Localização de robô móvel sujeita a ambientes sem características

Claudio de Souza Brito

CENTRO DE INFORMÁTICA
UNIVERSIDADE FEDERAL DA PARAÍBA

João Pessoa, 2022

Claudio de Souza Brito

Localização de robô móvel sujeita a ambientes sem características

Monografia apresentada ao curso Engenharia da computação
do Centro de Informática, da Universidade Federal da Paraíba,
como requisito para a obtenção do grau de Bacharel em Engenharia
da computação

Orientador: Tiago Pereira do Nascimento

Junho de 2022

Catalogação na publicação
Seção de Catalogação e Classificação

B862l Brito, Claudio de Souza.
Localização de robô móvel sujeita a ambientes sem características / Claudio de Souza Brito. - João Pessoa, 2022.
45 f. : il.

Orientação: Tiago Pereira do Nascimento.
TCC (Graduação) - UFPB/CI.

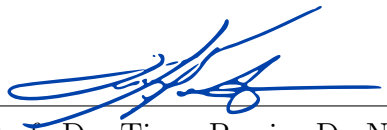
1. Robótica. 2. Algoritmo. 3. Trajetória. 4. Controlador. I. Nascimento, Tiago Pereira do. II. Título.

UFPB/CI CDU 004.896

Elaborado por ROGERIO FERREIRA MARQUES - CRB-15/690

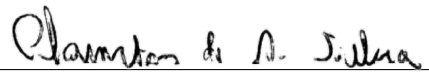
CENTRO DE INFORMÁTICA
UNIVERSIDADE FEDERAL DA PARAÍBA

Trabalho de Conclusão de Curso de Engenharia da computação intitulado ***Localização de robô móvel sujeita a ambientes sem características*** de autoria de Claudio de Souza Brito, aprovada pela banca examinadora constituída pelos seguintes professores:



Prof. Dr. Tiago Pereira Do Nascimento
Universidade Federal da Paraíba

Prof. Dr. Alisson Vasconcelos De Brito
Universidade Federal da Paraíba



Prof. Dr. Clauirton De Albuquerque Siebra
Universidade Federal da Paraíba

VERONICA MARIA LIMA SILVA:02558212397 Assinado de forma digital por VERONICA MARIA LIMA SILVA:02558212397 Dados: 2022.06.27 15:21:48 -03'00'

Coordenador(a) do Curso de Engenharia da Computação
Verônica Maria Lima Silva
CI/UFPB

João Pessoa, 23 de junho de 2022

Centro de Informática, Universidade Federal da Paraíba
Rua dos Escoteiros, Mangabeira VII, João Pessoa, Paraíba, Brasil CEP: 58058-600
Fone: +55 (83) 3216 7093 / Fax: +55 (83) 3216 7117

”Nem todos os que vagueiam estão perdidos”

J.R.R.Tolkien

AGRADECIMENTOS

Primeiramente agradeço aos meus pais por terem me educado, e lutado tanto para que eu tivesse as melhores condições de ensino para poder ter chegado até aqui. Agradeço à Universidade Federal da Paraíba e ao laboratório LaSER por ter me ensinado tanto. Também agradeço ao professor Dr. Tiago Pereira Do Nascimento por ter aceitado ser meu orientador, e me guiado durante esses últimos 12 meses de estudo. E por fim agradeço a empresa Automni por essa parceria.

RESUMO

A localização de robôs em ambientes sem características é um problema muito atual na área de robótica, presente em empresas de logística e *picking* que tem o objetivo de usar robôs para mover prateleiras de um armazém, a fim de poupar esforço humano. Para essa tarefa foram implementados algoritmos clássicos da área, como: o A* para planejamento de trajetória, e o NMPC (Nonlinear Model Predictive Control) como controlador. Usando as ferramentas disponíveis no ROS (Robotic Operating System), foi desenvolvido um mapa simulado para testes, além de integrados o EKF (Extended Kalman Filter) e o AMCL (Adaptive Monte Carlo Localization), para resultar em uma boa precisão com desvio padrão de poucos centímetros.

Palavras-chave: Robótica; Localização; Trajetória; Características; Controlador.

ABSTRACT

Robot localization in featureless environments is a current problem in robotics, present in logistics and picking companies that aim to use robots to move shelves in a warehouse, in order to save human effort. For this task, classical algorithms in the area were implemented, such as: A* for trajectory planning, and NMPC (Nonlinear Model Predictive Control) as a controller. Using the tools available in ROS (Robotic Operating System), a simulated map was developed for testing, in addition to integrating the EKF (Extended Kalman Filter) and AMCL (Adaptive Monte Carlo Localization), to result in a good accuracy with standard deviation of a few centimeters.

Key-words: Robotics; Localization; Trajectory; Featureless; Controller.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS

ROS – Robot Operating System

A* – A estrela

NMPC – NonLinear Model Predictive Controller

SLAM - Simultaneous Localization and Mapping

LiDAR - Light Detection And Ranging

LASER - Laboratório de Sistemas Embarcados e Robótica

EKF - Extended Kalman Filter

UFPB - Universidade Federal da Paraíba

AMCL - Adaptative Monte Carlo Localization

PRM - Probabilistic RoadMap

Sumário

1 INTRODUÇÃO

Robótica é a área da computação que estuda as técnicas e tecnologias para a concepção e uso de robôs. Os robôs são conjuntos de dispositivos que realizam atividades e movimentos simples ou complexos conforme foi programado. A robótica tem grande aplicação em diversas áreas desde a produção industrial[1], que introduz 85 mil robôs aos mercados todo ano, medicina[2], que possui IA e robótica como ferramentas indispensáveis no futuro, e também atividades domésticas com os famosos robôs de limpeza[3], onde nesta área a introdução da robótica é menos por profissionalismo e mais por conforto. É notável que a área de robótica existe para a de otimização de tempo e trabalho, esse padrão se repete neste projeto.

1.1 Definição do Problema e justificativa

Este projeto é em parceria com a empresa Automni, a qual financia a construção do robô de logística e *picking* L1BR desenvolvido no LaSER, laboratório da UFPB de robótica. Esta empresa está financiando a construção do primeiro robô L1BR, que tem a habilidade de movimentar prateleiras por baixo, evitando esforço humano, e que deverá ficar pronto até o fim do ano de 2021. O restante dos recursos para desenvolvimento do projeto será fornecido pelo LaSER.

Parecidos com os robôs de limpeza doméstica, os robôs usados em *warehouses* (armazéns) são terrestre e possuem estatura baixa, e são usados para movimentar prateleiras e outros itens pelo armazém, que é considerado um ambiente sem característica (como vãos e espaços abertos). Para um robô móvel de baixa estatura, isso se torna um fator complicador, um desafio na localização de robôs móveis, pois os cálculos de localização são feitos usando como base os objetos ao redor, um robô baixo, com um alcance *laser* limitado, tem dificuldade para conseguir captar o ambiente. Outro fator complicador é que as poucas características visuais existentes são repetitivas e similares (janelas, colunas, etc), podendo confundir o algoritmo. O LaSER, possui tal espaço com uma grande área de circulação e com características repetitivas, ambiente apropriado para experimentação.

A Figura 1 mostra a semelhança física entre robôs de limpeza e robôs de *warehouses*.

1.2 Premissas e Hipóteses

Há várias ferramentas no ramo da robótica que podem nos ajudar a resolver esse problema. Algoritmos do tipo SLAM conseguem entregar representações do mapa local através de sensores. O Hector SLAM[4], por exemplo, pois este foi criado para utilizar sensores *laser* como o LiDAR.

Figura 1: Robô de limpeza (esquerda) ao lado de robôs de logística e picking (direita)

Fonte: Montagem do autor[6][7].

Para a localização, a fusão de sensores através de filtros de Kalman[5], e algoritmos de extração e reconhecimento de marcos naturais para triangulação da posição do robô no ambiente, como feito em [8], são alternativas válidas para cumprir o objetivo.

Para por a localização em prova, é preciso que o robô se mova com uma trajetória planejada. Um planejador de trajetória, como o A*, e um controlador, como o NMPC, seriam o suficiente.

1.3 Objetivo geral

Este trabalho visa localizar um robô móvel *warehouse* em ambientes com poucas características, e com objetos repetitivos, se valendo de algoritmos de SLAM, feito com um sensor *laser* acoplado no robô, fusão de sensores e extração e reconhecimento de características (marcos naturais), processados pelo ROS, o conjunto de softwares para aplicação de robótica. Além de projetar os algoritmos de planejamento de trajetória e controle. Dessa forma teremos um robô capaz de se localizar no espaço com precisão, e se locomover até o destino com pouco erro. E podemos aplicar esse trabalho em armazéns, minimizando a complexidade das tarefas e otimizando o trabalho.

1.4 Objetivos específicos

A primeira etapa do projeto seria aplicar o algoritmo Hector SLAM no robô. Após isso o térreo do LaSER será modelado no Gazebo, um simulador 3D de robótica open source. Em seguida os algoritmos de trajetória e controle serão implementados, A* e NMPC respectivamente. Finalizando com a fusão de sensores usando filtro de Kalman

estendido (EKF) e o reconhecimento de marcos naturais (AMCL) serão adicionados ao trabalho, aumentando a eficiência de localização.

A Figura 2 mostra o sensor RPLIDAR A3, modelo utilizado no trabalho.

Figura 2: RPLIDAR A3, um tipo de sensor *laser* LiDAR

Fonte: Site oficial[9]

2 CONCEITOS GERAIS E REVISÃO DA LITERATURA

Neste capítulo serão introduzidos os conceitos sobre as ferramentas utilizadas neste projeto. Foi feita uma revisão bibliográfica para cada subtópico com o propósito de entender ao máximo suas propriedades, limitações, e seus potenciais.

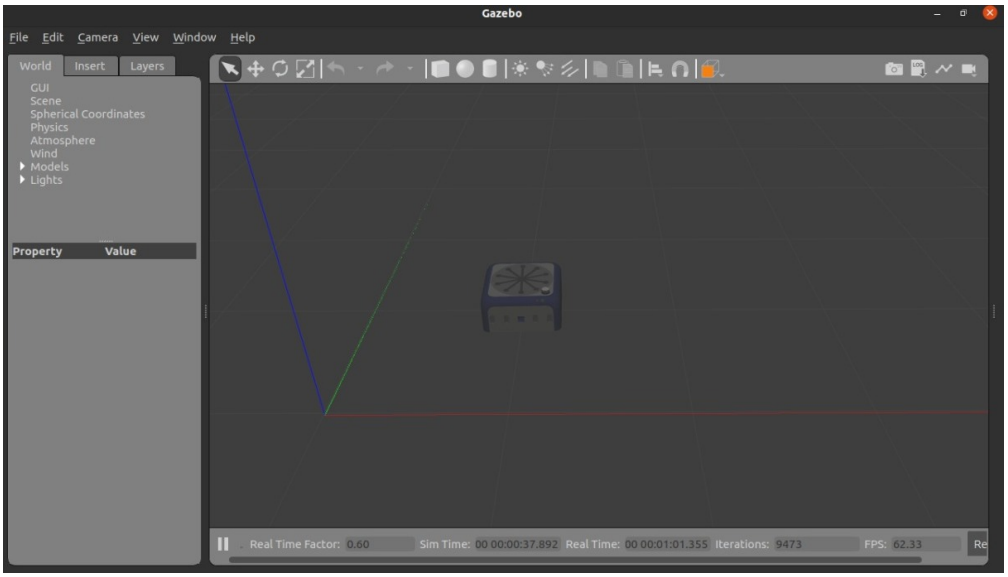
2.1 ROS

De acordo com a descrição ofinal do site, o *Robot Operating System* (ROS) é um conjunto de bibliotecas de *softwares* e ferramentas que ajudam a construir aplicações com robôs, e é *open source*. É o núcleo do projeto e o que permite que todas as partes se juntem.

Uma das ferramentas que vem com o ROS é o Gazebo, um *software* de simulação 3D que é capaz de renderizar modelos de vários tipos de extensão diferentes. O modelo do L1BR foi criado pelo aluno Jorge Luiz no projeto de extensão com o professor Tiago Nascimento. No Gazebo podemos ver o robô em tamanho real e do jeito que foi idealizado para ser. A construção física do mesmo irá seguir o modelo. A física é definida em arquivos à parte e servem para fazer a simulação mais consistente com a realidade, e inclui também a capacidade de se locomover de forma crível. Neste projeto usaremos esse *software* para construir o ambiente do LaSER.

A Figura 3 mostra o robô L1BR no software Gazebo.

Figura 3: Robô L1BR no Gazebo



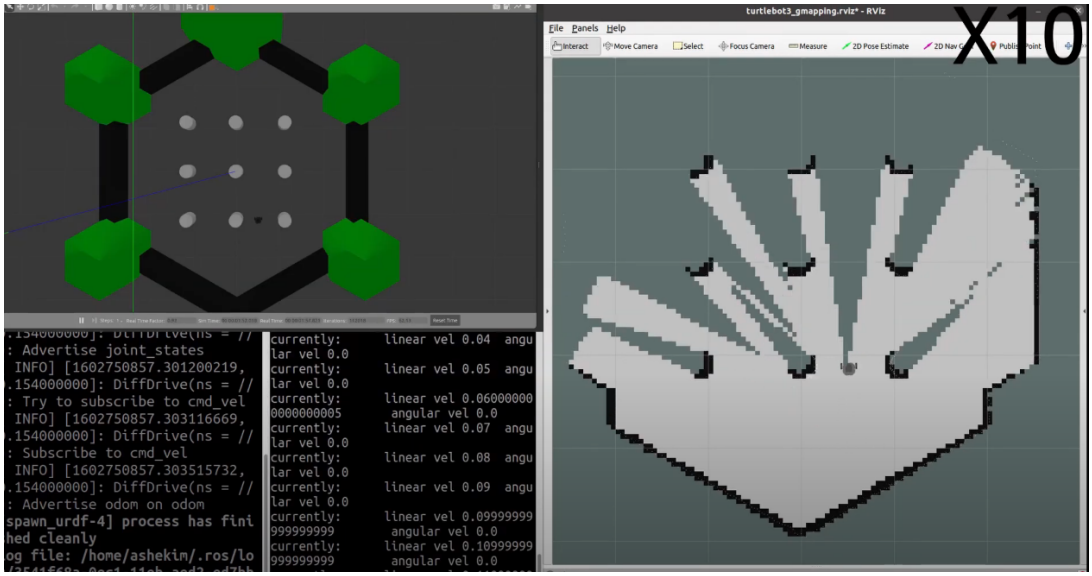
Fonte: *Print screen* feito pelo autor

O RViz é outra ferramenta poderosa do ROS, um visualizador 3D, mas diferente do Gazebo, ele não foca nas características físicas do ambiente, mas sim no efeito dos *plugins*

do robô. Caso tenha equipado uma câmera, é possível visualizar o conteúdo em tempo real de simulação, um sensor *laser* mostra quais pontos estão sendo interceptados. Além de promover a visualização dos efeitos dos sensores, ele também permite a utilização de algoritmos, como por exemplo os algoritmos SLAM, e sua representação visual.

A Figura 4 mostra a utilização simultânea do Gazebo (a esquerda), e o RViz (a direita). No RViz é possível ver o uso do algoritmo SLAM em processo de mapeamento.

Figura 4: Representação de mapeamento por SLAM pelo RViz



Fonte: Retirado de um vídeo da ROBOTIS[10]

2.2 SLAM

Existem vários tipos de SLAM, em [11] vemos várias derivações de SLAM que usam a câmera como sensor. Os chamados *visual* SLAM (vSLAM) usam exclusivamente as informações visuais e podem ser classificados em: algoritmo baseado em *features* (características de ambiente), algoritmo de método direto, e algoritmo baseado em câmeras RGB-D.

O primeiro utiliza certos pontos do ambiente para o mapeamento, ao capturar essas características específicas o algoritmo rastreia (*tracking*) suas posições e começa a formar o mapa (*mapping*) juntando-as de maneira consistente. Por fim, são utilizados algoritmos "assistentes"para otimizar o mapa, um dos usados na pesquisa é o EKF (Extended Kalman Filter), capaz de aumentar a precisão de medição ao combinar sensores. O segundo procura adquirir o mapa de maneira, como seu nome sugere, direta. Diferente do método anterior que procura por características específicas, este forma uma imagem rasa com toda a informação entregue pela câmera, e então a otimização é feita com base no erro entre o resultado criado e a imagem original.

E por fim, a técnica das câmeras RGB-D consistem em utilizar esta tecnologia já muito avançada, presente até mesmo no aparelho Kinect da Microsoft, que se tornaram menores e mais baratos ao longo do tempo. A câmera entrega as formas, texturas, e escala das imagens diretamente, o algoritmo entra no tratamento de profundidade, existem várias maneiras de fazer esse tratamento, tudo muito detalhado em [11].

Porém, diferente do que é exibido nesses projetos, o método SLAM que utilizaremos é Hector SLAM, que não utiliza a câmera, por mais que o robô L1BR a tenha, mas sim um sensor *laser* RPLIDAR, que tem 16MHz de frequência, e será usado com alcance de 14 metros. O uso de sensor *laser* é justificado por ter se tornado a opção mais comum de localização atualmente, e de custo baixo.

Como é possível ver em [12] o Hector SLAM não é o melhor SLAM que existe, porém, isso se deve à sensibilidade do algoritmo à rotação, o que não necessariamente é um problema. Além disso, ele foi criado para rodar com pouco poder computacional, o que significa que é eficiente. O Hector SLAM foi escolhido por uma questão de velocidade de processamento.

2.3 A*

O algoritmo A* (lê-se "A estrela") é um dos planejamentos de trajetória mais famosos de todos, seu princípio básico de direto ao ponto permite que possa ser implementado por qualquer programador sem exigir grandes complexidades, essa é com certeza uma das forças dessa técnica. Em [13] vemos um projeto com um ideal simples de testar técnicas convencionais de robótica com um modelo de robô caseiro, com o intuito de ver como o mesmo se sai com adversidades comuns, como por exemplo rodas defeituosas, controle dessincronizado, sensores com leitura incorreta. Pela proposta do projeto em [13], o código do A* utilizado não foge do básico.

Porém, a versão utilizada neste projeto levemente desvia do convencional, pois em vez de sempre buscar o melhor caminho possível, recorrendo à utilização de função de custo, que testa todos os caminhos possíveis, está em mente uma ideia mais simples: Encontrar o primeiro caminho possível. Tendo em vista que o objetivo final desse trabalho é promover um sistema de localização, controle, e planejamento de trajetória para robôs *picking*, o mais importante é ser eficiente. Para isso precisamos de um algoritmo que consiga entregar o objetivo rápido, adiantando a movimentação da máquina, isso reduz o custo computacional de tal maneira que compensa o fato de o robô está indo por uma rota mais demorada.

A justificativa de usar este algoritmo se deve ao fato de que, com o A*, muitas vezes o primeiro caminho encontrando já é o melhor caminho, isso ocorre quando entre a máquina e o destino final não existe obstáculo nenhum, um cenário bastante comum. Isso

não acontece com outras alternativas como o PRM (*probabilistic roadmap*) por exemplo. Se fosse utilizado o A* convencional, teríamos um desperdício de tempo e processamento.

2.4 Nonlinear Model Predictive Control - NMPC

O modelo de controle preditivo não linear é um controlador bastante complexo, que diferente da maioria das outras espécies trabalha com prever os estados futuros, em vez de focar em erros anteriores. Por mais que tenha ocorrido inúmeras melhorias tecnológicas em hardware e otimização, ainda é bastante difícil resolver os problemas do algoritmo em tempo real, principalmente em caso onde há perturbações, parâmetros desconhecidos, e erros de medida.

Uma pesquisa[14] propôs uma técnica de aproximar um NMPC robusto usando deep learning, e usar uma técnica de validação probabilística como medidor de qualidade, visando implementar em sistemas rápidos e em sistemas embarcados de custo baixo. O NMPC que será utilizado neste trabalho difere do visto na pesquisa, por mais que a proposta seja interessante. Foi usado uma versão mais tradicional do controlador, programado na linguagem C++, e com seu próprio looping de otimização sem uso de nada baseado em redes neurais. O foco deste projeto não se encontra no controle, que já provou funcionar de maneira concreta, mas sim na localização e uso de filtro de Kalman.

2.5 Ambientes sem características

Também conhecidos como *featureless environments* são uma problemática comum no ramo da robótica. Caracterizados como espaços sem muitos objetos, eles dificultam a tarefa de localização por *laser* visto que o mesmo necessita atingir superfícies para exercer sua função, e o problema é maior quando o alcance do sensor é curto. Às vezes ocorre de apenas um pequeno pedaço do ambiente ser dessa forma, como por exemplo quando o robô faz uma curva, e ao se aproximar da quina, a mesma ocupa metade da leitura, mas é considerada apenas como 1 (uma) característica.

Um artigo sobre localização de robôs de alta velocidade[15] propôs, utilizando sensor *laser*, 3 alternativas de contornar este problema:

- Usar a diferença entre as leituras ponto a ponto do sensor entre os tempos t1 e t2, dessa forma calculando a possível localização em x (coordenada horizontal), y (coordenada vertical) , e θ (ângulo de inclinação do robô), e usando uma função de erro para calcular a similaridade entre as leituras.
- Usar o EKF para combinar os dados do *laser* com a odometria das rodas, aumentando a precisão dos resultados, mas apenas em ambientes com poucas *features*, para poupar tempo de processamento.

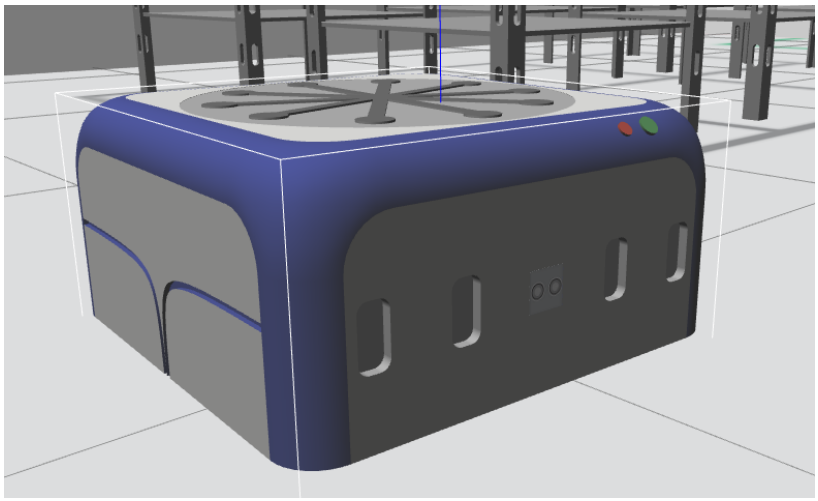
- Ao se encontrar em um ambiente com poucas características, o algoritmo iria criar novos pontos artificialmente usando cálculos de aproximação.

Um outro artigo[16] resolveu o mesmo problema com bases de dados. O robô armazenaria vários locais coletados previamente, dividiria cada local em pequenos pedaços, e transformaria cada um em matriz. Ao se locomover, o robô iria fazer um *matching* entre as matrizes do que os seus sensores estão captando, com as matrizes dos pedaços na base de dados, e então o algoritmo iria juntar os pedaços, como em um quebra cabeça, descobrindo a localização do robô, e aonde ele deve ir.

3 METODOLOGIA

O objetivo deste trabalho é ter sucesso em localizar um robô em um ambiente sem características, mais especificamente o robô L1BR (figura 5), que está sendo desenvolvido nesse momento por uma outra equipe dentro do laboratório LaSER da Universidade Federal da Paraíba (UFPB), também em parceria com a empresa Automni. Foi utilizado então um modelo 3D do robô na primeira parte do projeto, e após a construção do verdadeiro aparelho, houveram testes práticos.

Figura 5: Modelo 3D do robô L1BR representado no Gazebo



Fonte: *Screenshot* feita pelo autor

3.1 Softwares e Hardwares

A coleção de frameworks ROS é definitivamente a parte mais importante deste projeto, um ambiente feito para testes de robótica com todas as ferramentas necessárias para a confecção deste projeto, neste caso em específico estará sendo utilizado a versão noetic. Instalado junto com o ROS, o Gazebo foi bastante requisitado ao longo deste trabalho, o software se encontra na versão 11. Finalizando o pacote de robótica, o RViz, programa de visualização de sensores e atuadores, foi usado na versão noetic.

Os códigos foram todos escritos pelo Visual Studio Code, um editor de código-fonte desenvolvido pela Microsoft, em uma máquina LINUX Ubuntu 20.04. Máquina esta que possui um processador AMD A10-5757M APU, com placa gráfica integrada, 6GB de memória RAM DDR3, são requisições fracas e até ultrapassadas, porém não houve sinal de má performance, portanto não se espera que muitas outras máquinas sejam incapazes de executar este projeto. A situação melhora quando levado em consideração que a parte que mais exige poder computacional são os softwares de simulação, Gazebo e RViz, que

não serão utilizados na etapa física, e o algoritmo de planejamento de rota, que será executado online, poupando o processador para outras tarefas.

As últimas ferramentas utilizadas são as peças robóticas físicas, sendo elas o L1BR e o sensor que será acoplado.

3.2 Sensor *laser* e Hector SLAM

Foi adicionado ao modelo do L1BR um RPLIDAR, um sensor *laser* 2D, na parte superior do robô, este é possível de ser obtido pelos pacotes oficiais[17] desenvolvidos e distribuídos gratuitamente pela empresa Shanghai Slamtec. O item é capaz de acessar o tópico *scan* do ROS, e pode ser configurado por programação. Neste projeto utilizamos 360° e um longo alcance de 14 metros, visando capturar os elementos mais rapidamente.

Turtlebot é um kit de robô pessoal simples desenvolvido com o intuito de testes. A empresa ROBOTIS disponibilizou gratuitamente pacotes ROS[18] contendo diversos algoritmos comuns de estudo e trabalho, alguns deles são algoritmos SLAM que mesmo tendo sido criados para a série turtlebot, depois de algumas modificações podem funcionar para o L1BR também, portanto serão utilizados neste projeto.

O Hector SLAM recebe o *input* do sensor vindo pelo tópico *scan* e mapeia o ambiente, diferenciando obstáculos de espaços livres. Para movimentar o robô nesta fase, capturando cada ponto específico, foi utilizado um código de controle manual. A Figura 4 mostra como aparenta o processo em andamento. Após a figura do mapa ter sido completada, ela é salva para uso futuro.

3.3 Concepção do ambiente do LaSER simulado

Visto que a segunda, e mais importante, parte deste projeto se trata do robô físico no laboratório da universidade, a parte referente a simulação deve ser consistente, sendo assim, o térreo do LaSER foi construído em ambiente simulado e visualizado pelo Gazebo.

O primeiro passo se tratou de uma visita ao local para tirar as medidas necessárias para a simulação. Foram levados em conta: Os comprimentos das paredes principais, as posições dos objetos fixos, incluindo colunas, bancadas, armários, suas medidas exatas e as distâncias entre os objetos. As medições foram feitas com uma trena *laser*. A Figura 6 mostra um esboço das anotações feitas.

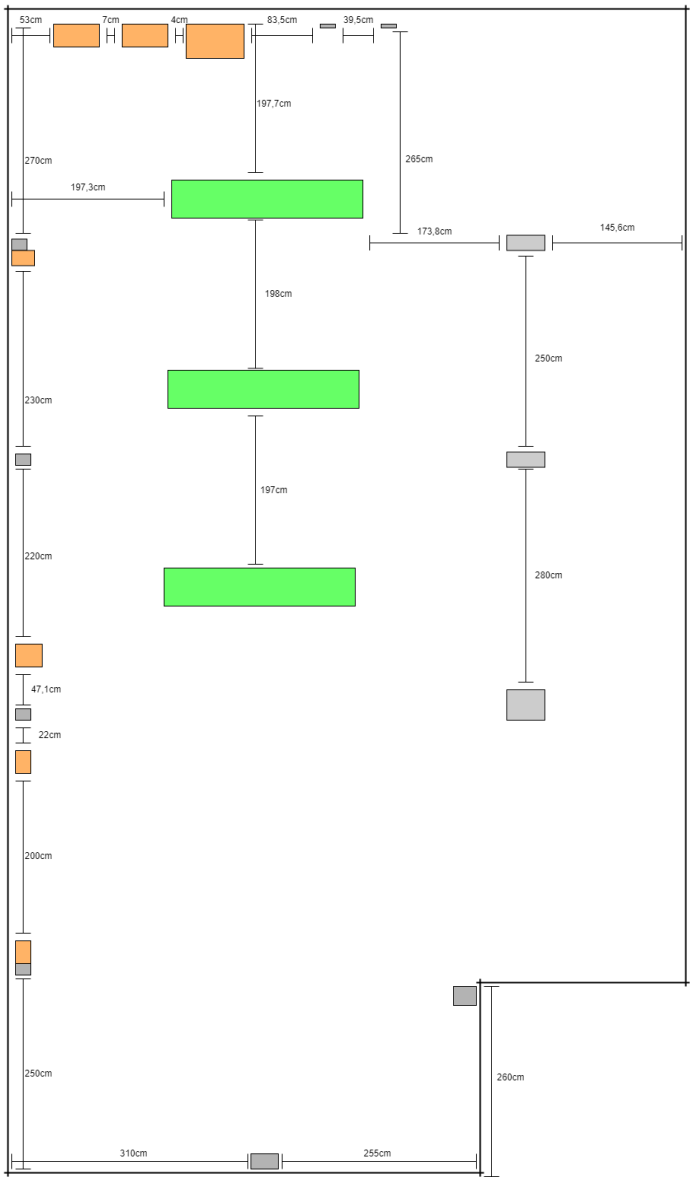
Há duas maneiras de fazer modelagem 3D. A primeira delas é criar os objetos usando a *tag geometry* no próprio arquivo *model.sdf* do modelo, cada objeto possui 3 (três) coordenadas de posição e tamanho, é possível fazer estruturas paralelepípedas de várias maneiras diferentes. As paredes são longas, mas finas, as colunas são menores

em comprimento, mas mais grossas. As bancadas são mais complicadas, foi preciso criar vários paralelepípedos, e combiná-los para fingir que é uma estrutura inteira.

A segunda maneira de modelagem é procurar modelos já prontos, criados em softwares como o Blender, e incluílos no projeto, precisando apenas focar na posição, essa estratégia foi usada para os diferentes armários. Infelizmente não pode ser feito o mesmo com as bancadas, o que evitaria bastante trabalho, pois o visual das bancadas do LaSER são bastante específicas, e precisava ser mais verídico à bancada real.

A Figura 6 mostra o esboço do laboratório feito com as medidas e distâncias coletadas. Sendo verde as bancadas, cinza as colunas, laranja os armários.

Figura 6: Esboço do laboratório



Fonte: Feito pelo autor no site app.diagrams

3.4 Dilatamento de mapa e Planejamento de rota

Antes de começar a planejar a trajetória, é preciso ter em mente que todos os algoritmos consideram o objeto como uma partícula, isto é, algo que tenha o tamanho de 1 (um) pixel. Porém é sabido que o robô possui sua largura, e se isso não for considerado, acontecerá do mesmo ficar preso nas paredes e quinas do ambiente, pois seu tamanho não foi considerado. A solução então seria engordar todos os obstáculos na proporção de largura do robô, ou seja, todos os obstáculos terão suas larguras aumentadas no valor igual (ou aproximado) à metade da largura do robô. Dessa forma, mesmo que o mesmo continue sendo considerado uma partícula pelo A*, passará distante dos obstáculos.

Como dito anteriormente, o algoritmo usado para planejar a rota entre pontos A e B será o A*, que tem como natureza verificar cada ponto disponível no mapa, verificando todos os possíveis caminhos. A linguagem de programação escolhida foi C++, é a linguagem mais comum nessa área, fortemente tipada, o que evita erros indesejados, além de ser fácil e eficiente.

Após definir quais são os nós de início e fim, o código entra em um looping onde ele verifica cada nó ainda não visitado, cada nó possui sua distância local, a distância entre o nó de início até aquele nó. Então é feito uma avaliação entre todos os seus vizinhos, caso o mesmo não seja um obstáculo, o algoritmo atualiza a distância local do vizinho como: distância-local-do-nó + ditância-entre-nó-e-vizinho. O código segue até que eventualmente chegará no nó final.

A distinção entre obstáculo e espaço livre é feita pelo algoritmo Hector SLAM, que atribui cores diferentes aos pixels onde o *laser* atinge e os que ele não atinge. A maneira escolhida para descobrir a distância entre o nó atual e o vizinho foi o cálculo de distância euclidiana, simples e eficiente. O pseudo-código abaixo mostra a ideia principal do algoritmo, que recebe como parâmetro 2 (dois) pontos, sendo *start* o nó de início e *end* o nó final, e então é traçado um caminho de pixel a pixel do ponto de partida até o ponto de chegada. As coordenadas de cada um desses pixels é gravado em um arquivo .txt posteriormente.

3.5 Controlador

Com o mapa e trajetória prontos, o próximo passo é a locomoção, como dito anteriormente será usado o modelo preditivo não linear (NMPC) possui seu diferencial por ser um sistema que trabalha com a previsão de movimento, em vez de focar no erro do passo anterior.

O algoritmo recebe as coordenadas atuais do robô e os dados da trajetória, e então calcula um número de passos até chegar naquele ponto da trajetória. O número de passos é

Algorithm 1: Pseudo-código do A* implementado

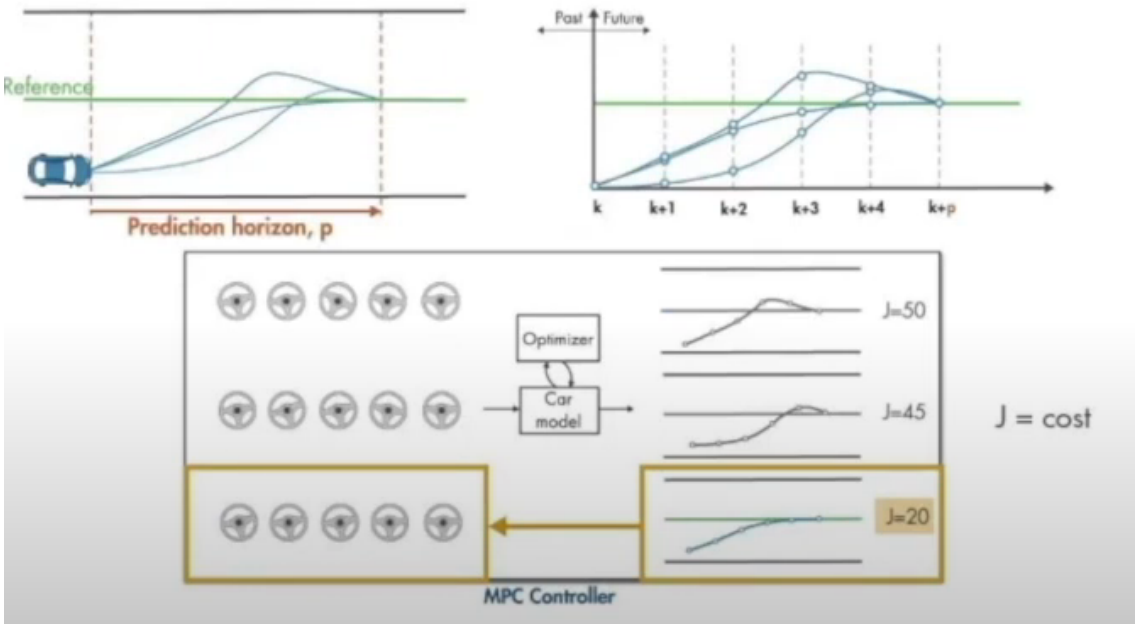
```
current ← start;
start.localGoal ← 0.0;
notTestedNodes.pushback(start);
while NOT notTestedNodes.empty() AND current != end do
    while NOT notTestedNodes.empty() AND notTestedNodes.front().visited
     do
        notTestedNodes.popfront();
    end
    if notTestedNodes.empty() then
        BREAK
    current ← notTestedNodes.front();
    current.visited ← TRUE;
    for EACH neighbour IN current.neighbours do
        if NOT neighbour.visited AND neighbour.obstacle == 0 then
            notTestedNodes.pushback(neighbour)
        LowerGoal ← current.localGoal + distance(current, neighbour);
        if LowerGoal < neighbour.localGoal then
            neighbour.parent ← current;
            neighbour.localGoal ← LowerGoal;

            neighbour.globalGoal
              ← neighbour.localGoal + distance(neighbour, end);
            end
        end
```

chamado de horizonte de previsão, isto é, o quanto o robô irá prever o movimento. Porém, para garantir que a trajetória planejada pelo robô é a mais ideal, a fase do horizonte de planejamento é repetida várias vezes, conseguindo valores diferentes todas as vezes, e então é executado um otimizador que calcula o custo de cada decisão de cada horizonte de predição. Aquele que conseguir menor custo é escolhido, e o resto do código segue esses valores escolhidos.

Um número pequeno de horizonte de previsão, entre 1 e 5, é bastante ideal, pois é rápido de ser calculado, e além disso, quando o robô chega no primeiro ponto do horizonte ele refaz todo o procedimento, criando outro novo horizonte. Ou seja, apenas o primeiro ponto é realmente importante, uma escolha de horizonte de predição longo irá apenas deixar o código mais lento, e talvez até causar um erro caso, por exemplo, um objeto indesejado apareça na área do horizonte de predição, mas o robô segue sem atualizar o mapa (que só é atualizado após a fase de controle) e acaba colidindo.

A Figura 7 ilustra o funcionamento do NMPC no processo de função de custo. É possível ver 3 diferentes caminhos feitos para chegar ao mesmo ponto, e o caminho menos custoso sendo escolhido.

Figura 7: Funcionamento do NMPC



Fonte: Vídeo aula feita pelo professor Tiago Nascimento[19]

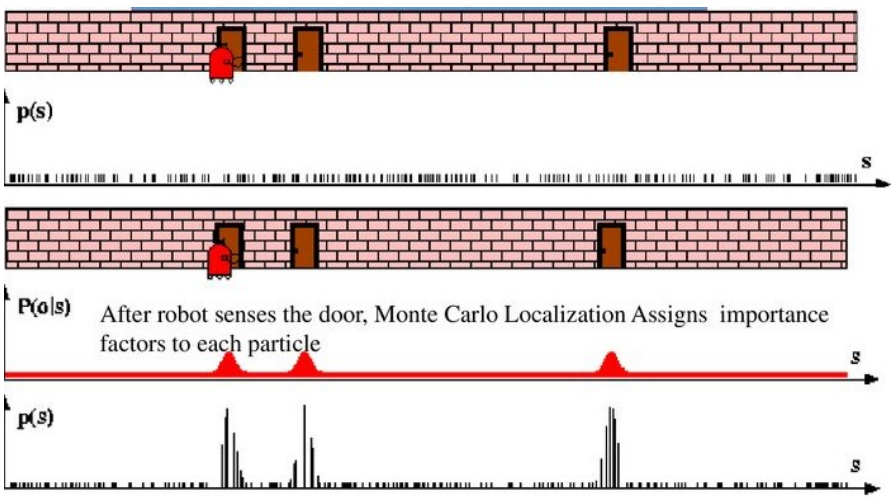
3.6 Localização em ambientes sem características

Ao fazer o planejamento de trajetória, o A* assume que o objeto (robô) seguirá aquele caminho sem erro (desvio), ou seja, ele assume que o robô sempre estará nas coordenadas corretas, bem localizado, principalmente em ambientes sem características.

Monte Carlo é uma espécie de filtro de partículas bastante usado na computação. Várias partículas são espalhadas no mapa de maneira aleatória, cada partícula tem suas medidas x, y, e θ e se movem junto com o robô dependendo da decisão do controlador. Depois disso, cada partícula é posta no modelo de medição que decidirá o peso de cada partícula, as partículas mais pesadas são aquelas que possuem mais chances de estarem sobre o robô. Sempre que o mesmo andar, as partículas levemente perdem levemente a precisão, pois não é possível prever perfeitamente onde o controlador levará o robô, mas quando o código for executado novamente, elas convergem. O modelo de medição utilizado foi o de marcos naturais, o sensor *laser* irá detectar pontos próximos, cada ponto é um marco, uma estrutura de referência. O código testa a distância de cada partícula a cada marco, e atribui peso de acordo com essa medida.

A Figura 8 ilustra o funcionamento do AMCL durante o processo de atribuição de peso das partículas. Importante notar como as partículas tem mais peso perto das portas (marco natural).

Figura 8: Funcionamento do Algoritmo Monte Carlo



Fonte: Vídeo aula feita pelo professor Tiago Nascimento[20]

Por mais que seja uma maneira muito boa de se localizar no ambiente, no que se refere à ambientes sem característica o sensor *laser* possuirá limitações, tendo poucos objetos para usar como referência, na sessão anterior foram citadas soluções para este problema. O artigo [15] propõe uma solução de criar marcos artificiais baseados nos verdadeiros. Dado uma distância fixa, um novo marco seria gerado a tal distância de um marco verdadeiro numa mesma linha. Supondo então que se trata de uma quina, e o alcance do sensor é curto e capta apenas o começo dela, o algoritmo irá notar a característica de parede pelo alinhamento dos pontos do sensor, a geração de novos pontos funcionaria ao criar seguindo o comprimento da parede, tal como se o alcance do sensor fosse maior.

Outra estratégia que o artigo mostra é combinar mais de um sensor, no caso do exemplo dado seria a odometria das rodas, ambos os valores seriam combinados no filtro de Kalman estendido, que consegue produzir uma espécie de "média"entre os dados. É um método bastante utilizado pela comunidade por ser bastante eficiente, e funciona melhor quando ambos os sensores usados no cálculo são independentes.

A proposta final deste trabalho é combinar duas dessas estratégias, o AMCL e o EKF. O algoritmo de Monte Carlo (ou AMCL, sigla para *Adaptive Monte Carlo Localization*) utiliza o cálculo de medida de pesos das partículas e o número de partículas utilizadas fora testado experimentalmente, a medida utilizada para o cálculo da distância entre as partículas e marcos naturais é a distância euclidiana, e a probabilidade é atribuída à partícula utilizando a equação de distribuição gaussiana, sendo: X a distância entre o robô e aquele marco, a média é a distância da partícula até o marco, e o desvio-padrão o fator aleatório. Como há chances do Monte Carlo ser impreciso, utilizamos o EKF para juntar os resultados de mais de um sensor (Odom e IMU), e então o resultado disto é passado para o AMCL, trazendo uma média satisfatória, e definindo a localização do robô em ambientes sem característica com precisão.

3.7 Integração das partes

Para juntar as etapas deste projeto, será também implementado um código principal chamado de *control.cpp*. O funcionamento deste código começaria com uma chamada da função para adquirir os valores de coordenadas do caminho planejado. Porém, os valores de coordenadas precisam ser compatíveis com os mesmos obtidos no ROS, pelo Gazebo principalmente. Sendo assim, é preciso realizar uma transformada na relação entre pixels e metros, uma simples aplicação da regra de 3 é o suficiente, dessa forma:

$$\frac{larguraDoAmbienteEmMetros}{x[i]} = \frac{larguraDoMapaEmPixel}{L[i]} \tag{1}$$

$$\frac{alturaDoAmbienteEmMetros}{y[i]} = \frac{alturaDoMapaEmPixel}{A[i]} \tag{2}$$

Sendo L o conjunto de coordenadas referentes ao eixo x de cada ponto da trajetória, A o conjunto de coordenadas referentes ao eixo y de cada ponto de trajetória, e i o ponto atual a ser transformado.

Após isso, para cada ponto na trajetória a ser seguido, alimentamos a função de controle com os valores da coordenada x do ponto, coordenada y do ponto, coordenada x atual do robô (obtido pelo AMCL), coordenada y atual do robô (obtido pelo AMCL), inclinação do robô em relação ao plano do chão, velocidade linear atual e velocidade angular atual, sendo esses 3 últimos obtidos pela odometria.

A função de controle retorna as velocidades lineares e angulares necessárias para que o robô alcance o ponto da vez, essas velocidades são passadas para o ROS, que faz o movimento ordenado. É preciso um certo tempo antes do próximo ciclo, fim de dar tempo para o robô se locomover, então um *delay* é chamado no fim do ciclo anterior. O pseudo-código abaixo ilustra o funcionamento do arquivo *control.cpp*.

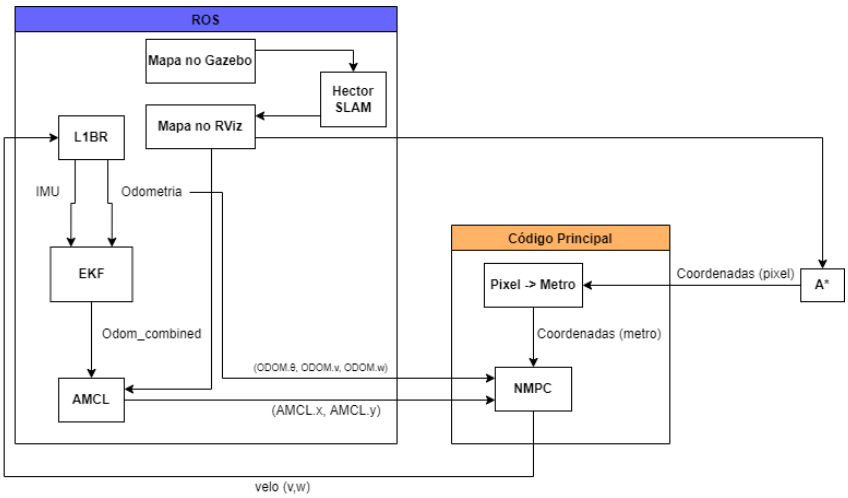
Algorithm 2: Pseudo-código do control.cpp implementado

```
L, A ← lerArquivo(coordenadas.txt);
x[] ← emptyArray();
y[] ← emptyArray();
for i = 0; i ≤ L.lenght(); i++ do
    x[i] ← larguraDoAmbienteEmMetros * L[i]/larguraDoMapaEmPixel;
    y[i] ← alturaDoAmbienteEmMetros * A[i]/alturaDoMapaEmPixel;
    end
    for i = 0; i ≤ x.lenght(); i++ do
        velo ←
          NMPC(x[i], y[i], AMCL.x, AMCL.y, ODOM.teta, ODOM.v, ODOM.w);

        velPub.publish(velo);
        delay();
        end
```

Houve um erro de indentação no algoritmo, é preciso observar que o segundo "For"só começa após o fim do primeiro.

A Figura 9 mostra o fluxograma do projeto inteiro. Eles são divididos em duas partes majoritariamente, uma parte referente ao ROS, e outra parte referente aos algoritmos externos. É possível também ver quais informações cada entidade passa para outra

Figura 9: Fluxograma de todo o projeto



Fonte: Feito pelo autor

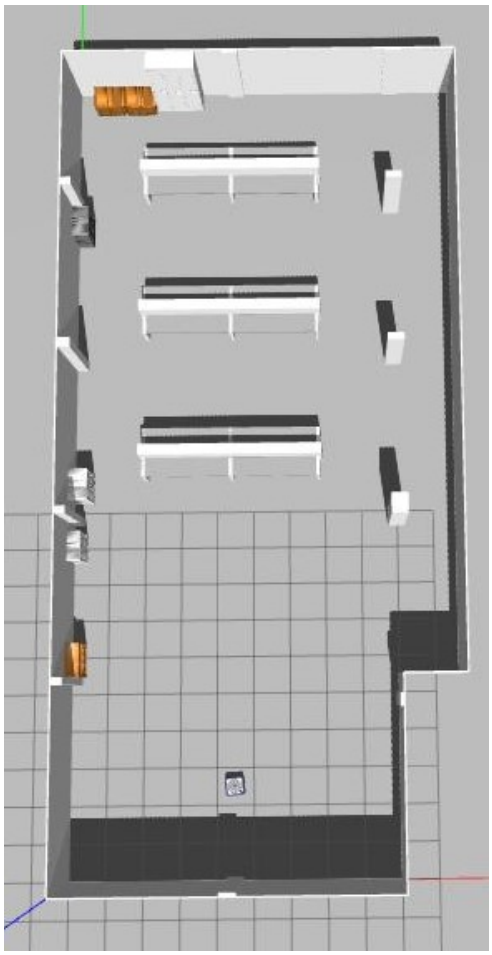
4 APRESENTAÇÃO E ANÁLISE DOS RESULTADOS

4.1 Modelagem, captura e dilatação do mapa

O Gazebo renderiza arquivos do tipo .sdf, e são por esses arquivos que é feita a modelagem. Foram criados objetos cujas medidas se encaixam no que fora medido anteriormente na Seção 3.3, assim como também foram utilizados alguns modelos de armários de um repositório gratuito de objetos no Blender.

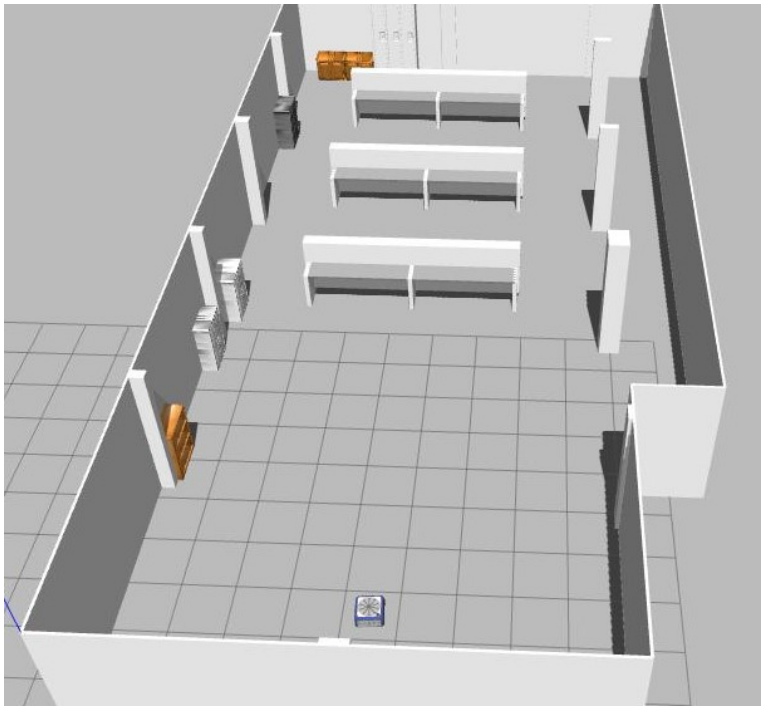
As Figuras 10 e 11 mostram o resultado da modelagem do laboratório LaSER.

Figura 10: Visão de cima do mapa modelado, espelhando a figura 6



Fonte: Screenshot feita pelo autor

Figura 11: Visão de lado do mapa modelado



Fonte: Screenshot feita pelo autor

Com o ambiente do robô pronto, foi usado um algoritmo de controle manual para mover o robô pelo mapa. Ao longo de sua passagem, o sensor *laser* atinge os objetos e suas formas ficam registradas no software de visualização RViz.

A Figura 12 mostra o mapa na visão do RViz após a utilização do mapeamento SLAM.

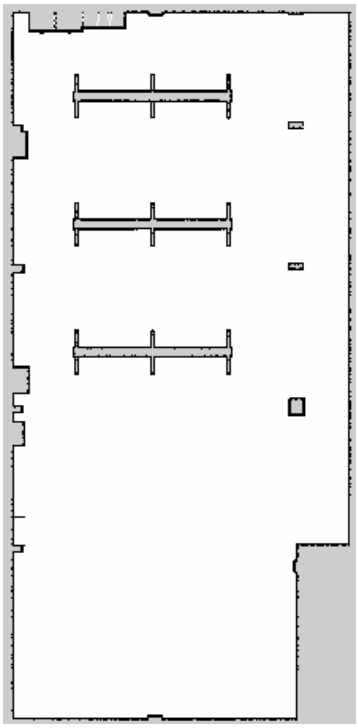
O arquivo de mapa será enviado para um código em python que identifica por cores quais partes são obstáculos/espaço indisponíveis e quais são espaços livres. Sendo branco espaço livre, cinza espaço indisponível, e preto obstáculo. Depois de identificar, ele trata de expandir todas as partes cinzas e pretas.

A Figura 13 mostra o resultado da dilatação, a alteração das cores faz parte do programa, agora dividido entre branco e preto, sendo obstáculo e espaço livre respectivamente.

4.2 Planejamento de rota e controle de trajetória com odometria

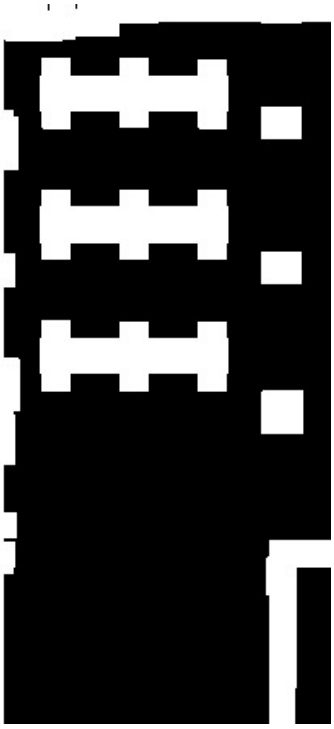
É com o mapa dilatado que o planejamento de trajetória é feito. Primeiramente, antes de verificar a precisão da localização, será testada a movimentação do robô em trajetórias específicas. Estas trajetórias são: vertical (figura 14), horizontal (figura 15), final (figura 16), esta última possuindo características vertical e horizontal.

Figura 12: Mapa obtido pelo sensor



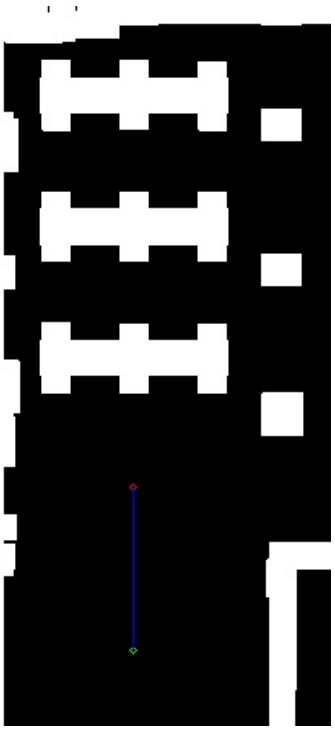
Fonte: Gerado pelo RViz

Figura 13: Mapa dilatado



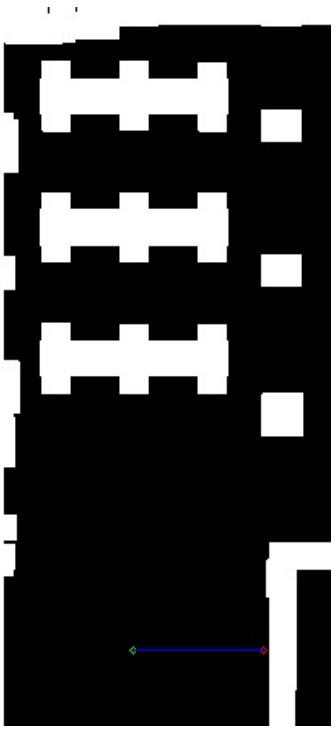
Fonte: Gerado pelo autor

Figura 14: Trajetória vertical



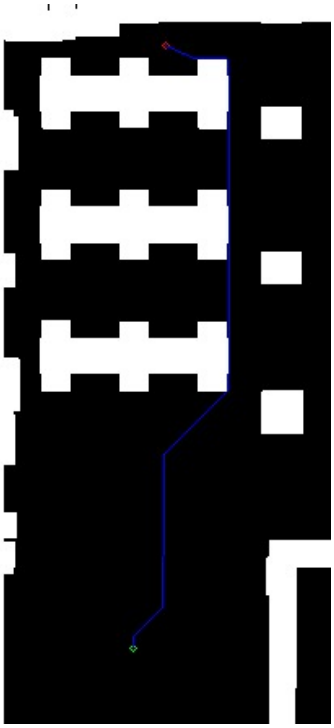
Fonte: Gerado pelo autor

Figura 15: Trajetória horizontal



Fonte: Gerado pelo autor

Figura 16: Trajetória final



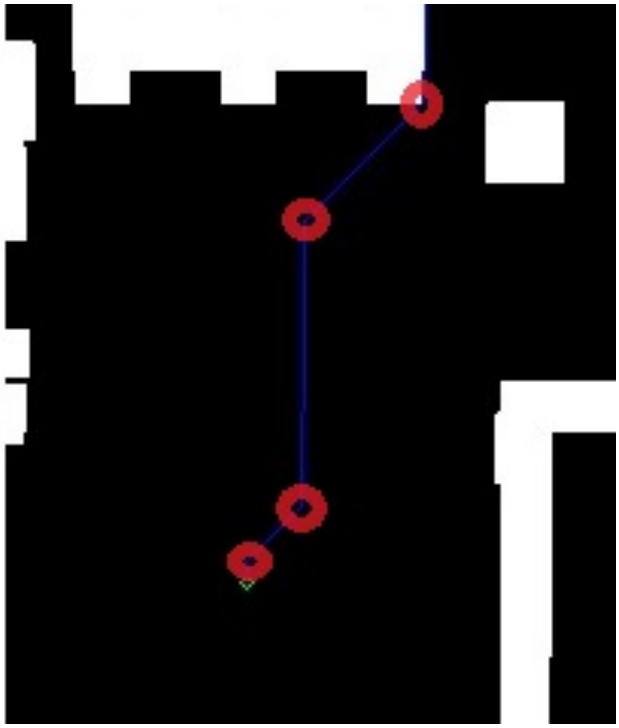
Fonte: Gerado pelo autor

Como ambos os EKF e AMCL ainda não foram implementados, serão utilizadas as informações apenas da odometria.

Quando se trata da trajetória vertical, o robô segue com quase perfeição, precisando corrigir o seu alinhamento apenas algumas vezes. Porém, ao realizar a trajetória na horizontal, o robô realiza um movimento de "ré"assim como um carro. Tomando distância e se inclinando para a direita, para então finalmente começar a andar na horizontal. Caso o robô julgue que saiu do alinhamento, ele performará o movimento de ré novamente.

Ao se locomover na trajetória final, a presença de trechos diagonais fez com que o robô tomasse decisões problemáticas. Como para cada ponto do trecho diagonal existe uma mudança horizontal, o robô alinha seu eixo para corresponder com a coordenada do ponto, mas logo depois vem um novo ponto, que também tem mudança na horizontal, então o robô deve alinhar novamente. Apenas após o código (ver Algorithm 2) ter passado do ponto do trecho diagonal, o robô finalmente se mexe em uma trajetória circular que começa do ponto de início do trecho diagonal até o fim (figura 17).

Figura 17: Pontos onde acontece a trajetória circular



Fonte: Gerado pelo autor

Essa trajetória circular pode ser bastante problemática em outros ambientes, felizmente o mapa do laboratório LaSER não causou problemas. Mas é importante levar em consideração que no segundo trecho diagonal o robô quase colide com a primeira coluna, corrigindo sua posição horizontal ao longo do resto do trajeto, se aproximando mais das mesas como previsto pelo caminho gerado pelo algoritmo A*.

Algo que possa ter contribuído com isso é o fato de que o robô não foi construído fisicamente ainda, portanto não é possível saber os verdadeiros valores da física do seu corpo (coeficientes de atrito, inércia, etc). Sendo assim, este trabalho lida com uma aproximação do que seria o comportamento físico do L1BR.

4.3 Integração com EKF e AMCL

Como dito na documentação do EKF[21], o filtro de Kalman estendido é obtido pela união de até 3 informações entregues pelo robô, sendo elas: Odometria, IMU(Inertial Measurement Unit), e Odometria visual. Este último se refere à terceira dimensão, a altura, como não é relevante neste projeto, será deixado de fora. Para utilizar o filtro de Kalman usamos o nó "robot_pose_ekf"e passamos como parâmetro o carcaça do robô. O resultado do EKF se chama *Odom_combined*.

O nó "amcl"cuida da integração com o algoritmo de Monte Carlo, para isso teremos que passar o resultado do EKF para que ele use como referência de Odometria (EKF

é considerado uma Odometria melhorada), assim como o mapa da figura 12 para que o mesmo tenha noção do ambiente. O AMCL possui outros parâmetros que serão explorados na sessão seguinte.

A Figura 18 mostra como ficou a *tf tree* do projeto, isto é, a formatação de todos os *frames* ou partes. *Left_wheel*, *right_wheel*, *right_front_wheel*, *left_front_wheel*, *right_rear_wheel*, *left_rear_wheel* são as rodas do robô, *camera_1* é a câmera, *piston* e *tray* dizem respeito ao pistão, *rplidar* é o RPLIDAR. A "carcaca"representa a carcaça do robô, e é o nó que mantém todas as componentes anteriores unidas. *Odom_combined* é a resultante do EKF e se conecta ao robô para obter as informações dos sensores dele. *Map* e *scanmatcher_frame* se conecta à árvore por *Odom_combined* para obter os valores da localização do robô, a fim de representá-lo nos mapas do Gazebo e RViz.

Por mais que o *frames* odom não esteja conectado com o resto da árvore, é confirmado que ele está servindo para alimentar o *Odom_combined*.

A Figura 19 mostra como fica o robô e o mapa com as partículas do AMCL funcionando. Os pontos verdes são a parte do mapa que o sensor consegue captar por estar no alcance de 14 metros. Os pontos vermelhos são as partículas que já estão convergindo no robô.

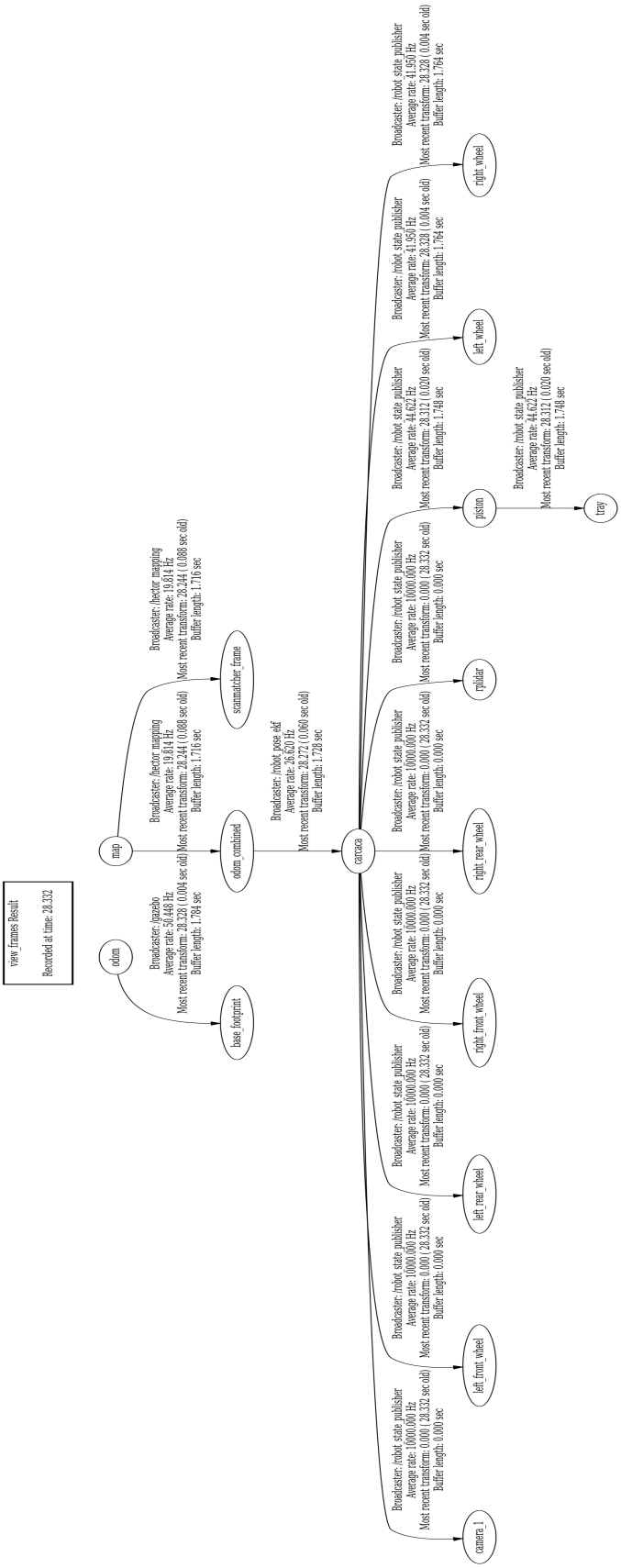
4.4 Experimentação

Existem vários parâmetros no nó do AMCL, todos eles possuem valores padrões que foram testados pelos seus desenvolvedores[22]. Como está sendo usado outro robô, não necessariamente os valores corresponderão bem, e será recomendado uma alteração nos valores, estes que são encontrados apenas por experimentação. Serão verificados os parâmetros: *odom_model_type*, *laser_max_beams*, *resample_interval*, *min_particles*, *max_particles*, *update_min_d*, *update_min_a*.

Serão exibidos gráficos, usando um código na linguagem python de diferença de performance entre os diferentes valores para cada parâmetro escolhido. Para decidir a melhor performance, foram retirados a média dos desvios padrões entre a posição estimada e a posição real ponto a ponto do código para cada coordenada. Vários testes são efetuados para ter certeza de que o experimento não será afetado por *outliers*. A melhor opção para o parâmetro será mantida para o teste do próximo, visto que eles possuem natureza independente, e portanto a alteração de um não afeta o outro. O andamento do processo será exibido em tabelas no final deste tópico.

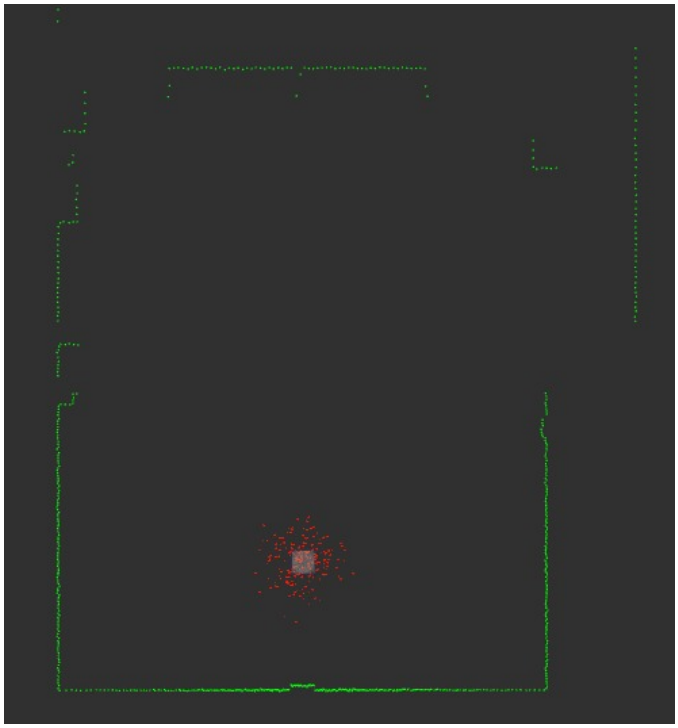
O primeiro parâmetro que é preciso ver é o *odom_model_type*, ou seja, o modelo de Odometria utilizado. Há dois valores possíveis "diff"(manobridade diferencial) e "omni"(manobridade omnidirecional), ambos são parecidos, a diferença é que a segunda opção leva em consideração a tendência do robô de transladar sem rotacionar, isto é,

Figura 18: TF tree



Fonte: Gerado pelo autor

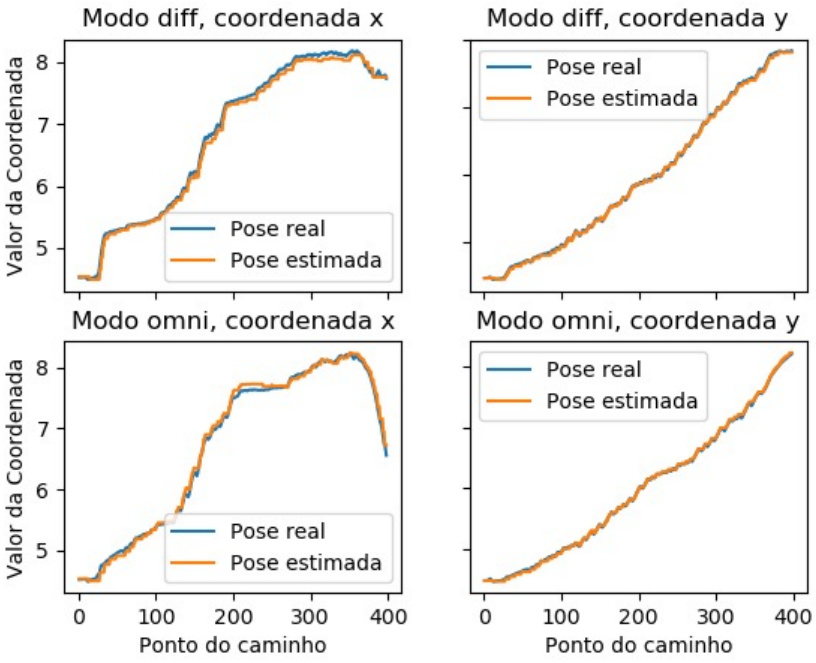
Figura 19: Partículas no RViz



Fonte: Screenshot feita pelo autor

andar perpendicular a algum plano.

Figura 20: Performance dos modos omni e diff

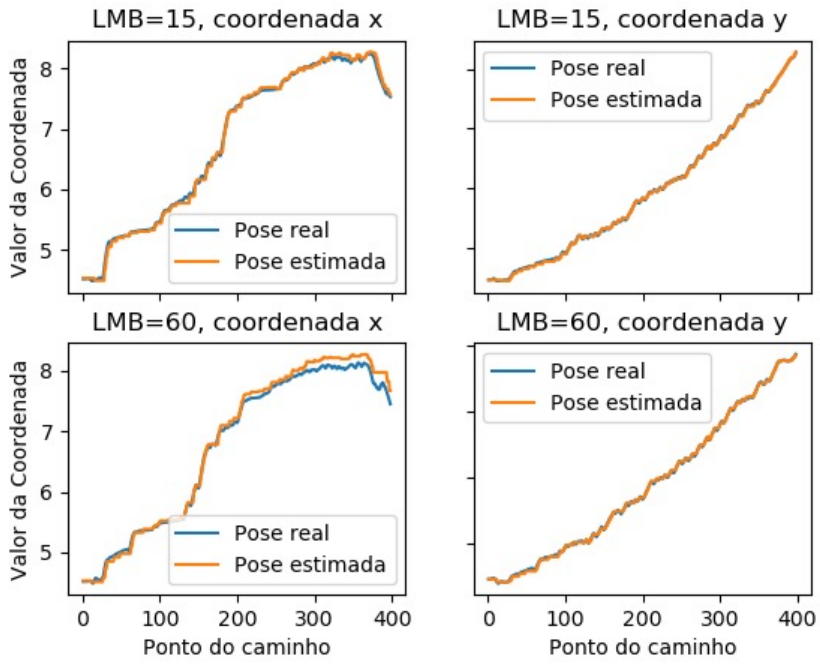


Fonte: Gerado pelo autor

Foi percebido, pela figura 20 as diferenças entre as poses reais e estimadas dos parâmetros. Os desvios padrões dos valores entre a pose real e a pose estimada para o modo *diff* foram de 0,030 e 0,058 para as coordenadas X e Y respectivamente. Enquanto que para o modo *omni* foram de 0,0181 e 0,0588. A superioridade do segundo modo é clara, provavelmente se ocorrida devido à vantagem de levar em consideração movimentos de translação sem rotação, que é característico deste projeto. Sendo assim o modo omni será mantido para os próximos testes.

O próximo parâmetro a ser testado é o *laser_max_beams* , que representa o valor de feixes de luz levados em consideração pelo algoritmo na hora de calcular a probabilidade de localização. O valor padrão é de 30, cuja performance pode ser vista na figura 20, serão testados então valores de 15 e 60, próximo do padrão. O motivo disto é que ao testar valores mais "absurdos"havia uma óbvia diminuição na qualidade de localização, dispensando qualquer necessidade de experimentação com gráficos e desvio padrão. Isso se deve ao fato de que leva mais tempo para processar a informação, e o valor de localização se torna atrasado ao longo do tempo.

Figura 21: Performance de diferentes valores para laser_max_beam



Fonte: Gerado pelo autor

Como é possível ver pela figura 21, a opção de diminuir os valores do parâmetro para 15 é a melhor, isso trás o desvio padrão médio para 0,0185 e 0,0408 (x e y). Esse valor será mantido para os próximos testes.

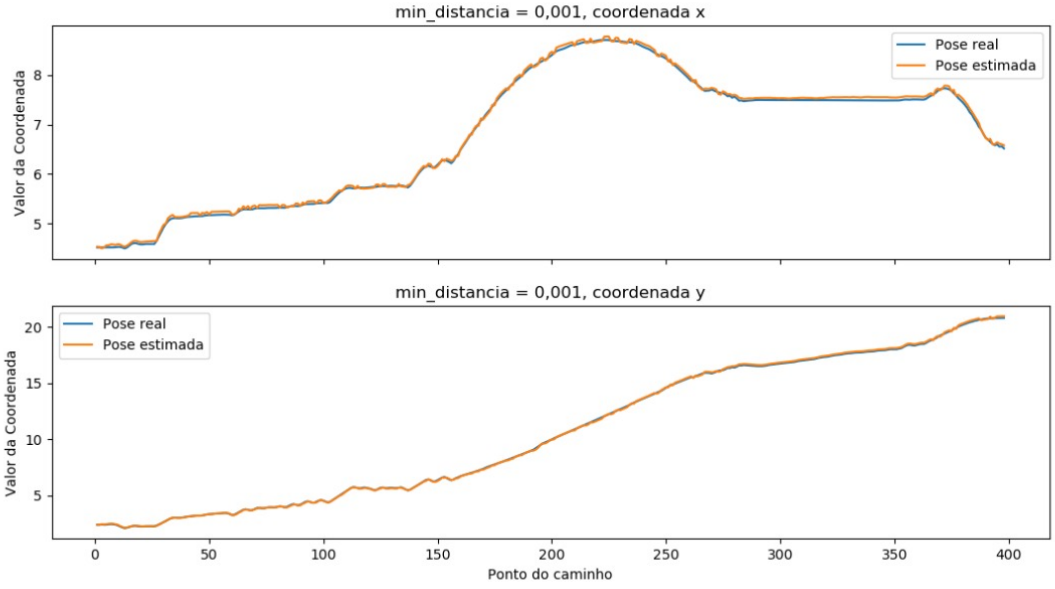
O *resample_interval* representa o número de atualizações no filtro do algoritmo

antes de fazer o *resampling*, que é a reorganização das partículas no espaço. O valor padrão deste é de 2, serão feitos testes com 1 e 3, não é o tipo de parâmetro que é possível distanciar do padrão sem comprometer o experimento, caso o valor seja muito alto, a leitura ficará atrasada.

Infelizmente não houve melhoria, com desvios padrões de 0,0353 e 0,0384 para *resample_interval* = 1, e 0,0212 e 0.0481 para *resample_interval* = 3, temos valores piores do que os apresentados anteriormente. Será dispensado o uso de gráfico nesse caso.

Em seguida será avaliado os parâmetros *update_min_d* e *update_min_a*, que tratam da distância mínima percorrida (linear e angular respectivamente) pelo robô para que o filtro seja atualizado. Seguindo a lógica de que quanto mais rápida seja a atualização, melhor será o trajeto do robô, o instinto é diminuir esses valores ao máximo. Sendo assim eles serão reduzidos dos seus valores padrões de 0,2 e 0,5236 para 0,001 em ambos.

Figura 22: Performance de distância mínima de atualização = 0,001



Fonte: Gerado pelo autor

Como indicavam a figura 22, a melhoria é bastante clara, não apenas no gráfico, mas durante a simulação foi possível ver uma convergência de partículas logo cedo. Os valores do desvio padrão foram de 0,0219 e 0,0279, infelizmente houve uma pequena queda de qualidade na coordenada x, porém a grande melhoria na coordenada y compensa bastante. Esses valores para *update_min_d* e *update_min_a* permanecerão no próximo teste.

Por enquanto, a tabela 1 representa o andamento dos experimentos, sendo "DP"abreviação para desvio padrão:

Tabela 1: Experimentação com paramêtros do AMCL

odom_type	*laser_beams*	*resample_interval*	*min_d/a*	DP em X	DP em Y
omni	30	2	0,2 e 0,5236	0,0181	0,0588
diff	30	2	0,2 e 0,5236	0,030	0,058
omni	15	2	0,2 e 0,5236	0,0185	0,0408
omni	60	2	0,2 e 0,5236	0,0405	0,0394
omni	15	1	0,2 e 0,5236	0,0353	0,0384
omni	15	3	0,2 e 0,5236	0,0212	0,0481
omni	15	2	0,001 e 0,001	0,0219	0,0279

Fonte: Gerado pelo autor com base nos valores obtidos

Os últimos parâmetros testados foram *min_particles* e *max_particles*, que ditam respectivamente os números mínimos e máximos de partículas usadas pelo filtro. O AMCL funciona de tal maneira que ele irá usar um número de partículas x que esteja dentro do intervalo definido pelo projetista. Esse número é decidido de acordo com a dificuldade atual do filtro em localizar o robô, usando como parâmetro o tópico de odometria e o resultado do sensor *laser*.

Os valores padrões são 100 e 5000, o limite mínimo será mantido pois já é um valor muito pequeno e o intervalo entre o mínimo e máximo será controlado pela alteração apenas pelo limite máximo. O objetivo é encontrar o intervalo de valores como melhor resultado e, em caso de empate, escolher o com menor limite máximo, o que diminui o esforço de processamento. A Tabela 2 mostra o resultado da média de desvios padrões para cada limite máximo testado.

Tabela 2: Desvios padrões para cada limite máximo de partículas

max_particles	Desvio padrão em X	Desvio padrão em Y
500	0,0174	0,0324
2500	0,0181	0,0292
3000	0,0199	0,0365
4000	0,0182	0,0371
6600	0,0170	0,0440

Fonte: Gerado pelo autor com base nos valores obtidos

É possível ver que os valores não divergem muito, o que indica que o AMCL não está tendo muitos problemas em encontrar o robô, provavelmente porque os outros parâmetros já foram otimizados, então a diferença de partículas usadas têm pouco efeito. Mesmo assim, 2500 mostra um melhor resultado em comparação com os outros, então será escolhido como valor definitivo.

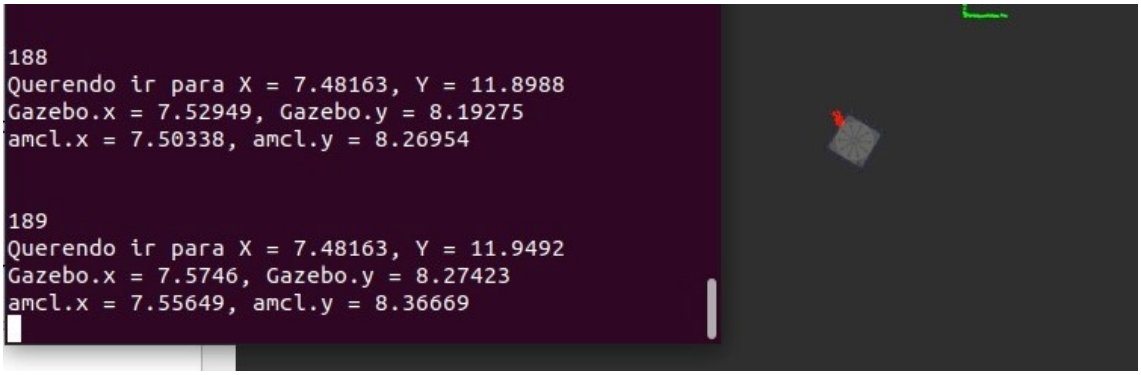
Um último experimento foi proposto para casos extremos de acidentes no armazém,

onde o robô pode sofrer uma translação inesperada, provavelmente resultante de uma batida ou empurro. Então para simular esse acidente, durante a execução do código, o robô foi tirado de sua posição atual para uma posição longe. Isso foi feito pelo Gazebo, a ferramenta de simulação do ROS.

O visual do RViz indica que os pontos convergem longe do robô, o que a princípio indicaria que houve uma falha na localização. Porém, durante a execução do código, é possível ver que os valores, em metros, do AMCL são muitos próximos ao do Gazebo, com diferença menor de 1cm para a coordenada X e de quase 10 cm para a coordenada Y.

A Figura 23 mostra a esquerda o andamento do programa e as informações sobre localização real (Gazebo), e estimada (AMCL). A direita vemos, pelo RViz, o robô L1BR com as partículas convergindo fora dele.

Figura 23: Resultado do experimento de simulação de acidente



Fonte: Screenshot feita pelo autor

Levando em consideração que o robô possui aproximadamente 50 cm de largura e altura, é incabível a convergência de partículas estar acontecendo fora do robô. Sendo assim é possível afirmar que não passa de um mero *bug* do RViz.

5 CONCLUSÕES E TRABALHOS FUTUROS

Este trabalho tem como objetivo localizar um robô terrestre móvel *warehouse* em ambiente sem características, tal qual armazéns com prateleiras, usando o laboratório LaSER como mapa de teste (modelado em 3D), e usando ferramentas clássicas da área de robótica (SLAM, A*, NMPC, EKF, AMCL) para alcançar esse objetivo.

O mapa é verossímil, representando bem o ambiente com falta de características que é o LaSER, este é o primeiro modelo 3D do laboratório, e ficará disponível para uso público. Este trabalho também é o primeiro a aplicar algoritmos ao L1BR, visto que acaba de ser criado, sendo assim, aqui fica registrado informações pertinentes para futuros usos desde robô.

O Hector SLAM se mostra um desafio para ser utilizado, pois as documentações dos nós do ROS são bastante escassas, mas depois de muitas pesquisas e tutoriais, a maneira correta de manipular os parâmetros foi encontrada.

A implementação do dilatamento de mapa e do A* são bastante direto ao ponto, uma questão de encontrar a linguagem e as ferramentas corretas.

O mesmo não pode ser dito do NMPC, que possui versões diferentes dependendo da fonte, portanto é necessário um aprofundamento teórico no algoritmo para descobrir a melhor versão. Além disso, a ausência dos verdadeiros valores da física do robô acabou custando bastante tempo para descobrir o porquê do controle estar tão irregular. Após descoberto o problema, foram usados valores da física do Turtlebot 2 da empresa ROBOTICS, visto que a carcaça do L1BR foi baseada na base dele, tendo assim uma aproximação do comportamento que o robô teria de fato.

O uso dos nós EKF e AMCL também são marcados pela falta de documentação do ROS, o primeiro exige uma manipulação de frames que demanda muita tentativa e erro até funcionar. Para substituir o *Odom* por *Odom_combined* é necessário criar um frame fantasma, que não existe ou interfere no programa, chamado *base_footprint*, pois o *Odom* não aceita estar presente na árvore sem estar conectado a um frame. Isso pode ser visto na figura 18. Já com o AMCL há um problema de conflito com a árvore tf, sendo que como o resultado dele não retorna para o ROS, mas sim é encaminhado para o NMPC, se faz necessário uma pesquisa entre os parâmetros para descobrir que é possível cortar a conexão do mesmo com a árvore, atribuindo o valor *false* ao parametro *tf_broadcast*. Não que isso atrapalhe o projeto.

O resultado final da localização é bastante satisfatório com desvio padrão médio pequeno, impedindo que o robô se perca. O mesmo se move seguindo a trajetória correta, salvo alguns movimentos diagonais onde, às vezes, ele performa movimentos circulares abertos, chegando perto de bater nas colunas. O código completo foi armazenado no site

Github[23].

Infelizmente, por uma questão de falta de tempo, não foi possível implementar uma das estratégias de localização planejada inicialmente, a criação de marcos artificiais, o que poderia ser um diferencial. Além disso houve atrasos na construção do robô físico, então este trabalho é encerrado apenas com a parte referente a simulação. Portanto, para possíveis projetos futuros, seria ideal concretizar essas duas promessas.

REFERÊNCIAS

[1] Cerqueira, Wagner. A robotização na produção industrial: Uol. Disponível em: https://mundoeducacao.uol.com.br/geografia/a-robotizacao-na-producao-industrial.htm. Acesso em: 8 Dez, 2021.

[2] J.F. Avila-Tomás, M.A. Mayer-Pujadas, V.J. Quesada-Varela, La inteligencia artificial y sus aplicaciones en medicina I: introducción antecedentes a la IA y robótica, Atención Primaria, Volume 52, Issue 10, 2020, Pages 778-784, ISSN 0212-6567, https://doi.org/10.1016/j.aprim.2020.04.013.

[3] Tondo, Stephanie. Robôs de limpeza viram sonho de consumo: veja as opções disponíveis no mercado: O Globo, 2021. Disponível em: https://oglobo.globo.com/economia/como-economizar/robos-de-limpeza-viram-sonho-de-consumo-veja-as-opcoes-disponiveis-no-mercado-24915406. Acesso em: 8 Dez, 2021.

[4] Kohlbrecher, Stefan. How to set up hector_slam for your robot: ROS.org, 2012. Disponível em: http://wiki.ros.org/hector_slam/Tutorials/SettingUpForYourRobot. Acesso em: 8 Dez, 2021.

[5] R F Junior, Jose. Filtro de Kalman: Medium, 2019. Disponível em: https://medium.com/@web2ajax/filtro-de-kalman-6e84f82993fc. Acesso em: 8 Dez, 2021.

[6] G, Filipe. Robô aspirador que também passa pano funciona por voz e limpa rápido. Disponível em: https://www.techtudo.com.br/noticias/2021/01/robo-aspirador-que-tambem-passa-pano-funciona-por-voz-e-limpa-rapido-ces2021.ghtml. Acesso em: 10 Jun, 2022.

[7] Face Middle East FZC. Testing Floors for G2P Automation. Disponível em: https://facemiddleeast.ae/testing-floors-g2p-automation/. Acesso em: 10 Jun, 2022.

[8] BEZERRA, Clauber Gomes. Localização de um robô móvel usando odometria e marcos naturais. 2004. 122 f. Dissertação (Mestrado em Automação e Sistemas; Engenharia de Computação; Telecomunicações) - Universidade Federal do Rio Grande do Norte, Natal, 2004.

[9] SLAMTEC. RPLIDAR A3 360 degree laser range scanner for indoor and outdoor applictaion. Disponível em: https://www.slamtec.com/en/Lidar/A3. Acesso em: 10 Jun, 2022.

[10] TurtleBot3 ROS1 Noetic Quick Start Guide for Noetic, 2020. 1 vídeo (4 min). Publicado pelo canal ROBOTIS OpenSourceTeam. Disponível em: https://www.youtube.com/watch?v=ji2kQXgCjeM&ab_channel=ROBOTISOpenSourceTeam. Acesso em: 10 Jun, 2022.

[11] Taketomi, T., Uchiyama, H. Ikeda, S. Visual SLAM algorithms: a survey from 2010 to 2016. IPSJ T Comput Vis Appl 9, 16 (2017). https://doi.org/10.1186/s41074-017-0027-2

[12] Pengtao Qu et al 2021 J. Phys.: Conf. Ser. 2024 012056

[13] Raza, Sayyed Jaffar Ali, et al. "Real-world modeling of a pathfinding robot using robot operating system (ROS)."arXiv preprint arXiv:1802.10138 (2018).

[14] Karg, Benjamin, Teodoro Alamo, and Sergio Lucia. "Probabilistic performance validation of deep learning-based robust NMPC controllers."arXiv preprint arXiv:1910.13906 (2019).

[15] Kai Lingemann, Andreas Nüchter, Joachim Hertzberg, Hartmut Surmann, High-speed laser localization for mobile robots, Robotics and Autonomous Systems, Volume 51, Issue 4, 2005, Pages 275-296, ISSN 0921-8890, https://doi.org/10.1016/j.robot.2005.02.004.

[16] Oh, J.; Han, C.; Lee, S. Condition-Invariant Robot Localization Using Global Sequence Alignment of Deep Features. Sensors 2021, 21, 4103. https://doi.org/10.3390/s21124103

[17] Kanning, T. RPLIDAR, 2019. Disponível em: http://wiki.ros.org/rplidar. Acesso em: 14 Dez, 2021.

[18] Lim D. Turtlebot3, 2018. Disponível em: http://wiki.ros.org/turtlebot3. Acesso em: 14 Dez, 2021.

[19] Aula 5 - Controle de Trajetória de Robôs - Parte2, 2020. 1 vídeo (53 min). Publicado pelo canal Laboratory of Systems Engineering and Robotics. Disponível em: www.youtube.com/watch?v=tm_mEKuNb6w&list=PLI1MB8Tq01INJdTRUKjs42LGaylORblJ8&index=11&ab_channel=LaboratoryofSystemsEngineeringandRobotics. Acesso em: 10 Jun, 2022.

[20] Aula 9 - Localização - Parte 2, 2020. 1 vídeo (29 min). Publicado pelo canal Laboratory of Systems Engineering and Robotics. Disponível em: https://www.youtube.com/watch?v=y_6aT0DfbGo&list=PLI1MB8Tq01INJdTRUKjs42LGaylORblJ8&index=15&ab_channel=LaboratoryofSystemsEngineeringandRobotics. Acesso em: 10 Jun, 2022.

[21] Sivalingam, D. robot_pose_ekf, 2022. Disponível em: http://wiki.ros.org/robot_pose_ekf. Acesso em: 05 Jun, 2022.

[22] AV. AMCL, 2020. Disponível em: http://wiki.ros.org/amcl. Acesso em: 07 Jun, 2022.

[23] S Brito, Claudio. SRC. João Pessoa: Github, 2022. Disponível em: https://github.com/claudiosouzabrito/src. Acesso em: 11 Jun, 2022.