UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO DE MESQUITA FILHO"
CAMPUS DE GUARATINGUETÁ

BRUNO GUEDES DA SILVA

Robô móvel autônomo para execução de tarefas de transportes de materiais em ambientes pré-definidos.

Guaratinguetá
2021

Bruno Guedes da Silva

Robô móvel autônomo para execução de tarefas de transportes de materiais em ambientes pré-definidos.

Trabalho de Graduação apresentado ao Conselho de Curso de Graduação em Engenharia Elétrica da Faculdade de Engenharia do Campus de Guaratinguetá, Universidade Estatual Paulista, como parte dos requisitos para obtenção do diploma de Graduação em Engenharia Elétrica .

Orientador: Prof. Dr. Leonardo Mesquita

Guaratinguetá
2021

Silva, Bruno Guedes da
S586r Robô móvel autônomo para execução de tarefas de transportes de materiais em ambientes pré-definidos / Bruno Guedes da Silva – Guaratinguetá, 2021.
50 f : il.
Bibliografia: f. 50

Trabalho de Graduação em Engenharia Elétrica – Universidade Estadual Paulista, Faculdade de Engenharia de Guaratinguetá, 2021.
Orientador: Prof. Dr. Leonardo Mesquita

1. Robôs. 2. Software - Desenvolvimento. 3. Transporte de materiais. I. Título.

CDU 681.3.06

UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO DE MESQUITA FILHO"
CAMPUS DE GUARATINGUETÁ

BRUNO GUEDES DA SILVA

ESTE TRABALHO DE GRADUAÇÃO FOI JULGADO ADEQUADO COMO PARTE DO REQUISITO PARA A OBTENÇÃO DO DIPLOMA DE **"GRADUANDO EM ENGENHARIA ELÉTRICA "**

APROVADO EM SUA FORMA FINAL PELO CONSELHO DE CURSO DE GRADUAÇÃO EM ENGENHARIA ELÉTRICA

Prof. Dr. DANIEL JULIEN BARROS DA SILVA SAMPAIO
Coordenador

BANCA EXAMINADORA:

Prof. Dr. Leonardo Mesquita
Orientador/UNESP-FEG

Prof. Dr. Samuel Euzédice de Lucena
UNESP-FEG

Prof. Dr. Daniel Julien Barros da Silva Sampaio
UNESP-FEG

Dezembro , 2021

DADOS CURRICULARES

BRUNO GUEDES DA SILVA

NASCIMENTO 14/08/1997 - Taubaté / SP

FILIAÇÃO Almir Jorge da Silva

Claudete Aparecida da Silva Guedes

2013 / 2015 Curso Técnico de Nível Médio - Eletroeletrônica

Colégio Técnico Industrial de Guaratinguetá - UNESP

2016 / 2021 Curso de Graduação - Engenharia Elétrica

Faculdade de Engenharia de Guaratinguetá - UNESP

RESUMO

Neste trabalho é apresentado um sistema de transporte de materiais com robôs móveis autônomos. A tarefa de identificação e transporte de um material de um local a outro é comum em armazéns e fábricas e consome o tempo de funcionários que poderiam executar atividades mais complexas, o que torna a automação da mesma muito atraente. O objetivo principal é o desenvolvimento de um software para que um robô móvel equipado com um braço robótico seja capaz de coletar, transportar e depositar objetos (materiais, equipamentos, etc.) entre prateleiras de forma autônoma. Para isso, utilizou-se o *Robot Operating System* como *framework* de desenvolvimento e implementou-se componentes necessários. A solução conta com capacidades de navegação para realização da locomoção do robô pelo ambiente, manipulação de um braço robótico para a interação com objetos e a detecção de objetos por meio de tags de realidade aumentada. Testes do sistema foram realizados em simulação, onde foi verificado a execução individual de cada componente e observou-se a correta operação de todos. O sistema completo foi testado em simulação executando diversas sequências de transporte e obteve-se uma taxa de sucesso de 91,7%. Por fim, também realizou-se testes com o robô real para a verificação da execução da sequência de coleta e depósito de um bloco com o robô estacionário.

PALAVRAS-CHAVE: Robô móvel autônomo. Braço robótico. *Pick-and-place*. AR-tag. Detecção de objetos. Navegação autônoma.

ABSTRACT

In this work, a material transportation system with autonomous mobile robots is presented. The task of identifying and transporting a material from one place to another is common in warehouses and factories and consumes the time of employees who could perform more complex activities, which makes the automation of this task very attractive. The main objective is to develop a software for a mobile robot equipped with a robotic arm to be able to autonomously pick, transport and place objects (materials, equipment, etc.) between shelves. For this, the Robot Operating System was used as a development framework and the necessary components were implemented. The solution has navigation capabilities to allow the robot to move around the environment, manipulation of a robotic arm to interact with objects, and object detection through augmented reality tags. System tests were performed in simulation, where the individual execution of each component was verified and the correct operation of all of them was confirmed. The complete system was tested in simulation by executing several transport sequences and a 91.7% success rate was obtained. Finally, tests were also performed with the real robot to verify the execution of a pick and place sequence of a block while the robot is stationary.

KEYWORDS: Autonomous mobile robot. Robotic arm. Pick-and-place. AR-tag. Object detection. Autonomous navigation.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

AMCL	*Adaptive Monte Carlo Localization*
AMR	*Autonomous Mobile Robot*
AR-tag	Tag de Realidade Aumentada
DWA	*Dynamic Window Approach*
IMU	*Inertial Measure Unit*
Lidar	*Light Detecting and Ranging*
ROS	*Robot Operating System*
Rviz	*ROS Vizualisation*
SDF	*Simulation Description Format*
SLAM	*Simultaneous Localization and Mapping*
YAML	*YAML Ain't Markup Language*

SUMÁRIO

1 INTRODUÇÃO

Atualmente, diversas empresas buscam constantemente melhorar sua produtividade através da automatização de processos. Para isso, tarefas manuais repetitivas ou que demandam grande esforço físico passaram a ser atribuídas a robôs. Nos armazéns e fábricas, a tarefa de coleta e transporte de materiais, que consiste em se deslocar até o local do material, identificar o material a ser coletado, coletá-lo e transportá-lo até um local de destino, é essencial, porém repetitiva e consome o tempo dos funcionários que poderiam estar concentrados em tarefas mais complexas. Para automatizar essa tarefa, torna-se interessante a implementação de robôs móveis autônomos, do inglês *autonomous mobile robot* (AMR), capazes de realizar a coleta e transporte de materiais sob demanda de forma independente.

Para a implementação de AMR para transporte de materiais, diferentes problemas devem ser resolvidos. Primeiramente, um AMR deve ser capaz de se mover pelo ambiente sem a intervenção humana, o que envolve a habilidade de estimar a sua própria localização e determinar uma trajetória ótima até seu destino. Além disso, é necessário detectar os objetos a serem transportados e determinar suas posições, de forma a permitir sua correta manipulação. Por fim, o AMR deve também ser capaz de planejar e executar movimentos com o braço robótico para interagir com os objetos a serem transportados, evitando a colisão com os demais objetos do ambiente.

Nesse trabalho, é apresentada uma solução de transporte de materiais que utiliza métodos de navegação autônoma, computação visual e manipulação de braço robótico para realizar a recuperação dos materiais. O sistema implementado para a solução consiste em um conjunto de componentes que adicionam funcionalidades ao robô e interagem entre si para realizar a tarefa de transporte de objetos. Para isso, utiliza-se o *framework Robot Operating System* (ROS), o qual disponibiliza diversas ferramentas customizáveis para métodos da robótica, facilitando a implementação de sistemas robóticos complexos e sua portabilidade entre a simulação e o robô real.

Dentre as ferramentas disponíveis pelo ROS destaca-se o MoveIt!, o qual permite o planejamento e controle de braços robóticos, e o ArUco, o qual detecta a posição de tags de realidade aumentada a partir da imagem de uma câmera e é utilizado para obter as posições de materiais a serem transportados.

1.1 OBJETIVOS

O trabalho possui os seguintes objetivos gerais:

- Desenvolver um sistema usando ROS para a coleta e transporte de materiais com um AMR;

- Implementação do sistema em ambiente de simulação com um cenário teste para realizar a validação do software;
- Implementação do sistema em um robô real para teste e validação.

O trabalho possui os seguinte objetivos específicos:

- Implementação e configuração dos módulos de localização, navegação e mapeamento do ROS;
- Desenvolvimento do programa para execução da tarefa *pick-and-place* pela garra robótica utilizando o *framework* MoveIt!;
- Modelagem de um ambiente simulado com o uso da ferramenta Gazebo;
- Implementação e configuração do módulo ArUco para identificação e localização de tags de realidade aumentada (AR-tag).

1.2 DIVISÃO DO TRABALHO

Esse trabalho é divido em seis capítulos.

O Capítulo 1 apresenta o tema e o problema tratado, assim como os objetivos gerais e específicos.

O Capítulo 2 apresenta os principais conceitos teóricos dos métodos utilizados no sistema.

O Capítulo 3 detalha os materiais utilizados para o desenvolvimento da solução, incluindo o robô, sensores, manipulador e as ferramentas computacionais ROS, MoveIt! e Gazebo.

O Capítulo 4 apresenta o desenvolvimento do sistema, mostrando a arquitetura geral do sistema e detalhando a implementação de cada módulo.

O Capítulo 5 detalha os testes realizados assim como os resultados obtidos.

O Capítulo 6 discute as conclusões do trabalho assim como apresenta propostas de trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Esse capítulo tem como objetivo apresentar os principais conceitos de robótica relevantes para o entendimento do trabalho desenvolvido. Os conceitos abordados são voltados à robótica móvel e às tarefas de navegação e manipulação.

2.1 ROBÔS MÓVEIS AUTÔNOMOS

Robôs, de maneira geral, são máquinas que executam uma determinada tarefa, capazes de sensorear, planejar e agir em um ambiente (CORKE, 2011). O sistema mecânico que constitui a parte física do robô é composto pelos elementos necessários para a execução da tarefa.

Comumente dois dispositivos estão presentes em robôs móveis, um dispositivo manipulador e um dispositivo de locomoção. Dispositivos manipuladores permitem ao robô executar ações que requerem a interação com objetos no ambiente e podem ser dotados de ferramentas, como equipamentos de solda, perfuração e corte, para realização da tarefa. Dispositivos de locomoção permitem ao robô se mover de um ponto a outro no ambiente, o que caracteriza o robô como móvel.

Robôs móveis autônomos devem ser capazes de realizar tarefas com mínima intervenção humana, podendo coletar informações de um ambiente dinâmico e planejar sobre o mesmo.

2.1.1 Sensoriamento

Sensores são cruciais para os robôs, fornecendo as informações necessárias para o planejamento de ações e podem ser divididos em duas categorias: sensores proprioceptivos, que medem os estados internos do robô, e sensores exteroceptivos que adquirem informação do ambiente ao seu redor (SICILIANO et al., 2009).

2.1.1.1 Sensores Proprioceptivos

Os sensores proprioceptivos fornecem ao robô informações de sua condição interna, podendo monitorar informações como temperatura, nível da bateria e velocidade.

Uma informação essencial para a realização do planejamento e controle de um robô é o conhecimento dos estados de seus atuadores, comumente motores e servomotores. Para isso são empregados *encoders* de posição e velocidade.

Os *encoders* de posição podem ser absolutos ou incrementais. *Encoders* absolutos são capazes de fornecer a posição angular em que o atuador se encontra, sendo útil para uso em braços robóticos, onde é necessário conhecer o ângulo de cada junta do braço para determinar sua configuração atual. Enquanto os *encoders* incrementais emitem um sinal a cada vez que a posição angular é incrementada ou decrementada uma quantidade fixa. A partir disso, é possível determinar o deslocamento angular atual relativo a quando o *encoder* foi inicializado.

Juntamente com a posição dos atuadores, é relevante conhecer a orientação e velocidade do robô como um todo, para isso utiliza-se geralmente uma unidade de medição inercial, do inglês *inertial measure unit* (IMU). A IMU combina dois sensores diferentes, um giroscópio e um acelerômetro, proporcionando medições da orientação relativa do robô, assim como velocidades translacionais e rotacionais.

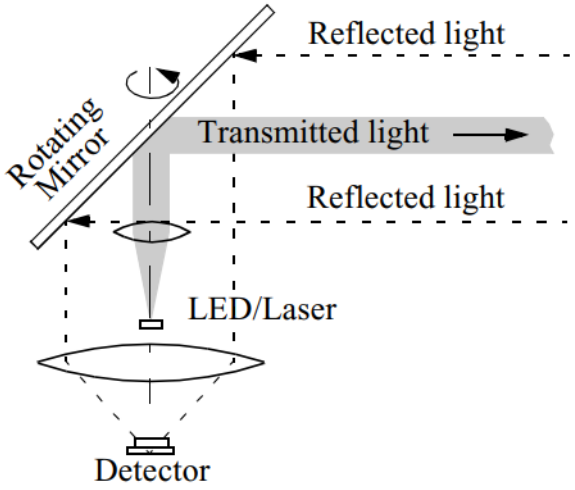
2.1.1.2 Sensores Exteroceptivos

Da mesma forma que um robô recebe informações de suas condições internas através de sensores proprioceptivos, os sensores exteroceptivos irão informar ao robô sobre o ambiente em que se encontra.

Para robôs móveis, é relevante receber informações sobre a distância até objetos próximos e a direção que os mesmos se encontram. Esses dados podem ser providos por sensores de distância.

Sensores laser são comumente empregados para tal finalidade por apresentarem boa exatidão e relativamente baixo custo. Tais sensores operam medindo a diferença de tempo ou fase entre um sinal luminoso enviado e o refletido. Além dos sensores lasers fixos, que medem a distância até objetos a partir de uma direção fixa, existem sensores denominados *light detecting and ranging* (lidar) que possuem um espelho montado em um mecanismo giratório o qual redireciona o sinal luminoso conforme Figura 1, permitindo a detecção de objetos em um plano como no exemplo da Figura 2.

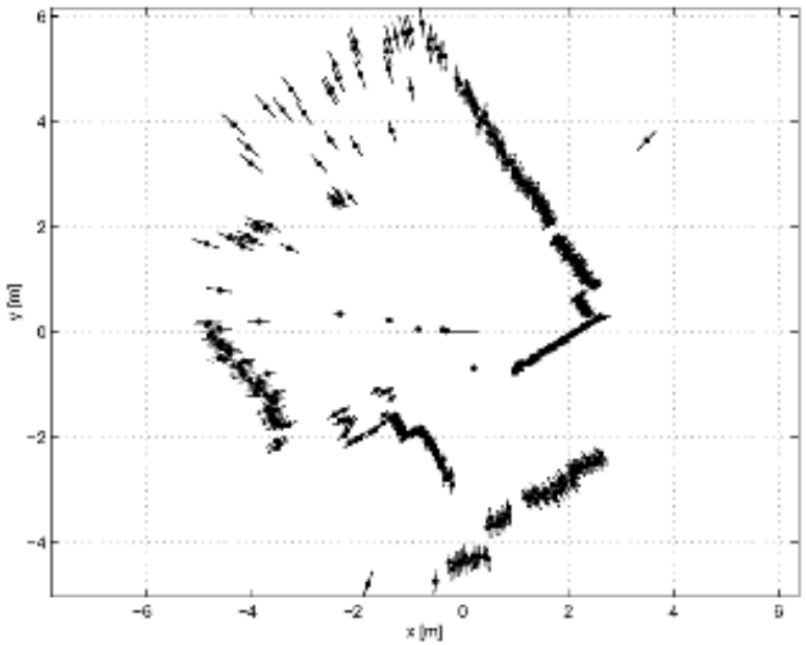
Figura 1 – Modelo de sensor laser com espelho rotativo.



Fonte: Siegwart e Nourbaksh (2004)

Uma enorme quantidade de informações do ambiente também pode ser obtida a partir da visão. Para isso câmeras são utilizadas para redirecionar e focar a luz refletida por objetos no ambiente no plano onde se encontra um sensor foto-sensitivo, o qual mede a intensidade da luz recebida. O sinal gerado pelo sensor é então processado para obter uma imagem digital. Diversos

Figura 2 – Exemplo de dados coletados por um lidar (comprimento das linhas indica a incerteza da medição).



Fonte: Siegwart e Nourbaksh (2004)

métodos podem ser utilizados para extrair informações como posição, orientação e classificação de objetos do ambiente. Além disso, muitos algoritmos realizam a extração de pontos relevantes em imagens obtidas sequencialmente por um robô móvel para realizar tarefas de localização e mapeamento.

2.1.2 Planejamento

Os sistemas de planejamento de um robô são responsáveis por conectarem as informações dos sensores com o controle dos atuadores, gerando comandos para execução de ações necessárias para alcançar um objetivo. Para robôs móveis, é essencial a presença de um sistema de planejamento para a navegação. O objetivo básico da navegação é permitir que o robô se desloque de um ponto a outro no ambiente. Para tal, três componentes principais são utilizados: localização, mapeamento e planejamento de rotas.

2.1.2.1 Localização

A localização do robô consiste em obter uma estimativa sobre a sua posição em relação ao ambiente. Um dos métodos mais simples para localização é o uso de dados obtidos pelos *ecoders* e IMU, em conjunto com o conhecimento prévio da estrutura física do robô, como tamanho e posição das rodas, para estimar a posição relativa do robô no ambiente em intervalos de tempo regulares. Esse método é denominado *dead reckoning*.

A grande desvantagem do *dead reckoning* é o acúmulo dos erros dos sensores em cada estimativa de posição, fazendo com que o erro da localização cresça com o passar do tempo. Para corrigir esse erro, aplica-se métodos que utilizam informações de sensores exteroceptivos, como lidar ou câmera. É necessário também o conhecimento do mapa do ambiente, o qual pode ser criado previamente ou construído pelo robô durante a navegação.

O algoritmo de localização de Monte Carlo é um dos métodos utilizados para correção do erro do *dead reckoning*, através da correspondência entre a observação de um sensor e o mapa do ambiente. O algoritmo utiliza uma técnica denominada filtro de partículas, onde diversos possíveis estados (localizações) do robô são mantidos e atualizados baseados no movimento realizado e nas observações do ambiente. Esse método é utilizado, pois permite a localização global do robô e é capaz de convergir para o valor correto, mesmo em sistemas dinâmicos não lineares.

A Figura 3 apresenta um exemplo unidimensional do funcionamento do algoritmo, o qual pode ser descrito nas seguintes etapas:

1. **Inicialização**: N partículas são inicializadas a partir de uma amostragem aleatória do espaço de estados possíveis do robô e atribuídas um peso bel uniforme (Figura 3 (a)).

2. **Medição do sensor**: é feita uma observação z do ambiente a partir de sensores e obtida, para cada partícula, a probabilidade $P(z|x)$ de que a observação z ocorra na posição x. Por fim atualizam-se os pesos bel proporcionalmente à probabilidade calculada. A Figura 3 (b) apresenta essa etapa com o robô realizando a observação de uma porta.

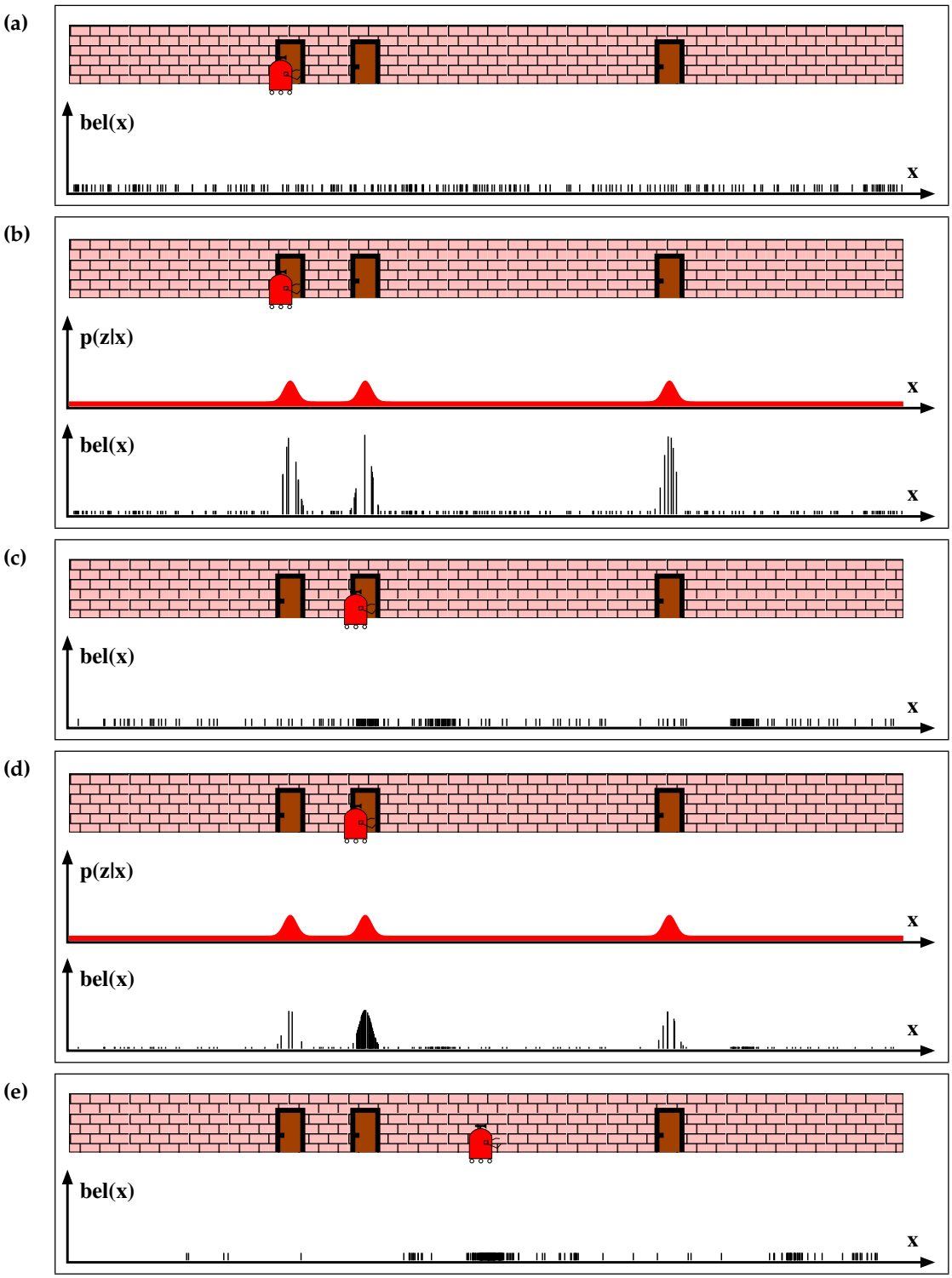
3. **Re-amostragem**: um novo conjunto de partículas é amostrado a partir do conjunto ponderado bel. Dessa forma, mais partículas serão selecionadas próximas a locais onde bel é alto.

4. **Deslocamento**: um comando de movimento é enviado ao robô e para cada partícula é feita uma predição de sua posição final após o movimento. As etapas de re-amostragem e deslocamento são apresentadas em conjunto na Figura 3 (c).

5. **Repetição dos passos 2, 3 e 4**: as etapas são repetidas levando à convergência das partículas.

O algoritmo de localização de Monte Carlo possui diferentes variações propostas para aprimorá-lo ou tratar casos específicos. Uma dessas variações é o algoritmo de Monte Carlo adaptativo, do inglês *adaptive Monte Carlo localization* (AMCL), no qual calcula-se, antes da etapa de re-amostragem, o número de partículas suficiente para representar a distribuição atual. Esse valor é proporcional à confiabilidade que se têm na posição do robô, ou seja, no início (Figura 3(a)) por não se saber a posição é necessário a utilização de várias partículas, enquanto em (e) a posição do robô é delimitada a somente algumas regiões e menos partículas

Figura 3 – Exemplificação em uma dimensão do algoritmo de localização de Monte Carlo.



Fonte: Thrun, Burgard e Fox (2005)

são utilizadas. O AMCL apresenta convergência mais rápida que o algoritmo tradicional, ao mesmo tempo em que melhora a performance e diminui o custo computacional (FOX, 2003).

2.1.2.2 Mapeamento

A tarefa de mapeamento possui o objetivo de construir uma representação do ambiente com a qual o robô possa se localizar e planejar rotas. Utilizam-se os dados obtidos por sensores exteroceptivos para estimar a posição dos objetos no ambiente. Entretanto, durante o mapeamento, é necessário que o robô se mova pelo ambiente de forma a identificá-lo por completo e, para tal, ele deve ser capaz de se localizar enquanto o mapa é construído. Esse problema é denominado de localização e mapeamento simultâneo, do inglês *Simultaneous localization and mapping* (SLAM), para o qual diversas técnicas podem ser utilizadas dependendo dos tipos de sensores existentes.

As *ocupancy grids* são mapas simples de se utilizar para a representação de um ambiente. A *occupancy grid* consiste de uma matriz que corresponde a uma grade quadricular do ambiente, cujas células possuem valores de 0 a 1 representando a probabilidade do espaço estar ocupado (MORAVEC; ELFES, 1985).

2.1.2.3 Planejamento de rotas

Dado uma posição final e um mapa, o planejamento de rota envolve identificar uma trajetória que fará o robô alcançar a posição final quando executada (SIEGWART; NOURBAKSH, 2004). A rota obtida também deve ser capaz de ser adaptada durante sua execução para evitar obstáculos dinâmicos identificados pelos sensores. Uma das técnicas comuns utiliza mapas de custo, onde um mapa é dividido em pequenas células uniformes, às quais são atribuídos custos proporcionais à distância até a posição final. Então, a grade é tratada como um grafo ponderado e um algoritmo de busca pode determinar uma rota ao obter o caminho de menor custo entre o nó da posição atual do robô e o nó da posição final.

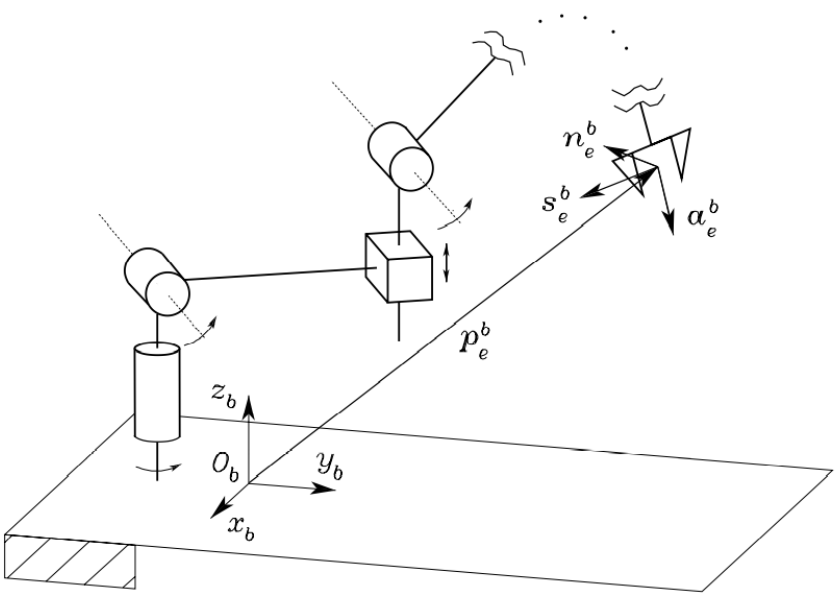
2.2 BRAÇOS ROBÓTICOS

Braços robóticos são sistemas de corpos ligados por juntas, onde cada junta possui um grau de liberdade linear ou rotacional. Tipicamente, um dos corpos da extremidade do sistema é fixado a uma base, enquanto a outra extremidade é livre para se mover no espaço e realizar tarefas. A extremidade livre é denominada *end-effector*.

A localização do *end-effector* no espaço é descrita pela sua posição e orientação, os quais requerem 6 dimensões no total. A Figura 4 mostra a representação de um braço robótico com diversas juntas lineares (cubos) e rotacionais (cilindros). A posição do *end-effector* é dada pelo vetor p, enquanto a orientação é dada pelo sistema n_e^b, s_e^b, a_e^b que corresponde à representação das coordenadas do *end-effector* em relação a origem O_b.

A cinemática é a área da mecânica que estuda o movimento de corpos ou sistemas de corpos sem considerar sua massa ou as forças atuantes (CORKE, 2011). A partir das equações de cinemática de um manipulador, pode-se determinar a posição final do mesmo a partir dos ângulos que cada junta apresenta, processo denominado cinemática direta, assim como realizar

Figura 4 – Descrição da posição e orientação de um *end-effector*



Fonte: Siciliano et al. (2009)

o processo inverso de determinar quais os ângulos necessários para atingir uma posição final, denominado cinemática inversa. Tais processos permitem a implementação de componentes de planejamento de trajetória para o *end-effector*, os quais determinam os valores das juntas em cada instante de tempo para que o *end-effetor* realize um movimento específico no espaço.

A dinâmica do braço também pode ser derivada de sua cinemática, considerando as massas e forças sobre o sistema, permitindo a implementação de controladores para os atuadores de cada junta.

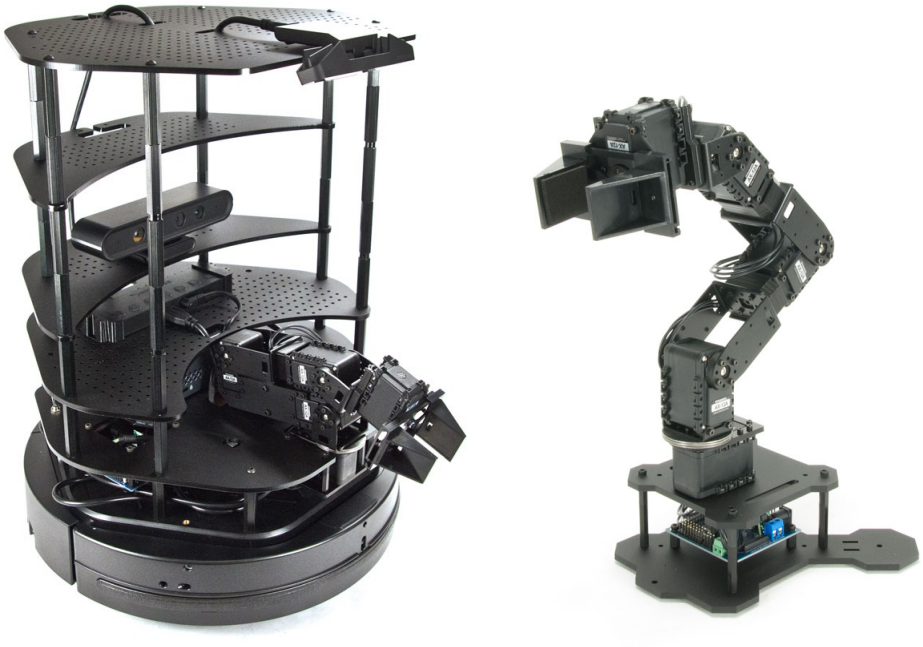
3 MATERIAIS

Para o trabalho diferentes materiais e ferramentas foram utilizados, de forma a implementar e testar o sistema desenvolvido. Esse capítulo apresenta o robô utilizado como base para desenvolvimento, assim como os softwares e *frameworks* usados.

3.1 TURTLEBOT2I

O Turtlebot 2i (Figura 5 (a)) é um robô móvel modular (TROSSEN ROBOTICS, 2021) voltado para pesquisa e baseado em ROS. O robô conta com um chassi modular e é equipado com sensores, mini computador e um braço robótico, permitindo sua utilização em soluções de automação e interação com o ambiente.

Figura 5 – Robô móvel para pesquisa Turtlebot 2i.



a) Turtlebot 2i b) PhantomX Pincher Arm

Fonte: Trossen Robotics (2021)

A base do Turtlebot 2i é um robô móvel Kobuki da Yujin Robot (Figura 6), a partir da qual a estrutura do robô é fixada. A base Kobuki possui duas rodas motorizadas e uma roda de rodízio para estabilidade, permitindo que, a partir do controle das velocidades de cada motor, o robô se locomova para frente e para trás, rotacione sobre o próprio eixo, ou realize uma trajetória em arco.

A base Kobuki também conta com sensores de precipício e de batida para medidas de segurança. As rodas possuem *encoders* que fornecem dados de posição angular para o ROS,

assim como uma IMU que fornece dados inerciais. Por fim, a base também vem equipada com conectores de energia que são utilizados para alimentar os demais componentes do Turtlebot 2i.

Figura 6 – Base móvel Kobuki.

Fonte: Yujin Robot (2021)

Duas câmeras estão presentes no Turtlebot 2i: a Orbbec Astra Pro com alcance de até 8 m e sensores para captura de imagens RGB e de profundidade, e a Intel RealSense SR-300 com 2 m de alcance também captura imagens RGB e de profundidade. Foi adicionado à base do robô um sensor lidar para realização de tarefas de mapeamento, localização e planejamento de rotas.

Acoplado ao robô, está o manipulador robótico PhantomX Pincher (Figura 5 (b)) com quatro graus de liberdade. As juntas são atuadas com servo motores e permite ao braço suportar uma carga máxima de 250 gramas. Os servos motores são controlados por microcontrolador já programados para interfacear com o ROS e o *framework* MoveIt! detalhados na seção 3.2 e subseção 3.2.1.

Todos componentes do Turtlebot 2i são conectados ao computador portátil Intel NUC6, o qual é equipado com processador Intel Celeron, 8 GB de memória RAM e SSD de 120 GB. O Intel NUC6 possui sistema operacional Ubuntu 18.04 e utiliza a versão Melodic Morenia do ROS para o controle do robô.

3.2 ROBOT OPERATING SYSTEM

O *Robot Operating System* é um *framework* gratuito e de código aberto que permite a criação de softwares para robôs de forma flexível e em escala. Consiste de uma coleção de ferramentas e bibliotecas para facilitar o desenvolvimento de tarefas complexas em diferentes plataformas robóticas (OPEN ROBOTICS, 2021).

Conforme Quigley et al. (2009), o ROS possui cinco características definidoras:

- **Peer-to-peer**: Sistemas que utilizam ROS consistem em diversos processos simultâneos executados em diferentes máquinas ou em somente uma e conectados entre si por uma

topologia *peer-to-peer*. Para isso, um processo denominado *master* é responsável por gerenciar a comunicação, permitindo que processos descubram um ao outro durante execução;

- **Baseada em ferramentas**: Os sistemas desenvolvidos com ROS consistem de diversas ferramentas e são executadas de forma distribuída permitindo que comportamentos complexos sejam gerados;
- **Multilinguagens**: Os componentes do ROS são de linguagem neutra, dando flexibilidade para os desenvolvedores criarem seus módulos utilizando diferentes linguagens de programação. Atualmente ROS fornece suporte para as linguagens C++, Python e Lisp, além de bibliotecas experimentais para Java, Lua, MATLAB, R, entre outras. Isso é obtido a partir de uma interface de descrição de linguagem, onde somente as mensagens que são trocadas entre os componentes através da rede *peer-to-peer* precisam ser definidas. Código para implementação da mensagem é gerado automaticamente pelo ROS para as linguagem suportadas, fornecendo uma interface para criação e leitura da mensagem. Isso efetivamente permite que componentes desenvolvidos em diferentes linguagem troquem facilmente informações entre si;
- **Fina**: O *framework* ROS foi desenvolvido de forma que os algoritmos complexos possam ser desenvolvidos e mantidos fora dos módulos do ROS, sendo necessário somente criar componentes pequenos que importem essa bibliotecas e exponham sua funcionalidade ao sistema. Isso permite que algoritmos sejam facilmente reutilizados em diferentes aplicações e pode-se beneficiar de programas de código aberto da comunidade;
- **Gratuito e de código aberto**: Todo código fonte do ROS é disponível publicamente, permitindo contribuições contínuas da comunidade e que aplicações comercias e gratuitas sejam criadas.

Tendo essas características como base, Quigley et al. (2009) define os quatro conceitos principais utilizados na implementação de sistemas em ROS:

- **Nós**: Cada módulo que executa um processo computacional é denominado um nó. Os nós podem se comunicar entre si por conexões peer-to-peer e cada um é responsável por executar uma tarefa específica;
- **Mensagens**: A comunicação realizada entre os nós utiliza mensagens para troca de dados. Cada mensagem é um dado estruturado, o qual é estritamente tipado e definido utilizando a interface de definição de linguagem;
- **Tópicos**: O envio e recebimento de mensagens é realizado através de tópicos. Nós que enviam dados *publicam* mensagens em um determinado tópico, enquanto nós interessados em receberem estes dados devem se *subscrever* no mesmo tópico para acessarem as

mensagens. Dessa forma, os nós não estão cientes um do outro e comunicações do tipo um para um, um para N e N para N são possíveis;

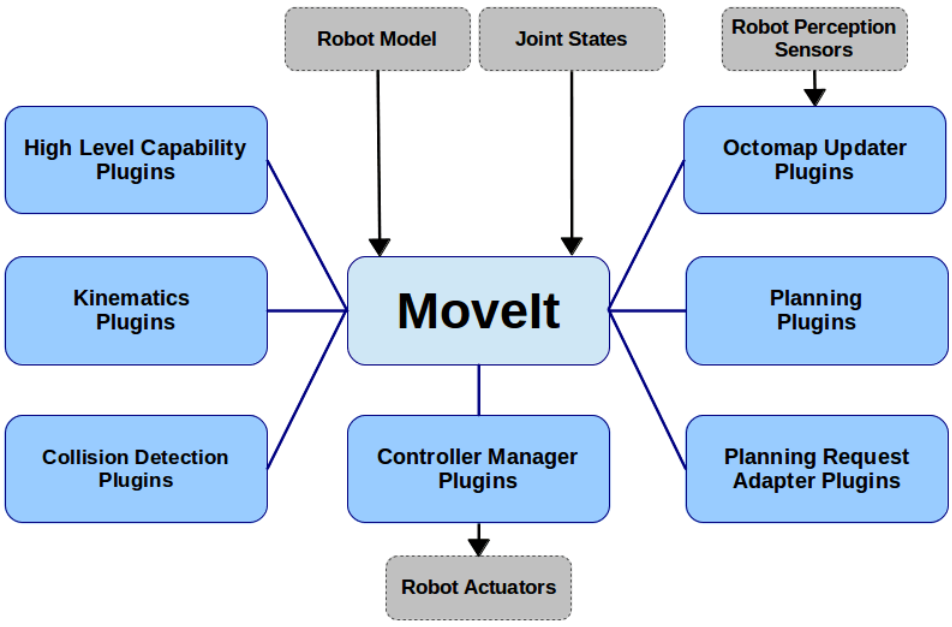
- **Serviços**: Serviços permitem a troca de mensagens entre nós assim como os tópicos, porém realizam isso de forma síncrona. Os dados trocados entre nós através de serviços são definidos em pares de mensagens, sendo um correspondente à solicitação e outro à resposta. Diferentemente dos tópicos, somente um nó pode anunciar um serviço.

Os arquivos que constituem um software em ROS são organizados em unidades chamadas pacotes. Um pacote pode conter o código de um ou mais nós, arquivos de definição de mensagens e/ou serviços, arquivos de configuração utilizados, datasets, entre outros arquivos relevantes. Os pacotes também são os componentes bases para distribuição, ou seja, os componentes dos ROS são fornecidos através de pacotes, os quais podem ser específicos, possuindo somente um nó, ou mais gerais, possuindo até mesmo diversos outros pacotes aninhados.

3.2.1 MoveIt!

MoveIt! é um *framework* de planejamento de movimento desenvolvido em ROS e fornece uma variedade de ferramentas para utilização de braços robóticos. O MoveIt! possui componentes para o planejamento de movimento, detecção de colisão, cálculo de cinemática direta e inversa e outros implementados através de plugins. Isso é obtido com a utilização de interfaces construídas com o sistema de mensagens do ROS e permite que um ou mais componentes sejam facilmente substituídos ou customizados sem a necessidade de alterar os demais. A Figura 7 apresenta os principais plugins disponíveis no MoveIt!.

Figura 7 – Plugins disponíveis pela interface do MoveIt!.



Fonte: Sucan e Chitta (2021)

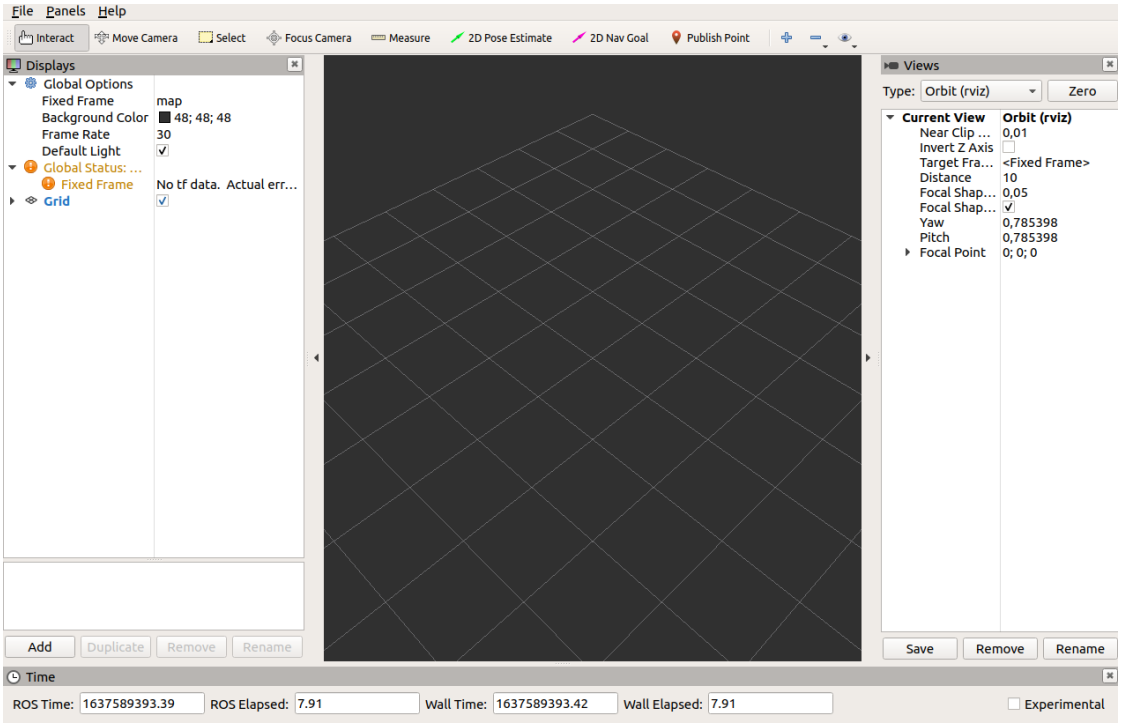
O componente central da Figura 7 representa dois importantes nós ROS disponibilizados pelo MoveIt!. O primeiro nó é chamado *move_group* e é o principal componente presente no pacote MoveIt! para utilizar as capacidades fornecidas pelos seus plugins. Esse nó integra os componentes para fornecer uma interface única com ações e serviços a serem chamados pelo usuário. Durante sua inicialização, o nó busca por parâmetros de configuração em arquivos previamente definidos de forma a determinar quais plugins e funcionalidades devem ser utilizados.

O segundo nó é o *planning_scene*, o qual mantém uma representação do ambiente em que o braço está presente. O nó *planning_scene* recebe informações sobre a posição das juntas do manipulador, informações de sensores configurados e informações fornecidas diretamente por mensagens. Todas esses dados são utilizadas pelo *move_group* para comunicar as informações necessárias aos plugins durante tarefas de planejamento e detecção de colisão.

3.2.2 ROS Vizualization

O ROS fornece junto de seu pacote básico uma ferramenta de visualização 3D chamada ROS Vizualization (Rviz). O Rviz consiste de uma interface gráfica que permite a observação dos dados transmitidos pelos tópicos de forma gráfica. A Figura 8 mostra a tela principal da interface do Rviz, onde observa-se à esquerda a lista de displays, ao centro a tela de visualização 3D e à direita o painel de controle da câmera.

Figura 8 – Tela principal da interface gráfica do Rviz.

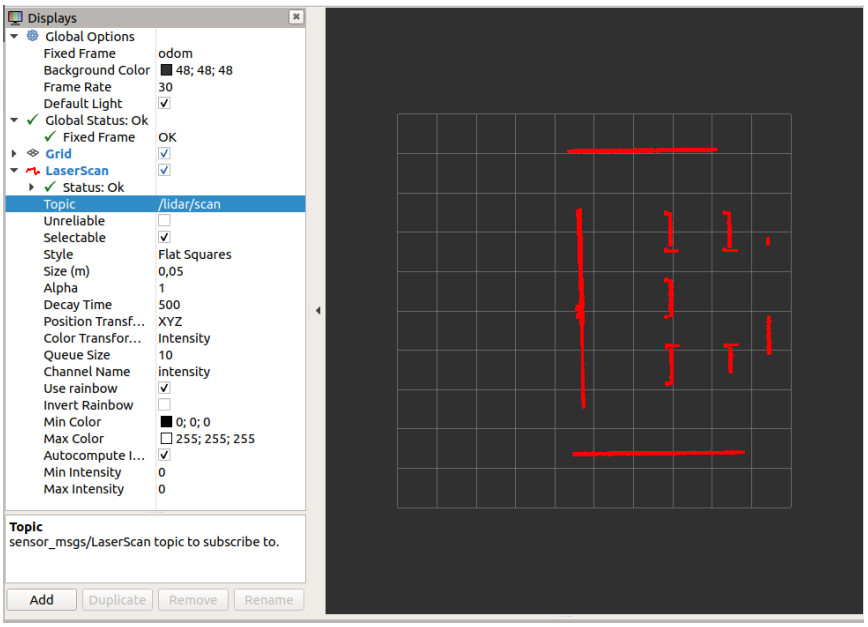


Fonte: Open Robotics (2021)

O Rviz possui pluguins padrões para a visualização de mensagens de tipos básicos do ROS,

como as transmitidas por câmeras e lidars, assim como mapas, trajetórias e modelos cinemáticos. Cada tópico a ser visualizado deve ser associado a um display, o qual fará com que o Rviz se subscreva ao tópico em questão e irá exibir no modelo 3D do ambiente uma representação montada a partir dos dados recebidos. Por exemplo, para a visualização de dados de um sensor lidar, deve-se utilizar o display do tipo *Laser Scan*, o qual deve ser associado a um tópico com mensagens do tipo *sensor_msgs/LaserScan* e então os dados recebidos serão exibidos como uma série de pontos nas distâncias medidas pelo lidar conforme a Figura 9.

Figura 9 – Exemplo de visualização de dados lidar no Rviz.



Fonte: Produção do próprio autor.

Adicionalmente, o Rviz possui plugins que permitem o envio de comandos para alguns nós de planejamento do ROS. Entre eles se destaca a integração com o MoveIt!, onde é possível visualizar o braço robótico e enviar posições objetivo do *end-effector* para planejar e executar trajetórias, e também a integração com o planejamento de rotas, sendo possível indicar uma posição no mapa para a qual o robô deve se deslocar.

3.3 GAZEBO

O Gazebo é uma ferramenta de simulação 3D de código aberto centrada no teste e desenvolvimento de robôs em ambientes internos e externos. O simulador Gazebo oferece uma variedade de modelos prontos de robôs e sensores, além de motores de física, os quais são métodos para simular a física newtoniana que rege a interação entre os objetos do ambiente, e interfaces gráficas que permitem a visualização do ambiente e monitoração de dados como posição, velocidade e força dos objetos no ambiente.

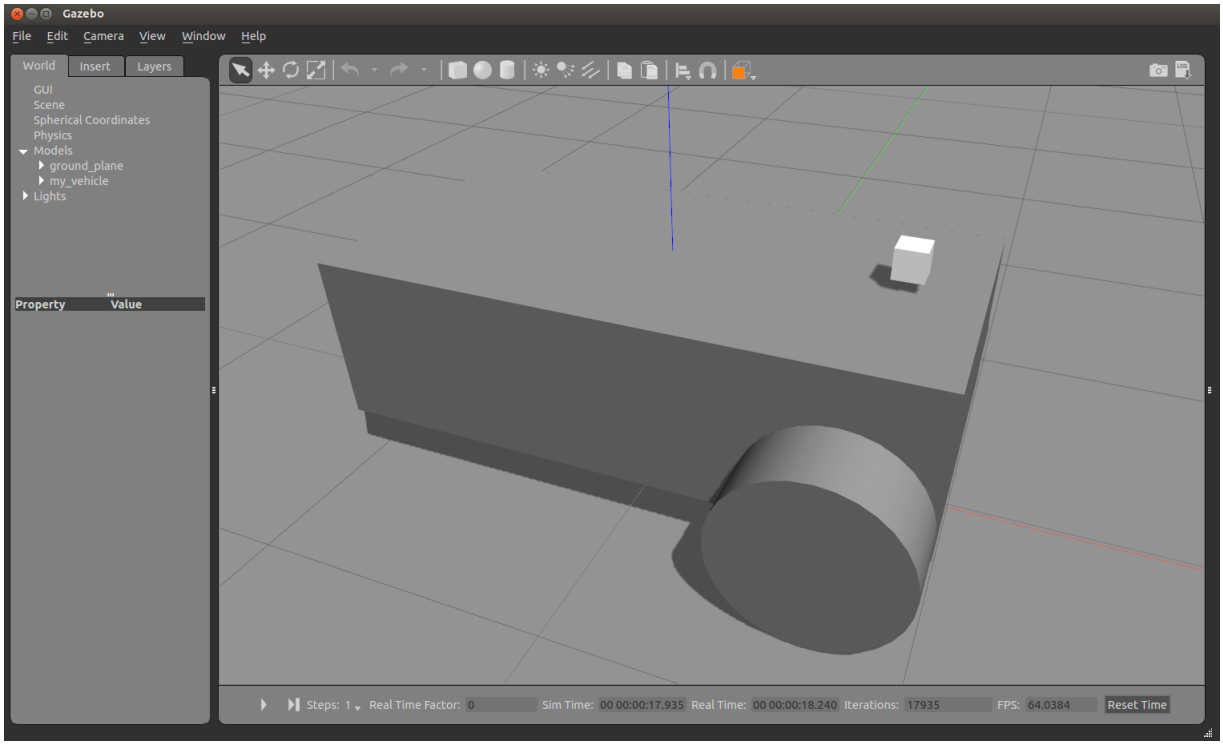
Um cenário simulado no Gazebo é chamado de *mundo*, o qual é definido por um arquivo que usa o Formato de Descrição de Simulação, do inglês *Simulation Description Format* (SDF).

Nesse arquivo está contida a descrição dos diversos elementos presentes em uma simulação. Um elemento da simulação é chamado de *modelo*, podendo representar um objeto, um sensor, um robô, um foco de luz, entre outros.

Pode-se também criar arquivos SDF contendo somente um modelo, os quais podem ser então utilizados para compor um modelo mais complexo ou um arquivo de um mundo. Esse recurso é extremamente útil para construção de mundos complexos, com a presença de diversos elementos dinâmicos.

O simulador Gazebo oferece uma interface gráfica do usuário (Figura 10) para interação com o cenário simulado e para a criação de modelos e mundos, sendo dispensável a escrita manual de arquivos SDF de modelos simples.

Figura 10 – Exemplo de tela da interface gráfica do usuário do Gazebo.



Fonte: Open Source Robotics Foundation (2021)

A integração do simulador Gazebo com o sistema ROS pode ser facilmente implementada com o uso dos plugins fornecido pelo Gazebo. Plugins do Gazebo utilizam sua API para acessarem e/ou modificarem informações do mundo simulado e expõem estas funcionalidades em tópicos ou serviços do ROS, permitindo assim que os nós do sistema ROS interajam com a simulação. Diversos plugins estão disponíveis por padrão no Gazebo, como plugins para câmeras, lidar, controladores diferenciais, sensores inerciais e outros, além de ser possível encontrar facilmente outros plugins desenvolvidos pela comunidade e disponíveis como código aberto.

4 IMPLEMENTAÇÃO

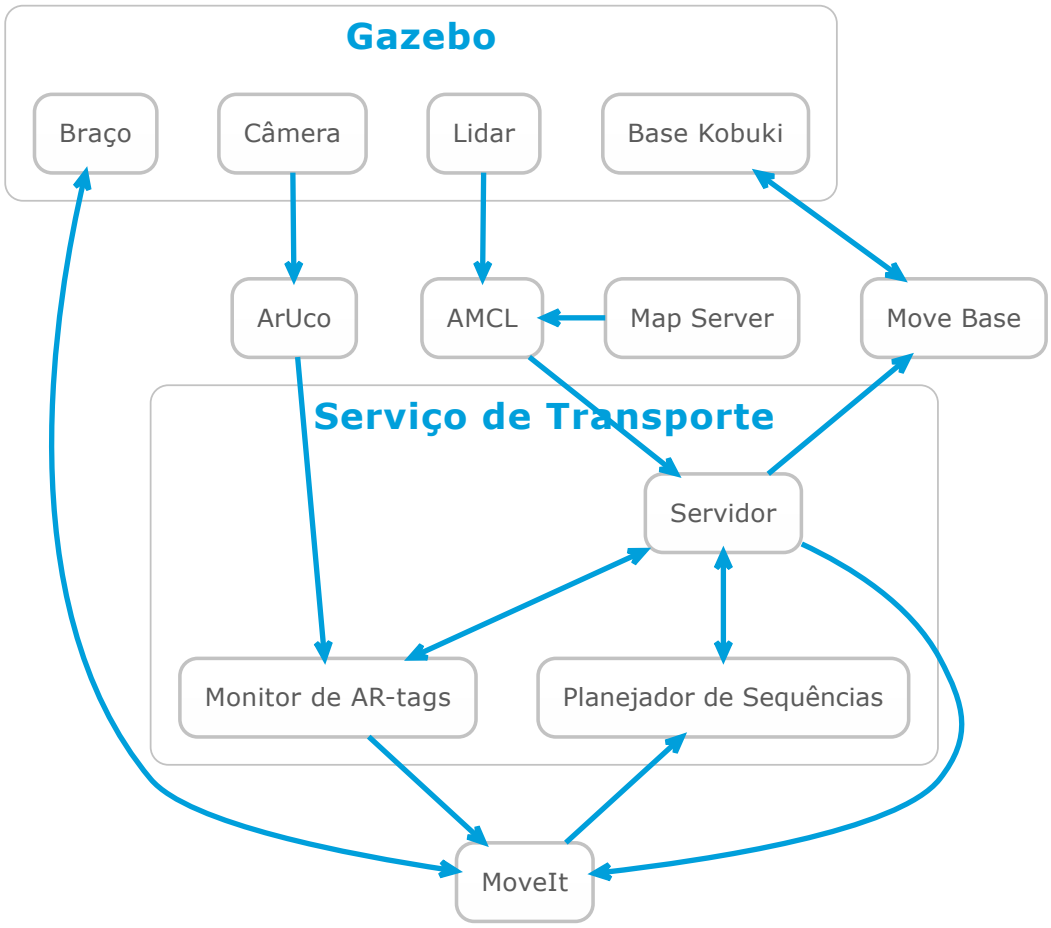
Nesse capítulo é descrito o sistema, detalhando os módulos criados com suas principais características e configurações.

4.1 DESCRIÇÃO DO SISTEMA

O sistema proposto realiza o controle do robô Turtlebot 2i utilizando o ROS. Foram implementados componentes de localização, mapeamento e planejamento de rotas para permitir a locomoção do robô; módulo de detecção de AR-tags para determinar a posição de prateleiras e blocos; e componentes para manipulação de blocos com o *framework* MoveIt!.

A Figura 11 apresenta a organização dos principais componentes que constituem o sistema proposto, demonstrando a direção em que os dados são compartilhados entre os nós. O bloco Gazebo refere-se a simulação utilizada para teste do sistema e é intercambiável com os componentes do robô real.

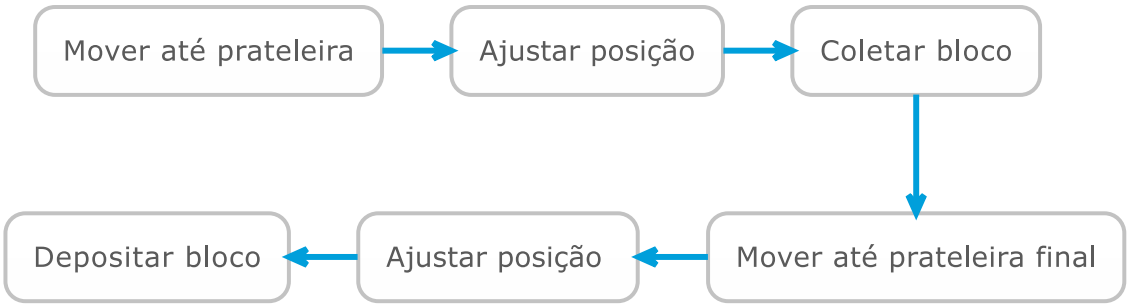
Figura 11 – Diagrama de blocos do sistema implementado.



Fonte: Produção do próprio autor.

O componente chamado Serviço de Transporte, detalhado na seção 4.6, realiza a integração das diferentes capacidades do sistema de forma que o robô apresente o comportamento de alto nível desejado. Esse comportamento é descrito pelo diagrama na Figura 12, o qual possui seis passos principais e são detalhados na subseção 4.6.1. O objetivo é fazer o robô, quando requisitado, transportar um bloco da prateleira A até a prateleira B de maneira autônoma.

Figura 12 – Diagrama de sequência do comportamento de alto nível do robô.



Fonte: Produção do próprio autor.

Nas seções seguintes são detalhadas as implementações dos componentes da Figura 11 separados por funcionalidade, apresentando os nós ROS utilizados para cada um.

4.2 LOCALIZAÇÃO E MAPEAMENTO

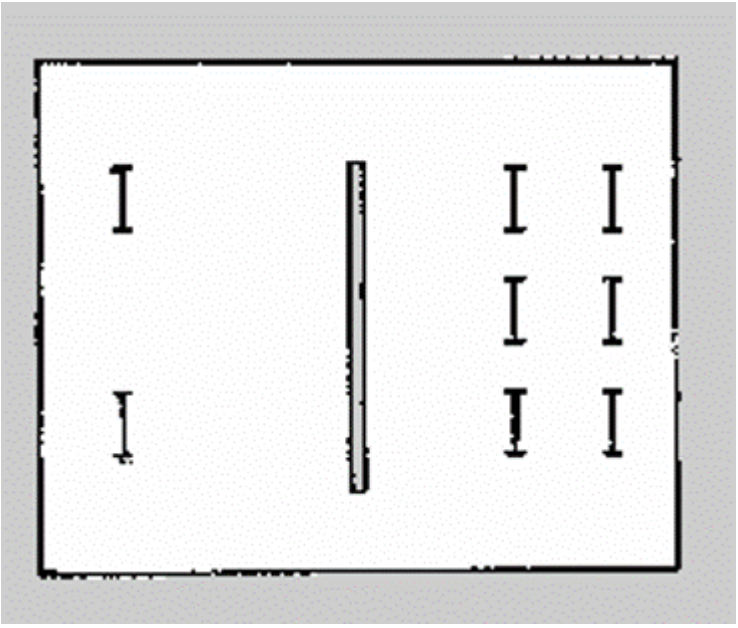
A capacidade de estimar a própria localização é fornecida ao robô através do componente AMCL, o qual é implementado pelo nó *amcl*. Para isso, é necessário o uso de um mapa do ambiente que é disponibilizado pelo componente Map Server utilizado o nó *map_server*.

4.2.1 Map Server

O *map_server* é um nó disponível no pacote de mesmo nome *navigation* do ROS, o qual acessa um arquivo de descrição de um mapa e fornece um serviço ROS para outros nós que desejam a informação. A Figura 13 apresenta o arquivo que descreve o mapa utilizado para a simulação conforme a seção 5. Nesse arquivo o mapa é descrito por uma imagem bidimensional em escala de cinza, na qual células ocupadas, ou seja, onde o robô não pode passar, como paredes e prateleiras, são representados por píxeis pretos e células livres, onde o robô pode circular, são representados pelo píxeis brancos. Os demais locais não mapeados são considerados incertos e apresentam a cor cinza.

Para a configuração do nó, é necessário fornecer um arquivo YAML (YAML Ain't Markup Language) especificando o caminho para o arquivo do mapa, assim como outros parâmetros de configuração apresentados na Tabela 1.

Figura 13 – Grade de ocupação para mapa do cenário simulado.



Fonte: Produção do próprio autor.

Tabela 1 – Parâmetros utilizados na configuração do mapa.

Parâmetro	Valor	Descrição
resolution	0,05	Indica quantos metros equivale um pixel da imagem
origin	x: -8 y: -8 z: 0	Posição do pixel do canto esquerdo inferior da imagem
occupied_thresh	0,65	Define o valore limite da probabilidade que indica espaço ocupado no mapa
free_thresh	0,196	Define o valore limite da probabilidade que indica espaço livre no mapa

Fonte: Produção do próprio autor.

4.2.2 Localização Adaptativa de Monte Carlo

O nó *amcl* implementa o algoritmo de localização probabilístico proposto por Fox (2003) e está disponível no pacote de *navigation* do ROS. O nó recebe mensagens do sensor laser, da odometria da base móvel e do mapa e publica a posição do robô no mapa. Diversos parâmetros podem ser passados ao nó para especificar as características do filtro de partículas, do modelo de sensor laser e do modelo da odometria. A Tabela 2 apresenta os principais parâmetros utilizados para o Turtlebot 2i.

Tabela 2 – Parâmetros utilizados para o algoritmo de Localização Adaptativa de Monte Carlo.

Parâmetro	Valor	Descrição
intial_pose	x: 0,0 y: 0,0 z: 0,0	Posição inicial do robô em relação ao mapa para inicialização do algoritmo.
particles	min: 100 max: 5000	Limite mínimo e máximo do número de partículas amostradas.
odom_model_type	'diff'	Indica o modelo cinemático da base. Pode ser diferencial ou omnidirecional.
odom_alpha	1: 0,2 2: 0,2 3: 0,4 4: 0,2	Definem o ruído esperado nas medições da odometria do robô.
update_min_d	0,2	Distância translacional em metros a ser percorrida para desencadear uma medição do sensor e re-amostragem do filtro de partículas.
update_min_a	π / 6	Deslocamento angular em radianos necessário para desencadear uma medição do sensor e re-amostragem do filtro de partículas.
recovery_alpha	fast: 0,002 slow: 0,15	Fator de decaimento exponencial utilizado para definir quando iniciar comportamento de recuperação.

Fonte: Produção do próprio autor.

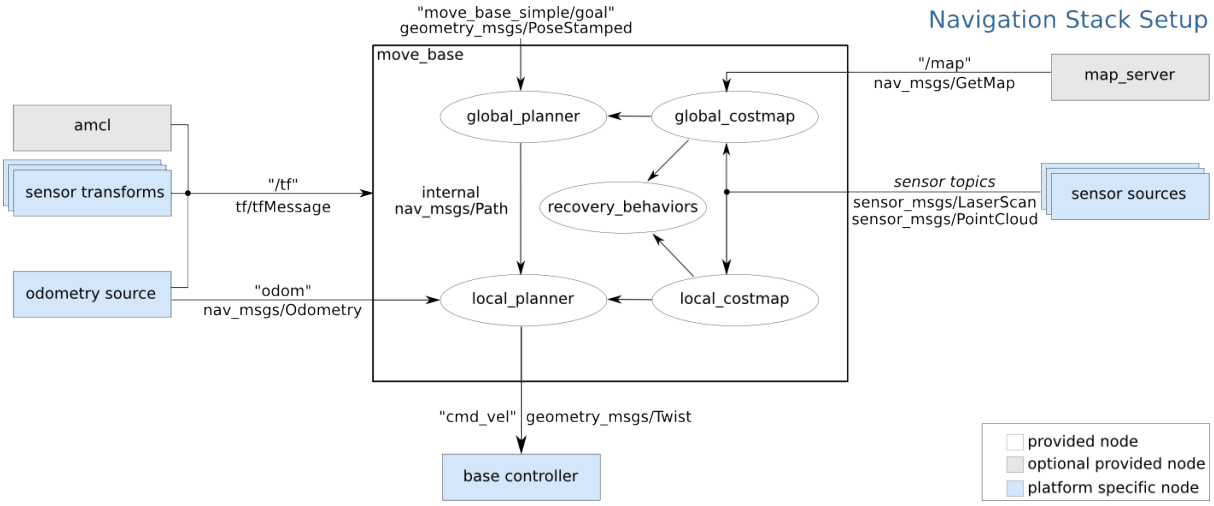
4.3 PLANEJAMENTO DE ROTAS

Dentro do pacote *navigation* do ROS o nó que fornece a funcionalidade de planejamento e execução de rotas é chamado *move_base*. O *move_base* inicializa e mantém outros nós responsáveis por partes específicas da tarefa de navegação, permitindo que, a partir de um arquivo de configuração, diferentes algoritmos sejam utilizados em cada um. A Figura 14 apresenta a arquitetura do sistema usado pelo *move_base*, assim como suas relações com outros nós.

A operação do *move_base* consiste em receber uma posição objetivo para o robô e tentar alcançá-la com a base móvel. Isso é obtido com o uso de dois nós de planejamento, um global e um local. O planejamento global é responsável por obter uma rota entre a posição atual do robô e o objetivo, enquanto o planejamento local considera somente uma pequena região ao redor do robô e deve gerar comandos para seguir a rota global. Os mapas de custo consistem em grades de ocupação que indicam os locais em que o robô deve ou não transitar através da atribuição de pesos a cada ponto do mapa. Esses pesos são valores inteiros de 0 a 254 que são considerados pelos algoritmos de planejamento para gerarem rotas sem colisões.

Os mapas de custo mantidos pelo *move_base* podem ser configurados por um arquivo

Figura 14 – Arquitetura dos nós mantidos pelo *move_base* e suas entradas e saídas.



Fonte: Open Robotics (2021)

YAML. A partir dessas configurações é determinada a área ao redor dos obstáculos na qual o robô não pode transitar, assim como uma região na qual os custos são propagados de maneira inversamente proporcional à distância dos obstáculos, de forma que durante o planejamento o robô prefira rotas distantes dos obstáculos. A Tabela 3 apresenta os principais parâmetros utilizados para configuração do mapa de custo.

Tabela 3 – Parâmetros utilizados para a configuração dos mapas de custo.

Parâmetro	Valor	Descrição
robot_radius	0,18	Raio da base do robô. Utilizado para determinar a região ao redor dos obstáculos na qual ocorreria uma colisão.
inflation_radius	0,55	Distância máxima a partir do obstáculo, na qual os custos serão propagados.
cost_scaling_factor	5,0	Coeficiente do decaimento exponencial considerado na propagação dos custos entre o raio do robô até o raio de inflação.

Fonte: Produção do próprio autor.

Ao mapa de custo também é informado o tópico onde os dados do sensor laser são publicados. Isso permite que o mapa seja atualizado dinamicamente caso objetos não presentes no mapa inicial aparecem ao redor do robô. Esses objetos são então considerados no planejamento de rotas, o qual é refeito se necessário de forma a desviar do objeto detectado.

4.3.1 Planejamento Global

Uma rota global é obtida entre a posição atual do robô e a posição objetivo utilizando o algoritmo de Dijkstra proposto por Dijkstra et al. (1959). Como apresentado na Figura 14 o nó

que realiza o planejamento global é chamado *global_planner* e recebe o mapa de custo global como parâmetro.

4.3.2 Planejamento Local

A partir da trajetória global obtida pelo *global_planner*, o nó *local_planner* é responsável por gerar comandos de velocidade para que o robô siga essa trajetória. Para isso, uma rota cinemática local é determinada utilizando a abordagem de janela dinâmica (DWA). Conforme Fox, Burgard e Thrun (1997) e Open Robotics (2021), na DWA a faixa de possíveis valores de velocidade aceitos pelo controlador é amostrada discretamente. Cada amostra é utilizada para predizer a trajetória executada pelo robô caso seja aplicada ao controlador por um curto período de tempo. Por fim, as trajetórias são avaliadas com base na sua proximidade a obstáculos, ao objetivo final e a rota global e então a melhor trajetória é escolhida. A Tabela 4 apresenta os parâmetros utilizados para configuração do nó *local_planner*.

Tabela 4 – Parâmetros utilizados para a configuração do nó *local_planner*.

Parâmetro	Valor	Descrição
acc_lim	x: 1,0 y: 0,0	O valor limite da aceleração do robô em m/s^2 para x e y e rad/s^2.
max_vel_x	0,5	A máxima velocidade translacional permitida do robô em m/s.
min_vel_x	0,0	A mínima velocidade translacional permitida do robô em m/s.
max_vel_theta	1,0	A máxima velocidade rotacional permitida do robô em rad/s.
min_vel_theta	0,4	A mínima velocidade rotacional permitida do robô em rad/s.
holomic_robot	false	Determina qual tipo de comando de velocidade deve ser gerado pelo nó.
goal_tolerance	yaw: 0,3 xy: 0,15	Configura a tolerância aceita para atingir a posição final.
sim_time	1,0	Período de tempo de simulação de cada amostra.
path_distance_bias	64	Peso referente ao quanto o planejador deve seguir a trajetória global.
goal_distance_bias	24	Peso referente ao quanto o planejador deve buscar a posição final.
occdist_scale	0,5	Peso referente ao quanto o planejador deve desviar dos obstáculos.

Fonte: Produção do próprio autor.

4.4 DETECÇÃO DE BLOCOS

Para que o robô possa ajustar sua posição em relação às prateleiras e coletar e depositar os objetos é necessário que as posições relativas dos mesmos sejam medidas, para isso utilizou-se AR-tags do pacote ArUco acopladas às prateleiras e aos blocos transportados.

Está disponível também, o pacote *aruco_ros*, o qual encapsula as funcionalidades de detecção de AR-tags em nó ROS. Esse nó pode ser inicializado passando parâmetros que indicam o tópico onde a imagem da câmera utilizada para detecção é publicado, assim como outras informações necessárias para determinar a posição das tags, como tamanho e posição de referência da câmera.

Optou-se pela utilização de AR-tags de tamanhos diferentes para a identificação de prateleiras e de blocos, de forma que as AR-tags presentes nas prateleiras são maiores, facilitando a detecção em distâncias maiores. Para isso utilizou-se dois nós *aruco_ros*, pois somente um tamanho de AR-tag pode ser definido por nó.

4.5 BRAÇO ROBÓTICO - MOVEIT!

O *framework* MoveIt! foi utilizado para o planejamento de movimento do braço robótico. Conforme descrito em subseção 3.2.1, o MoveIt! possui diversos plugins para adicionar funcionalidades ao seu componente principal. Para implementação do sistema utilizou-se a interface gráfica de configuração do MoveIt! para gerar os arquivos de configuração e para os nós e plugins padrões necessários para o controle do braço utilizado.

Dentre os plugins padrões utilizados pelo MoveIt!, substituiu-se o plugin de cinemática responsável por obter os ângulos das juntas do robô necessários para alcançar uma determinada posição no espaço. Isso foi necessário pois o plugin padrão considera que o braço robótico possui seis graus de liberdade, enquanto o PhantomX Pincher Arm possui somente quatro, dessa forma ele possui algumas restrições de posições e orientações que pode alcançar. Optou-se então pela utilização do plugin *IKFast*, o qual gera equações para solução analítica da cinemática do braço a partir do arquivo de descrição do mesmo. O plugin foi configurado conforme tutorial disponível na documentação online do MoveIt! (SUCAN; CHITTA, 2019).

4.6 SERVIÇO DE TRANSPORTE

O componente de Serviço de Transporte se subdivide em três componentes. O comportamento de alto nível do robô apresentando na Figura 12 é realizado pelo componente Servidor, o qual expõe um serviço ROS nomeado *transport_material*. Esse serviço recebe como mensagem de entrada três números de identificação: o bloco a ser transportado, a prateleira de coleta e a prateleira de depósito e então realiza a sequência de passos descrita. Para isso utilizou-se dois componentes auxiliares descritos nas subseções seguintes.

4.6.1 Servidor

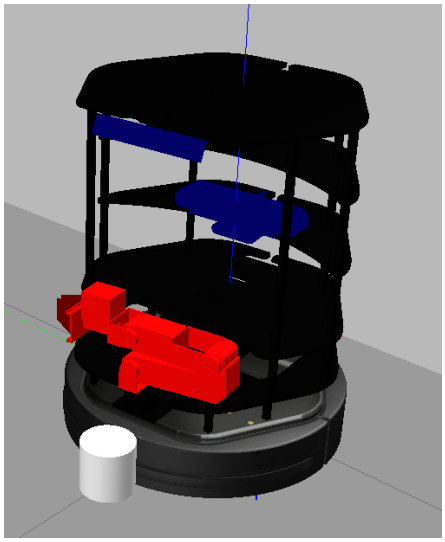
Neste componente é implementado o serviço *transport_material*. Em sua inicialização o nó cria instâncias dos componentes auxiliares Monitor de AR-tags e Planejador de Sequência *pick-and-place*, assim como dos nós *move_base*, utilizado para enviar objetivos de navegação, e *move_group*, utilizado para enviar comandos de planejamento e execução de movimentos do braço.

Ao receber uma mensagem o serviço executa a seguinte sequência de comandos:

1. Recolher o braço.
2. Ir até a prateleira de coleta.
3. Ajustar a posição do robô.
4. Coletar objeto.
5. Recolher o braço.
6. Ir até a prateleira de depósito.
7. Depositar objeto.
8. Recolher o braço.

Para recolher o braço, um comando contendo os valores dos ângulos desejados das juntas dos braços é enviado para o *move_group*, de forma que o MoveIt! realize o planejamento e execução dos movimentos para posicionar o braço, como mostrado na Figura 15.

Figura 15 – Posição do braço retraído.



Fonte: Open Robotics (2021)

As posições das prateleiras estão gravadas no servidor. Dessa forma, ao ser necessário se locomover até uma prateleira, basta que a localização desejada seja enviada para o *move_base* de forma que uma rota seja planejada e seguida. Entretanto, o navegador possui erro e não posiciona o robô em um local ótimo para realizar a coleta do objeto. Por isso, utiliza-se a posição da AR-tag detectada pelo ArUco correspondente à prateleira para ajustar a posição do robô enviando comandos de velocidade diretamente à base Kobuki.

Após o ajuste de posição, o robô se encontra à frente da prateleira e então, com o uso do Monitor de AR-tags, são adicionados objetos de colisão referentes à prateleira e ao bloco a *planning_scene* do MoveIt!. Então pode-se gerar mensagens de *Grasp* ou *Place Location*, dependendo da etapa executada no momento, com uso do Planejador de Sequências, as quais são então passadas para o MoveIt! para que o movimento seja planejado e executado.

4.6.2 Monitor de AR-tags

Esse componente subscreve-se nos tópicos do ArUco para receber as posições das AR-tags detectadas nos blocos e prateleiras. Sempre que uma nova posição é recebida, obtém-se a média das últimas trinta posições da AR-tag detectada para ser utilizada pelo MoveIt! durante a sequência de *pick-and-place*. Isso garante que a posição das AR-tags esteja atualizada e com pouco ruído.

Adicionalmente, o componente mantém uma interface com o nó *planning_scene* do MoveIt! e possui uma função que pode ser chamada pelo componente Servidor para adicionar objetos de colisão correspondentes a prateleiras e blocos quando necessário.

4.6.3 Planejador de sequências *pick-and-place*

O MoveIt! possui métodos para planejar uma sequência de coleta ou depósito completa, permitindo assim planejar o movimento por completo antes de realizá-lo e descobrir se é possível ou não de ser executado. Entretanto, ainda é necessário informar posições específicas necessárias para o movimento. Essas posições são organizadas em uma mensagem ROS, sendo a mensagem para a sequência de coleta chamada de *Grasp* e a para depósito de *Place Location*.

A mensagem *Grasp* possui os seguintes campos:

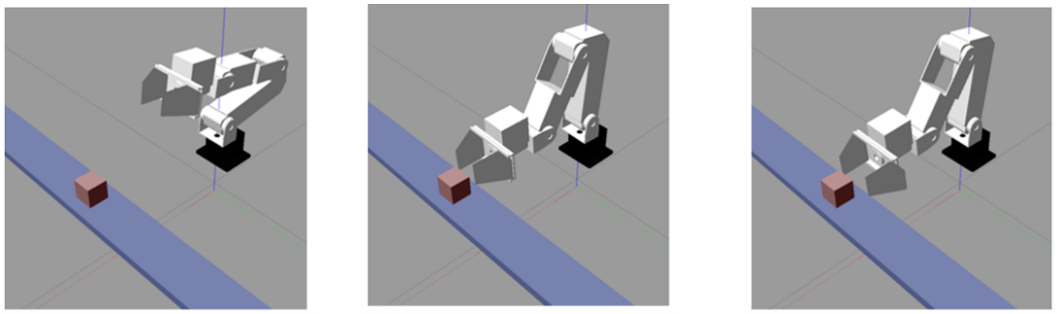
- *pre_grasp_posture*: Define a abertura que a garra do braço deve ter antes de pegar o objeto. É importante que o valor informado seja maior que o tamanho do objeto a ser pego, pois caso contrário não será possível posicionar a garra de forma que o objeto esteja entre os dedos da garra.

- *grasp_posture*: Define a abertura que a garra do braço deve ter para pegar o objeto. O valor a ser informado depende do tamanho do objeto assim como o material de que é feita a garra, pois deve ser pequeno o suficiente para segurar o objeto com firmeza, mas não deformá-lo, caso seja frágil.

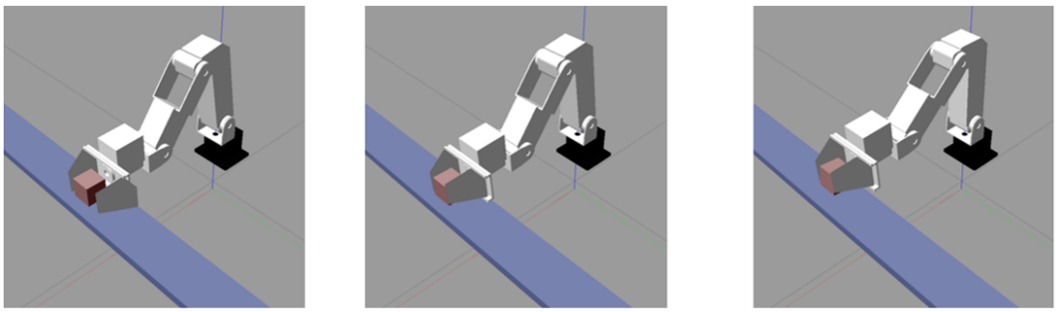
- *grasp_pose*: É a posição em que o braço deve estar para poder pegar o objeto, ou seja, uma posição na qual o objeto esteja dentro dos dedos da garra.
- *pre_grasp_approach*: A direção e distância na qual a garra deve se aproximar do bloco.
- *post_grasp_retreat*: A direção e distância na qual a garra deve levantar o bloco após prendê-lo na garra.
- *grasp_quality*: O método do MoveIt! para planejar uma sequência de coleta também aceita um vetor contendo várias mensagens *Grasp* ao invés de somente uma. Nessa situação, as possíveis sequências são classificadas segundo o valor presente no campo *grasp_quality* de cada uma. A definição da métrica utilizada para definir o valor da qualidade da mensagem é responsabilidade do nó que publica a mensagem, porém é opcional.

Os campos da mensagem podem ser vistos como uma sequência de passos para coletar um objeto, conforme ilustrado na Figura 16. Primeiro o braço do robô se moverá para uma posição *pré_grasp*, definida pela posição *grasp_pose* menos *pre_grasp_approach*. Então a pinça assumirá a abertura de *pre_grasp_posture* e depois se moverá em direção ao objeto na direção de aproximação do *pre_grasp_approach*. Neste ponto, a garra deve estar posicionada sobre o objeto e então fechar até uma abertura de *grasp_posture* para agarrar o objeto. Finalmente, o braço move-se na direção de *post_grasp_retreat* para levantar o objeto.

Figura 16 – Sequência de movimentos executados a partir de uma mensagem *Grasp*.



1. Posição inicial
2. Posição de aproximação
3. Abertura de aproximação



4. Posição de coleta
5. Abertura de coleta
6. Levantar objeto

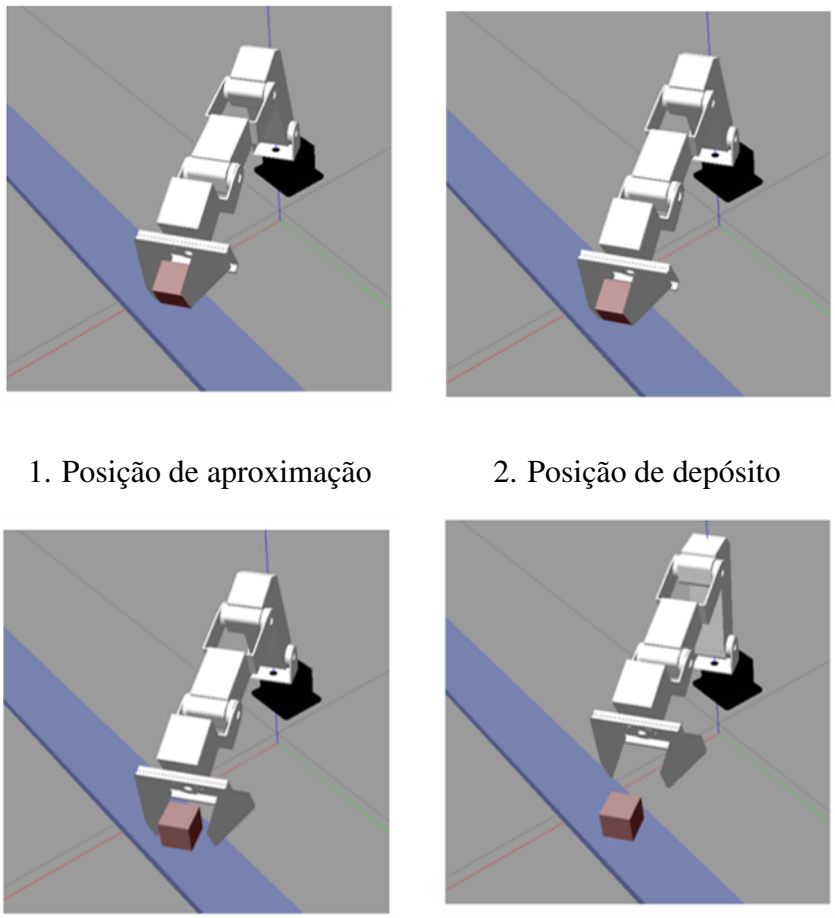
Fonte: Produção do próprio autor.

Semelhantemente, a mensagem *Place Location* possui os seguintes campos:

- *pre_place_approach*: A direção e distância para se aproximar do local de depósito do objeto.
- *place_pose*: A posição em que o objeto deve ser deixado.
- *post_place_posture*: A abertura que a garra deve ter de forma que o objeto seja solto.
- *post_place_retreat*: A direção e distância que a garra deve se mover após soltar o objeto.

A mensagem *Place Location* também define uma sequência de passos a serem executados pelo braço robótico conforme a Figura 17.

Figura 17 – Sequência de movimentos executados a partir de uma mensagem *Place Location*.



1. Posição de aproximação
2. Posição de depósito
3. Soltar objeto
4. Recuo da garra

Fonte: Produção do próprio autor.

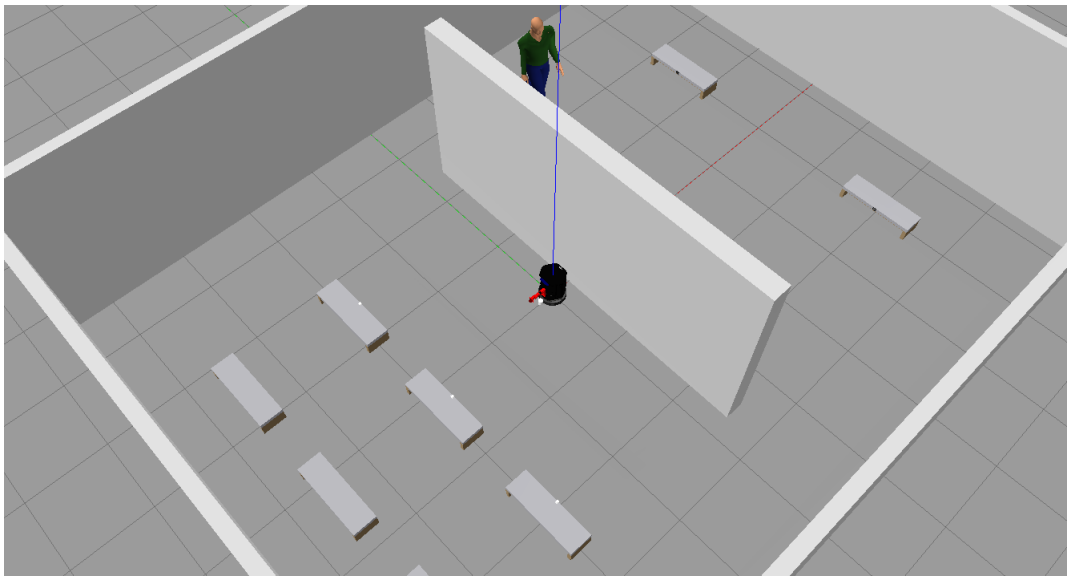
O componente Planejador de Sequências possui dois métodos principais, os quais constroem mensagens *Grasp* e *Place Location* a partir da posição de um objeto de colisão presente na *planning_scene* do MoveIt!. Como parâmetro, pode-se definir o número de mensagens a serem geradas e então a função retornará um vetor com diferentes mensagens, nas quais cada

uma considera um ângulo de aproximação aleatório entre 0° e 90°. Com isso o MoveIt! pode planejar o movimento de maneira variada e obter um que não cause colisões.

5 RESULTADOS

A avaliação do sistema foi feita através de simulação utilizando o *software* Gazebo e os dados publicados pelos nós, por meio de tópicos, foram visualizados através do RViz. Para isso foi modelado um cenário de simulação contendo elementos como o robô Turtlebot 2i, prateleiras e blocos marcados com AR-tags. O cenário modelado, apresentado na Figura 18, é um ambiente interno retangular com uma parede central utilizada para separar os ambientes de coleta e depósito dos blocos. Esse cenário foi utilizado para representar uma versão simplificada de um armazém geral.

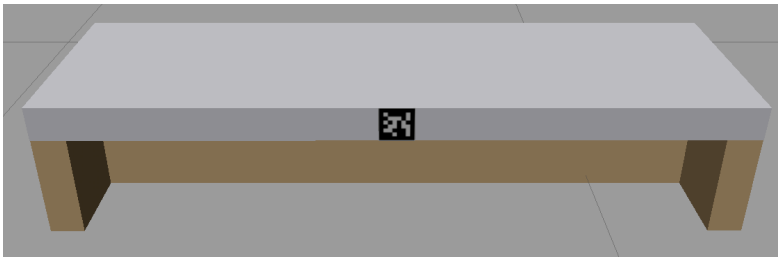
Figura 18 – Visão geral do cenário modelado para a ferramenta de simulação Gazebo.



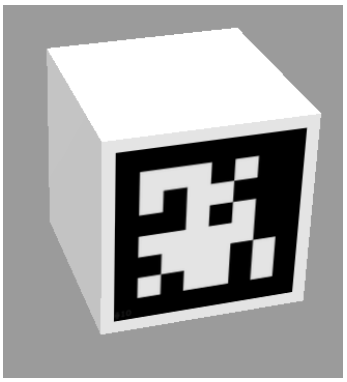
Fonte: Produção do próprio autor.

A Figura 19 apresenta os modelos de simulação utilizados. A prateleira foi feita de forma a ter uma altura mínima para ser alcançada pelo robô, além disso possui um apoio central de forma que o sensor laser do robô possa identificar a presença de um obstáculo. O bloco consiste em um cubo de três centímetros com uma AR-tag ArUco posicionada em sua face frontal. O modelo do Turtlebot 2i é composto por sua base móvel, câmeras, sensor laser e braço PhantomX Pincher Arm, conforme a sua versão real, descrita na seção 3.1. O modelo do robô possui plugins individuais para a simulação dos sensores e controladores presentes, permitindo que dados sejam publicados e/ou recebidos de tópicos ROS.

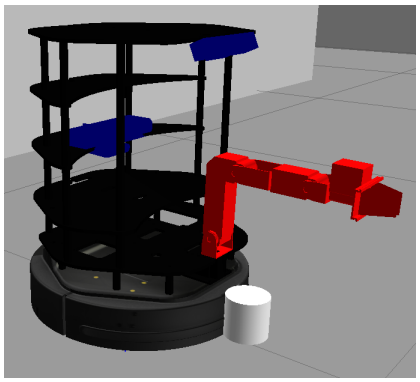
Figura 19 – Modelos criados no Gazebo para a simulação.



(a) Prateleira



(b) Bloco



(c) Turtlebot2i

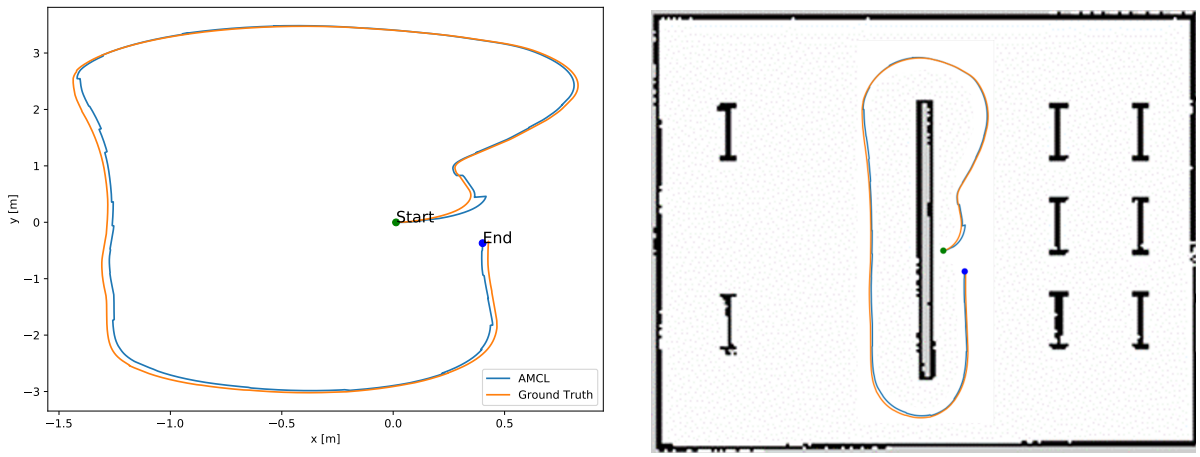
Fonte: Produção do próprio autor.

Foi adicionado ao cenário um modelo de um humano que percorre uma trajetória em linha reta repetidamente durante a simulação. Isso acrescenta um elemento dinâmico à cena, de forma que o robô deve detectá-lo e desviar evitando colisões.

5.1 AMCL

Foram feitos experimentos para verificar o erro da localização estimada pelo AMCL. Para isso, obteve-se os dados gerados pelo nó *amcl*, configurado conforme os parâmetros apresentados na Tabela 2, e a posição verdadeira do Gazebo durante a realização de 7 trajetórias executadas pelo robô com controle manual no cenário modelado. A Figura 20 apresenta um exemplo de trajetória, onde pode-se observar que a posição estimada pelo AMCL (em azul) apresenta um desvio da posição real (em laranja) no início da trajetória e após o deslocamento do robô a posição é reajustada, conforme esperado e descrito na subseção 2.1.2.1. Nota-se também, no segmento da trajetória à esquerda do gráfico (aproximadamente ao longo da reta $x = -1{,}2$ m), que a estimativa do nó *amcl* está deslocada em relação à posição real, o que se deve ao ambiente possuir marcos simples e com pouca distinção entre si, pois as paredes e prateleiras são retas e todas paralelas entre si. Ainda assim o desvio registrado é baixo, mostrando robustez no método e não afeta a navegação do robô.

Figura 20 – Exemplo de trajetória para teste da localização por AMCL.



(a) Trajetória ampliada. (b) Trajetória executada

Fonte: Produção do próprio autor.

A partir dos dados coletados na execução das trajetórias, calculou-se o erro absoluto médio da localização de todas as 7 trajetórias para as coordenadas x, y e orientação θ. Os valores são apresentados na Tabela 5.

Tabela 5 – Erros absolutos médios da localização AMCL obtidos pelo controle manual do robô na trajetória ilustrada na Figura 20.

Coordenada	Erro
x [m]	0,0284
y [m]	0,0285
θ [rad]	0,4760

Fonte: Produção do próprio autor.

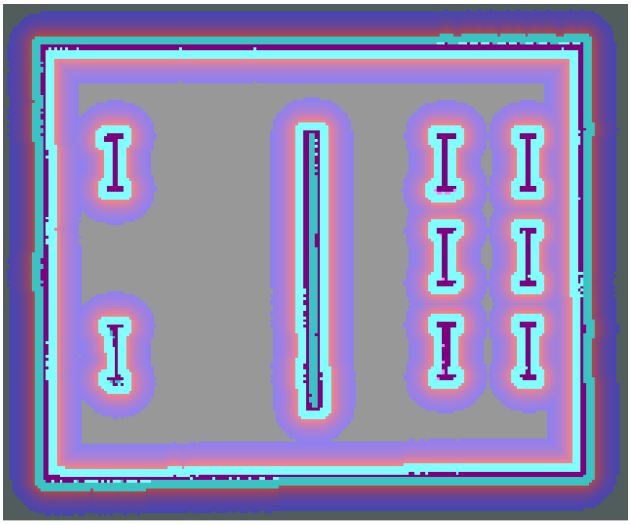
O erro médio apresentado na Tabela 5 indica um desvio na posição estimada em relação à posição real do robô de aproximadamente 3 cm nos eixos x e y, e de 30° na orientação. Esse erro mostrou-se aceitável, por se tratar de um cenário de espaço amplo onde o robô não precisa navegar em locais estreitos, além de ser baixo o suficiente para não prejudicar o posicionamento do robô a uma distância adequada das prateleiras para realização do *pick-and-place*.

5.2 PLANEJAMENTO DE ROTAS

A Figura 21 apresenta a visualização do mapa de custo global mantido pelo nó *move_base* utilizando os parâmetros definidos na Tabela 3, no qual as áreas em ciano são os obstáculos e as regiões em vermelho à lilás são os pesos da região de segurança ao redor deles, sendo que a cor vermelha indica pesos maiores.

De forma a verificar se as rotas foram corretamente geradas pelo algoritmo de Dijkstra e corretamente navegadas pelo robô, diversas posições de destino foram alimentadas ao nó *move_base* através da interface Rviz. A Figura 22 mostra dois exemplos de rotas testadas,

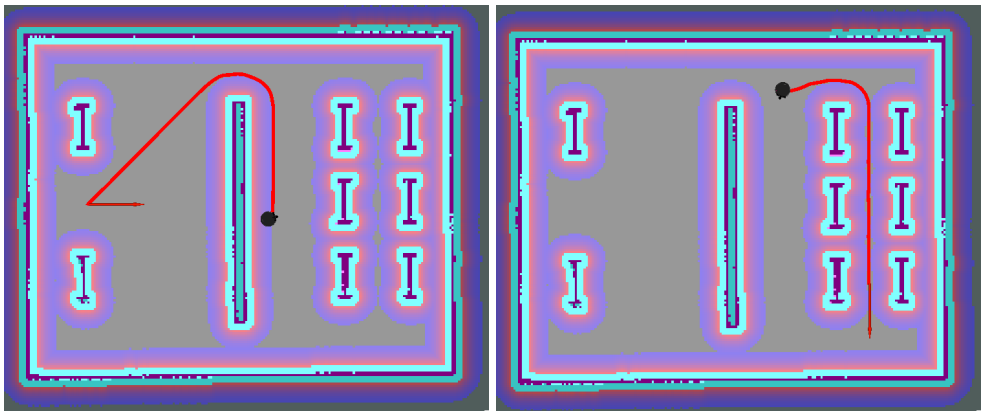
Figura 21 – Mapa de custo mantido pelo *move_base*.



Fonte: Produção do próprio autor.

onde pode ser observado como a trajetória gerada (em vermelho) contorna a parede central na Figura 22a mantendo a distância devido a região em lilás ao redor da parede que indica locais com peso no mapa. Da mesma forma, na Figura 22b a trajetória gerada mantém o robô equidistante das prateleiras por ser o caminho de menor peso até o destino.

Figura 22 – Exemplos de rotas geradas pelo *global_planner*, onde a seta vermelha representa a posição e orientação de destino.



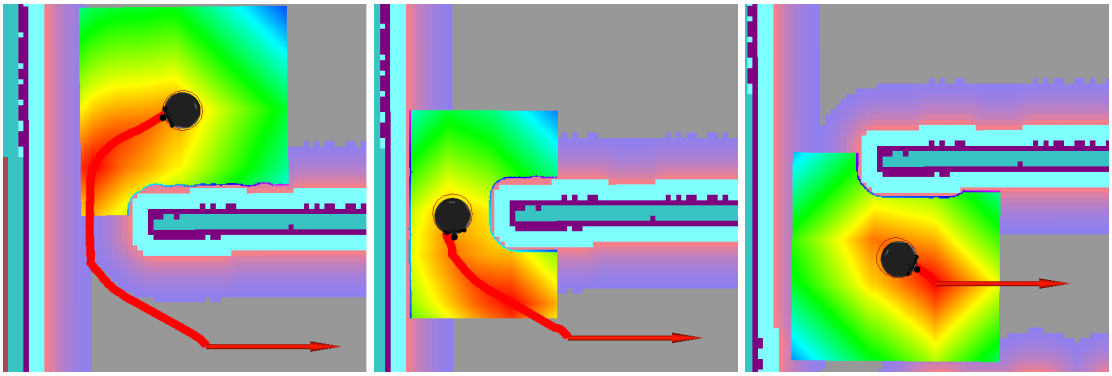
(a) Exemplo de desvio da parede. (b) Exemplo de desvio das prateleiras.

Fonte: Produção do próprio autor.

Quanto ao planejamento local, pode-se observar a grade de custos gerada pelo algoritmo DWA, o qual foi configurado conforme os parâmetros da Tabela 4, e sua modificação conforme o robô se aproxima do objetivo. A Figura 23 apresenta os custos do algoritmo DWA em três instantes durante a execução de uma trajetória, onde os custos são representados por uma escala de cores do vermelho, menor custo, ao azul, maior custo. Pode-se observar como o robô tende a desviar de objetos ao seguir as trajetórias de menor custo. No fim da trajetória, nota-se também

que os custos menores se concentram ao redor da posição objetivo.

Figura 23 – Visualização do custo total das trajetórias previstas pelo algoritmo DWA.

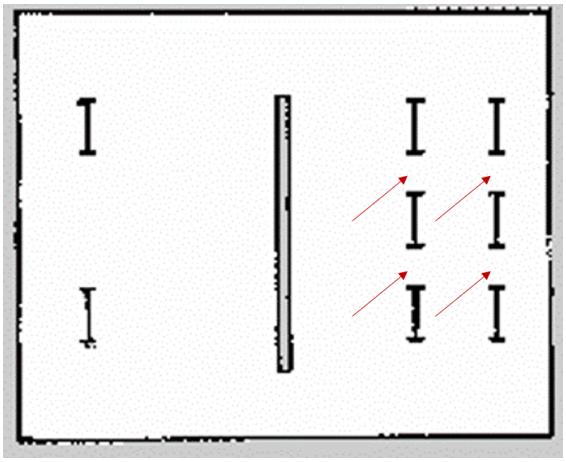


(a) Início da trajetória. (b) Meio da trajetória. (c) Próximo ao fim da trajetória.

Fonte: Produção do próprio autor.

O planejamento de rotas não apresentou erros ao gerar rotas, porém durante a navegação do robô através das rotas geradas observou-se dificuldades em se mover entre as prateleiras, conforme os locais indicados na Figura 24. Isso ocorreu, pois o espaço disponível para o robô transitar era menor do que o diâmetro de sua base, o qual inicialmente foi configurado para um valor acima do diâmetro real do robô para garantir sua segurança. Após a diminuição da configuração do raio do robô (Tabela 3), o robô pôde transitar entre as prateleiras com maior facilidade.

Figura 24 – Locais entre prateleiras em que o robô apresenta dificuldades para atravessar.



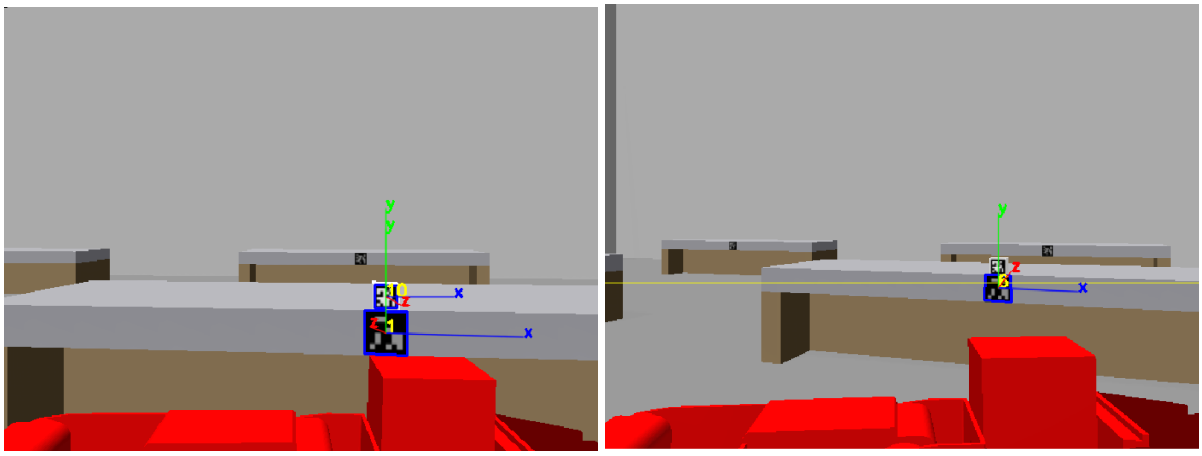
Fonte: Produção do próprio autor.

5.3 ARUCO

O nó do ArUco retorna a imagem da câmera sobreposta ao identificador único (ID) de cada AR-tag detectada. Dessa forma é possível visualizar e confirmar a correta identificação das

AR-tags a partir do Rviz, como apresentado na Figura 25a, onde as AR-tags com IDs 1 para a prateleira e 10 para o bloco são identificados.

Figura 25 – Exemplos de visualização de AR-tags detectadas pelo nó ArUco.



(a) Identificação correta. (b) Identificação errada.

Fonte: Produção do próprio autor.

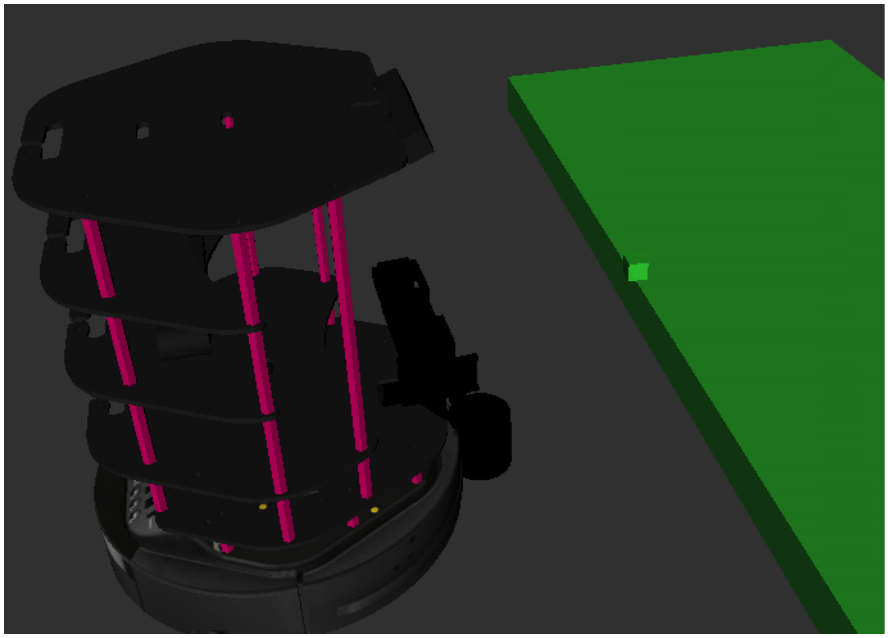
Observou-se que, quando o robô está em movimento, as AR-tags são identificadas incorretamente, retornando IDs errados, como pode ser visto no exemplo da Figura 25b, onde o ID da prateleira 1 é erroneamente identificado como 6. Esse erro ocorre com baixa frequência e não afeta o desempenho na identificação das posições das prateleiras e blocos, pois a posição considerada para realizar a sequência de *pick-and-place* é a média das últimas 30 medições obtidas.

5.4 MOVEIT!

Com o uso do Rviz pode-se observar a cena mantida pelo *planning_scene*, isso permite verificar o correto posicionamento dos objetos de colisão correspondentes às prateleiras e blocos. Como exemplo, a Figura 26 apresenta a cena do MoveIt! após o passo de ajuste da posição do robô em relação à prateleira, que é realizado usando a posição detectada da AR-tag presente na prateleira, e previamente a execução da sequência de *pick-and-place*, onde se observa o robô a 40 cm da prateleira. A sequência do *pick-and-place* também pode ser observada pelo Rviz e os seus passos acompanhados juntamente com o Gazebo.

A execução do *pick-and-place* se mostrou robusta para pequenas variações de posição (± 3 cm) e orientação do bloco ($\pm 10°$), porém, devido ao curto comprimento do braço e o fato de possuir somente 4 graus de liberdade, ainda há bastante limite em seu alcance.

Figura 26 – Visualização da cena de planejamento do MoveIt! pelo RViz.



Fonte: Produção do próprio autor.

5.5 SERVIÇO DE TRANSPORTE

Foi enviado ao serviço de transporte 146 diferentes requisições para mover os blocos de uma prateleira a outra e obteve-se uma taxa de sucesso de 91,7%. A Figura 27 e Figura 28 apresentam uma sequência[1] de ações executadas pelo robô para cumprir a requisição de transporte recebida.

Observou-se nos testes das requisições de transporte um comportamento satisfatório do robô, onde todos os passos definidos na Figura 12 foram executados como esperados repetidamente. Os erros observados foram em maioria devido ao componente do MoveIt!, onde o planejamento da sequência de *pick-and-place* resulta em falha pois o bloco se encontra em uma posição que o braço não pode alcançar. Isso se deve principalmente pelo fato de o braço possuir somente 4 graus de liberdade, o que limita as orientações que o braço pode assumir e impede a ação de pegar o bloco em uma posição arbitrária.

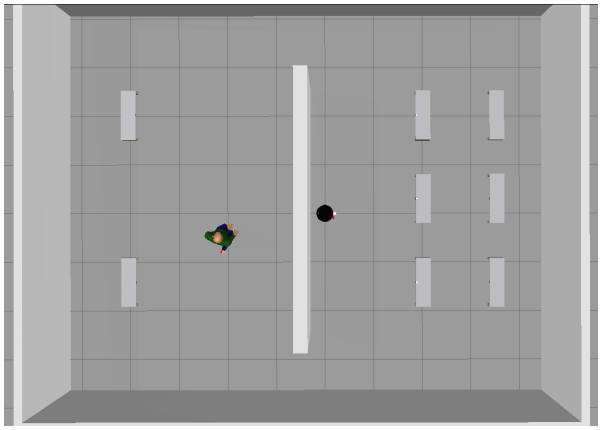
5.6 ROBÔ REAL

A execução da sequência de *pick-and-place* foi realizada com o robô parado e posicionado à frente de uma caixa com uma AR-tag representando uma prateleira e um bloco com uma AR-tag menor. Com isso, foi possível verificar a aplicação dos componentes de detecção de AR-tags, MoveIt! e construção de sequências *pick-and-place*. A Figura 29 apresenta instantes da sequência[2].

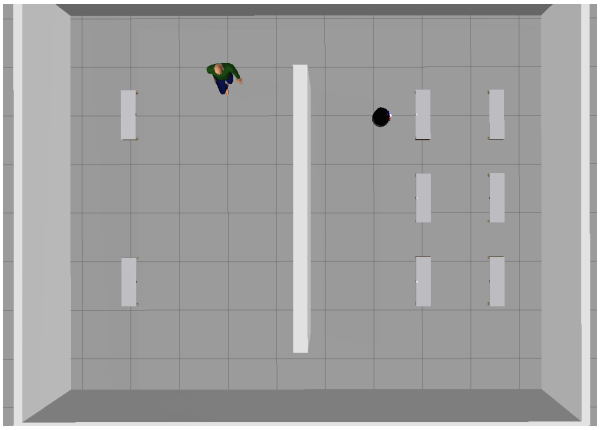
[1] A execução completa das ações foi gravada e pode ser visualizada em https://www.youtube.com
[2] A sequência completa está disponível para visualização em https://www.youtube.com

Outros experimentos não foram realizados no robô real devido ao sensor lidar estar danificado, impossibilitando a utilização das capacidades de navegação do robô.

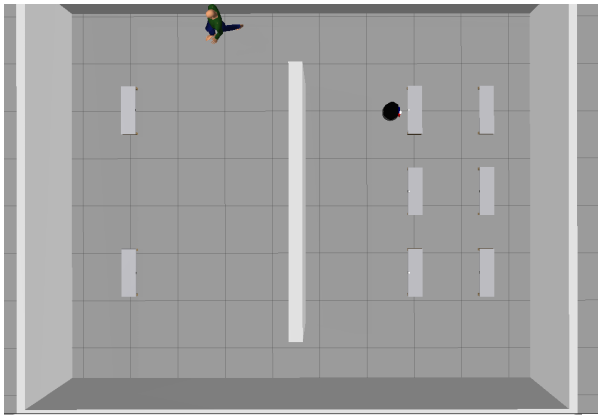
Figura 27 – Sequência de ações para transporte de um bloco.



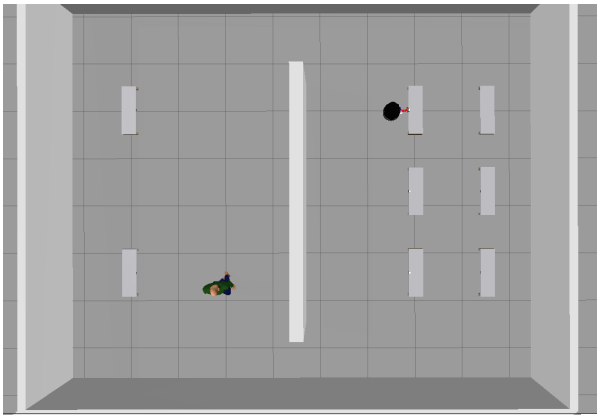
(a) Inicialização.



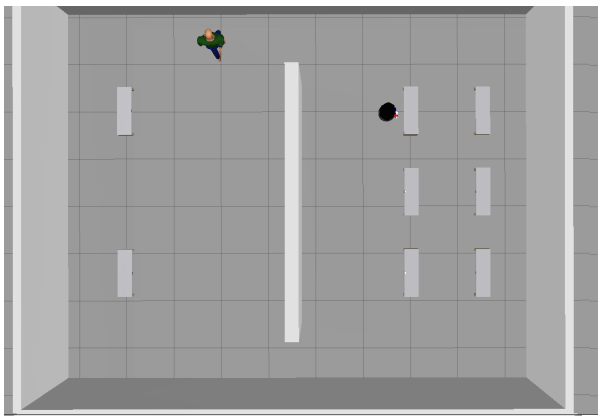
(b) Mover até a prateleira.



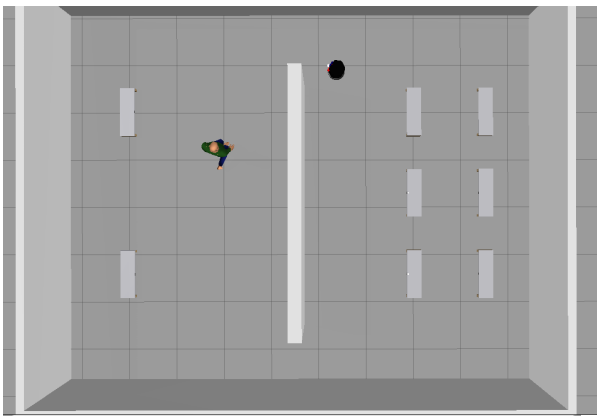
(c) Ajustar posição.



(d) Pegar bloco.



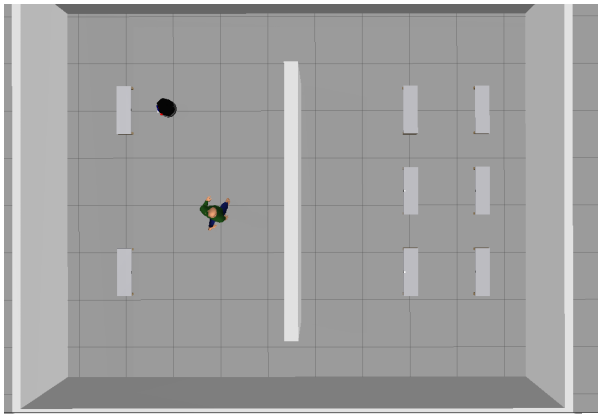
(e) Recolher o braço.



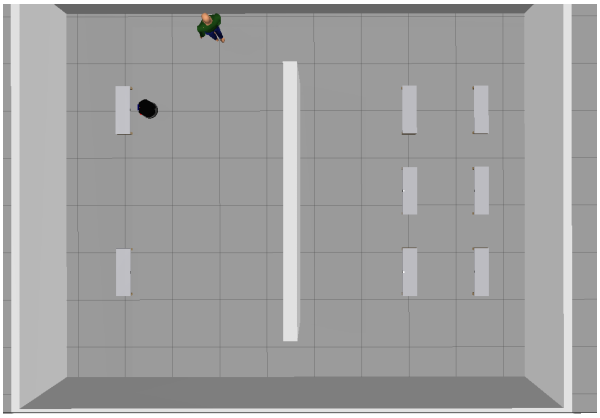
(f) Transportar bloco.

Fonte: Produção do próprio autor.

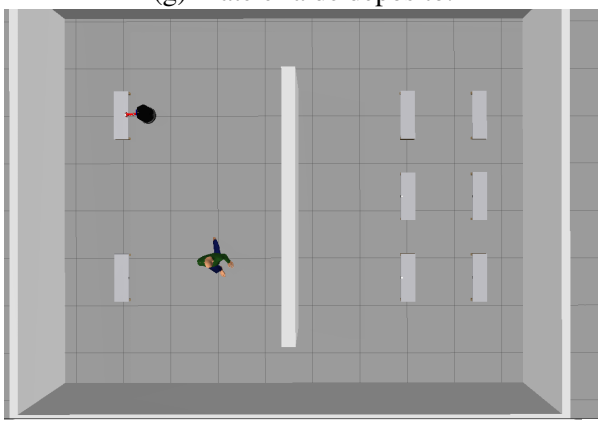
Figura 28 – Sequência de ações para transporte de um bloco (continuação).



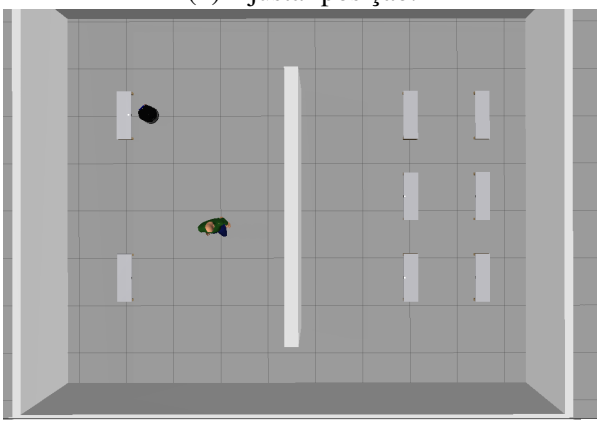
(g) Prateleira de depósito.



(h) Ajustar posição.



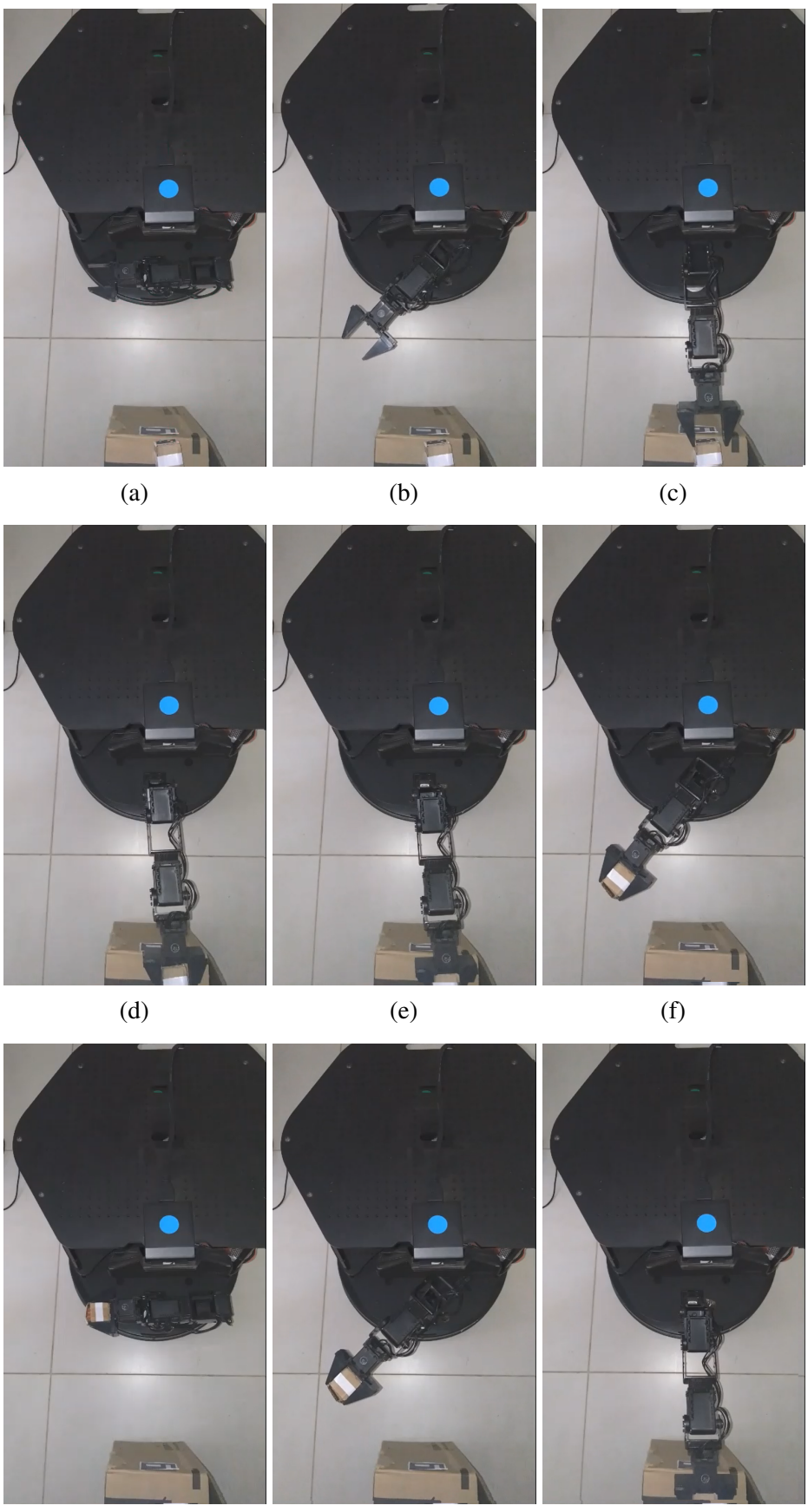
(i) Depositar bloco.



(j) Recolher braço

Fonte: Produção do próprio autor.

Figura 29 – Sequência *pick-and-place* executada em robô real.



(a) (b) (c)

(d) (e) (f)

(g) (h) (i)

Fonte: Produção do próprio autor.

6 CONCLUSÃO

Nesse trabalho, foi apresentado um sistema para implementação de um AMR para transporte de materiais, integrando capacidades de navegação em ambientes internos e manipulação de objetos. Os testes realizados em simulação demonstraram a efetividade dos componentes individuais e do sistema no cumprimento da tarefa.

A integração de componentes do robô se deu pela utilização do ROS. O pacote de navegação permitiu a integração de módulos de localização e planejamento de rotas. O *framework* MoveIt! possibilitou uma implementação facilitada dos componentes de manipulação do braço robótico e a execução da tarefa de *pick-and-place*.

O trabalho apresentou desafios durante seu desenvolvimento. Desde o início, foi necessário realizar o estudo de novos conceitos de robótica, como as etapas de mapeamento, localização e planejamento de rotas para a navegação de um robô, além de ferramentas como ROS e MoveIt!. Em especial, destaca-se a implementação do algoritmo para obtenção das posições para realização do *pick-and-place*, onde foi necessário entender em detalhes o funcionamento do processo de *pick-and-place* do MoveIt! para correto uso com um braço de 4 graus de liberdade.

Como trabalho futuro, é necessário explorar a aplicação do sistema completo no robô real e avaliar seu comportamento em diferentes ambientes. Um ponto de interesse para estudo é a implementação e avaliação de outros métodos de computação visual para a detecção das prateleira e dos blocos. Por fim, destaca-se o potencial de implementação do sistema de forma descentralizada, a partir da utilização de servidores e conexões de alta velocidade, como o 5G, permitindo diminuir a carga computacional do robô e manter sua confiabilidade.

REFERÊNCIAS

CORKE, P. **Robotics, vision and control**: fundamental algorithms in matlab. Berlin: Springer, 2011. 570 p. Disponível em: <https://doi.org/10.1007/978-3-642-20144-8>. Acesso em: 20 nov. 2021.

DIJKSTRA, E. W. et al. A note on two problems in connexion with graphs. **Numerische mathematik**, Berlin, v. 1, n. 1, p. 269–271, 1959.

FOX, D. Adapting the sample size in particle filters through kld-sampling. **The international Journal of robotics research**, SAGE Publications, Thousand Oaks, v. 22, n. 12, p. 985–1003, 2003.

FOX, D.; BURGARD, W.; THRUN, S. The dynamic window approach to collision avoidance. **IEEE Robotics Automation Magazine**, New York - EUA, v. 4, n. 1, p. 23–33, 1997. Disponível em: <https://doi.org/10.1109/100.580977>. Acesso em: 20 nov. 2021.

MORAVEC, H.; ELFES, A. High resolution maps from wide angle sonar. In: **Proceedings. 1985 IEEE International Conference on Robotics and Automation**. [s.n.], 1985. v. 2, p. 116–121. Disponível em: <https://doi.org/10.1109/ROBOT.1985.1087316>. Acesso em: 15 nov. 2021.

OPEN ROBOTICS. **Robot operating system**. 2021. Disponível em: <https://www.ros.org/about-ros/>. Acesso em: 10 nov. 2021.

OPEN SOURCE ROBOTICS FOUNDATION. **Gazebo**. 2021. Disponível em: <http://gazebosim.org/>. Acesso em: 12 nov. 2021.

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.

SICILIANO, B. et al. **Robotics**: modelling, planning and control. Londres: Springer, 2009. 632 p. Disponível em: <https://doi.org/10.1007/978-1-84628-642-1>. Acesso em: 20 nov. 2021.

SIEGWART, R.; NOURBAKSH, I. R. **Introduction to autonomous mobile robots**. Londres: MIT Press, 2004. 321 p.

SUCAN, I. A.; CHITTA, S. **IKFast kinematics solver!** 2019. Disponível em: <http://docs.ros.org/en/melodic/api/moveit_tutorials/html/doc/ikfast/ikfast_tutorial.html>. Acesso em: 17 nov. 2021.

SUCAN, I. A.; CHITTA, S. **Moveit!** 2021. Disponível em: <https://moveit.ros.org/>. Acesso em: 12 nov. 2021.

THRUN, S.; BURGARD, W.; FOX, D. **Probabilistic robotics**. Cambridge: MIT Press, 2005.

TROSSEN ROBOTICS. **Turtlebot 2i mobile ROS platform**. 2021. Disponível em: <https://www.trossenrobotics.com/interbotix-turtlebot-2i-mobile-ros-platform.aspx>. Acesso em: 10 nov. 2021.

YUJIN ROBOT. **Kobuki**. 2021. Disponível em: <http://kobuki.yujinrobot.com/about2/>. Acesso em: 10 nov. 2021.