UNIVERSIDADE FEDERAL DE GOIÁS
ESCOLA DE ENGENHARIA ELÉTRICA, MECÂNICA E DE COMPUTAÇÃO

FELLIPE ADORNO CLAUDINO DA COSTA

Avaliação de Algoritmos de Localização e Mapeamento Simultâneos Aplicada a Competição *Robocup@home*

GOIÂNIA
2019

EMC

TERMO DE CIÊNCIA E DE AUTORIZAÇÃO PARA DISPONIBILIZAR VERSÕES ELETRÔNICAS DE TRABALHO DE CONCLUSÃO DE CURSO DE GRADUAÇÃO NO REPOSITÓRIO INSTITUCIONAL DA UFG

Na qualidade de titular dos direitos de autor, autorizo a Universidade Federal de Goiás (UFG) a disponibilizar, gratuitamente, por meio do Repositório Institucional (RI/UFG), regulamentado pela Resolução CEPEC nº 1204/2014, sem ressarcimento dos direitos autorais, de acordo com a Lei nº 9610/98, o documento conforme permissões assinaladas abaixo, para fins de leitura, impressão e/ou *download*, a título de divulgação da produção científica brasileira, a partir desta data.

1. Identificação do Trabalho de Conclusão de Curso de Graduação (TCCG):

Nome completo do autor: Fellipe Adorno Claudino da Costa

Título do trabalho: Avaliação de Algoritmos de Localização e Mapeamento Simultâneos Aplicada a Competição Robocup@home

2. Informações de acesso ao documento:

Concorda com a liberação total do documento [x] SIM [ ] NÃO[1]

Havendo concordância com a disponibilização eletrônica, torna-se imprescindível o envio do(s) arquivo(s) em formato digital PDF do TCCG.

Fellipe Adorno Claudino da Costa
(Nome completo do autor)[2]

Ciente e de acordo:

Marco Antonio Assfalk de Oliveira
(Nome completo do orientador)[2]

Data: 2019 / 12 / 13

(Marco Antonio Assfalk de Oliveira)

[1] Neste caso o documento será embargado por até um ano a partir da data de defesa. A extensão deste prazo suscita justificativa junto à coordenação do curso. Os dados do documento não serão disponibilizados durante o período de embargo.

Casos de embargo:

- Solicitação de registro de patente;
- Submissão de artigo em revista científica;
- Publicação como capítulo de livro;
- Publicação da dissertação/tese em livro.

[2] As assinaturas devem ser originais sendo assinadas no próprio documento, imagens coladas não serão aceitas.

FELLIPE ADORNO CLAUDINO DA COSTA

Avaliação de Algoritmos de Localização e Mapeamento Simultâneos Aplicada a Competição *Robocup@home*

Trabalho de Conclusão de Curso submetido à Universidade Federal de Goiás, como requisito necessário para obtenção do grau de Bacharel em Engenharia Elétrica

Goiânia, julho de 2019

Ficha de identificação da obra elaborada pelo autor, através do
Programa de Geração Automática do Sistema de Bibliotecas da UFG.

Adorno Claudino da Costa, Fellipe
Avaliação de Algoritmos de Localização e Mapeamento Simultâneos Aplicada a Competição Robocup@home [manuscrito] / Fellipe Adorno Claudino da Costa. - 2019.
liii, 53 f.: il.

Orientador: Prof. Dr. Marco Antonio Assfalk de Oliveira.
Trabalho de Conclusão de Curso (Graduação) - Universidade Federal de Goiás, Escola de Engenharia Elétrica, Mecânica e de Computação (EMC), Engenharia Elétrica, Goiânia, 2019.
Bibliografia.
Inclui mapas, gráfico, tabelas, algoritmos, lista de figuras, lista de tabelas.

1. localização. 2. mapeamento. 3. SLAM. 4. gmapping. 5. vinySLAM. I. Antonio Assfalk de Oliveira, Marco, orient. II. Título.

CDU 621.03

UNIVERSIDADE FEDERAL DE GOIÁS

FELLIPE ADORNO CLAUDINO DA COSTA

Esta Monografia foi julgada adequada para a obtenção do título de Bacharel em Engenharia Elétrica, sendo aprovada em sua forma final pela banca examinadora:

Orientador(a): Prof. Dr. Marco Antonio Assfalk de Oliveira
Universidade Federal de Goiás - UFG

Ms. Lucas da Silva Assis
Universidade Federal de Goiás - UFG

Prof. Dr. Cássio Dener Noronha Vinhal
Universidade Federal de Goiás - UFG

Goiânia, 13 de dezembro de 2019

Resumo

A atual monografia trata da aplicação e comparação de soluções de Localização e Mapeamento Simultâneos para robôs assistentes da categoria *Open Challenge Robocup@home* na competição Latino Americana de Robótica. Para tal foi utilizado o robô Zordon do grupo de robótica Pequi Mecânico da Universidade Federal de Goiás: um autômato capaz de locomover sobre o piso de uma casa, detectar pessoas, interagir com elas e manipular objetos. Nesse sentido, buscou-se avaliar se o algoritmo *vinySLAM* sobressai frente a atual abordagem de mapeamento por SLAM utilizada pelo Pequi Mecânico, *gmapping*. Para tanto, foi proposto um *ground-truth* de trajetória, criado a partir de dados coletados de sensores como odometria, unidade de medida inercial e LIDAR, movido por comandos de navegação remotos em uma casa com quartos, sala e cozinha, para simular a aplicação de SLAM em cena compatível com encontrada em competição. Para tal elaborou-se experimentos de demarcação do ambiente e execução dos algoritmos pesquisados dentro do *framework Robot Operating System*. Considerou-se a complexidade computacional e a performance de algoritmos para avaliar qual melhor se adapta a realidade do robô. Verificou-se que a abordagem *vinySLAM* apresentou resultados quantitativos de localização e qualitativos de mapeamento semelhante a *gmapping* sendo que em complexidade temporal alcançou melhor desfecho.

Palavras-chave: localização; mapeamento; SLAM, *robocup@home*; robótica móvel; Gmapping; VinySLAM.

Abstract

The current monography deals with applications and evaluation of Simultaneous Localization and Mapping approaches to assistants robots of *Open Challenge Robocup@home* category in Latin American Robotics Competition. For this purpose, the Zordon robot from the Pequi Mecânico robotics group of the Federal University of Goiás was used: an automaton capable of moving on the floor of a house, detecting people, interacting with them and manipulating objects. In this sense, was tried to evaluate if the *vinySLAM* algorithm stands out against the current SLAM mapping approach used by Pequi Mecânico, *gmapping*. Therefore a ground-truth trajectory was proposed, created from data collected by sensors such as odometry, inertial measurement unit and LIDAR, moved by tele-guided navigation commands in a house with bedrooms, living room and kitchen, simulating the application of SLAM in a competition-compatible scene. For this, experiments were carried out to demarcate the environment and execute searched algorithms within the *framework Robot Operating System*. Computational complexity and algorithm performance were considered to evaluate which one best fits the reality of the brazillian robot. That work found out that the *vinySLAM* approach presented quantitative localization and qualitative mapping results similar to *gmapping*, and in temporal complexity reached a better outcome.

Keywords: localization; mapping; SLAM; *robocup@home*; mobile robotics; Gmapping; VinySLAM.

Lista de ilustrações

Lista de tabelas

Sumário

1 Introdução

1.1 Problemática

Quando um robô autônomo tem que se locomover por ambiente interno (e.g. casa, escritório, shopping center, etc.), as tarefas de localização e mapeamento tornam-se pré-requisitos essenciais para a navegação robótica e são limitadas pela complexidade dos softwares utilizados e pela acurácias dos sensores embarcados. Um dos principais problemas na robótica está relacionado ao mapeamento e a localização simultâneos, do inglês *Simultaneous Localization and Mapping*. Neste caso, o robô propõe um mapa do ambiente e, simultaneamente, se localiza neste mapa (Thrun et al. 2000). Desse modo, robôs de aplicação doméstica, são exemplos de autômatos que devem mover-se sobre ambientes como citado.

A liga *RoboCup@home* tem o objetivo de desenvolver e implementar tecnologias de robô assistivo e de serviços autonômos, sendo essencial para futuras aplicações domesticas pessoais (Wisspeintner et al. 2009). Nessa competição, além da navegação robótica, o robô deve reconhecer e manipular objetos, detectar seres humanos, processar algoritmos de reconhecimento de fala e gestos para compreender comandos, além de saber interagir com pessoas em linguagem natural.

O grupo de robótica da Universidade Federal de Goiás, Pequi Mecânico, visa no ano de 2019 participar dessa categoria na Competição Latino Americana de Robótica (LARC), para tal, está em desenvolvimento uma base móvel de baixo custo, acoplada com um manipulador de 6 graus de liberdade, rodando softwares otimizados para microprocessadores embarcados de poder de processamento limitado. Por conveniência, toda infraestrutura de software esta sendo desenvolvida dentro do ambiente do ROS (*Robot Operating System*).

Pensando em adotar as melhores práticas de navegação baseada em mapas, a localização feita isoladamente poderia ser um solução aceitável. No entanto, na tarefa de localização o robô explicitamente tenta se localizar coletando dados dos sensores e, então, atualizando sua provável posição no respectivo mapa do ambiente (Siegwart, Nourbakhsh e Scaramuzza 2011). Desse modo, existe uma vasta grama de algoritmos de localização como: *EKF Localization* (Smith e Cheeseman 1986, Chen, Hu e McDonald-Maier 2012) que fundem dados odométricos com informações de sensores de alcance de laser; Localização por *Multi-Hypothesis Tracking* como em Jensfelt e Kristensen (2001), que estende as características do anterior representando suas crenças em multiplas gaussianas; Localização baseado filtros de partículas (Montemerlo et al. 2002, Montemerlo et al. 2003, Fox 2002), que substituem crenças sobre os objetos encontrados em partículas, gerando no final um

filtro de partículas para se localizar.

Contudo, esses algoritmos precisam de um mapa que descreve o modelo do ambiente a se locomover. Para tal, a tarefa de mapear precisamente ambientes extensos como uma casa demanda uma boa localização, tornando o mapeamento um problema de "quem surgiu primeiro, o ovo ou a galinha?" como dito por Thrun et al. (2000) vem dizer. Assim os algoritmos de *Simultaneous Localization and Mapping* (SLAM), respondem a questão fazendo com que tanto a localização quanto o mapeamento sejam feitas simultaneamente.

Dois algoritmos estão disponíveis em pacotes do *ROS*, que podem ser considerados *SLAMs* do estado-da-arte e geraram bons resultados em robôs embarcados com limitação de tempo real (Filatov et al. 2017) são: *gmapping* (Grisetti et al. 2007) e *vinySlam* (Huletski, Kartashov e Krinkin 2017). O primeiro utiliza melhorias sobre a proposta de (Hahnel et al. 2003), apresentando técnicas adaptativas para reduzir o número de partículas no filtro *RAO-Blackwellized* além de um processo de seleção de operações de amostragem na qual reduz seriamente o problema da depleção das partículas; o segundo método incorpora sobre o *tinySLAM* (Steux e Hamzaqui 2010) o Modelo de Crença Transferível, melhorando sua robustez e acurácia.

1.2 Objetivo

1.2.1 Objetivo Geral

O principal objetivo desse trabalho é comparar os resultados da aplicação dos algoritmos *gmapping* e *vinySLAM* dentro do robô da modalidade *RoboCup@home* dasafio *open challenge* da equipe de robótica, Pequi Mecânico, da Universidade Federal de Goiás. Analisar qual será o melhor algoritmo que responde de forma eficiênte pelo *trade-off* de acurácia e custo computacional.

1.2.2 Objetivos Específicos

Tendo em visa a escassez de *datasets* de ambientes interno como casa e apartamentos, esse projeto final de curso objetiva a criação de um conjunto de dados para ser utilizados para avaliação do mesmo e pela comunidade do *Robocup@home* em futuras avaliações de algoritmos para navegação autônoma de robôs.

1.3 Organização

A próxima seção aborda uma explanação sobre o robô assistente Zordon do grupo de robótica Pequi Mecânico. Lista quais sensores e atuadores foram implantados no robô

para essa pesquisa. Ainda mais, 2.2 resume os nós e pacotes do *framework Robot Operating System* (ROS) utilizados.

A seção 3 define a revisão bibliográfica dessa pesquisa. Sintetiza o modelo probabilístico de um típico algoritmo de Localização e Mapeamento Simultâneo e revisa as abordagens dos algoritmos *gmapping* e *vinySLAM*.

A metodologia que compõe as técnicas de coleta de dados e a implementação do método de comparação dos resultados dos algoritmos de SLAM podem ser vistos na seção 4. Conquanto os resultados e análise deste para o método comparação pode ser visto na seção 5.

E por fim a seção 6 conclui sobre os resultados e demonstram se os objetivos propostos na seção de introdução da monografia foram concluídos. Se as perguntas e problemas apresentados inicialmente foram respondidas e/ou esclarecidas.

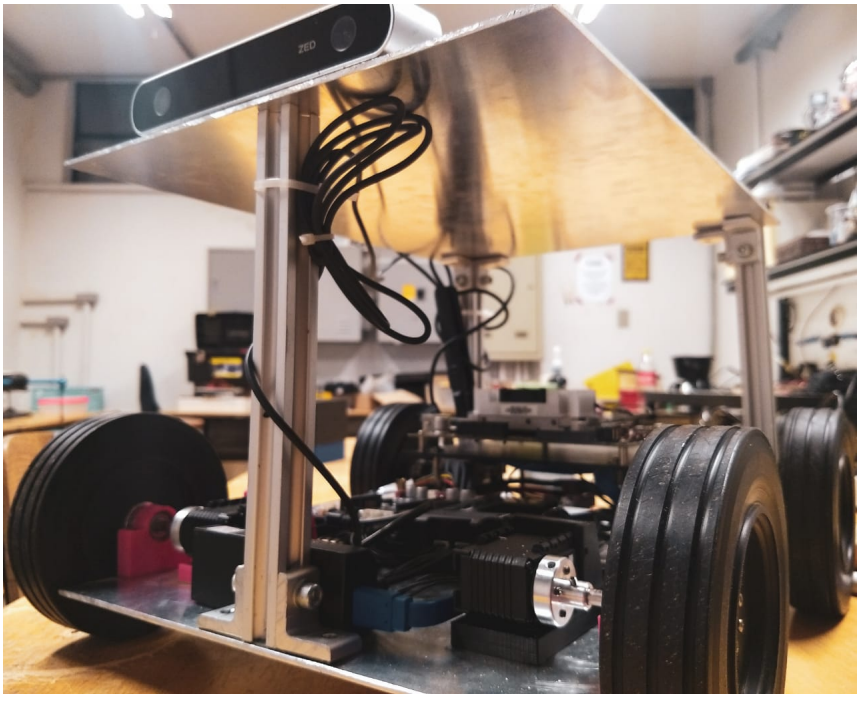
2 Robô Zordon

A equipe de robótica da Universidade Federal de Goiás, Pequi Mecânico, buscará, no segundo semestre de 2019, se integrar ao rol de equipes participantes da categoria *Open Challenge RoboCup@home* na *Latin American Robot Competition*. Para tal, está em desenvolvimento o robô Zordon: um robô autônomo de baixo custo, capaz de executar as inúmeras atividades associadas a tarefas domésticas, com sua estrutura mecânica toda desenvolvida pelos integrantes da equipe goiana.

Essa estrutura leva um manipulador de 6 graus de liberdade, com juntas atuadas por motores *XYZrobot Smart Servo A1-16*. Uma base de 4 motores de corrente contínua *Metal Gearmotors* de 25 milímetros de diâmetro, 21,6 kg.cm e 100 rotações por minuto, montados de forma a adequar uma base móvel de eixo diferencial (duas rodas para cada lado da base) como mostra a Figura 1 abaixo.

Para executar os mais variados software relacionados à elaboração das tarefas, o Zordon possui, como microprocessador embarcado, uma *Jetson TX2 Developer Kit* da *NVIDIA Corporation* com seus 6 núcleos de processamento (dois NVIDIA Denver e quatro ARM® Cortex®-A57 MPCore, ambos de arquitetura de 64 bits), quatro gigabytes de

Figura 1 – Robô Zordon



Fonte: dados coletados na pesquisa

memória RAM e com uma placa de vídeo de 256-core da arquitetura *NVIDIA Pascal*^{TM}.

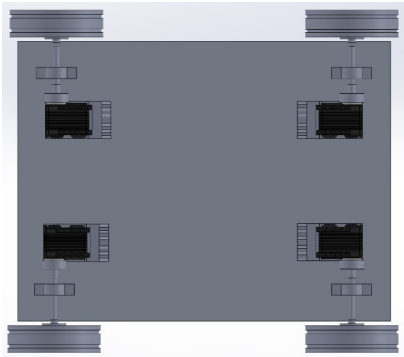
A respeito dos sensores, o tipo de dados que eles fornecem à camada de percepção do robô, seu modelo e sua acurácia, essa etapa abordará esses temas. Como esses dados serão processados por aplicações executando dentro do ambiente ROS, a seção 2.2 faz uma explicação resumida do *Robot Operating System*: sua organização em *stacks*/pacotes; estrutura de nós, tópicos e serviços; tipos de mensagens de sensores e navegação; pacotes de tratamento de dados utilizados; pacotes de aquisição sensorial; e uma breve explanação sobre o utilitário *rosbag* que será utilizado como meio de coleta de dados, como consta no capítulo 4.1.

2.1 Percepção robótica

2.1.1 Sensor odométrico

Na maioria das aplicações de robôs móveis dois métodos de estimadores de posição básicos são empregados juntos: posicionamento relativo e absoluto (Borenstein e Feng 1996). Como exemplo de posicionamento relativo, odometria é o método de localização mais comumente usado; ele rovê boa acurária para curto período, não é dispendioso, e permite taxas de amostragens muito altas (Borenstein et al. 1997). O robô Zordon possui quatro motores com encoder de rotação de 48 pulsos por rotação cada. Assim pode ser possível estimar a velocidade de rotação das rodas e o deslocamento linear delas.

Figura 2 – Visão de planta do Robô Zordon



Fonte: dados coletados na pesquisa

Considerando que as rodas estão dispostas como mostra a Figura 2, onde as duas rodas da esquerda trabalham na mesma velocidade, tal como as duas rodas da direita. Pode-se dizer que a base do robô Zordon comporta-se como uma base de um único

eixo diferencial situado na linha central. Desse modo, as equações 2.1 e 2.2 descrevem o incremento linear e o angular respectivamente.

$$\Delta S = K\frac{\Delta C_{direita} + \Delta C_{esquerda}}{2} \tag{2.1}$$

$$\Delta \alpha = K\frac{\Delta C_{direita} - \Delta C_{esquerda}}{W} \tag{2.2}$$

$$K = \frac{2\pi Rr}{N} \tag{2.3}$$

Nesse caso, $\Delta C_{direita}$ e $\Delta C_{esquerda}$ representam o incremento de pulsos contados pelos encoders da direita e esquerda respectivamente, o parâmetro K (equação 2.3) é dado por metros por contagem, R é o raio da roda dado por metros por revolução, N é a resolução dos encoders dado por contagens por revolução e r é a taxa de redução da caixa de engrenagem. Todas essas fórmulas foram deduzidas em (Crowley e Reignier 1992).

A informação odométrica do robô é definida com esses incrementos representando o deslocamento linear e angular da base referentes a posição e orientação de partida. Pode-se ainda derivar essas equações e chegar na velocidade linear e angular do robô.

Todavia, mesmo que a odometria apresente uma ótima solução em tempo real para o problema do posicionamento relativo, ela sempre apresenta erros. Borenstein et al. (1997) diz que esses erros podem ser categorizados em dois grupos: erros sistemáticos e não-sistemáticos. Enquanto os erros sistemáticos estão relacionados às imperfeições cinemáticas do robô, como diâmetro da roda e distância entre as rodas mal medidos, erros não sistemáticos resultam da interação não perfeita entre as rodas e o piso em contato. Embora esses erros podem ser minimizados por técnicas probabilísticas como em Ljung (1979), a incerteza sobre a posição e a orientação do robô cresce com o quadrado do incremento linear e angular, como mostra a equação 2.4 retirada de Crowley e Reignier (1992).

$$C_W = \Delta S^2 \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & K_{drift} \end{bmatrix} + \Delta \alpha^2 \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & K_{rot} \end{bmatrix} \tag{2.4}$$

$$\tag{2.5}$$

Onde C_W corresponde a incerteza devido a translação, rotação e suas derivadas; K_{drift} representa uma estimativa do *drift* da base dado em graus-quadrado por metros-quadrado, e o parâmetro adimensional K_{rot} é a estimativa de erro angular ao executar um movimento de rotação em torno do próprio eixo 360º. Essa media C_W serve para atualizar a matriz de covariância que representa a incerteza de se estar, para um dado momento, em um ponto (x, y) com ângulo θ.

2.1.2 Unidade de Medida Inercial

A Unidade de Medida Inercial (do inglês *Inertial Measurement Unit*, aka IMU) é um dispositivo eletrônico capaz de estimar navegação inercial, a qual Borenstein et al. (1997) categoriza como outro método de medida de posição relativa (também chamado de *dead-reckoning*).

Com os dados providos do IMU, acelerações rotacional e translacionais são estimadas pelo giroscópio e acelerômetro respectivamente. Esses tipos de sensores são implementados dentro circuito integrado e podem ser acessados por algum tipo de interfaceamento eletrônico. O dispositivo, incorporado ao robô Zordon, BNO055 da empresa Bosch Sensortec (Sensortec 2016) possui o protocolo de comunicação serial *i2c*. Desse modo, com apenas dois fios pode-se adquirir as medidas de orientação e aceleração passando apenas o endereço lógico do componente e comandos detalhados em seu *datasheet* para o sensor e esperar por uma resposta do mesmo. A taxa de atualização do sensor é de cerca de 120 Hz, se tornando um adequado tipo de sensoriamento para aplicações em tempo real, pois espera-se que o robô tome decisões de navegação a menos de 10 Hz.

Uma outra vantagem do *CI* da Bosch é que ele possui um magnetômetro (método de posicionamento absoluto, capaz de aferir orientação referente ao campo magnético terrestre) e um processo de estimação de estado não-linear, como Filtro de Kalman Extendido, capaz de fundir os dados de orientação e deslocamento em uma única informação mais precisa, sem o prejuízo na taxa de amostragem.

Apesar deste IMU ser o estimador de orientação mais preciso (principalmente devido aos filtros internos como EKF) entre os sensores embarcados do robô do Pequi Mecânico, é o que possui a maior incerteza associada ao deslocamento. Isso acontece já que, para medir o deslocamento, é preciso integrar a aceleração duas vezes, esse tipo de algoritmo incorpora *drift* com o tempo (Borenstein et al. 1997). Além do mais esse dispositivo tem aumento de ruídos quando há grande distorção do campo magnético da terra, como em ambientes próximos a estruturas metálicas e linhas de transmissão de energia (Borenstein e Feng 1996).

2.1.3 LIDAR 2D

Um LIDAR (*Light Detection And Ranging*) é um tipo de sensor óptico de detecção remota capaz de medir a distância entre o emissor e objetos. Esse tipo de sensor envia pulsos de ondas eletromagnéticas guiadas como os lasers e espera pelo retorno desses pulsos, a partir do fato que a velocidade da luz é constante para um material homogêneo (ou quase homogêneo como o ar), pode-se estimar a distância entre o objeto refletor e o sensor.

Repetindo esse procedimento, variando o ângulo de emissão do feixe de luz, obtém-

se a distância dos objetos ao redor, de forma a montar uma representação 2D do meio no qual o sensor está inserido. Caso a quantidade de pulsos recebidos de uma certa direção seja inferior a um limite pré-definido, então o equipamento pode inferir que o objeto detectado não possuí uma forma bem estabelecida (e.g. vidros, espelhos, objetos muito próximos ou muito distantes). Nesse caso, o ponto referente ao objeto pode ser tratado como um valor não identificado.

Para a robótica, essa solução sensorial pode ser usada em aplicações de mapeamento e localização. Pode ser considerado um sensor para construção de mapas que serve para medidas de posição absoluta segundo Borenstein et al. (1997). Com um mapa de duas dimensões formado, um algoritmo de localização, por exemplo, pode estimar a distância do robô dos pontos detectados e assim melhor calcular o erro de *drift* dos sensores de odometria e IMU. KartoSLAM [1], CoreSLAM[2], HectorSLAM [3], Gmapping [4], LagoSLAM [5] são exemplos de algoritmos de SLAM baseados em mapa que utilizam esses dispositivos como sua principal medida do ambiente de operação e possui sua implementação dentro do ROS.

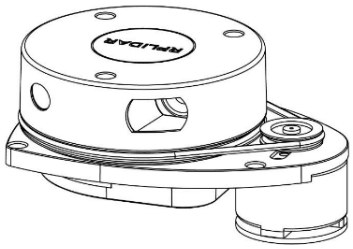
O robô Zordon possui um LIDAR de baixo custo da marca *RPLIDAR A1* produzido pela empresa Slamtec Co., Ltd. Este é capaz de detectar objetos em campo de 360 graus e raio de 12 m (dimensões mínimas de 15 cm no alcance máximo), com 8000 amostras por segundo, publicando varreduras de laser a 10 Hz. É importante destacar que quanto maior a frequência de operação deste tipo de sensor, mais rápido os algoritmos de SLAM convergem em resultados de localização e mapeamento. Todavia, devido as baixas velocidades de operação do processo de navegação do robô Zordon, essa frequência apresenta ser satisfatória para a localização e mapeamento de objetos da cena de navegação. A Figura 3 mostra um croqui do LIDAR do Pequi Mecânico.

2.2 *Robot Operating System*

As aplicações de software do robô Zordon estão todas desenvolvidas dentro do ROS. Para manipulação do braço robótico utiliza-se o *framework* de movimentação MoveIt, onde o trabalho de referência foi publicado por Chitta, Sucan e Cousins (2012); para tarefas de *speech recognition* e *object detection* estão em desenvolvimento pacotes que integram API bem desenvolvidas (e.g Google *speech-to-text* e OpenCV) com o controle de comportamento; na navegação, o planejamento de rotas, os mapas de obstáculos, dentre outros algoritmos, ficam a cargo da *stack* de navegação nativa do ROS retratado por (Marder-Eppstein et al. 2010).

[1] http://www.ros.org/wiki/karto
[2] http://www.ros.org/wiki/coreslam
[3] http://www.ros.org/wiki/hector_slam
[4] http://www.ros.org/wiki/gmapping
[5] https://github.com/rrg-polito/rrg-polito-ros-pkg

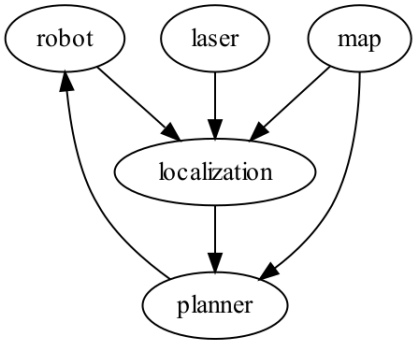
Figura 3 – Croqui do sensor RPLIDAR A1.



Fonte: SLAMTECH

O ROS é um conjunto de bibliotecas, *framework* e utilitários, capaz de promover o melhor da programação distribuída e paralela para aplicações robóticas. A infraestrutura básica desse sistema é nó. Essa abstração é capaz de prover ou requerer dados contidos em mensagens por via de tópicos ou serviços. Um nó implementa uma funcionalidade específica de robótica e são processos que executam computação (Quigley et al. 2009). Tal como mostrado na Figura 4, nós são capazes de adquirir dados de um laser e um mapa, processar esses dados em um algoritmo de localização, obter a informação de localização e planejar uma rota de locomoção para uma base.

Figura 4 – Diagrama de relacionamento entre nós



Fonte: produzido pelo autor

Cada elipse desse diagrama retrata um nó com sua funcionalidade especificada pelo nome contido dentro. As setas representam as mensagens trocadas entre os nós. Nesse caso as mensagens são organizadas sobre a abstração de um tópico. Esse tipo de mensagem trabalha sobre o padrão *publisher/subscriber*. Nessa situação, as setas originando em cada no representam suas publicações e as chegantes suas subscrições.

Com essa forma de organização pode-se promover comunicações assíncronas entre

os nós. No entanto, alguns algoritmos não funcionariam dessa maneira, nesse caso as mensagens do tipo serviço são as melhores soluções. Em suma, um serviço é um tipo de mensagem onde os nós que a utilizam implementam o padrão de comunicação distribuída *request/response*. Ou seja, dado um nó A que demanda um serviço de outro B (e.g. o resultado da detecção de um objeto) esse espera pelo fim do processo do nó requisitado, implementando assim uma comunicação síncrona.

O conteúdo das mensagens são padronizados em múltiplas formas, desde números inteiros, coleção de caracteres, até estruturas de mensagens mais complexas como imagens. A seção seguinte trata dos padrões de mensagens relacionados aos sensores supracitados, enquanto as seções posteriores uma rápida explicação de nós importantes para esse trabalho e o básico do utilitário do *rosbag* serão retratados.

2.2.1 Tipos de mensagens sensoriais

Mensagens protocoladas são mensagens cuja seu conteúdo possui um conjunto de estruturas de dados capazes se serem implementados em todas as linguagens que o ROS disponibiliza uma biblioteca de códigos. Assim, um nó programado na linguagem de programação *python* pode publicar dados em um tópico enquanto um nó programado em *C++* pode subscrever este tópico e processar o conteúdo normalmente.

Para encapsular dados de sensores como odometria, existe o pacote de mensagens nav_msgs (ROSWiki nav_msgs) contendo a mensagem do tipo Odometry. Esse tipo de mensagem possui como contudo um cabeçalho, o nome do frame (eixo de coordenadas de referência) cuja a odometria está relacionada, uma estrutura para posição e orientação do robô contendo sua covariância e uma estrutura idêntica para dados de velocidade.

Por outro lado, os tipos de mensagens dos sensores IMU e LIDAR encontram-se no pacote de mensagens sensor_msgs (ROSWiki std_msgs). Sendo que o IMU é encapsulado por mensagens do tipo Imu (com um quatérnion representado a orientação, um vetor de três dimensões para a velocidade angular e outro vetor 3d para aceleração linear) e enquanto para o LIDAR o mesmo acontece com as mensagens do tipo LaserScan (contendo como principal atributo um *array* de ponto flutuante associado às distâncias dos objetos detectados).

2.2.2 *ROS Wrappers* de sensores

Algumas vezes um determinado sensor possui um conjunto de bibliotecas de software que foi implementado por uma empresa ou uma comunidade sem levar em conta a infraestrutura do ROS em seu código, mas esse sensor é necessário para inúmeras aplicações rodando no ROS. Então, a estratégia mais interessante a adotar é criar um nó capaz de acessar essas bibliotecas e converter os dados providos por elas em tipos de mensagens

protocoladas para ser entregues ao sistema. Nesse caso, o pacote que contém o nó pode ser chamado de um *ROS Wrapper* desse sensor.

Para esse trabalho foi utilizado dois *ROS Wrappers* que abrangem a aquisição de dados dos sensores IMU e LIDAR. O primeiro foi desenvolvido por membros do próprio grupo de robótica, utilizando a biblioteca em *C++* do *CI* BNO055 acessível ao público. Já o segundo foi utilizado o pacote *rplidar* que se encontra em (ROSWiki rplidar) como um pacote oficializado pelos organizadores do ROS.

3 *Simultaneous Localization and Mapping*

Modelos de medida descrevem o processo de formação pelo qual as medidas dos sensores são geradas no mundo físico (Thrun et al. 2000). Assim como modelo do movimento do robô, essas abstrações são formuladas de forma probabilística, levando em conta as incertezas associadas às medidas e ao controle. DURRANT-WHYTE; BAILEY (2006) denominam modelos de medidas como modelo de observações. Estes autores e Thrun et al. consideram o SLAM um problema probabilístico na qual surge quando o robô não tem acesso ao mapa do ambiente e também não tem acesso a sua própria posição.

Sem localização e mapeamento prévios, o problema do mapeamento e localização simultâneos leva a distribuição probabilística (equação 3.1), na qual o qual descreve a probabilidade a posteriori da localização dos pontos de referência (m) e da posição do veículo (x_k dado o conjunto de observações adquiridas ($Z_{0:k}$) e entradas de controle desde o instante zero até o k-ésimo instante ($U_{0:k}$), junto com o estado inicial do veículo (x_0).

$$P(x_k, m|Z_{0:k}, U_{0:k}, x_0) \tag{3.1}$$

O modelo de observação (medidas) descrevem a probabilidade de obter uma observação z_k quando a localização do robô e a localização dos pontos de referência são conhecidos e podem ser generalizados pela equação (3.2) logo abaixo.

$$P(z_k|x_k, m) \tag{3.2}$$

O modelo do movimento do robô pode ser descrito em termos de uma distribuição de probabilidade sobre transições de estados de um processo markoviano na qual o estado atual (x_k) depende apenas do estado anterior (x_{k-1}) e das entradas de controle no instante k, como mostra a equação (3.3).

$$P(x_k|x_{k-1}, u_k) \tag{3.3}$$

O algoritmo genérico do SLAM pode ser implementado sequencialmente em um padrão recursivo em dois passos: predição (equação 3.4, como uma probabilidade a priori) e correção (equação 3.5) (Thrun et al. 2000).

$$P(x_k, m|Z_{0:k-1}, U_{0:k}, x_0) = \int P(x_k, |x_{k-1}, u_k) \times P(x_{k-1}, m|Z_{0:k-1}, U_{0:k-1}, x_0) dx_{k-1} \tag{3.4}$$

$$P(x_k, m|Z_{0:k}, U_{0:k}, x_0) = \frac{P(z_k|x_k, m)P(x_k, m|Z_{0:k-1}, U_{0:k}, x_0)}{P(z_k|Z_{0:k-1}, U_{0:k})} \tag{3.5}$$

Assumindo que a localização do robô é conhecida (ou no mínimo determinística) em todo o momento, construir o mapa resume em computar a densidade condicional $P(m|X_{0:k}, Z_{0:k}, U_{0:k})$. Por outro lado, encontrar a localização do robô compreende-se em computar a densidade condicional $P(x_k|Z_{0:k}, U_{0,k}, m)$ assumindo que a localização dos pontos de referência são conhecidos com boa precisão.

Há algumas soluções para o problema do SLAM, no geral todas envolvem buscar uma representação apropriada tanto do modelo de observação quanto do modelo de movimentação do robô para permitir eficiência no método e na computação realizada dos passos recursivos (3.4 e 3.5).

Tipicamente, existem duas abordagens para a representação de mapas de ambiente. São os mapas geométricos ou topológicos (Oliveira 2008). Os mapas geométricos referem aqueles em que a posição dos objetos tais como a distância entre eles são bem definidas. Por outro lado, os mapas topológicos representam o ambiente a partir de elos, que definem os objetos, conectados a nós (Oliveira 2008). Dessa forma, essa monografia limita-se na busca por algoritmos de mapeamento e localização simultâneas que atendem as especificações dos mapas geométricos.

Dentre as soluções a mais conhecidas, e a primeira solução para o problema probabilístico do SLAM baseado em mapas geométricos, foi a EKF SLAM (Smith e Cheeseman 1986). Essa abordagem representa o modelos probabilísticos da equação 3.2 e 3.3, na forma de um modelo de espaço de estados com erro aditivo gaussiano utilizando o filtro de Kalman estendido como cerne da solução. Como essa resposta emprega linearização em cima de modelos não-lineares a convergência e a consistência do algoritmo é prejudicada quando a não-linearidade torna-se grandiosa.

3.1 Gmapping

O filtro de partículas Rao-Blackwellized pode ser aplicado ao problema do SLAM para descrever o modelo de movimentação do robô como um conjunto de amostras levadas à distribuição probabilística não gaussianas. O algoritmo do FastSLAM (Montemerlo et al. 2002) introduz esse tipo de abordagem que leva em sua construção amostragem Monte Carlo recursiva, ou filtragem por partícula, sendo o primeiro diretamente a representar o modelo de processos não lineares e distribuição não gaussiana para posição (Durrant-Whyte e Bailey 2006).

A alta dimensão do espaço de estados do problema do SLAM torna aplicações diretas de filtros de partículas computacionalmente inviável. No entanto, é possível reduzir

o espaço de amostragem aplicando Rao-Blackwellization. O estado das juntas do SLAM podem ser particionado de acordo com a regra do produto 3.6.

$$P(X_{0:k}, m|Z_{0:k}, U_{0:k}, x_0) = P(m|X_{0:k}, Z_{0:k})P(X_{0:k}|Z_{0:k}, U_{0:k}, x_0) \tag{3.6}$$

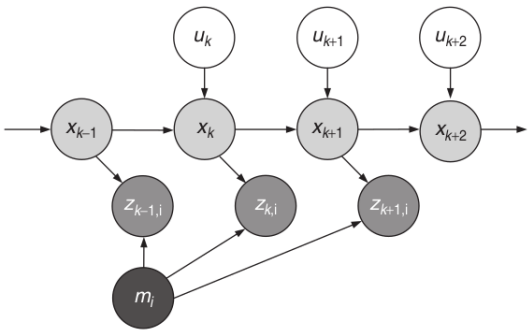
Veja que o problema do SLAM passa a levar em conta a trajetória percorrida pelo robô em vez da posição momentânea dele. Aqui pode-se considerar as entradas de controle $U_{0:k}$ como os valores odométricos medidos desde do instante zero até k.

Se $P(m|X_{0:k}, Z_{0:k})$ pode ser representado analiticamente, apenas $P(X_{0:k}|Z_{0:k}, U_{0:k}, x_0)$ precisa ser amostrada ($X_{0:k}^{(i)} \sim P(X_{0:k}|Z_{0:k}, U_{0:k}, x_0)$). Assim o mapa pode ser composto, em função de cada partícula, por uma distribuição gaussiana independente (equação 3.7).

$$P(m|X_{0:k}^{(i)}, Z_{0:k}) = \prod_{j}^{M} P(m_j|X_{0:k}^{(i)}, Z_{0:k}) \tag{3.7}$$

Dado a trajetória do robô $X_{0:k}$ em vez da posição momentânea deste, os pontos de referência do mapa tornam-se independentes. Em outras palavras, se o histórico da posição é conhecido com exatidão, então, desde que as observações no mapa são condicionalmente independente, os estados no mapa também são independentes (veja Figura 5). Essa propriedade do FastSLAM que o torna mais rápido do que o EKF-SLAM, já que o mapa pode ser representado por gaussianas independentes, sua complexidade passa a ser linear em vez de quadrática.

Figura 5 – Independência do mapa com a trajetória do robô



Fonte: (Grisetti et al. 2007)

A principal teoria por traz dos filtros de partículas é a *sequential important sampling* (SIS). Um filtro SIR Rao-Blackwellized para mapeamento incrementalmente processa as observações e as leituras odométricas na medida em que elas estão disponíveis. Assim sendo, esse processo responde pela atualização de um conjunto de amostras, representando

a probabilidade a *posteori* sobre o mapa e a trajetória do robô, como descrito nos quatro passos abaixo:

1. *Amostragem*: a próxima geração de partículas $x_k^{(i)}$ é obtida a partir da geração atual $x_{k-1}^{(i)}$, amostrando por uma distribuição proposta $\pi(x_k^{(i)}|X_{0:k-1}, Z_{0:k}, u_k)$.

2. *Ponderação por Importância*: um peso importante individual $w^{(i)}$ é registrado para cada partícula de acordo com

$$w_k^{(i)} = w_{k-1}^{(i)} \frac{P(z_k|X_{0:k}^{(i)}, Z_{0:k-1})P(x_k^{(i)}|x_{k-1}^{(i)}, u_k)}{\pi(x_k^{(i)}|X_{0:k-1}^{(i)}, Z_{0:k}, u_k)} \tag{3.8}$$

   onde,

$$P(z_k|X_{0:k}, Z_{0:k-1}) = \int P(z_k|x_k, m)P(m|X_{0:k-1}, Z_{0:k-1})dm \tag{3.9}$$

3. *Reamostragem*: partículas com baixo peso importante, são geralmente substituídas por amostras com alto peso importante. Assim sendo, com um conjunto finito de amostras pode aproximar uma distribuição contínua. Além do mais, permite aplicar o filtro de partículas em situações em que a distribuição proposta difere da verdadeira distribuição que modela as transições de estados.

4. *Estimando o mapa*: para cada amostra $x_k^{(i)}$, o correspondente mapa estimado $m_k^{(i)}$ é computado baseada na trajetória do robô e nas observações ($P(m_k^{(i)}|X_{0:k}^{(i)}, Z_{0:k}, x_0)$).

Grisetti et al. (2007) propõe melhorias no âmbito da distribuição proposta e uma nova técnica de reamostragem adaptativa. Nessa solução, também chamada de *gmapping*, a primeira melhoria é similar à definida pelo algoritmo do FastSLAM-2 (Montemerlo et al. 2003). Em vez de usar uma distribuição proposta estática como FastSLAM, esse algoritmo computa no decorrer da execução uma distribuição proposta melhorada permitindo usar mais informações obtidas pelos sensores enquanto gera as partículas. A segunda melhoria mantém uma variedade razoável de partículas, desse modo reduz o risco da depleção de partícula, que ocorre quando a quantidade a quantidade de partículas é insuficiente para representar a população.

3.2 VinySLAM

As abordagem de SLAM que utilizam filtro de partículas são geralmente melhores dos que as implementações utilizando EKF. Todavia, em ambiente embarcado de baixo custo, nenhuma consegue computar em tempo real (aqui adota-se 20 Hz como referência). Para tal, seria necessário algum algoritmo capaz de processar os passos de atualização dos estados de forma rápida e com acurácia. Steux e Hamzaqui (2010) propõe uma solução de SLAM (tinySLAM) barata computacionalmente e de fácil de implementação (i.e. um

SLAM implementado com apenas duzentas linhas de código na linguagem de programação C). No entanto, (Santos, Portugal e Rocha 2013) e (Filatov et al. 2017), mostrara que essa abordagem não conseguiu ser robusta para aplicações robóticas em ambiente *indoor*.

Huletski, Kartashov e Krinkin não ignoram a proposta do *tinySLAM*, eles melhoram a abordagem baseando no combinador de varredura de Monte Carlo (do inglês *Monte Carlo scan matcher*) que calcula a posição atual com base nas observações e no mapa usando passeio aleatório. Logo após encontrar a posição esse algoritmo atualiza as células do mapa de grade que representa o Modelo de Crença Transferível (Smets e Kennes 1994). Esses autores metaforizam a comparação entre o algoritmo de Steux e Hamzaqui e sua proposta com as seguintes palavras (considere uma livre tradução): "tinySLAM não é estável (como uma pilha de pedras) e as mudanças propostas agem como videiras que 'apertam' suas principais partes para aprimorar robustez". Daí que surge o nome VinySLAM (i.e. "videiras" vem do inglês *vines*).

As sub seções 3.2.1 a 3.2.3 resumirão a estimativa de ocupação de células do mapa de grade com o Modelo de Crença Transferível (do inglês Transferable Belief Model, i.e. TBM), uma breve explanação sobre aquilo que o algoritmo do tinySLAM é importante para a presente seção e por fim uma síntese do vinySLAM junto com um comparativo deste algoritmo frente ao gmapping.

3.2.1 TinySLAM

O algoritmo *tinySLAM* consiste de duas principais operações: calculo de distância entre o resultado de um *laser scan* (i.e. varredura de obstáculos providos pelo LIDAR) com o mapa, e atualização do mapa. Para encontrar a localização do veículo essa abordagem usa uma implementação "simples do algoritmo de Monte-Carlo para coincidir a atual varredura de laser com o mapa" (Steux e Hamzaqui 2010). O algoritmo de *scan matcher* apresentado logo abaixo (1) sintetiza essa implementação.

Percebe-se que para cada varredura de laser dado, o algoritmo de *scan matcher* busca pela posição na vizinhança dos valores preditos para minimizar a discrepância entre o mapa (Huletski, Kartashov e Krinkin 2017). Nesse algoritmo [1] a função $random\_normal$ é uma randomização normalizada na média dos valores de posição e $ts\_distance\_scan\_to\_map$ pode ser vista como uma função de custo da busca de Monte-Carlo ou uma função de

[1] Retirado diretamente da implementação disponível como código-fonte aberto em <https://github.com/OpenSLAM-org/openslam_tinyslam acessado em 15 outubro de 2019>.

Likelihood para cada partícula (isto é hipotese) do filtro (Steux e Hamzaqui 2010).

Algoritmo 1: monte_carlo_search

Entrada: $map, scan, start\_pos, sigma\_xy, sigma\_theta, stop$

Saída: Distância entre map e scan

```
1  início
2      counter ← 0
3      current_pos ← start_pos
4      current_dist ← ts_distance_scan_to_map(scan, map, current_pos)
5      best_dist ← current_dist
6      repita
7          current_post.x ← random_nomal(current_pos.x, sigma_xy)
8          current_post.y ← random_nomal(current_pos.y, sigma_xy)
9          current_post.theta ← random_nomal(current_pos.theta, sigma_theta)
10         se current_dist < best_dist então
11             best_dist ← current_dist
12         senão
13             counter ← counter + 1
14         fim
15     até counter < stop;
16     retorna best_dist
17 fim
```

Devido a busca do algoritmo de Monte-Carlo possuir complexidade temporal elevada, geralmente adorta-se a posição inicial do veículo dado pele odometria como posição de inicialização (*current_pos* do algoritmo 1). Assim sendo, geralmente com a incorporação dessa fonte de dados no algoritmo reduz o tempo gasto para inferir sobre a localização e o mapeamento.

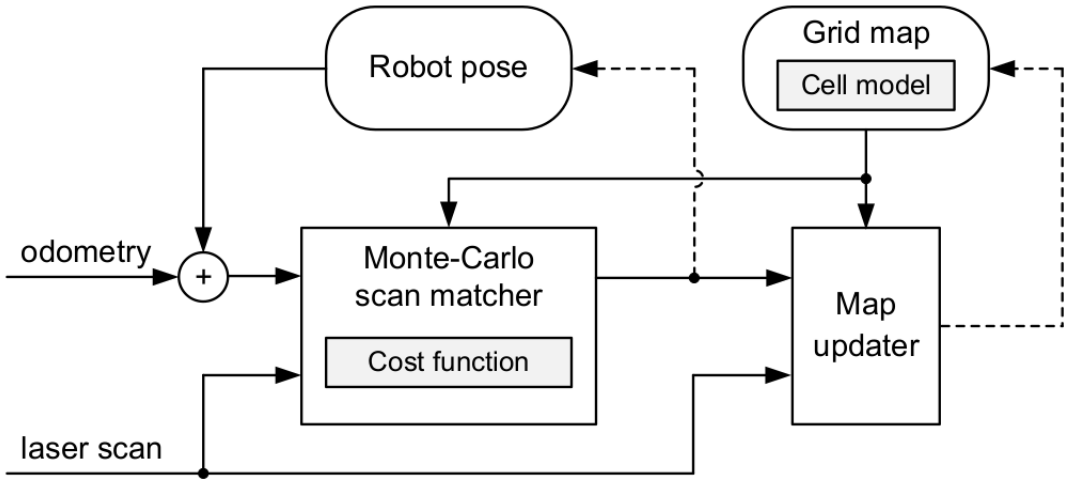
Outro procedimento de suma importância desse algoritmo é a atualização do mapa. Aqui é realizado utilizando uma implementação modificada do algoritmo de Bresenham. Dessa forma, são extraídos linhas a partir dos pontos de varredura de laser e estes são adicionados ao mapa de uma maneira especial a fim de facilitar a convergência do *scan matcher* (Steux e Hamzaqui 2010).

Huletski, Kartashov e Krinkin dizem que, para o algoritmo *tinySLAM*, o mapa é modificado (atualizado) na medida em que observações são disponíveis a ele e sempre logo após a passagem de *scan matching* (busca pela menor distância que faça coincidir o mapa previamente obtido com a varredura de laser).

No caso do *tinySLAM* ainda, o conjunto de células dos mapa de grade a ser preenchido com informações de ocupação é determinado por um simples algoritmo de

ray tracing, onde cada célula desse conjunto tem sua ocupação em função da posição dos obstáculos em linha direta com sensor de varredura de laser.

Figura 6 – *Workflow* do *tinySLAM*



Fonte: retirado de (Huletski, Kartashov e Krinkin 2017)

Além do mais, este algoritmo utiliza da estratégia de transbordar as células ocupadas (modificar a informação de ocupação das células vizinhas) afim de aumentar a robustez do mapeamento. Sendo o transbordamento diretamente proporcional a distância dos obstáculos encontrados.

A figura 6 sintetiza o fluxo de trabalho do algoritmo *tinySLAM* e também do algoritmo *vinySLAM*, sendo que este último incorpora modificações nos procedimentos simbolizados pelos retângulos cinza (i.e. "*Cost Function*"e "*Cell model*").

3.2.2 Modelo de Crença Transferível

O objetivo do Modelo de Crença Transferível (MCT) é representar conhecimento como crenças quantificadas. O modelo fornece uma maneira de atualizar informação rastreada a partir de novas observações assim como manter a diferença entre estados de incerteza e desconhecimento (Huletski, Kartashov e Krinkin 2017).

Em vez de aplicar um modelo de conhecimento sobre o estado das células dos mapas de grade que forneça apenas hipóteses sobre se elas estão ocupadas ou vazias, *vinySLAM* adotou novos dois estados: conflito e desconhecido. Sendo assim o conhecimento sobre as células dos mapas de ocupação podem ser definidos pela Assinatura de Crença Básica (ACB) que representa uma função para assinar crença a cada hipótese de estado (ocupada, vazia, conflito e desconhecido).

Duas observações de fontes confiáveis representadas pela ACB podem ser fundidas usando a regra conjuntiva:

$$acb_{1\oplus 2}(C) = \sum_{A\cap B=C} acb_1(A).acb_2(B) \tag{3.10}$$

onde:

- A, B, C são hipóteses analisadas pelo Modelo de Crença Transferível;
- acb_1 e acb_2 são os valores de ACB das observações analisadas;
- e $acb_{1\bigoplus 2}$ representa a ACB das observações fundidas.

No *vinySLAM* o Modelo de Crença Transferível, a partir de um mapa de ocupação - células representadas pela probabilidade de estar ocupada p_o- pode-se obter o mapa de crença - células representada pela *acb* de cada estado. As equações 3.11 a 3.14, ilustram essa transformação, onde q representa a qualidade da observação de cada ponto de varredura de laser.

$$acb(ocupada) = p_o.q \tag{3.11}$$

$$acb(vazia) = (1 - p_o).q \tag{3.12}$$

$$acb(desconhecida) = 1 - acb(ocupada) - acb(vazia) \tag{3.13}$$

$$acb(conflito) = 0 \tag{3.14}$$

Sabe-se que o MTC usa a regra conjuntiva com normalização pelo conflito para atualizar as células do mapa de crença (veja equação 3.15). Por outro lado para recuperar o mapa de ocupação dado o mapa de crença pode-se utilizar da transformação definida na equação 3.16 (de acordo com (Smets 2005)).

$$acb'_{1\oplus 2}(A) = \begin{cases} \frac{acb_{1\oplus 2}(A)}{1-acb_{1\oplus 2}(\Theta)}, & A \neq \Theta \\ 0, & A = \Theta \end{cases} \tag{3.15}$$

$$p_o = acb(ocupado) + 0,5.acb(desconhecido) \tag{3.16}$$

3.2.3 Função de custo

O *vinySLAM*, a função de custo para o algoritmo de *scan matcher* (ts_distance_scan_to_map no algoritmo 1) foi modificada para funcionar com o Modelo de Crença Transferível. Enquanto no *tinySLAM* o objetivo dessa função era servir como uma heurística que calcula a distância relativa entre o mapa e cada varredura de laser, nessa abordagem o processo sintetiza-se na soma das discrepâncias ponderadas com base no modelo de crença.

A equação 3.17 descreve a discrepância em função das Assinaturas de Crença Básica de cada hipótese do modelo a partir de duas observações (1 e 2). Note que por facilidade de notação as hipóteses foram reduzidas a letra inicial de seu nome.

$$discrepância = |acb_1(o) - acb_2(o)| + acb_{1\oplus 2}(c) + acb_1(d) + acb_2(d) \tag{3.17}$$

Por outro lado as equações 3.18 e 3.19 definem os valores de pesos para a soma ponderada da nova função de custo.

$$w(p) = w'(p).\sqrt{\rho_p} \tag{3.18}$$

$$w'(p) = \begin{cases} 3, & 0.9 < |cos(\theta_p)|) \\ 2, & 0,8 < |cos(\theta_p)|) \leq 0,9 \\ |cos(\theta_p) + sin(\theta_p)| & senão \end{cases} \tag{3.19}$$

Nesse caso ρ_p é a distância de um obstáculo, θ_p é a direção do obstáculo e p é o ponto da varredura de laser. Dessa forma, haverá um grande peso para pontos na varredura de laser posicionados a frente do robô, permitindo portanto, uma maior robustez em localização dentro de corredores.

4 Metodologia

O trabalho de conclusão de curso presente, objetiva comparar os algoritmos de SLAM baseado em mapas geográficos citados na seção anterior com foco em buscar a melhor solução de mapeamento e localização simultânea para o robô assistente Zordon da categoria *robocup@home open challange* na Competição Latino Americana de Robótica. Para tal, é importante que as soluções avaliadas contemplem as seguintes diretrizes:

a) mapeamento robusto;

b) localização precisa;

c) inferência de localização e mapeamento rápida para navegação em tempo real;

d) complexidade computacional reduzida.

Num primeiro momento, quando deseja-se avaliar um mapeamento robusto de ambiente *indoor* (item a), comparações são necessárias. Para tal pode-se adotar de práticas de análise qualitativa sobre o resultado do mapeamento, com inferências na qualidade do mapeamento dos cantos, nas bordas, nas superfícies oblíquas e no transbordamento de píxeis das linhas retilíneas como paredes. (Nardi et al. 2015) define algumas métricas de comparações qualitativas de mapeamento provido por SLAMs. Tais métricas e o motivo de (Nardi et al. 2015) terem as definidos são:

1. proporção de células ocupadas e não ocupadas (livres), dessa forma pode-se inferir sobre o transbordamento de pixeis nas paredes, cuja erros no mapeamento podem produzir múltiplas visualizações de uma mesma parede;

2. quantidade de cantos no mapa mensura a precisão do mapa, pois dado um mapa impreciso espera-se que ele deva mapear mais cantos do que realmente existem no ambiente, devido à sobreposição de algumas partes inconcisas, como curvaturas de paredes retilíneas;

3. quantidade de áreas totalmente fechadas (e.g. um cômodo sem porta ou janelas visualizadas) como mesma explicação do item anterior, pois a sobreposição de pixeis podem ser detectadas caso um cômodo ou sala seja mapeado sem saída.

Vale ressaltar ainda que, desde que o pesquisador possua em mãos representações geométricas precisas do ambiente de testes, tais como plantas baixas ou mapeamento prévio confiável, ele pode usufruir de técnicas quantitativas de comparação. Pode-se chamar

essas representações precisas (isto é, gabaritos da representação geométrica do ambiente de teste) de *ground-truth* de mapas.

Para avaliar a qualidade dos mapas geométricos que alguns algoritmos de SLAM produzem, é necessário investigar o erro entre os mapas gerados e o mapa de *ground-truth*. Uma possível métrica, usada por (Santos, Portugal e Rocha 2013), pode ser utilizada para o cálculo desse erro em mapas de ocupação [1]. Nesse caso foi utilizada a classificação dos píxeis dos mapas pelo algoritmo *k-nearest neighbor* (k-nn ou K vizinhos próximos em tradução livre) da seguinte maneira: primeiro, converta o mapa gerado e o mapa de *ground-truth* em modelos binários. Esse mapa deve conter apenas os limites e obstáculos da cena observada. Depois os mapas foram sobrepostos de forma em que eles estivessem alinhados. Logo após, para cada célula ocupada do mapa do SLAM foi calculada sua distância ao mapa de *ground-truth* pelo busca do k-nn com parâmetro $k = 1$. Por fim soma-se todas as distâncias e divide pelo número de células ocupadas do mapa de *ground-truth*. Segundo (Santos, Portugal e Rocha 2013) essa métrica produz uma medida normalizada da distância em termos de células na qual pode ser aplicada em qualquer mapa de grade de ocupação desde que um mapa de *ground-truth* esteja disponível.

No entanto, para elaborar uma bom gabarito de representação de um ambiente típico de *robocup@home* deve-se levar em consideração o posicionamento preciso dos moveis e eletrodomésticos de uma casa com no mínimo um quarto, sala, cozinha e corredores. A figura 7 representa o *layout* da arena da competição internacional realizada na Alemanha em 2019.

No quesito b, localização precisa, geralmente abordagens quantitativas, com base em *ground truth* de trajetória percorrida pelo robô, são as mais adotadas para avaliar a performance de localização dos algoritmos de SLAM. (Korkmaz, Yılmaz e Durdu 2016) mostra que com base em uma referência precisa de trajetória, a comparação com a localização obtida pelos métodos de localização e mapeamento simultâneos pode ser dada pelo indicador Raiz do Erro Quadrático Médio (REQM), sendo que o menor REQM resulta na solução mais precisa. A equação que define o REQM pode ser e vista logo abaixo.

$$REQM = \sqrt{\frac{\sum_i^n (x_i - \widehat{x}_i)^2 + (y_i - \widehat{y}_i)^2}{n}} \tag{4.1}$$

Note que $(\widehat{x}, \widehat{y})$ denota pontos observados pelos algoritmos de SLAM enquanto (x, y) os pontos reais em que o veículo se encontra no *ground-truth* de trajetória.

Comparado com *ground truth* de mapa, um *ground truth* de trajetória é mais fácil de se obter na prática. Assim, embora essa afirmação leva a como usar um mapa de *ground truth* para avaliar o mapa estimado, comparações entre a trajetória percorrida

[1] Mapa de grade cuja cada célula representa um valor de 0 a 1, onde menor valor representa a menor probabilidade da célula estiver ocupada por um osbtáculo e quando maior o valor maior a probabilidade

Figura 7 – *Layout* da Arena 2019

Fonte: (Github de RoboCup@Home)

pelo robô e o *ground truth* de trajetória é frequentemente uma das métricas mais usadas em situações reais (Filatov et al. 2017). Dessa forma, a presente monografia ignora a comparação quantitativa de mapas e parte pela busca do algoritmo que produz a melhor localização. (Santos, Portugal e Rocha 2013).

Na sequência, o tempo de resposta dos algoritmos de SLAM é considerado rápido (item c) se para atualizar a localização e o mapa esses algoritmos demandam menos tempo do que geraram resultados para navegação. Geralmente um SLAM baseado em alcance de *laser* é considerado em tempo real caso a frequência de atualização dos dados inferidos seja tão rápida quanto a taxa de atualização do sensor de entrada (e.g. Lidar).

Sabendo que a taxa de publicação das mensagens do LIDAR é de 10 Hz e que foi utilizado uma implementação dos algoritmos de SLAM tal que a inferência sobre a localização ocorre na medida que uma mensagem de *Laser Scan* é recebida. Espera-se que a taxa de atualização de localização ocorra próxima de 5 Hz (aqui poderá ser um valor

abaixo da taxa de publicação do Lidar pois o algoritmo poderia computar em tempo maior do que o período de inserção de duas mensagens de entrada).

A equipe de robótica Pequi Mecânico quando optou pela aquisição do LIDAR da *SLAMTECH* sabia da limitação de tempo de amostragem deste, no entanto, pensaram em adquiri-lo para ser embarcado em aplicações de robótica móvel de veículos pequenos e com capacidade reduzida de locomoção. Espera-se que o robô Zordon desloque no máximo a 30 cm/s (devido a limitações de hardware). Assim sabendo que a localização inferida pelos SLAMs corrigem a resposta de odometria filtrada pelo filtro de Kalman, uma frequência de correção de aproximadamente 5Hz pode ser considerada ideal para a aplicação, visto que o algoritmo corrige nó máximo um deslocamento de até 6 cm, abaixo portanto do erro médio esperado pela inferência.

Sabe-se que a localização dada pela odometria das rodas fundida com a localização do IMU possui incertezas que crescem com o acúmulo de deslocamento angular e linear. Portanto, uma maneira de corrigir essa localização seria atualizá-la com uma fonte de observação de incerteza limitada no âmbito dos algoritmos providos pelo pacote *robot_localization*, como *ekf_localization_node* ou *ukf_localization_node*. Os algoritmos de SLAM produzem essa fonte de localização confiável para atualização dos filtros preditivos na frequência de atualização de suas inferências.

No entanto, dentro do ROS, o processo de atualização da localização odométrica é construída de maneira diferente. Aqui a correção do *drift* da odometria filtrada é o processo que dá forma ao objetivo de localização desses algoritmos. Para tal adota-se transformações em cima de eixos coordenados ligados em uma estrutura de dados de árvore (.i.e. *tf*[2]).

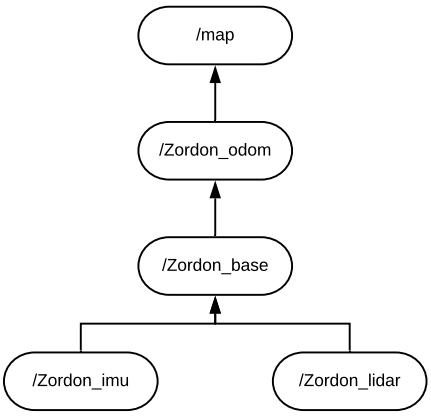
A figura 8 mostra como a árvore de eixos coordenados encontram-se definida no robô Zordon. Observa-se que os sensores possuem seus próprios eixos coordenados (*frames*) deslocados em referência ao eixo coordenado que define a posição central do veículo. Por sua vez o veículo locomove-se em referência ao *frame* da odometria. Sabendo que os algoritmos de SLAM estudados constroem mapas dos obstáculos visíveis pelo sensor LIDAR, a sua inferência de localização será sempre em relação a esses mapas. Daí a correção do *drift* de odometria é a transformação algébrica inversa do eixo de coordenada da base locomotora em referência ao *frame* do mapa gerado, aplicando esse valor em deslocamento do *frame* da odometria.

Com essa abordagem, enquanto o algoritmo de SLAM não computa a inferência de localização o veículo poderá ser localizado pela posição dada pela odometria filtrada com as informações do IMU.

Não obstante, a complexidade computacional (item d) é tida como proporcional ao tempo de resposta da abordagem de localização e mapeamento simultâneo. Isto pois, para

[2] Uma explanação detalhada do pacote *tf* encontra-se em (Foote 2013).

Figura 8 – Relação da árvore de transformações de eixos coordenados do robô Zordon



Fonte: produzido pelo autor.

um mesmo algoritmo de SLAM, quanto mais frequente é sua atualização, usualmente, mais utilização de CPU este demanda. Para tal, torna-se interessante limitar a frequência de atualização da inferência produzida pelo SLAM afim de alcançar um tempo de resposta mínimo que garanta as prerrogativas da navegação. Adortou-se aqui uma frequência de publicação da correção de odometria de 5 Hz e permitindo que o mapeamento feito pelos algoritmos *gmapping* e *vinySLAM* ocorra em tempo maior devido ao fato que os algoritmos de SLAM supra apenas a parte de mapeamento da competição, não a parte de navegação autônoma que demanda um mapa construído visualizado em tempo real. Levando em conta, assim, que a localização continuará a ser feita pelo algoritmo *Adaptive Monte Carlo Localization*.

4.1 Experimento em cena simulada de @home

Para permitir comparar os algoritmos de SLAM de forma justa (i.e. sem viés), foi feito num primeiro momento uma coleta de dados dos sensores do robô Zordon em um ensaio de navegação guiada por comandos remotos em uma casa com características semelhantes ao cenário de competição da categoria Robocup@home. Sendo esses dados providos dos sensores de odometria, lidar e IMU, pode-se montar uma conjunto de dados (também conhecido como bag de dados) armazenados na memória não-volátil do computador de bordo do robô.

Primeiramente definiu 23 marcações na casa de modo a permitir que o robô movimente sequencialmente por elas. Essas marcações foram criadas na casa com fita adesiva de cor verde para melhor visualização do operador do veículo. Adotou-se distâncias

proporcionais ao tamanho das cerâmicas do piso da casa (nesse caso cada cerâmica possui o formato de um quadrado de 43,5 cm). Uma sequência que simplifica essa marcação esta descrita a seguir:

1. define um ponto de inicialização na casa;
2. movimente uma distância X (proporcional ao comprimento de uma cerâmica) para frente até aproximar-se de um obstáculo;
3. demarca o ponto alcançado
4. girar 90º em sentido tal que permita desviar do obstáculo sem prejuízo de acquição de dados da cena observada;
5. repita o item 3 e 4 até conseguir demarcar todos os pontos que permita a navegação de todos os cômodos de uma casa.

Vale ressaltar que a casa para simulação de um cenário de @home possui vário níveis de pisos e foi escolhido para essa experimentação apenas um nível que possui 1 quarto com cama e guarda-roupas, uma sala com estantes e sofás, uma cozinha com eletrodomésticos típicos de uma casa de @home (armários, geladeira, fogão, etc.), e uma dispensa com móveis e armários. Todas as portas foram devidamente fechadas, tanto as de madeira quanto as de vidro.

Esse conjunto de dados foi adquirido pelo utilitário de armazenamento de informações do ROS *rosbag*. A sequência de comandos do ROS para executar a coleta de dados pode ser descrita de seguinte maneira:

1. inicializar todos os sensores e nós de publicação de dados destes;
2. energizar a base de navegação e inicializar o nó *rosserial_python* para permitir a correta iteração entre o acionamento dos motores com o ROS;
3. executar o nó *ekf_localization_node* do pacote *robot_localization* com parâmetros que permitam a filtragem de valores da odometria a partir de dados do IMU;
4. preparar o nó de controle remoto (*teleop_twist_key* do pacote *teleop*) para aplicar velocidades à base do veículo;
5. inicializar a coleta de dados providos pelos tópicos de publicação do algoritmos dos itens anteriores;
6. guiar o veículo com o programa do item 4 aos pontos demarcados na casa;

7. para cada ponto demarcado publicar uma mensagem literal no tópico */checkpoint* com a mensagem "cheguei";

8. ao finalizar a locomoção guiada do veículo interromper o *rosbag* do item 5.

4.2 Avaliação dos dados coletados

A partir do momento em que o conjunto de dados foi salvo, então reproduziu ele novamente no ROS em outro computador com ele instalado. Nesse sentido, aplicou-se a reprodução do *bag* para teste de performance dos algoritmos analisados por essa monografia dentro do microprocessador embarcado *Jetson TX2* da equipe de robótica Pequi Mecânico.

Para cada algoritmo realizou-se 30 ensaios de performance em cima dos dados coletados. Cada ensaio de performance consistia basicamente das seguintes regras de execução dentro do ambiente ROS:

1. configurar o parâmetro *"use_sim_time"* para valor booliano verdadeiro;

2. definir uma velocidade de reprodução do *bag* coletado para 5 vezes mais rápido do que a velocidade original deste;

3. inicializar o nó do algoritmo de SLAM com os parâmetros de configuração devidamente escolhidos para navegação autônoma de veículos em um ambiente semelhante à pista de competição do @home;

4. executar o script desenvolvido em Python *publish_point.py*

5. inicializar um novo *bag* para coletar apenas as publicações dos pontos produzidos pelo algoritmo do item anterior;

6. reproduzir o *bag* coletado no estágio de navegação por comandos remotos com o utilitário *rosbag play* com velocidade de reprodução definida no item 2;

7. ao terminar a reprodução do bag do item anterior interromper o comando produzido no item 5.

O utilitário desenvolvido em Python *publish_point.py* traduz em pontos (x, y, z) da posição da base observados pelo algoritmo de SLAM experimentado para cada instante em que o tópico */checkpoint* recebeu uma mensagem "cheguei". No fim, o ultimo *bag* coletado contém um arquivo armazenado contendo todos os 23 pontos inferidos por cada SLAM.

Tendo os pontos inferidos por cada SLAM pode-se comparar com o *ground-truth* de trajetória construído pelas marcações das fitas verdes 4.1 e medidos por via de trenas. O erro de distância pode ser dado pela distância euclidiana no plano (equação 4.2)

$$d = \sqrt{(x_i - \widehat{x}_i)^2 + (y_i - \widehat{y}_i)^2} \tag{4.2}$$

Para limitar o consumo de CPU dos processos computacionais dos algoritmos de SLAM dessa monografia, adotou-se do utilitário *cpulimit* [3] sistema operacional Linux. O método para avaliar qual algoritmo representa a melhor performance com o menor consumo de processadores pode ser descrito a seguir:

1. executa ambos os algoritmos sem limitação de CPU;
2. deixe ele percorrer 5 minutos de execução;
3. enquanto ocorre o item 2, execute os comandos do ros:
    - rosrun tf view_frames (aqui modificado para ouvir o tópico /tf por 5 min);
    - rosrun hz /map -w 1000 (para visualizar a taxa de publicação do mapa)
4. verifica se a taxa de publicação do mapa encontra dentro de um valor aceitável 0,2 Hz;
5. verifica se a taxa de publicação da transformação /map -> /odom que encontra-se no arquivo *.pdf* do primeiro comando do item 3 esta acima de 5 Hz;
6. aplique o limitador de CPU (cpulimit) diminuindo 3% da CPU gasta anteriormente e repita os itens 2 a 6 até o item 4 e 5 produzir resultado não desejado.

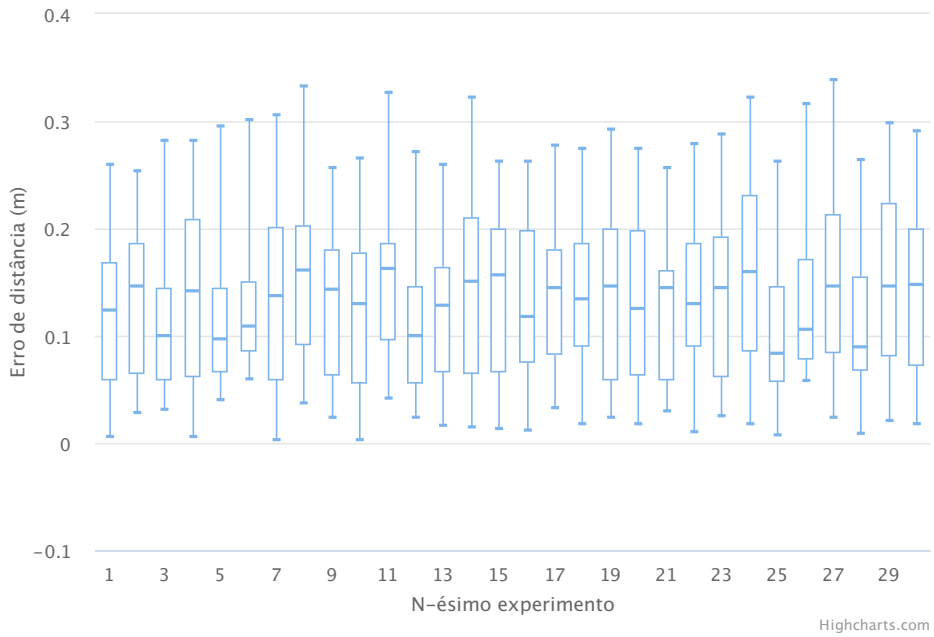
[3] Algumas versões do sistema operacional Linux não possui esse utilitário nativamente, pode-se encontrar o código-fonte assim como descrição de sua instalação no site <https://github.com/opsengine/cpulimit> acessado em 15 de novembro de 2019.

5 Resultados e Análise

Este capítulo apresenta os resultados observados no experimento de comparação da performance dos algoritmos *gmapping* e *vinySLAM*.

Primeiramente, buscou-se encontrar os erros de deslocamento para cada ponto demarcado no chão (como consta em 4.2 equação 4.2). Para os 30 experimentos realizados (a partir 30 a média e variância basicamente não muda), em cima do mesmo *bag* de dados coletados, para o algoritmo *gmapping*, esses erros podem ser visto na Figura 9. Note que a distribuição dos erros apresenta-se uniforme, com mediana próxima de 13cm e com erro menor do que 40cm para o pior caso da amostragem, sendo ainda que a distribuição dos erros (primeiro ao terceiro quartil) encontra-se densa.

Figura 9 – Distribuição dos erros de deslocamento por experimento do algoritmo *gmapping*



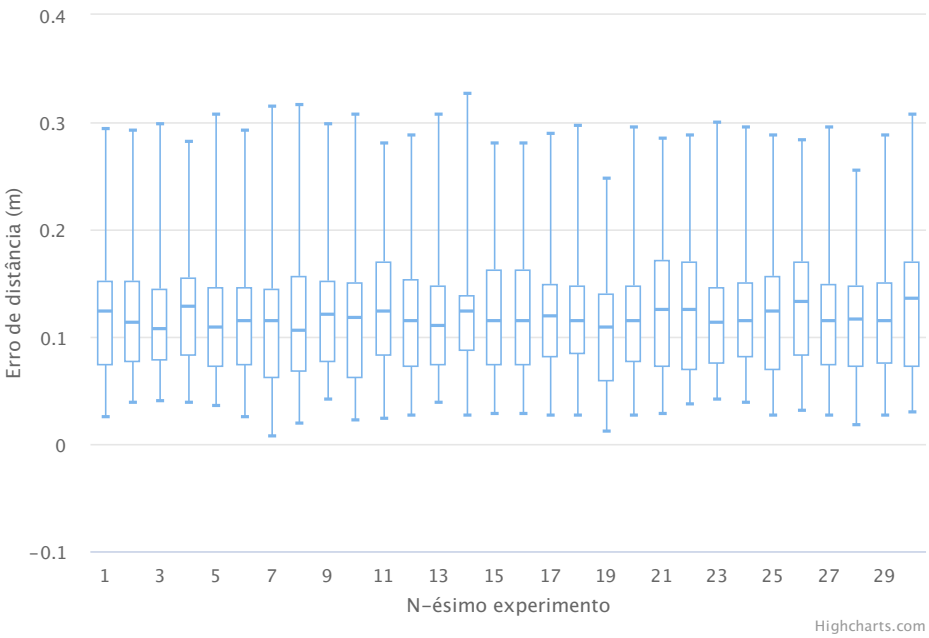
Fonte: dados coletados.

A partir dessa visualização pode-se observar que o algoritmo *gmapping* consegue eliminar os erros não sistêmicos de localização, diferente da localização baseada apenas em odometria e IMU onde derrapagem e mal modelagem de controle leva ao acúmulo de erro incremental.

Não obstante, a figura 10 traz a distribuição dos erros de deslocamento para cada ponto demarcado aplicado aos 10 experimentos de localização e mapeamento do algoritmo *vinySLAM*. Percebe-se que esse algoritmo produz erros de deslocamento máximo inferior

a 0,35cm e com distribuição concentrada entre 15cm a 7cm, com mediana variando em torno de 10cm a 13cm.

Figura 10 – Distribuição dos erros de deslocamento por experimento do algoritmo *vinySLAM*



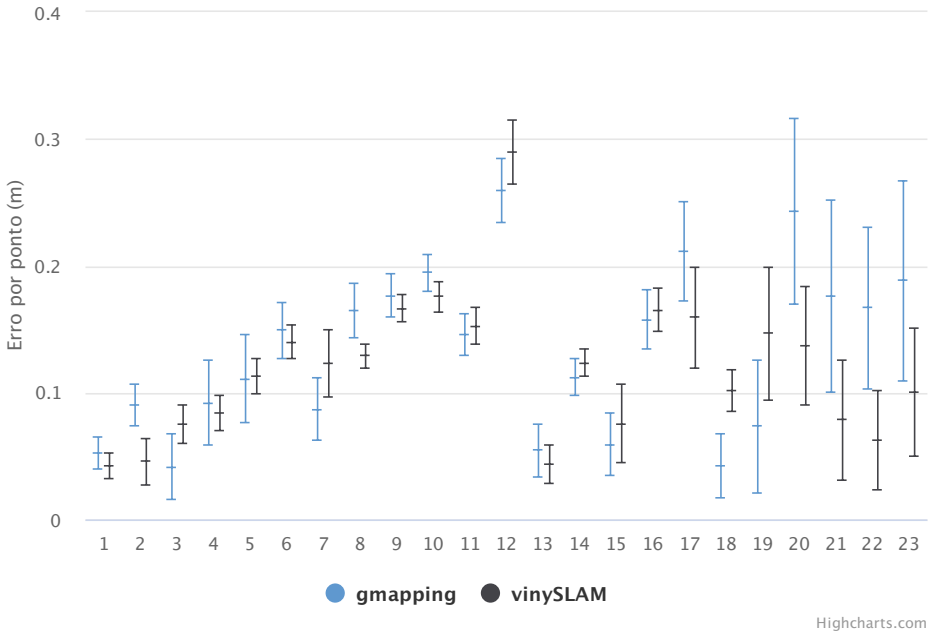
Fonte: dados coletados.

Observa-se que para esse algoritmo a distribuição de erros de deslocamento em função das demarcações é uniforme nos dez experimentos. Isto é devido ao modelo de discrepância baseado no Modelo de Crença Transferível para cálculo da função de custo do algoritmo de *scan matcher* é uma implementação que traz resultados compatível com os encontrados no algoritmo *gmapping*.

Por outro lado, observou-se que alguns pontos específicos obteve maior dificuldade de localização. A figura 11 mostra o erro de cada um dos 26 pontos do percurso em ordem crescente, sendo executado a média e o desvio padrão referente aos 30 experimentos para cada algoritmo. Observa-se que o 12º ponto (localizado na sala frente ao sofá voltado para estante) foi o de maior erro encontrado. Analisando o percurso, observa-se na figura 12 que esse ponto possui o maior erro da odometria, assim, pode-se afirmar que as modalidades de SLAM são dependentes a informação odométrica e tendem a falhar quando estão imersas nesse contexto.

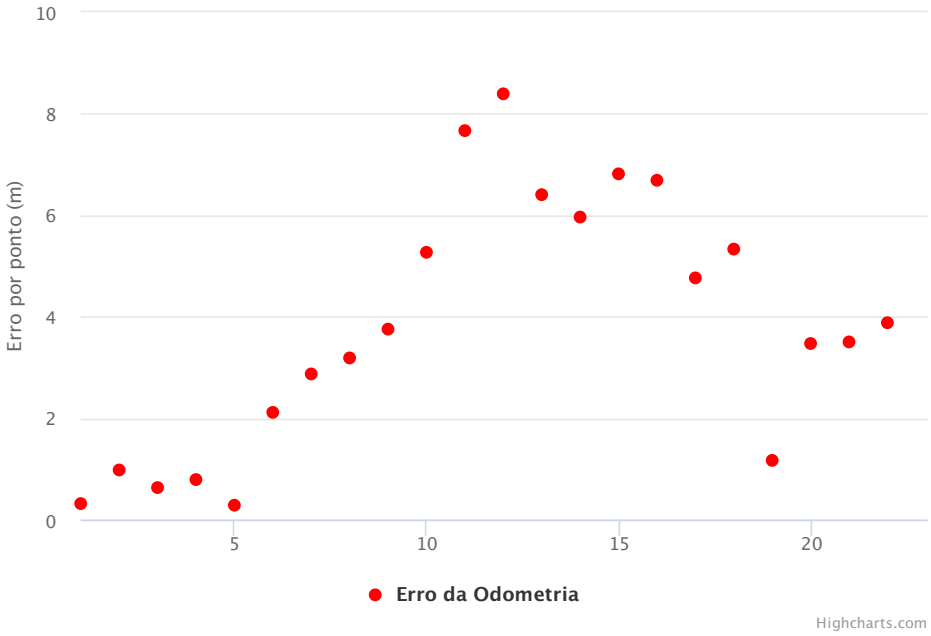
A fim de comparação entre os algoritmos *gmapping* e *vinySLAM* buscou-se encontrar um estimador capaz de aferir sobre a qualidade da localização inferida por estes. A média da raiz do erro quadrático médio e o desvio padrão calculado para os dez experimentos de cada algoritmo encontra-se na tabela 5.

Figura 11 – Distribuição dos erros de deslocamento por experimento do algoritmo



Fonte: dados coletados.

Figura 12 – Distribuição dos erros de deslocamento da odometria por experimento



Fonte: dados coletados.

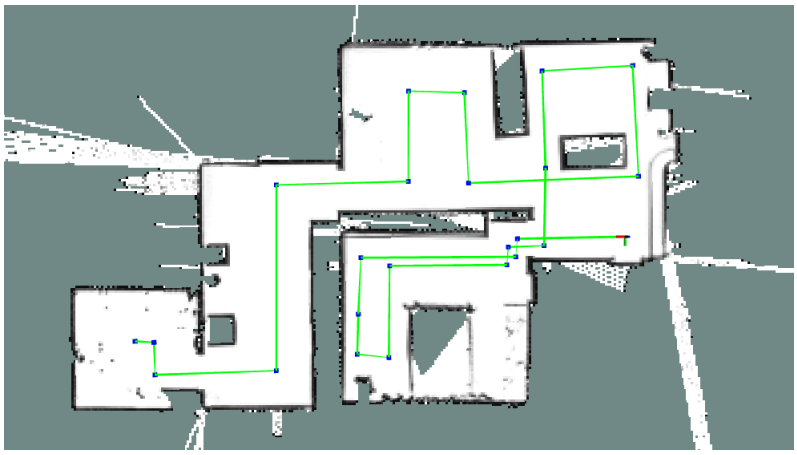
Tabela 1 – Raiz do erro quadrático médio por algoritmo

	gmapping (m)	*vinySLAM* (m)
REQM	0.15084 ± 0.01758	0.13330 ± 0.00453

Aqui observa-se que o algoritmo *vinySLAM* obteve o melhor resultado de localização, com menor erro encontrado para esse estimador.

Concomitante a análise dos resultados de localização dos algoritmos de SLAM estuados, pôde-se chegar em resultados de mapeamento compatíveis com a realidade de simulação de uma casa de @home. As figuras 13 e 14 mostra os mapas inferidos pelos algoritmos no final da navegação comandada remotamente pelos 23 pontos.

Figura 13 – Mapa gerado pelo algoritmo *gmapping*



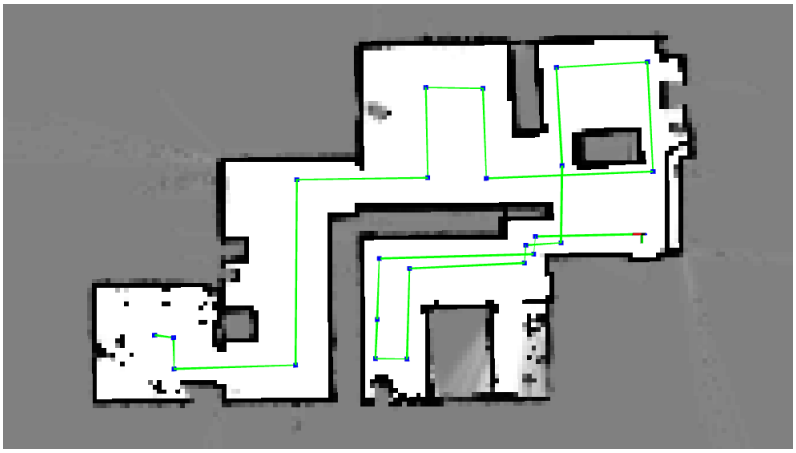
Fonte: coletado na pesquisa.

Nessas imagens, os quadrados azuis definem os pontos de localização encontrados por cada SLAM quando o robô passe por cima da demarcação do piso. As retas verdes são seguimentos que ilustram os caminhos de locomoção do veículo. Vale ressaltar que a visualização das linhas verdes não representa corretamente a locomoção do robô durante o percurso, mas apenas uma visualização estimada de sua trajetória.

Note que ambos os mapas gerados são semelhantes. Todavia, o mapa gerado pelo *vinySLAM* apresentou melhor resultados qualitativos (menos transbordamentos de pixeis nas bordas) do que o *gmapping*.

Outro ponto positivo do algoritmo *vinySLAM* está no modelo de geração dos mapas dada pela inferência sobre a probabilidade de as células do mapa de ocupância estejam ocupadas. Nessa abordagem, as células cuja a varredura de laser não alcançou com qualidade, a probabilidade delas serem ocupadas é menor do que no algoritmo *gmapping*.

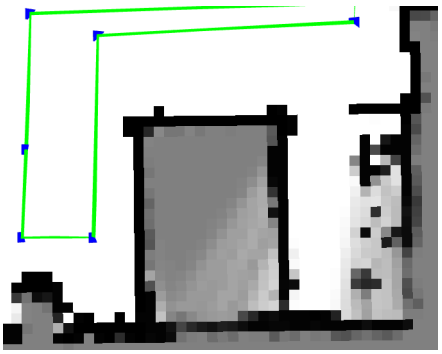
Figura 14 – Mapa gerado pelo algoritmo *vinySLAM*



Fonte: coletado na pesquisa.

Isso permite ao operador do veículo identificar possíveis pontos de falha de localização antes da etapa de mapeamento da competição de *@home*. Ainda mais, o modelo baseado em crenças transferíveis (equações 3.11 a 3.19), por usar a qualidade do sinal de LASER do LIDAR conseguiu no caso da figura 15, inferir melhor que a cama como um obstáculo.

Figura 15 – Cama visualizada pelo algoritmo *vinySLAM*



Fonte: coletado na pesquisa.

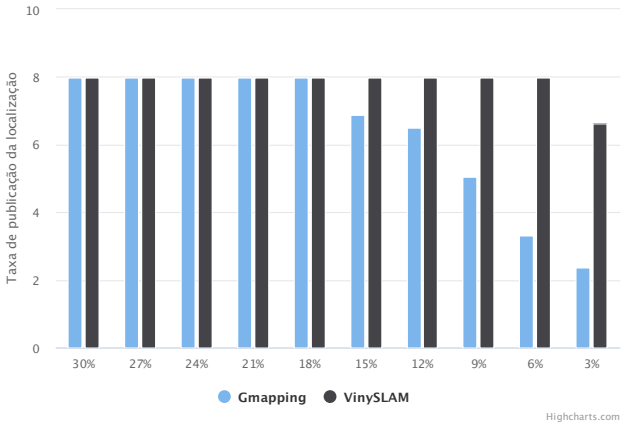
Doravante, avaliando os resultados de consumo computacional dos algoritmos (5) dentro dos 6 núcleos do microprocessador da *Nvidia Jetson TX2*, pode-se perceber que o algoritmo *vinySLAM* alcançou 3% de CPU enquanto *gmapping* resultou em 10%. Vale ressaltar que com esses valores resultaram em uma taxa de publicação da localização média acima de 5Hz.

A figura 16 apresenta a taxa de publicação média a cada 60 segundos de um único experimento em função da limitação de CPU imposta aos processos computacionais dos

Tabela 2 – Uso médio CPU por algoritmo em porcentagem

	gmapping	*vinySLAM*
Uso de CPU	10%	3%

Figura 16 – Taxa de publicação da localização em função do consumo de CPU por algoritmo.



Fonte: coletado na pesquisa.

algoritmos estudados. Percebe-se então que o algoritmo *gmapping* a aproximadamente 9% de CPU começa a ter sua taxa de publicação de localização abaixo do limitante proposto por essa pesquisa 4.

No caso do *vinySLAM* não foi possível limitar seu processo abaixo de 3% para averiguar o ponto crítico da taxa de publicação pois o utilitário *cpulimit* não consegue impor limitação de CPU para valores próximos de zero. Portanto, pode-se concluir que o algoritmo *vinySLAM* possui o melhor custo benefício de performance versus complexidade computacional.

6 Conclusão

Esse trabalho comparou os resultados da aplicação dos algoritmos de Localização e Mapeamento Simultâneo no robô Zordon da modalidade *open challenge RoboCup@home* da primeira equipe de robótica desta categoria no Pequi Mecânico e da Universidade Federal de Goiás. Analisou qual seria o melhor algoritmo que responde de forma eficiente pelo *trade-off* de acurácia e custo computacional.

Para tal estudou os algoritmos *gmapping* e *vinySLAM*. O primeiro sendo uma implementação melhorada do famoso algoritmo de localização e mapeamento simultâneo *FAST-SLAM* (filho direto do ramo de SLAMs que herdam características do primeiro SLAM probabilístico, i.e. EKF-SLAM), onde a incorporação do filtro de partículas Rao-Blackwellized juntamente com uma nova técnica de reamostragem adaptativa.

No mesmo sentido, o algoritmo *vinySLAM* define-se como uma melhoria da abordagem barata computacionalmente *tinySLAM*. Sendo que este implementa melhorias na função de custo da busca heurística dada pelo algoritmo de Monte-Carlo (aqui adaptado para proporcionar uma implementação de baixa complexidade computacional de *scan matcher*); além da melhoria no âmbito do Modelo de Crenças Transferíveis para realização do mapeamento de grade.

Não obstante, esse projeto constratou soluções de Localização e Mapeamento Simultâneo com localização baseada em mapa, demostrando soluções como *Adaptative Monte-Carlo Localization*, *EKF Localization* e *Multi-Hypothesis Tracking* que resulta apenas sobre a localização e ignora inferência sobre o mapeamento.

Desenvolveu uma metodologia de coleta de dados sensoriais baseado no *framework* de aplicações robóticas ROS. Dado que esse sistema de programação voltado a robôs e veículos autônomos possui uma vasta gama de pacotes e utilitários de código aberto, adotou-se do uso de mensagens protocoladas para sensores e nós que subscrevem e publicam seus valores.

Desse modo, fez-se uso dos pacotes *robot_localization*, *tf*, *teleop*, *rplidar*, *rosserial* e *slam-constructor*. E dos utilitários *rosbag* para coleta padronizada de dados e *view_frames* para avaliar a performance da localização. Além do uso do utilitário exógeno ao ROS *cpulimit* para realizar limitações dos processos computacionais de cada SLAM estudado.

Buscou-se ainda implementar um método de avaliação de performance para melhor concluir sobre a comparação de algoritmos de SLAM focados na proposta do robô Zordon. Sendo que esse método pôde trazer resposta aos objetivos dessa pesquisa.

Além do mais, competições acadêmicas elevam a qualidade da ciência dos centros

de pesquisa. A categoria *robocup@home* surge com o objetivo de fomentar o elo de iteração homem-robô com intuito de alcançar o ponto de singularidade tecnológica onde homens e máquinas possam compartilhar esforços em função do bem-estar individual e social.

Nesse sentido, localização torna-se uma das tarefas mais relevantes para a competições que envolvam veículos móveis, dado a necessidade de responder as perguntas: "onde o robô está?"e "para onde o robô deva ir?"em ambiente com existência de obstáculos. Logo a navegação poderia se resumir em como controlar os atuadores da base em função de trajetórias estipuladas por veículos autônomos.

Conquanto, para planejar e caminhar sobre trajetórias de ambientes complexos uma casa ou um *shopping center*, é necessário uma representação metrificada do ambiente a se locomover. Isso leva ao problema de mapear. Foi explicado que no estado-da-arte da abordagem de mapeamento, os algoritmos de SLAM sobressaem aos demais. Portanto, a relevância desse tema é deveras importante para o meio acadêmico da robótica.

Esta pesquisa constatou que o algoritmo *vinySLAM* é um pouco melhor do que *gmapping* em performance de localização, produz mapas semelhante e qualitativamente iguais. Contudo, no critério de resposta rápida ele destaca-se como a melhor abordagem de localização e mapeamento simultâneo para a equipe do grupo de robótica do Pequi Mecânico.

Por conseguinte, o objetivo geral proposto por essa monografia foi alcançado. Todavia conclui-se que a método de comparação adotado não alcança o objetivo secundário de prover um *framework* de comparações de algoritmos de SLAM, devido a baixa qualidade do *ground-truth* de trajetória proposto comparado com os existentes na literatura.

No mais, os resultados alcançados do algoritmo *vinySLAM* poderia ser aplicado futuramente em outras modalidades de categorias do grupo de robótica da Universidade Federal de Goiás, tais como *IEEE open* e *O DESAFIO DE ROBÓTICA PETROBRAS/ROBOCUP BRASIL* na qual a maior limitação de *hardware* poderia levar à aplicação dessa abordagem em microprocessadores menos parrudos como *raspberry pi* ou *Nvidia Jetson Nano*.

Referências

Borenstein, J. et al. Mobile robot positioning: Sensors and techniques. *Journal of robotic systems*, Wiley Online Library, v. 14, n. 4, p. 231–249, 1997. Citado 4 vezes nas páginas 19, 20, 21 e 22.

BORENSTEIN, J.; FENG, L. Measurement and correction of systematic odometry errors in mobile robots. *IEEE Transactions on robotics and automation*, New York, NY: Institute of Electrical and Electronics Engineers, c1989-c2004., v. 12, n. 6, p. 869–880, 1996. Citado 2 vezes nas páginas 19 e 21.

CHEN, L.; HU, H.; MCDONALD-MAIER, K. Ekf based mobile robot localization. In: IEEE. *2012 Third International Conference on Emerging Security Technologies*. [S.l.], 2012. p. 149–154. Citado na página 15.

Chitta, Sucan e Cousins. Moveit![ros topics]. *IEEE Robotics & Automation Magazine*, IEEE, v. 19, n. 1, p. 18–19, 2012. Citado na página 22.

Crowley e Reignier. Asynchronous control of rotation and translation for a robot vehicle. *Robotics and Autonomous Systems*, Elsevier, v. 10, n. 4, p. 243–251, 1992. Citado na página 20.

DURRANT-WHYTE, H.; BAILEY, T. Simultaneous localization and mapping: part i. *IEEE robotics & automation magazine*, IEEE, v. 13, n. 2, p. 99–110, 2006. Citado 2 vezes nas páginas 26 e 27.

FILATOV, A. et al. 2d slam quality evaluation methods. In: IEEE. *2017 21st Conference of Open Innovations Association (FRUCT)*. [S.l.], 2017. p. 120–126. Citado 3 vezes nas páginas 16, 30 e 37.

FOOTE, T. tf: The transform library. In: *Technologies for Practical Robot Applications (TePRA), 2013 IEEE International Conference on*. [S.l.: s.n.], 2013. (Open-Source Software workshop), p. 1–6. ISSN 2325-0526. Citado na página 38.

FOX, D. Kld-sampling: Adaptive particle filters. In: *Advances in neural information processing systems*. [S.l.: s.n.], 2002. p. 713–720. Citado na página 15.

GITHUB de RoboCup@Home. <https://github.com/RoboCupAtHome/GermanOpen2019>. Accessed: 2019-12-03. Citado na página 37.

Grisetti, G. et al. Improved techniques for grid mapping with rao-blackwellized particle filters. *IEEE transactions on Robotics*, v. 23, n. 1, p. 34, 2007. Citado 3 vezes nas páginas 16, 28 e 29.

HAHNEL, D. et al. An efficient fastslam algorithm for generating maps of large-scale cyclic environments from raw laser range measurements. In: IEEE. *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003)(Cat. No. 03CH37453)*. [S.l.], 2003. v. 1, p. 206–211. Citado na página 16.

Huletski, Kartashov e Krinkin. Vinyslam: an indoor slam method for low-cost platforms based on the transferable belief model. In: IEEE. *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. [S.l.], 2017. p. 6770–6776. Citado 4 vezes nas páginas 16, 30, 31 e 32.

Jensfelt e Kristensen. Active global localization for a mobile robot using multiple hypothesis tracking. *IEEE Transactions on Robotics and Automation*, IEEE, v. 17, n. 5, p. 748–760, 2001. Citado na página 15.

KORKMAZ, M.; YILMAZ, N.; DURDU, A. Comparison of the slam algorithms: Hangar experiments. In: EDP SCIENCES. *MATEC Web of Conferences*. [S.l.], 2016. v. 42, p. 03009. Citado na página 36.

Ljung. Asymptotic behavior of the extended kalman filter as a parameter estimator for linear systems. *IEEE Transactions on Automatic Control*, IEEE, v. 24, n. 1, p. 36–50, 1979. Citado na página 20.

MARDER-EPPSTEIN, E. et al. The office marathon: Robust navigation in an indoor office environment. In: *International Conference on Robotics and Automation*. [S.l.: s.n.], 2010. Citado na página 22.

MONTEMERLO, M. et al. Fastslam: A factored solution to the simultaneous localization and mapping problem. *Aaai/iaai*, v. 593598, 2002. Citado 2 vezes nas páginas 15 e 27.

MONTEMERLO, M. et al. Fastslam 2.0: An improved particle filtering algorithm for simultaneous localization and mapping that provably converges. In: *IJCAI*. [S.l.: s.n.], 2003. p. 1151–1156. Citado 2 vezes nas páginas 15 e 29.

NARDI, L. et al. Introducing slambench, a performance and accuracy benchmarking methodology for slam. In: IEEE. *2015 IEEE International Conference on Robotics and Automation (ICRA)*. [S.l.], 2015. p. 5783–5790. Citado na página 35.

OLIVEIRA, P. R. G. d. *Auto-localização e construção de mapas de ambiente para robôs móveis baseados em visão omnidirecional estéreo*. Tese (Doutorado) — Universidade de São Paulo, 2008. Citado na página 27.

QUIGLEY, M. et al. Ros: an open-source robot operating system. In: KOBE, JAPAN. *ICRA workshop on open source software*. [S.l.], 2009. v. 3, n. 3.2, p. 5. Citado na página 23.

ROSWIKI nav_msgs. <http://wiki.ros.org/nav_msgs>. Accessed: 2019-10-17. Citado na página 24.

ROSWIKI rplidar. <http://wiki.ros.org/rplidar>. Accessed: 2019-10-17. Citado na página 25.

ROSWIKI std_msgs. <http://wiki.ros.org/std_msgs>. Accessed: 2019-10-17. Citado na página 24.

SANTOS, J. M.; PORTUGAL, D.; ROCHA, R. P. An evaluation of 2d slam techniques available in robot operating system. In: IEEE. *2013 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*. [S.l.], 2013. p. 1–6. Citado 3 vezes nas páginas 30, 36 e 37.

SENSORTEC, B. Bno055 intelligent 9-axis absolute orientation sensor. *Bosch Sensortec, Baden-Württemberg, Germany*, 2016. Citado na página 21.

SIEGWART, R.; NOURBAKHSH, I. R.; SCARAMUZZA, D. Autonomous mobile robots. *A Bradford Book*, MIT press, 2011. Citado na página 15.

SMETS, P. Decision making in the tbm: the necessity of the pignistic transformation. *International Journal of Approximate Reasoning*, Elsevier, v. 38, n. 2, p. 133–147, 2005. Citado na página 33.

SMETS, P.; KENNES, R. The transferable belief model. *Artificial intelligence*, Elsevier, v. 66, n. 2, p. 191–234, 1994. Citado na página 30.

SMITH, R. C.; CHEESEMAN, P. On the representation and estimation of spatial uncertainty. *The international journal of Robotics Research*, Sage Publications Sage CA: Thousand Oaks, CA, v. 5, n. 4, p. 56–68, 1986. Citado 2 vezes nas páginas 15 e 27.

Steux e Hamzaqui. tinyslam: A slam algorithm in less than 200 lines c-language program. In: IEEE. *2010 11th International Conference on Control Automation Robotics & Vision*. [S.l.], 2010. p. 1975–1979. Citado 4 vezes nas páginas 16, 29, 30 e 31.

Thrun, S. et al. Probabilistic algorithms and the interactive museum tour-guide robot minerva. *The International Journal of Robotics Research*, SAGE Publications, v. 19, n. 11, p. 972–999, 2000. Citado 3 vezes nas páginas 15, 16 e 26.

WISSPEINTNER, T. et al. Robocup@home: Scientific competition and benchmarking for domestic service robots. *Interaction Studies*, v. 10, p. 392–426, 12 2009. Citado na página 15.