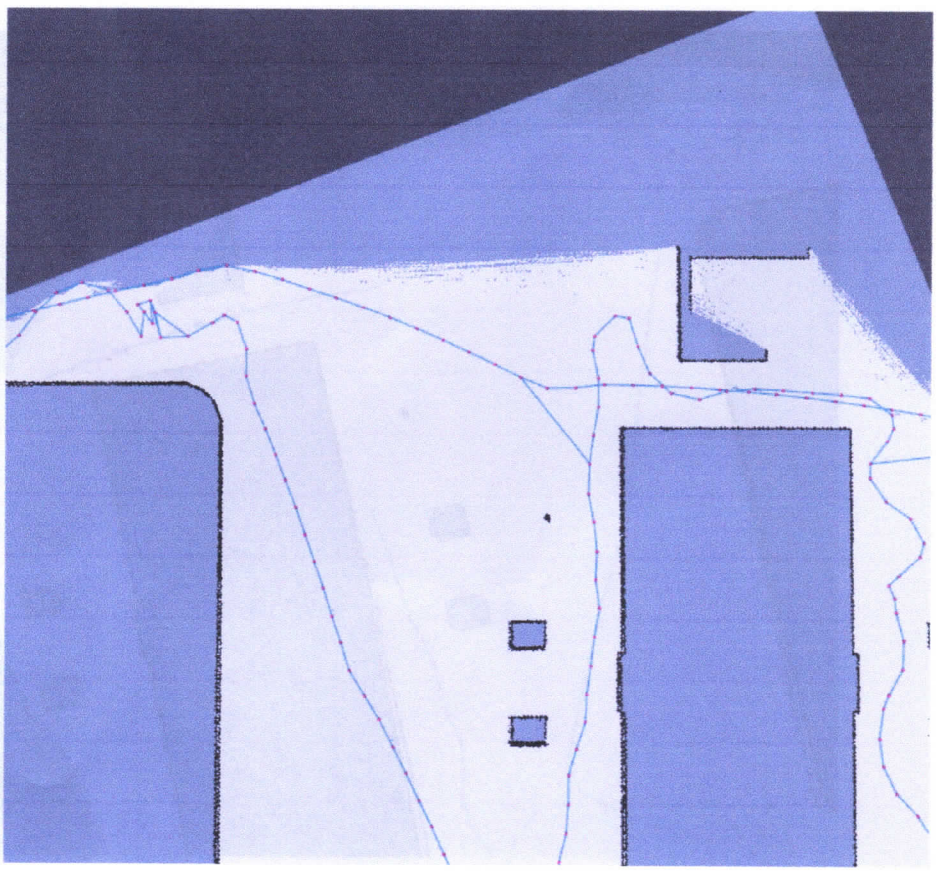


Figura 9 – Conjunto de Nós e visualização da obtida pelo lidar para outro nó.



Fonte: Autoria Própria.

Figura 11 – Mapa após diversas outras observações.

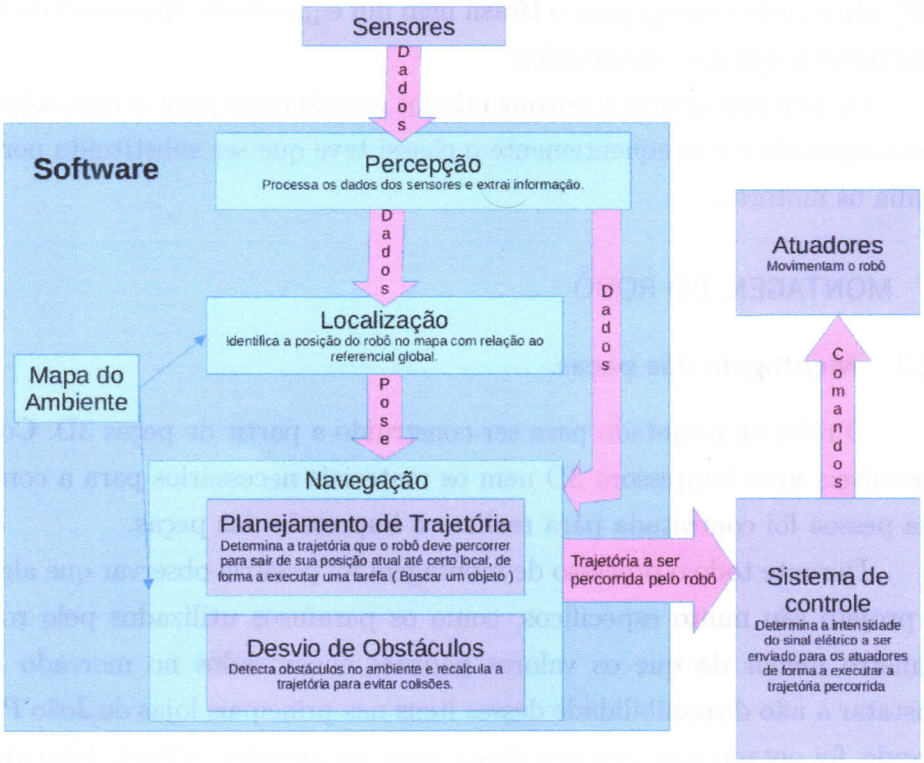


3.4 NAVEGAÇÃO

Navegação é a tarefa principal da robótica móvel e depende do correto funcionamento de todas as técnicas citadas anteriormente além de percepção, controle de movimento e diversas outras. Ao projetar um robô móvel desejamos que este realize alguma tarefa útil a humanidade e para isso é necessária a navegação.

Navegação compreende partir de uma posição inicial conhecida, determinada na etapa de Localização e utilizando um mapa, que geralmente vai ser criado com as técnicas de LMS, determinar um estado final alvo para a posição do robô $p = [x, y, \theta]$ planejar e executar uma trajetória para chegar a posição alvo, além de desviar de obstáculos dinâmicos que podem surgir no ambiente. o problema da navegação é abordado em (SIEGWART; NOURBAKHSH; SCARAMUZZA, 2011). Abaixo temos uma Figura onde estão ilustrados os processos descritos anteriormente e a forma como estes se interconectam para tornar o robô capaz de realizar alguma tarefa.

Figura 12 – Diagrama funcional para um robô móvel.



Fonte: Própria.

4 DESENVOLVIMENTO

Neste capítulo será descrito como uma implementação para solução do problema de LMS foi utilizada e aplicada para um robô real. Para essa implementação o projeto do robô Nanosaur (BONGHI, 2020) será reproduzido com algumas modificações, A técnica de LMS utilizada é a de *Smoothing* com a utilização de um *posegraph* para criar um mapa de características.

O processo de implementação inicia com a pesquisa de preço e montagem do robô e compreende o entendimento do funcionamento e correta configuração do software utilizado e o entendimento geral dos algoritmos usados, descritos na fundamentação teórica.

4.1 PESQUISA DE PREÇO

Para construir os itens listados no quadro 1 foi necessário fazer uma pesquisa de preço com a comparação entre diversos sites na busca de um orçamento que tornasse o projeto realizável, para alguns dos itens na lista de itens recomendados em (BONGHI, 2020) não existia entrega para o Brasil nem um equivalente disponível de forma que alguns itens tiveram que ser substituídos.

Os motores originais encomendados inicialmente para o robô não funcionaram da forma esperada e consequentemente o chassi teve que ser substituído por um modelo que incluía os motores.

4.2 MONTAGEM DO ROBÔ

4.2.1 Montagem das peças.

O robô foi projetado para ser construído a partir de peças 3D. Como não estavam disponíveis uma impressora 3D nem os materiais necessários para a construção do robô, uma pessoa foi contratada para realizar a impressão das peças.

Durante todo o processo de montagem foi possível observar que alguns itens usados no projeto são muito específicos, como os parafusos utilizados pelo robô que tem um diâmetro menor do que os valores padrões encontrados no mercado Brasileiro. Após constatar a não disponibilidade desses itens nas principais lojas de João Pessoa e Campina Grande, foi optado pela procura desses itens em grandes redes de lojas virtuais onde estes foram encontrados.

A primeira tentativa de montar o robô e fazê-lo funcionar fracassou. O robô após a montagem, não era capaz de vencer a inércia e iniciar o movimento. Após diversos testes foi descoberto que o material utilizado para realizar a impressão das rodas tinha um grau de dureza mais elevado do que o utilizado no projeto original, o que aumenta as perdas mecânicas por fricção pois as rodas estão sempre pressionadas e isso pode ter impossibilitado aos motores iniciar o movimento.

Após constatada essa falha na impressão, o material apropriado para imprimir as rodas do robô foi utilizado em uma nova impressão o que resultou em um material muito mais elástico, reduzindo as perdas por fricção. Entretanto, mesmo com a troca do material, ainda havia dificuldade em fazer o robô se mover.

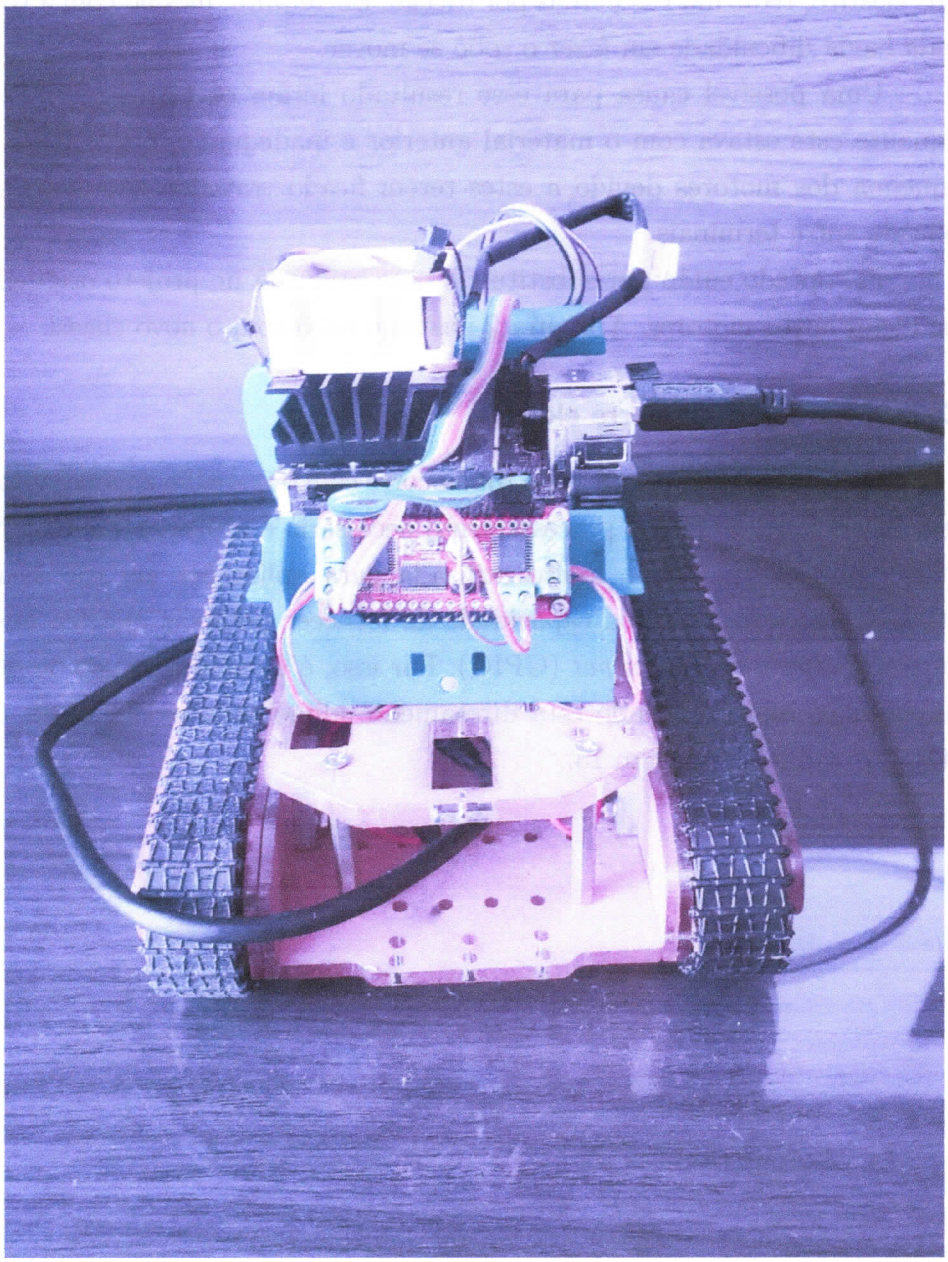
Uma possível causa para esse resultado foram os testes realizados com o robô enquanto este estava com o material anterior e inadequado, o que pode ter ocasionado a queima dos motores devido a estes terem ficado travados enquanto uma tensão era aplicada entre terminais.

Foi optado então por substituir a base utilizada no projeto original por um chassi que inclui novos motores. A Figura 2 ilustra o robô com o novo chassi.

4.2.2 Montagem da parte elétrica.

O correto funcionamento do robô ocorre com o trabalho em conjunto de duas placas eletrônicas. Para realizar o processamento necessário é utilizada a Jetson Nano (NVIDIA, 2022b), que mesmo sendo muito utilizada para aplicações em robótica e inteligência artificial, não tem uma boa capacidade de fornecer potência elétrica nos seus pinos de *General Purpose Input Output* (GPIO). Por isso, é necessário o uso de outra placa para alimentar os motores, sendo esta diretamente alimentada pela bateria. As duas placas estão ilustradas na imagem 13.

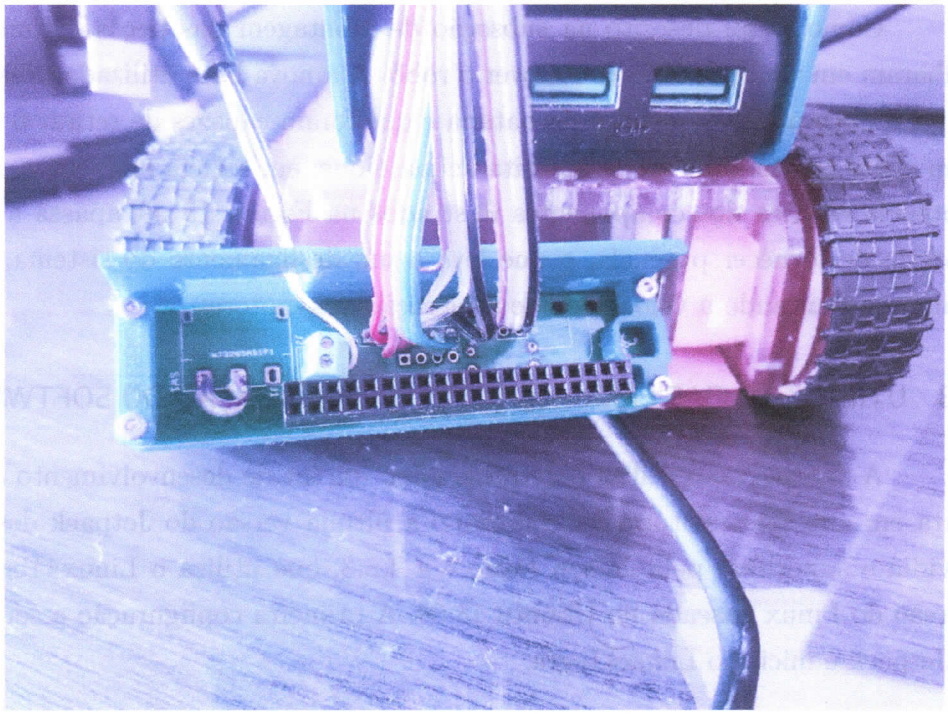
Figura 13 – Robo Nanosaur: Placa vermelha a frente alimenta os motores e Nvidia Jetson realiza o processamento.



Fonte: Autoria Própria.

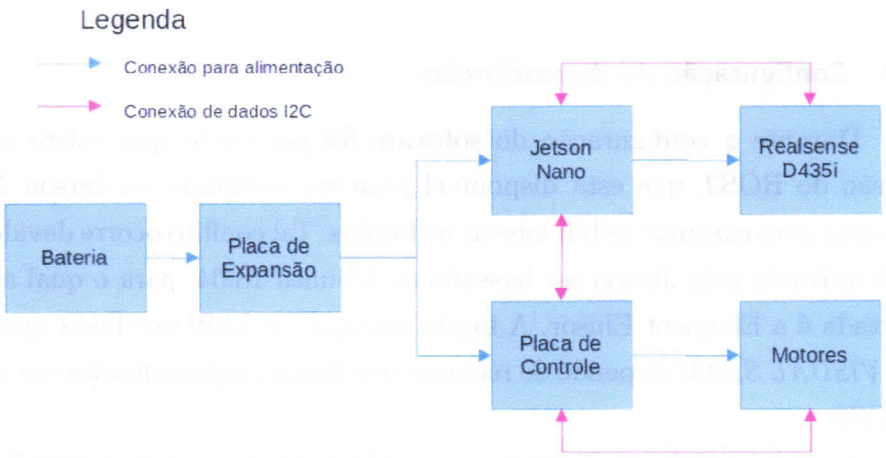
Para que o sistema funcione é necessária a comunicação entre as duas placas utilizando o protocolo *Inter-Integrated Circuit* (I2C) com a devida conexão elétrica entre as portas GPIO da Jetson Nano e as entradas da placa de controle. Essa conexão ocorre com o intermédio de placa de expansão desenvolvida por Raffaelo Bongui e ilustrada na Figura 14. Um conjunto de fios e conectores conectados com solda interligam todos os componentes. Na Figura 15 temos um diagrama de blocos do sistema em funcionamento.

Figura 14 – Placa de expansão: Possibilita a interconexão entre a bateria, a placa de controle e a Jetson Nano.



Fonte: Autoria Própria.

Figura 15 – Diagrama de Blocos dos principais componentes do robô.



Fonte: Autoria Própria.

Os LED's utilizados no projeto original foram removidos devido à instabilidade da conexão I2C com esses componentes. Devido a forma como o software foi desenvolvido,

qualquer falha na conexão com os LED's interrompe toda a comunicação entre o robô e o computador, o que tornou necessário sua remoção.

Como já foi descrito na subseção de montagem das peças, os motores originais falharam em vencer a inércia e mover o robô. A a nova base utilizada possui motores que exigem mais potência elétrica da bateria e que foram capazes de retirar o robô da inércia. Um efeito colateral da troca descrita acima é que, em alguns momentos, a demanda por potência elétrica dos componentes ilustrados na Figura 15 ultrapassa a capacidade da bateria de fornecer potência, o que leva a um desligamento do sistema. Esse efeito foi mitigado limitando a velocidade de aceleração dos motores.

4.3 UTILIZAÇÃO DAS FERRAMENTAS E CONFIGURAÇÃO DO SOFTWARE

A Jetson Nano utiliza o Jetpack como um kit de desenvolvimento de software. Na data em que este relatório foi elaborado a última versão do Jetpack desenvolvida pela Nvidia e suportada pela Jetson Nano é a 4.6.3, que utiliza o Linux4Tegra 32.7.3, uma versão do Linux baseada no Ubuntu 18.04. A primeira configuração a ser feita é instalar o Jetpack e iniciar o Linux4Tegra.

4.3.1 Comunicação com placa

Para poder iniciar a configurar o ambiente do Jetpack 4.6.3, inicialmente é necessário estabelecer alguma forma de comunicação com a placa. Os métodos escolhidos foram usar a comunicação por meio do ssh e de aplicativos como *Teamviewer* a depender de qual tarefa estava sendo executada no momento.

4.3.2 Configuração de dependências

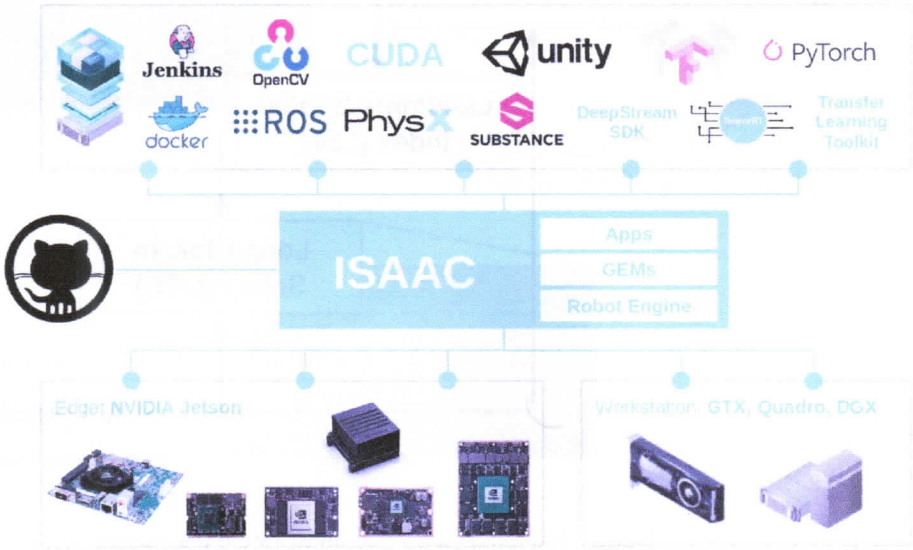
Durante a configuração do software foi percebido que existe um conflito entre a versão do ROS2, que está disponível para ser instalada na Jetson Nano, e a versão necessária para executar as bibliotecas utilizadas. Tal conflito ocorre devido ao Linux4Tegra 32.7.3 utilizado pela Jetson ser baseado no Ubuntu 18.04, para o qual a versão do ROS2 suportada é a Eloquent Elusor. A implementação de LMS escolhida que utiliza o *ISAAC ROS VISUAL SLAM* depende de recursos que foram implementados em versões superiores do ROS2.

A solução utilizada é criar um contêiner com todas as dependências necessárias à execução do algoritmo de LMS implementado no ISAAC ROS VISUAL SLAM. Os contêineres utilizados já estavam disponíveis no projeto original, mas foi preciso fazer algumas modificações para remover o uso dos LEDs do código principal. O entendimento sobre o funcionamento do Docker foi necessário tanto para modificar o código como para solucionar possíveis erros e executar algumas funcionalidades, como para entender o que estava acontecendo.

4.4 TÉCNICA DE LMS

O conjunto de pacotes e ferramentas *Isaac ROS* fornecido pela Nvidia para alguns de seus dispositivos será utilizado. Abaixo temos uma imagem do ecossistema do *Isaac ROS*.

Figura 16 – Ecossistema do ISAAC ROS.



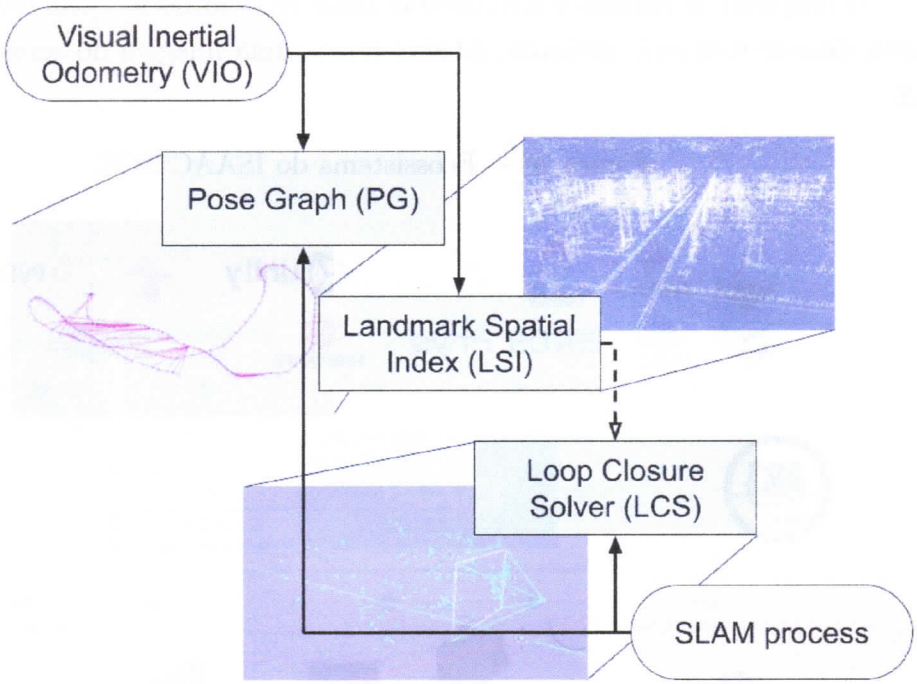
Fonte: Documentação Isac ROS (NVIDIA, 2018).

Das ferramentas fornecidas pelo Isaac ROS, o pacote *Isaac ROS VISUAL SLAM* será utilizado. Esse pacote requer uma câmera que tenha visão estéreo para realizar o processo de construção do mapa, portanto a câmera Realsense D435i será utilizada.

Como foi visto na fundamentação teórica, o processo de LMS utilizando *posegraph* utiliza algum sensor para fornecer a informação de odometria e assim fazer uma estimativa inicial para a posição dos nós e suas restrições. No pacote *Isaac ROS visual SLAM*, o sensor utilizado para calcular a odometria é a própria câmera. A odometria é calculada por padrão usando as imagens estéreo da câmera e utilizando odometria visual (AQEL *et al.*, 2016) para estimar o deslocamento realizado pelo robô a partir das imagens da câmera.

Em ambientes em que a qualidade da odometria visual pode ser comprometida, como baixa luminosidade ou ambientes homogêneos o algoritmo muda a forma de calcular a odometria, usando o *Inertial Measurement Unit* (IMU) da realsense com a integração das leituras desse sensor para obter uma leitura de deslocamento. Na Figura 17 temos um diagrama com a descrição do processo de LMS implementado nesse pacote.

Figura 17 – Funcionamento Isaac ROS VISUAL SLAM.



Fonte: Documentação Isac ROS (NVIDIA, 2018).

4.5 RESULTADOS DE SIMULAÇÃO

O algoritmo implementado no Pacote *Isaac Ros Visual SLAM* foi testado em ambiente real. Para testar a qualidade de sua implementação, foi criado um ambiente semelhante ao ambiente real ilustrado na Figura 18. Como não havia um modelo de simulação que incluísse tanto a realsense como o IMU Foi usado como sensor apenas a câmera estéreo da Realsense.

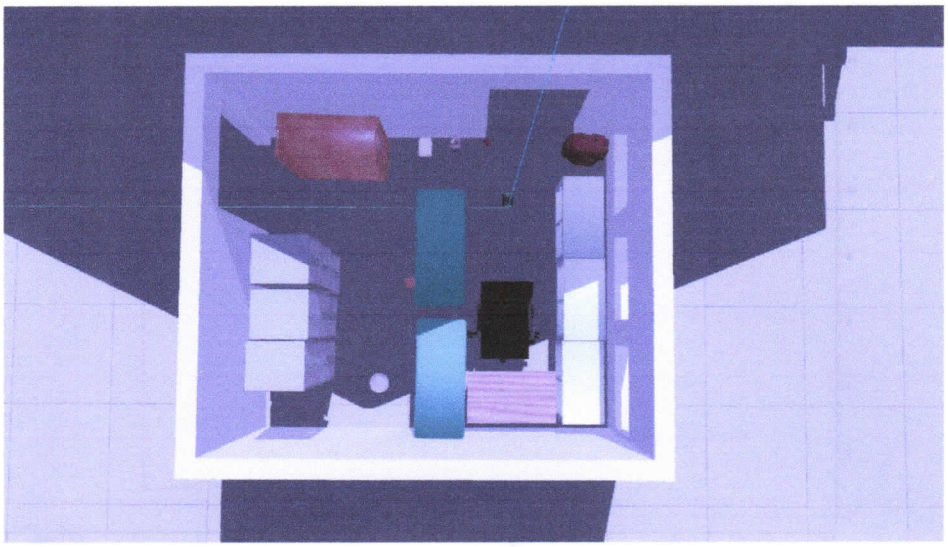
A pose do robô foi comparada com o *Ground Truth* retornado pelo Gazebo resultando na Figura 19 e em um erro ilustrado na Figura 20 .

Como pode ser visto no gráfico de erros ilustrado na Figura20,em determinado momento o erro de localização cresce indefinidamente. A conclusão é que o sensor IMU é necessário para possibilitar a correta localização para o robô.

4.6 RESULTADOS ROBÔ REAL

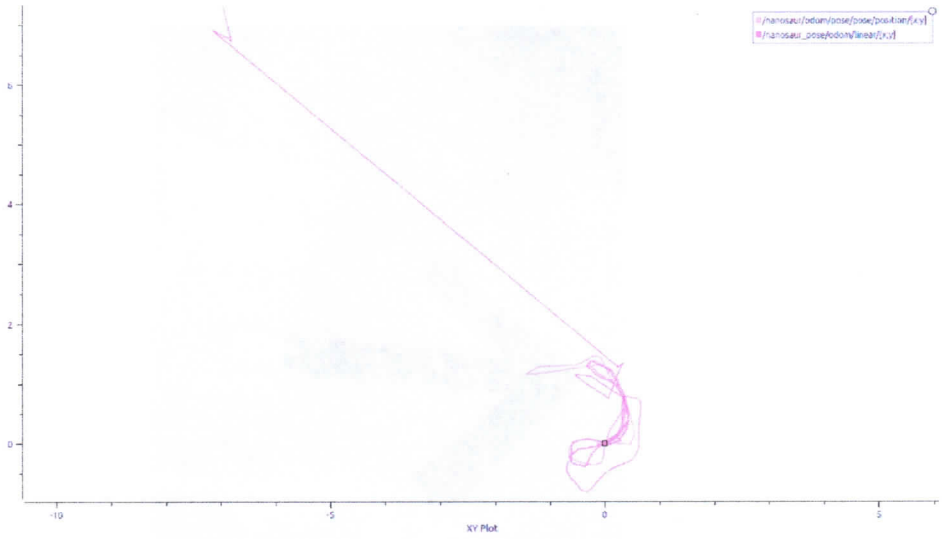
Após a correta configuração das dependências, da correta montagem e configuração do robô, o desempenho da técnica de LMS escolhida foi testado com a realização de dois experimentos descritos abaixo

Figura 18 – Ambiente de Simulação do robô.



Fonte: Autoria Própria.

Figura 19 – *ground truth* em laranja e localização fornecida pelo pacote em rosa.

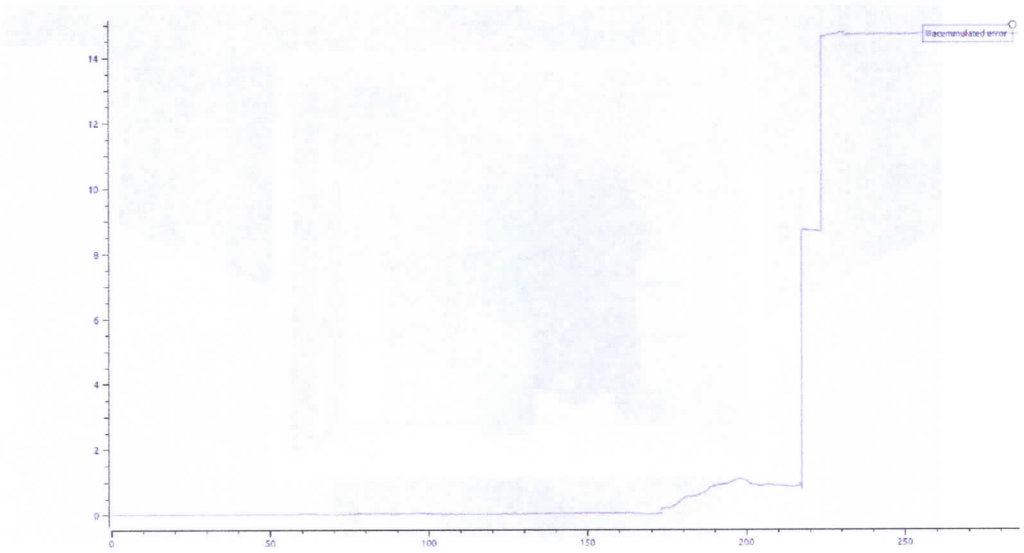


Fonte: Autoria Própria.

4.6.1 Experimento 1

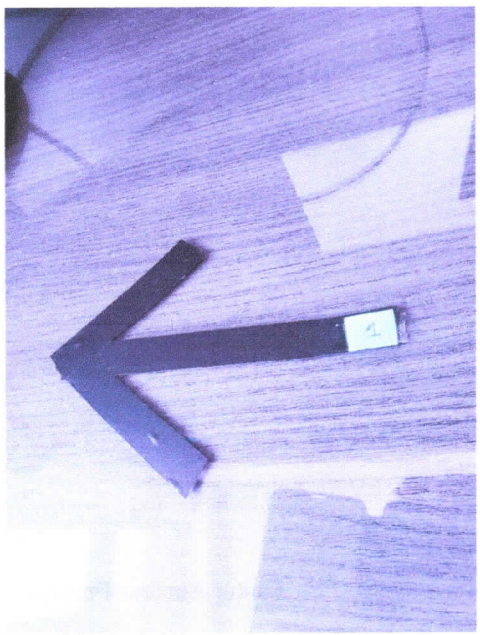
O primeiro experimento consiste em deslocar o robô manualmente de forma a movimentá-lo nas 3 dimensões do espaço por uma área de aproximadamente 0.25 metros quadrados. O local onde o robô se encontra quando o processo de LMS é iniciado foi demarcado por fita isolante, como ilustrado na Figura 21.

Figura 20 – Erro de posição.



Fonte: Autoria Própria.

Figura 21 – Demarcação do ponto onde o processo de LMS para o experimento 1 inicia.

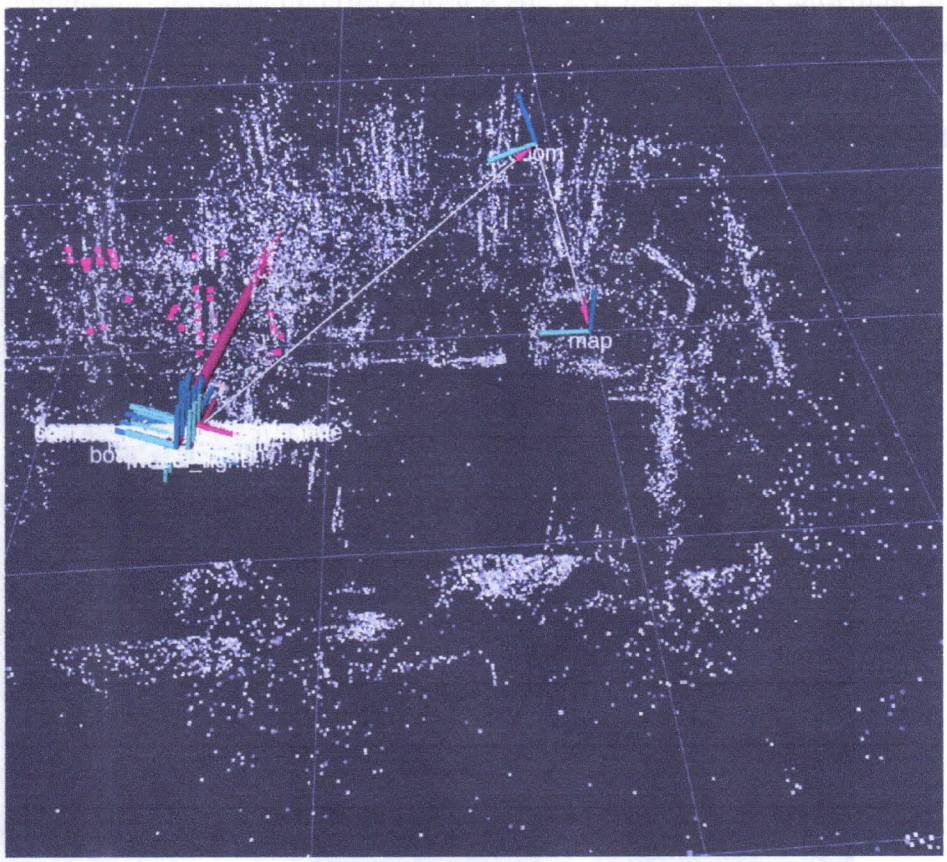


Fonte: Autoria Própria.

Devido a forma como o algoritmo foi desenvolvido o *frame* de referência global *map* descrito anteriormente na fundamentação teórica é definido por padrão como sendo o ponto onde o processo de LMS é iniciado. Dessa forma, a transformação linear entre os sistemas de coordenadas *map* e *base link* representa o deslocamento do robô com relação ao ponto inicial ilustrado na Figura 21.

Portanto, uma forma de testar a eficiência do processo de LMS é retornar o robô ao ponto inicial demarcado na Figura 21 e testar se os sistemas de coordenadas se encontram alinhados. Com esse objetivo, ao fim do experimento o robô foi retornado ao ponto inicial. A imagem 22 ilustra os resultados obtidos.

Figura 22 – Mapa encontrado e coordenadas map odom e base link ao final do experimento.



Fonte: Autoria Própria.

Na imagem 22 o deslocamento entre os sistemas de coordenadas *map odom* e *base link* é representado por seta amarela, os nomes *map* e *base link* de seus respectivos sistemas é visível e o nome base link não é visível devido a sua ocultação pelos nomes de outros sistemas de coordenadas auxiliares do robô.

O resultado obtido difere do esperado para o funcionamento apropriado, pois caso a localização estivesse funcionando corretamente, os sistemas base link e map deveriam estar alinhados quando o robô retorna ao ponto inicial. A partir desse resultado é possível concluir que o processo de LMS para esse robô não funciona de forma apropriada quando existe um deslocamento nas 3 dimensões.

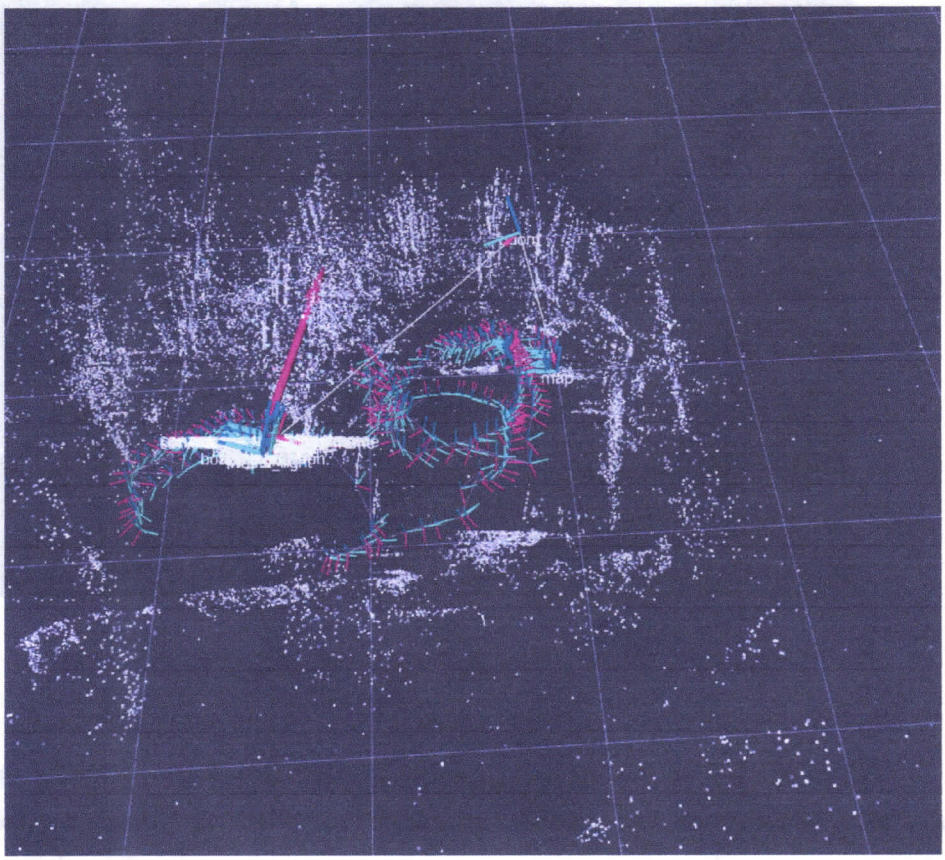
A possível causa desse resultado é que o processo de odometria visual não têm informação suficiente para determinar seu deslocamento com precisão uma vez que a

câmera irá gravar imagens que estão a frente do robô e não obtém informação sobre os elementos que estão acima do robô na coordenada z do frame de referência *map*.

Dessa forma, para estimar o deslocamento da coordenada z, o robô têm apenas a informação do sensor IMU presente na câmera. Com o processo de integração das leituras do IMU, há o acúmulo do erro nessa coordenada, o erro causa a criação de um mapa incorreto, o que pode fazer com que as associações encontradas por meio do processo de *scan matching* entre as leituras dos sensores estejam erradas, causando a propagação do erro para outras coordenadas e a falha do processo de LMS.

Na imagem 23 temos a informação contida na Figura 22 com a adição de mais uma informação: os nós encontrados durante o processo de LMS pelo método de *Smothing*.

Figura 23 – Mapa encontrado,coordenadas *map odom* e *base link* e trajetória.



Fonte: Autoria Própria.

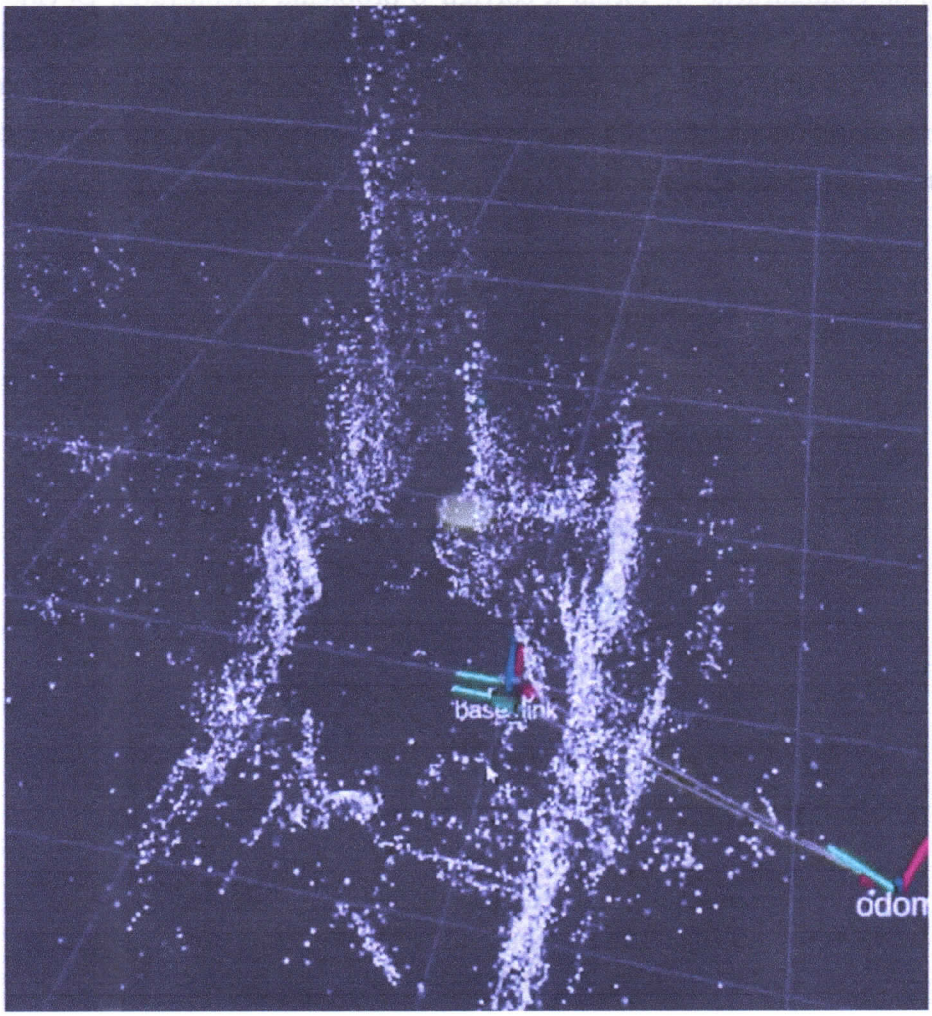
4.6.2 Experimento 2

O experimento 2 consistiu em deslocar o robô em duas dimensões, enviando comandos de velocidade por meio da estrutura do ROS e fazendo-o percorrer o ambiente enquanto executa a técnica de LMS. Da mesma forma como ocorreu no Experimento 1 o local onde o robô iniciar a mapear o ambiente foi demarcado, ao final do experimento o

robô foi retornado a esse local e foi verificado se os sistemas de coordenadas *map* e *base link* se encontravam alinhados.

Na imagem 24 extraída ao final do experimento os sistemas de coordenadas *map base link* e *odom*, além do mapa criado, podem ser visualizados. É possível perceber que os sistemas de coordenadas *base link* e *odom* se encontram muito próximos, podendo a pequena diferença encontrada entre estes ser justificada pelo fato de, mesmo com a marcação, ser difícil voltar exatamente a mesma posição inicial definida e também devido ao erro acumulado durante a trajetória.

Figura 24 – Mapa encontrado e coordenadas *map odom* e *base link* ao final do experimento.



Fonte: Autoria Própria.

4.6.3 O mapa como uma forma de localizar o robô

Após a criação de um mapa, a próxima etapa a ser realizada é a de salvar esse mapa e utilizá-lo para realizar a localização do robô. O pacote Isaac ROS Visual SLAM

utilizado disponibiliza uma ação no ROS para realizar o armazenamento do mapa e outra para carregar esse mapa e iniciar o processo de localização do robô.

Após a criação dos mapas ilustrados nas figuras 23 24 a ação de armazenamento do mapa foi executada com a seleção do contêiner *nanosaur_perception_1* responsável pela execução do algoritmo de LMS e a execução da ação de armazenamento do mapa, utilizando a interface apropriada.

Após salvar o mapa dentro do contêiner, foi constatado que um arquivo *dwg* era criado dentro do mesmo e que esse processo interrompia a execução do conteiner logo após o sarmazenamento do mapa e tornava o contêiner *nanosaur_perception_1* inutilizável

Uma possível solução é refazer o contêiner com a mudança de algum dos parâmetros de configuração de forma a corrigir o problema encontrado. Devido as dificuldades encontradas, descritas na próxima seção do trabalho, que reduziram significativamente o tempo disponível para o projeto, não foi possível corrigir o problema encontrado. Portanto, não serão apresentadas análises quantitativas sobre a precisão do mapa criado, esse tópico é então um possível trabalho futuro a ser realizado.

5 CONCLUSÃO

O desenvolvimento desse projeto teve como motivação principal obter conhecimento sobre o processo de LMS. Parte desse objetivo foi alcançado ao pesquisar sobre o processo de LMS e as principais técnicas utilizadas na literatura existente.

A partir dessa pesquisa, parte dos objetivos do trabalho foram alcançados e também foi possível elaborar a fundamentação teórica onde o funcionamento das técnicas de LMS é explicado, além de serem mencionadas as principais técnicas de LMS.

A outra parte dos objetivos foi alcançada com a aplicação prática da técnicas de LMS por *Smoothing* utilizando *posegraphs*, essa técnica foi aplicada em ambiente de simulação e em ambiente real utilizando o pacote Isaac ROS Visual Slam como foi descrito durante o desenvolvimento.

Outro simulador que poderia ter sido usado para testar as técnicas de LMS é o Isaac Sim (NVIDIA, 2019), desenvolvido pela Nvidia, a mesma empresa que fabrica a placa Jetson Nano. Até o momento da publicação deste trabalho, a placa gráfica com menor poder de processamento que têm capacidade de executar esse simulador é a GeForce RTX 2070 muito superior a placa gráfica disponível para realizar o projeto. De acordo com (NVIDIA, 2022a) o poder de processamento da *Graphical Processing Unit* (GPU) considerada mínima necessária é de 7.5, enquanto o poder de processamento da GPU disponível é de 5.0. Portanto, devido as limitações de Hardware, não foram feitas simulações com o Isaac Sim da Nvidia.

Durante o desenvolvimento da etapa prática do projeto foram encontradas dificuldades que podem ser agrupadas em dois tipos diferentes. O primeiro tipo de dificuldade compreende o aprendizado das ferramentas descritas na seção de Métodos e Ferramentas, sendo dedicados um certo número de horas a transpor essa barreira.

O segundo tipo de dificuldade compreende limitações no hardware utilizado, uma vez que nem todos os itens listados no projeto original possuíam entrega para o Brasil e alguns dos itens com a entrega não chegaram com a qualidade esperada como foi descrito em mais detalhes na seção de desenvolvimento.

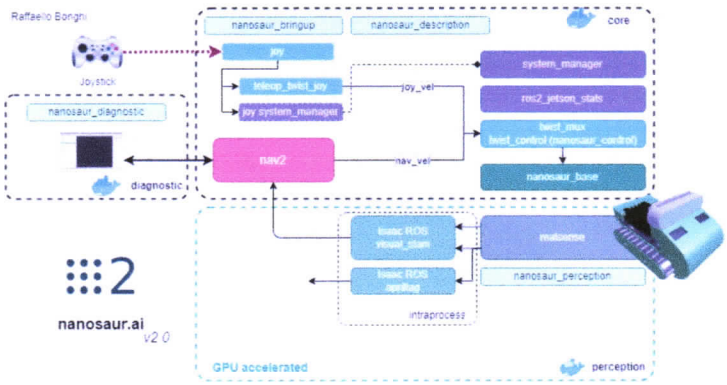
5.1 TRABALHO FUTURO

Após a criação do mapa do ambiente as próximas etapas que tornariam esse robô autônomo são a utilização desse mapa para se localizar e navegar o ambiente, com esse objetivo, os próximos trabalhos que podem ser desenvolvidos são:

- Pesquisar quais são os serviços de armazenamento em nuvem disponíveis e utilizar computação em nuvem para simular técnicas de LMS com o Isaac Sim.
- Integrar o robô com bibliotecas de Navegação como a Navigation2, como mostrado na figura 25 abaixo o próprio projetista do robô Nanosaur têm como objetivo realizar

essa etapa, mas até a data da publicação desse trabalho essa etapa não havia sido implementada.

Figura 25 – Projeto Nanosaur.



Fonte: (BONGHI, 2020).

- Após completar a etapa anterior, outro possível trabalho a ser desenvolvido é o de definir uma trajetória a ser percorrida pelo robô com o uso da Navigation2 enquanto a precisão da Localização utilizando o mapa gerado é testada.

REFERÊNCIAS

AQEL, Mohammad *et al.* Review of visual odometry: types, approaches, challenges, and applications. **SpringerPlus**, v. 5, dez. 2016. DOI: 10.1186/s40064-016-3573-7.

BONGHI, Raffaello. **Nanosaur: The smallest NVIDIA Jetson dinosaur robot.** 2020. Disponível em: https://nanosaur.ai/. Acesso em: 30 dez. 2022.

GERKEY, Brian. **Why ROS2?** 2014. Disponível em: https://design.ros2.org/articles/why_ros2.html. Acesso em: 15 jan. 2023.

HESS, Wolfgang *et al.* Real-time loop closure in 2D LIDAR SLAM. *In:* p. 1271–1278. DOI: 10.1109/ICRA.2016.7487258.

HUANG, Shoudong; DISSANAYAKE, Gamini. Robot Localization: An Introduction. *In:* WILEY Encyclopedia of Electrical and Electronics Engineering. [*S.l.*]: John Wiley & Sons, Ltd, 2016. P. 1–10. ISBN 9780471346081. DOI: https://doi.org/10.1002/047134608X.W8318. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/047134608X.W8318. Disponível em: https://onlinelibrary.wiley.com/doi/abs/10.1002/047134608X.W8318.

INFRASTRUCTURE, ROS. **ROS Enhanced Proposal.** 2016. Disponível em: https://github.com/ros-infrastructure/rep/blob/master/rep-0105.rst. Acesso em: 27 nov. 2022.

KOHLBRECHER, S. *et al.* A Flexible and Scalable SLAM System with Full 3D Motion Estimation. *In:* IEEE. PROC. IEEE International Symposium on Safety, Security and Rescue Robotics (SSRR). [*S.l.*: *s.n.*], nov. 2011.

LU, Feng; MILIOS, Evangelos. Globally Consistent Range Scan Alignment for Environment Mapping. **Auton. Robots**, v. 4, p. 333–349, out. 1997. DOI: 10.1023/A:1008854305733.

NVIDIA. **Cuda GPUS**. Disponível em: https://developer.nvidia.com/cuda-gpus. Acesso em: 23 jan. 2022.

_____. **Documentação Isaac ROS**. 2018. Disponível em: https://docs.nvidia.com/isaac/doc/overview.html. Acesso em: 8 jan. 2022.

_____. **Isaac Sim documentation**. 2019. Disponível em: https://docs.omniverse.nvidia.com/app_isaacsim/app_isaacsim/requirements.html. Acesso em: 23 jan. 2022.

NVIDIA. **Kit de desenvolvimento Jetson Nano**. 2022. Disponível em: https://developer.nvidia.com/embedded/jetson-nano-developer-kit. Acesso em: 8 jan. 2022.

PANZIERI, Stefano *et al.* A low cost vision based localization system for mobile robots, dez. 2022.

SAMAN, Abu Bakar Sayuti H M; LOTFY, Ahmed Hesham. An implementation of SLAM with extended Kalman filter. *In:* 2016 6th International Conference on Intelligent and Advanced Systems (ICIAS). [*S.l.*: *s.n.*], 2016. P. 1–4. DOI: 10.1109/ICIAS.2016.7824105.

SIEGWART, Roland; NOURBAKHSH, Illah R.; SCARAMUZZA, Davide. **Introduction to Autonomous Mobile Robots**. 2nd. [*S.l.*]: The MIT Press, 2011. ISBN 0262015358.

THRUN, S. Learning Metric-Topological Maps for Indoor Mobile Robot Navigation. **Artificial Intelligence**, v. 99, n. 1, p. 21–71, 1998.

THRUN, Sebastian; BURGARD, Wolfram; FOX, Dieter. **Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)**. [*S.l.*]: The MIT Press, 2005. ISBN 0262201623.

THRUN, Sebastian; MONTEMERLO, Michael. The Graph SLAM Algorithm with Applications to Large-Scale Mapping of Urban Structures. **I. J. Robotic Res.**, v. 25, p. 403–429, mai. 2006. DOI: 10.1177/0278364906065387.

YOUSIF, Khalid; BAB-HADIASHAR, Alireza; HOSEINNEZHAD, Reza. An Overview to Visual Odometry and Visual SLAM: Applications to Mobile Robotics. **Intelligent Industrial Systems**, v. 1, nov. 2015. DOI: 10.1007/s40903-015-0032-7.