

CENTRO UNIVERSITÁRIO FEI

**MURILO DARCE BORGES SILVA
RODRIGO SIMÕES RUY**

**IDENTIFICAÇÃO DE DIFERENÇAS DE DESEMPENHO ENTRE SISTEMAS
ROBÓTICOS SIMULADOS COM E SEM SOFTWARE CONTEINERIZADO,
UTILIZANDO SIMULADOR GAZEBO, ROS 2 E DOCKER.**

São Bernardo do Campo
2025

MURILO DARCE BORGES SILVA
RODRIGO SIMÕES RUY

**IDENTIFICAÇÃO DE DIFERENÇAS DE DESEMPENHO ENTRE SISTEMAS
ROBÓTICOS SIMULADOS COM E SEM SOFTWARE CONTEINERIZADO,
UTILIZANDO SIMULADOR GAZEBO, ROS 2 E DOCKER.**

Trabalho de Conclusão de Curso apresentado ao
Centro Universitário FEI, como parte dos requisitos
necessários para obtenção do título de Bacharel em
Ciência da Computação. Orientado pelo Prof. Dr.
Leonardo Anjoletto Ferreira.

São Bernardo do Campo
2025

MURILO DARCE BORGES SILVA
RODRIGO SIMÕES RUY

**IDENTIFICAÇÃO DE DIFERENÇAS DE DESEMPENHO ENTRE SISTEMAS
ROBÓTICOS SIMULADOS COM E SEM SOFTWARE CONTEINERIZADO,
UTILIZANDO SIMULADOR GAZEBO, ROS 2 E DOCKER.**

Trabalho de Conclusão de Curso apresentado ao
Centro Universitário FEI, como parte dos requisitos
necessários para obtenção do título de Bacharel em
Ciência da Computação.

Comissão julgadora

Prof. Dr. Leonardo Anjoletto Ferreira

Prof. Dr. Plinio Thomaz Aquino Junior

Prof. Dr. Fagner de Assis Moura Pimentel

São Bernardo do Campo

06/06/2025

RESUMO

Containerização é uma ferramenta muito útil quando se lida com projetos que precisam de diferentes dependências ou programas que podem ter conflitos entre si, que precisam de uma grande quantidade de configuração inicial, ou que precisam de portabilidade. Este projeto visa identificar e mostrar a diferença de desempenho entre robôs, que utilizam o Robot Operating System 2 (ROS 2), simulados com e sem containerização. Para verificar esta diferença, os testes serão realizados no ambiente simulado do software Gazebo Classic. Os resultados mostram que o desempenho da simulação não é impactado pelo uso do Docker, que ocorre um pequeno aumento do uso de memória, porém sem variações significativas no uso do processador, indicando que a simulação usando containers é viável durante o processo de desenvolvimento.

Palavras-chave: Robô, Robot Operating System 2 (ROS 2), Gazebo Classic, Docker, Containerização.

ABSTRACT

Containerization is a very useful tool when dealing with projects that require different dependencies or programs that may conflict with each other, that need a large amount of initial configuration, or that need portability. This project aims to identify and show the performance difference between robots, using Robot Operating System 2 (ROS 2), simulated with and without containerization. To verify this difference, tests will be performed in the simulated environment of the Gazebo Classic software. The results show that the simulation performance is not impacted by the use of Docker, that there is a small increase in memory usage, but without significant variations in processor usage, indicating that simulation using containers is viable during the development process.

Keywords: Robot, Robot Operating System 2 (ROS 2), Gazebo Classic, Docker, Containerization.

SUMÁRIO

1	INTRODUÇÃO	5
1.1	OBJETIVO	5
1.2	ESTRUTURA DO TRABALHO	5
2	CONCEITOS FUNDAMENTAIS	6
3	TRABALHOS RELACIONADOS	9
4	METODOLOGIA	11
5	EXPERIMENTOS E RESULTADOS[EXPLICAR FIGURAS]	12
5.1	RESULTADOS SETUP	12
5.2	RESULTADOS BASE	13
5.3	RESULTADOS PATRULHEIROS	14
5.4	RESULTADOS BASE + DOCKER	14
5.5	RESULTADOS PATRULHEIROS + DOCKER	15
6	DISCUSSÃO E CONCLUSÃO	20
6.1	TRABALHOS FUTUROS	20
6.2	AGRADECIMENTOS	20
	REFERÊNCIAS	21

1 INTRODUÇÃO

A conteinerização é uma ferramenta poderosa no campo de desenvolvimento e implementação, disponibilizando certa camada de isolamento entre componentes de um projeto, assegurando que estes não irão conflitar, seja por funções internas ou dependências de versões diferentes sendo utilizadas. No campo da robótica, conteinerização é vista como uma técnica para facilitar o desenvolvimento, portabilidade e consistência em projetos de robótica, mas há um problema com relação a isso, não foram feitas pesquisas detalhando sobre a mudança de desempenho que ocorre entre a integração do ROS 2 (Meta-sistema operacional utilizado em robôs) com relação ao Docker e quais efeitos um robô pode acabar sofrendo com esta integração.

1.1 OBJETIVO

O objetivo deste projeto é o de documentar e avaliar o desempenho de um robô simulado com e sem conteinerização em arenas simuladas baseadas na arena da sala K4-04 do Centro Universitário FEI sendo executado no software Gazebo Classic.

1.2 ESTRUTURA DO TRABALHO

O decorrer deste projeto está dividido da seguinte maneira: na seção 2, serão abordados os conceitos e ferramentas utilizados no projeto junto para que o leitor possa entender com clareza o tema abordado no projeto e os termos utilizados.

Na seção 3, são abordados os trabalhos que possuem alguma relação com o projeto, como métricas, ideias e etc.

A Seção 4, é abordadaa metodologia utilizada para o desenvolvimento deste projeto, demonstrando as técnicas utilizadas e os passos a serem realizados para atingir o objetivo final.

Na seção 5, são abordados os experimentos e resultados obtidos ao longo do projeto, como os experimentos foram executados, os problemas obtidos e, no final, mostrar os resultados que foram obtidos, explicando o que era esperado e o que acabou por ser obtido.

Na seção 6, são abordadas a discussão, conclusão e os trabalhos futuros. São explicados os resultados obtidos do projeto, o que foi concluído e, no final, explicar alguns passos que não foram realizados e que estão disponíveis. para algum aluno desenvolver no futuro.

2 CONCEITOS FUNDAMENTAIS

Este capítulo aborda os conceitos teóricos e as ferramentas utilizadas no desenvolvimento deste trabalho, com o intuito de descrever e relacionar estes conceitos, para haver entendimento nas etapas posteriores abordadas.

O principal conceito a ser abordado é a Containerização, uma forma de virtualização feita para ser mais rápida e flexível que a emulação, é um processo de implantação que consegue ser executado em diversos dispositivos e sistemas operacionais. Isso ocorre pelo fato de que um contêiner consegue armazenar os arquivos e bibliotecas para ser executado, permitindo a um usuário executar uma aplicação de outro sistema operacional no sistema operacional que o mesmo possua. Além disso, o contêiner permite que falhas ocorram sem afetar outros processos que não estão agrupados no mesmo. (WEN et al., 2023). A containerização será feita usando o Docker que é uma plataforma utilizada para desenvolvimento, envio e funcionamento de aplicações de maneira separada da infraestrutura por conta da conteinerização. Por conta deste fator, o usuário consegue gerir as aplicações da mesma maneira que gera sua infraestrutura. Outro fator importante é que o Docker permite que as aplicações desenvolvidas sejam testadas e executadas com menos atraso do que a maneira convencional. Contêineres são bons para fluxos de integrações e entregas de trabalho contínuas. (DOCKER, 2025)

Para realizar os testes no robô simulado, será utilizado o ROS2 (Robot Operating System 2), que é um meta-sistema operacional um sistema operacional para robôs, o mesmo realiza funções similares a outros sistemas operacionais, com exceção de controles de CPU, pois o mesmo executa processos robóticos (planejamento de movimento, navegação, manipulação de objetos, entre outros), fluxos de dados, entre outros. Possui bibliotecas e ferramentas que executam códigos em múltiplos computadores. Este conceito é utilizado pelo ROS 2, sendo o método mais utilizado nos robôs atuais, sendo uma camada acima de um sistema operacional real, que oferece abstrações e serviços para os sistemas robóticos. O ROS 2 é justamente um meta-sistema operacional de código aberto utilizado para auxiliar a desenvolver aplicações para robôs. O mesmo possui serviços que outros sistemas operacionais normalmente possuem, mas com o foco maior para a área da robótica, facilitando comunicação entre processos, funções que se comunicam com as demais e entre muitos outros. Para o desenvolvimento do projeto, será utilizado o ROS 2, que mantém o conceito modular e distribuído, mas possui melhorias e mais funcionalidades que o ROS original (Figura 1) (OPEN; SOURCE, 2018). Para conectar as aplicações, foram utilizados DDS (Data Distribution Service), que são protocolos Middleware. Os

Middlewares são uma camada de software que conecta as aplicações a um sistema operacional, permitindo uma comunicação e compartilhamento de dados mais simples entre os componentes de um sistema. Esta facilidade permite que os desenvolvedores foquem no desenvolvimento das principais funções de uma aplicação, pois a comunicação entre a aplicação e o sistema operacional está sendo feita pelo middleware. O DDS é um protocolo middleware e uma API para conexão centrada em dados, este protocolo integra os componentes de um sistema que muitas aplicações precisam, como arquitetura escalável, confiabilidade e prover conectividade de dados de baixa latência. Este protocolo foi criado pela OMG (Object Management Group). Para este projeto, foram utilizados dois tipos de DDS, o primeiro é o Fast DDS uma implementação de DDS feita em C++, possui uma biblioteca que oferece uma API e protocolo de comunicação que disponibiliza um modelo Publisher-Subscriber centrado em dados (DCPS). Este modelo visa ser eficiente e confiável para distribuir as informações para o sistema em tempo real. O segundo DDS utilizado é o Cyclone DDS, que é uma implementação de DDS com alto desempenho, permite que os desenvolvedores que o utilizam possam criar "gêmeos" digitais das entidades de seus sistemas, permitindo compartilhar estados, eventos, fluxos de dados e mensagens pela rede em tempo real, visa ser rápido, consistente e seguro.

Os testes simulados foram executados em modelos baseados na arena presente na sala K4-04 do Centro Universitário FEI, para isso, será utilizado o software Gazebo Simulator Classic que é um software usado para desenvolver simulações, possui diversos projetos de código aberto para que os interessados possam utilizar e desenvolver suas próprias simulações. Neste software estão presentes também diversos modelos, tanto como objetos como também robôs.(Figura 2). Para realizar estes testes, será utilizado o modelo virtual do robô TurtleBot3 Burger que é um robô customizável de preço acessível ao público baseado no modelo ROS para ser utilizado como um material educativo, de pesquisas, entretenimento pessoal e etc, é um robô que foi desenvolvido com o intuito de ser barato, por conta disto, o mesmo não possui uma grande funcionalidade ou qualidade, mas o mesmo compensa na relação da quantidade de aplicações que o mesmo consegue realizar.(Figura 3) (ROBOTIS, 2025).



Figura 1 – Exemplo do ROS 2, utilizando turtlesim, ROS e RQT. Fonte: Autores, 2025

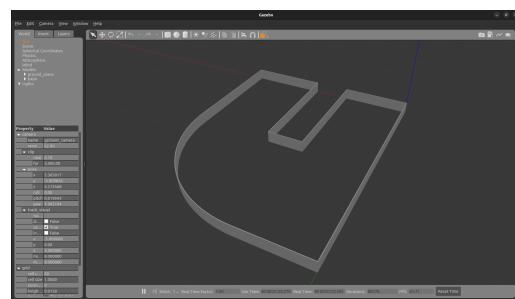


Figura 2 – Exemplo do Gazebo Simulator Classic com a arena Presente na K4-04. Fonte: Autores, 2025

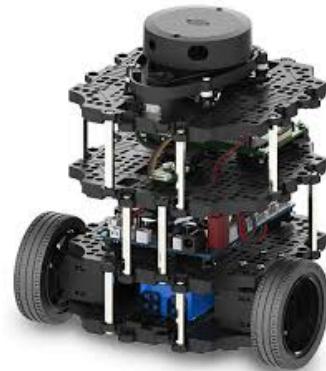


Figura 3 – TurtleBot3 Burger. Fonte: FOUNDATION, 2025

3 TRABALHOS RELACIONADOS

De acordo com (WEN et al., 2023), seu projeto auxilia com relação ao entendimento da conteinerização em sistemas embarcados e também com relação ao seu desempenho. No artigo, é detalhada a utilização de diferentes formas de conteinerização e seu efeito no desempenho em diferentes tipos de hardware. Foram realizados testes gerais que envolviam CPU, memória, rede e disco, em três ambientes diferentes: máquina virtual, contêiner e um contêiner dentro de uma máquina virtual. Com isso, foi concluído que as máquinas virtuais e os contêineres possuem um desempenho semelhante ao bare-metal (servidor físico que é de uso exclusivo para apenas um cliente, sem a camada de virtualização que fica entre o hardware e o sistema operacional), onde entre a CPU, rede e memória, sofria uma perda de no máximo 5% enquanto no disco a diferença era de até 35%. Foi observado que o Docker e a KVM (máquina virtual baseada no Kernel) foram 5 a 10% mais lentos que o bare-metal, com o Docker sendo mais lento ainda na primeira inicialização, mas levando a concluir que conteinerização e virtualização podem ser utilizados em aplicações para automóveis.

Segundo o autor (SOBIERAJ; KOTYŃSKI, 2024), seu projeto auxilia no entendimento dos conceitos de avaliação do Docker com relação a outros sistemas operacionais. Para verificar a diferença entre os sistemas operacionais, foram utilizados os recém-instalados Sistemas Operacionais que eram MacOS Ventura, Ubuntu 22.04 e Windows 10 rodando em um MacBook Pro 13. Os testes consistiam em "estressar" o Docker com relação à CPU, rede e na resiliência do mesmo. Após realizar os testes, foi observado que o sistema operacional possui uma importante influência sobre o desempenho presente no container docker, alguns possuíam benefícios com relação a outros em uma determinada categoria, o macOS se destacou com relação nos dados obtidos nas configurações utilizadas nos sistemas docker, não sofrendo grandes perdas de desempenho, o Linux se mostrou mais eficiente no geral, por conta de ser uma função que raramente utilizam de escrita no disco, se mostrando uma escolha melhor que o MacOS, o Windows acabou não se beneficiando tanto quanto os outros, a não ser pela taxa de transferência de rede entre os contêineres, assim como o linux o windows possui problemas com a velocidade em que a escrita é feita e com isso. Este artigo auxilia no entendimento com relação aos tipos de testes que podem ser realizados para analisar o desempenho do Docker, auxiliando de uma maneira que possa ser um pontapé inicial para o desenvolvimento dos testes com o Docker, e como medir o desempenho de um contêiner.

Conforme (WEN et al., 2024), o artigo aborda a arquitetura baseada em microserviços para sistemas automotivos. Cada serviço foi realizado em contêineres separados, pois os testes realizados identificaram que este método é viável e acaba por melhorar a latência existente em sistemas operacionais Linux sem contêineres, que obteve uma latência de 5 a 8% end-to-end, além de reduzir a latência máxima, o que mostra a vantagem no uso de conteinerização para os sistemas automotivos em tempo real. Foi concluído que o ROS 2, utilizado para avaliar a arquitetura de microserviços para uma aplicação real de direção autônoma, foi de extrema importância por conta de sua arquitetura distribuída que é baseada em nós e possui comunicação DDS, o que levou ao isolamento das funções do veículo e facilitou a migração para contêineres. Na conteinerização, o ROS 2 perde um pouco de seu desempenho, mas em aplicações complexas como o Autoware, a mesma acaba por melhorar, reduzindo o uso de CPU e memória. Este artigo auxilia no entendimento do uso da conteinerização com relação ao ROS 2, seus problemas e seus acertos.

4 METODOLOGIA

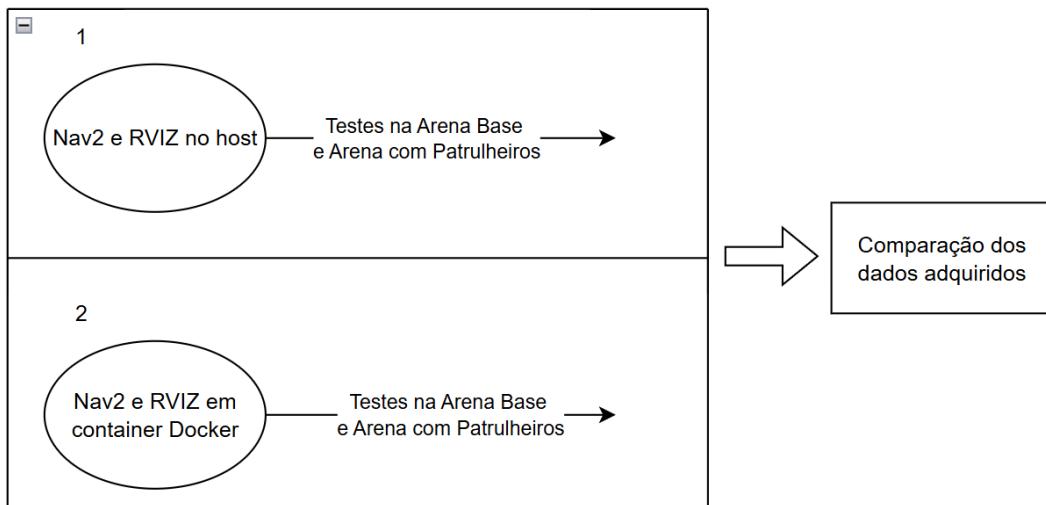


Figura 4 – Fluxograma do projeto

A metodologia (Figura 4) é dividida em 2 partes, sem Docker e com Docker, respectivamente, sendo que a integração com o Docker é na parte de navegação, isolando a stack nav2 e rviz do host. Todos os testes são executados em uma arena similar à presente na sala K404 da FEI. Foi utilizado o programa nmon para obter os dados durante os testes, obtendo o desempenho do sistema inteiro, servindo para garantir que todas as estatísticas estão sendo comparadas de forma íntegra. Foram utilizados scripts bash para tornar a execução dos testes consistente, tendo cada parte do teste também em scripts, facilitando a criação/modificação de testes personalizados. Foi criado um pacote ROS2 (tcc) para organizar scripts, mapas e arquivos launch, sendo que alguns destes são modificações de arquivos presentes em outros pacotes deste projeto. Foi utilizado um arquivo modificado de AMCL (burger.yaml) para imediatamente inicializar a localização do turtlebot, tornando os testes completamente autônomos e mais consistentes, já que a localização manual pode ser rejeitada. No primeiro teste, o robô simplesmente precisa percorrer a arena de ponta a ponta, tendo um SLAM pré-feito para o auxiliar, sem obstáculo algum. No segundo teste, o robô também precisa percorrer a arena de ponta a ponta com um SLAM pré-feito, mas dessa vez a arena está populada por outros robôs "patrulheiros", que servem para atrapalhar o trajeto do robô principal, testando sua resiliência quanto à mudança de rotas com e sem Docker. Configurações e arquivos para facilitar o setup estão presentes no repositório.

5 EXPERIMENTOS E RESULTADOS[EXPLICAR FIGURAS]

Os experimentos utilizam uma arena praticamente idêntica à arena presente na instituição, na sala K4-04 (Figura 2). O motivo de esta arena ter sido escolhida se deu por conta da comparação que poderia futuramente ser realizada entre os testes virtuais e reais.

Experimentos feitos: Experimento Setup: Testes iniciais para confirmar o comportamento de ROS2 dentro de um container Docker Experimento Base: Simulações utilizando o mapa base sem patrulheiros Experimento Patrulheiros: Simulações utilizando o mapa base com patrulheiros Experimento Base+Docker: Simulações utilizando mapa base sem patrulheiros, com a stack nav2 e rviz dentro de um container Docker Experimento Patrulheiros + Docker: Simulações utilizando o mapa base com patrulheiros, com a stack nav2 e rviz dentro de um container Docker.

5.1 RESULTADOS SETUP

Foram feitos testes Para implementar a parte simulada proposta, foi utilizado o manual (ROBOTIS, 2025) e pacotes (ROBOTIS-GIT, 2025) disponíveis pelo grupo ROBOTIS, tornando o desenvolvimento desta parte rápido. A utilização e desenvolvimento dos projetos ROS2 dentro do Docker foi facilitada pela utilização de workflows no GitHub, onde as imagens de teste foram automaticamente construídas e publicadas como pacotes no repositório, o que reduziu o tempo do processo de construir as imagens localmente, que demorava de mais 20 minutos para no máximo 5 minutos, além de também as disponibilizar para outros usarem.

Houve certas dificuldades para integrar o ROS2 dentro do contêiner Docker com o ROS2 nativo, que lidaria com a simulação Gazebo. Foi descoberto que o FastDDS (middleware utilizado por padrão pelo ROS2 Humble), não interage de forma consistente com Docker. Ele era capaz de compartilhar os tópicos entre os ambientes, mas causava falha na publicação e recebimento de mensagens, não mostrando nenhuma.

Para solucionar isso, o FastDDS foi substituído por outro middleware disponível para ROS2 Humble, CycloneDDS, o qual foi utilizado especificamente no container Docker.

```

root@BattleStation25:~/home/rodrigo/...          rodrigo@BattleStation25: ~
rodrigo@BattleStation25: $ echo $RMW_IMPLEMENTATION
rodrigo@BattleStation25: $ ros2 topic list
/parameter_events
/rosout
rodrigo@BattleStation25: $ ros2 run demo_nodes_cpp talker
[INFO] [1748231727.656969225] [talker]: Publishing: 'Hello World: 1'
[INFO] [1748231728.656933660] [talker]: Publishing: 'Hello World: 2'
[INFO] [1748231729.656980784] [talker]: Publishing: 'Hello World: 3'
[INFO] [1748231730.657285389] [talker]: Publishing: 'Hello World: 4'
[INFO] [1748231731.657301886] [talker]: Publishing: 'Hello World: 5'
[INFO] [1748231732.657292011] [talker]: Publishing: 'Hello World: 6'
[INFO] [1748231733.657285603] [talker]: Publishing: 'Hello World: 7'
[INFO] [1748231734.657286758] [talker]: Publishing: 'Hello World: 8'
[INFO] [1748231735.657267963] [talker]: Publishing: 'Hello World: 9'
[INFO] [1748231736.657290746] [talker]: Publishing: 'Hello World: 10'
[INFO] [1748231737.656933981] [talker]: Publishing: 'Hello World: 11'
[INFO] [1748231738.656962986] [talker]: Publishing: 'Hello World: 12'
[INFO] [1748231739.656961720] [talker]: Publishing: 'Hello World: 13'
[INFO] [1748231740.657492615] [talker]: Publishing: 'Hello World: 14'
[INFO] [1748231741.657030920] [talker]: Publishing: 'Hello World: 15'
[INFO] [1748231742.656963833] [talker]: Publishing: 'Hello World: 16'
[INFO] [1748231743.657117598] [talker]: Publishing: 'Hello World: 17'
[INFO] [1748231744.657020480] [talker]: Publishing: 'Hello World: 18'

root@BattleStation25:~/dockerteste           root@BattleStation25:~/dockerteste
rodrigo@BattleStation25: $ sudo su
[sudo] password for rodrigo:
root@BattleStation25:/home/rodrigo# docker exec -it tcc-ros2-1 bash
root@BattleStation25:/dockerteste# ls
src
root@BattleStation25:/dockerteste# echo $RMW_IMPLEMENTATION
root@BattleStation25:/dockerteste# ros2 topic list
/parameter_events
/rosout
root@BattleStation25:/dockerteste# ros2 topic list
/chatter
/parameter_events
/rosout
root@BattleStation25:/dockerteste# ros2 topic echo /chatter

```

Figura 5 – Falha utilizando RMW padrão (FastDDS)

```

root@BattleStation25:~/home/rodrigo/...          rodrigo@BattleStation25: ~
rodrigo@BattleStation25: $ echo $RMW_IMPLEMENTATION
rodrigo@BattleStation25: $ ros2 topic list
/chatter
/parameter_events
/rosout
root@BattleStation25:/dockerteste# ros2 topic echo /chatter
^Croot@BattleStation25:/dockerteste# ros2 topic list
/parameter_events
/rosout
root@BattleStation25:/dockerteste# export RMW_IMPLEMENTATION=rmw_cyclonedds_cpp
root@BattleStation25:/dockerteste# ros2 topic list
/chatter
/parameter_events
/rosout
root@BattleStation25:/dockerteste# ros2 topic echo /chatter
data: 'Hello World: 18'
...
data: 'Hello World: 19'
...
data: 'Hello World: 20'
...
data: 'Hello World: 21'
...
data: 'Hello World: 22'
...
data: 'Hello World: 23'
...
data: 'Hello World: 24'

root@BattleStation25: $ ros2 run demo_nodes_cpp talker
[INFO] [1748231966.319745640] [talker]: Publishing: 'Hello World: 1'
[INFO] [1748231967.319316139] [talker]: Publishing: 'Hello World: 2'
[INFO] [1748231968.319311374] [talker]: Publishing: 'Hello World: 3'
[INFO] [1748231969.319474912] [talker]: Publishing: 'Hello World: 4'
[INFO] [1748231970.3197159519] [talker]: Publishing: 'Hello World: 5'
[INFO] [1748231971.319770725] [talker]: Publishing: 'Hello World: 6'
[INFO] [1748231972.3198134291] [talker]: Publishing: 'Hello World: 7'
[INFO] [1748231973.3198134291] [talker]: Publishing: 'Hello World: 8'

```

Figura 6 – Sucesso utilizando RMW CycloneDDS

5.2 RESULTADOS BASE

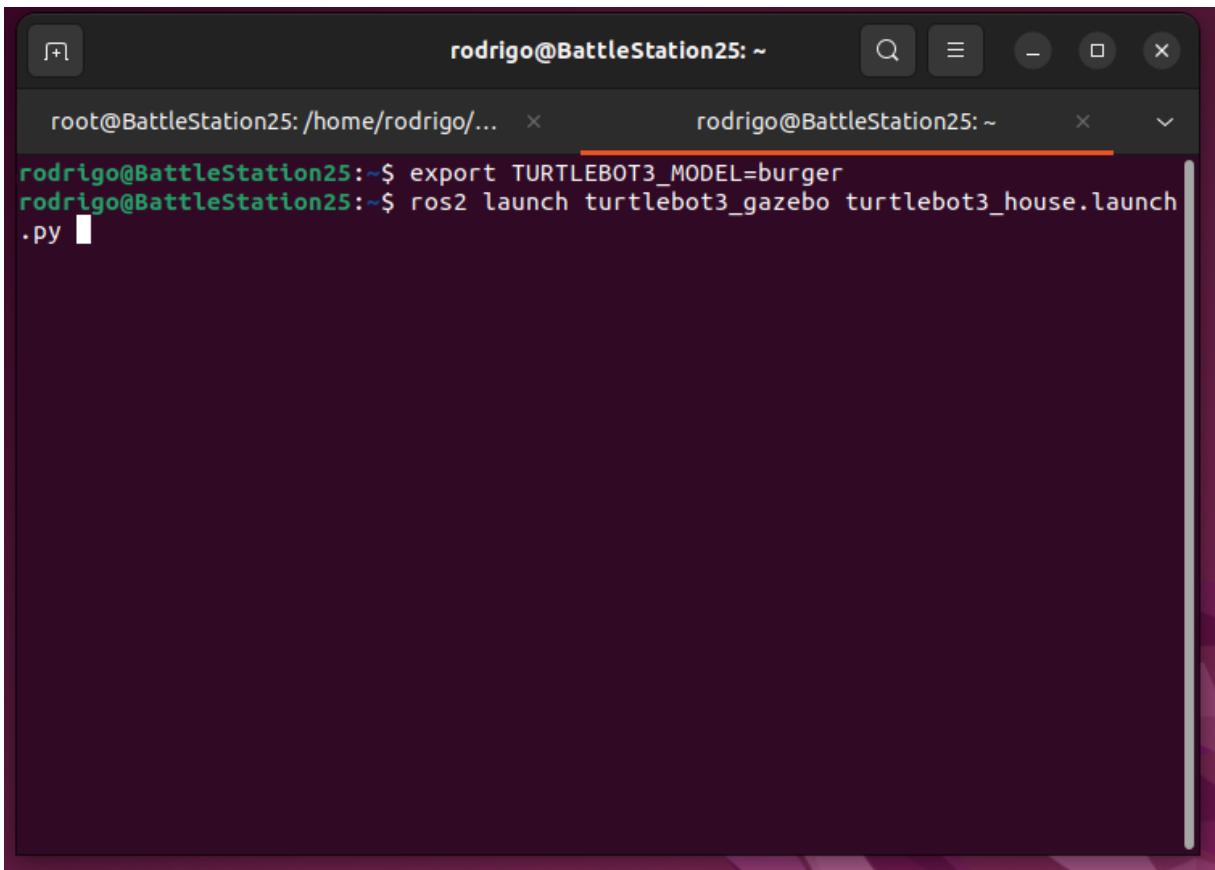
-A EDITAR- Foram feitos testes para adquirir os dados de desempenho de simulações sem Docker, utilizando a arena base sem patrulheiros. 1- Rodar o comando no diretório do repositório (INSERIR COMANDO) 2 - Imagens durante a execução do teste 3- Mostrar dados via nmonvisualizer

```

services:
  ros2:
    image: ghcr.io/joca2511/tcc_docker:main
    network_mode: host
    privileged: true
    environment:
      - ROS_DOMAIN_ID=10
      - TURTLEBOT3_MODEL=burger
      - RMW_IMPLEMENTATION=rmw_cyclonedds_cpp
    stdin_open: true
    tty: true

```

Figura 7 – Compose proposto para testes

A screenshot of a terminal window titled "rodrigo@BattleStation25: ~". The window contains two tabs: "root@BattleStation25: /home/rodrigo/..." and "rodrigo@BattleStation25: ~". The bottom tab is active and shows the command: "rodrigo@BattleStation25:~\$ export TURTLEBOT3_MODEL=burger" followed by "rodrigo@BattleStation25:~\$ ros2 launch turtlebot3_gazebo turtlebot3_house.launch.py".

```
rodrigo@BattleStation25:~$ export TURTLEBOT3_MODEL=burger
rodrigo@BattleStation25:~$ ros2 launch turtlebot3_gazebo turtlebot3_house.launch.py
```

Figura 8 – Simulação Gazebo é lançada pelo host

5.3 RESULTADOS PATRULHEIROS

Foram feitos testes para adquirir os dados de desempenho de simulações sem Docker, utilizando a arena base com patrulheiros. 1- Rodar o comando no diretório do repositório (INSERIR COMANDO) 2 - Imagens durante a execução do teste. 3- Mostrar dados via nmonvisualizer

5.4 RESULTADOS BASE + DOCKER

Foram feitos testes para adquirir os dados de desempenho de simulações com Docker, utilizando a arena base sem patrulheiros. 1- Rodar o comando no diretório do repositório (INSERIR COMANDO) 2 - Imagens durante a execução do teste. 3- Mostrar dados via nmonvisualizer

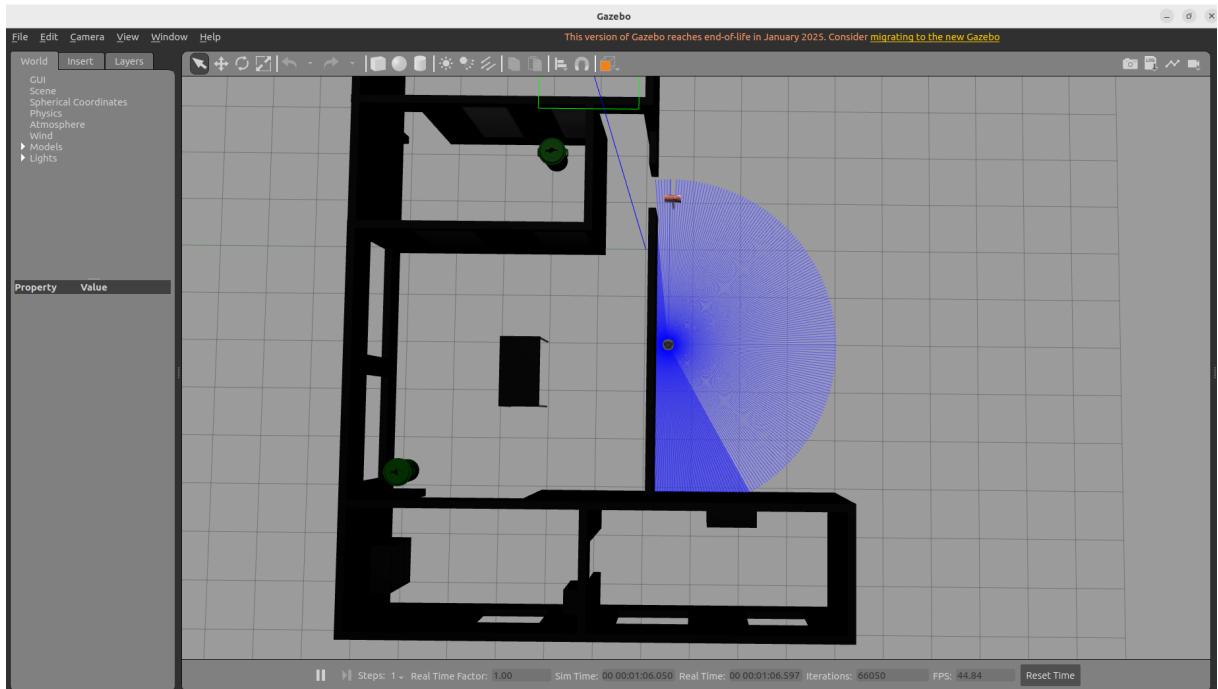


Figura 9 – Aparência inicial da simulação Gazebo

5.5 RESULTADOS PATRULHEIROS + DOCKER

Foram feitos testes para adquirir os dados de desempenho de simulações com Docker, utilizando a arena base com patrulheiros. 1- Rodar o comando no diretório do repositório (INSERIR COMANDO) 2 - Imagens durante a execução do teste. 3- Mostrar dados via nmonvisualizer.

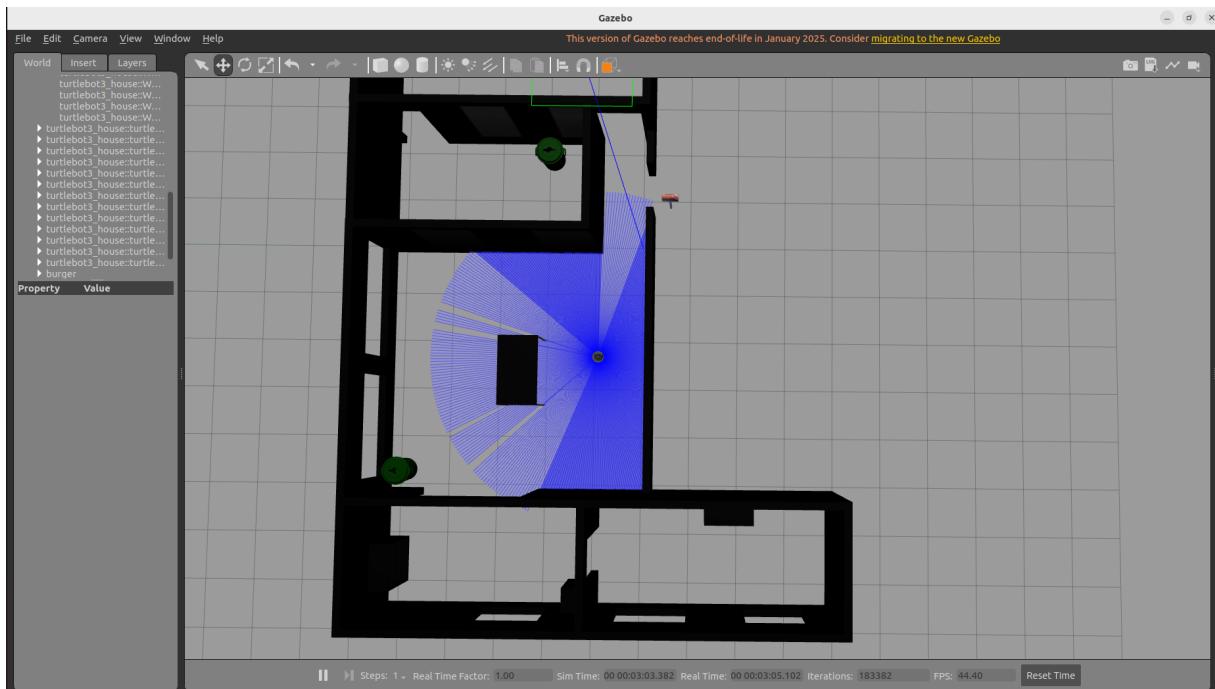


Figura 10 – Turtlebot3 Burger é movido para dentro da casa

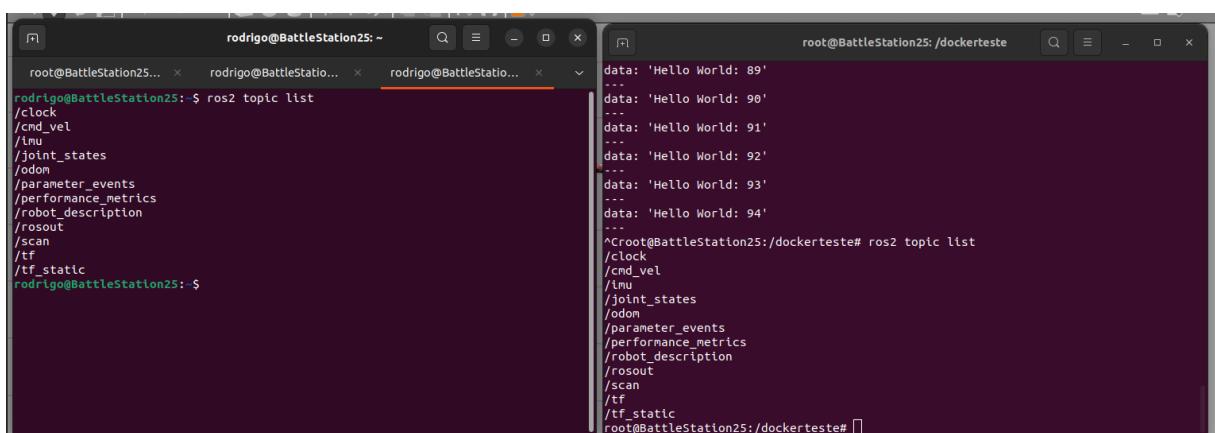
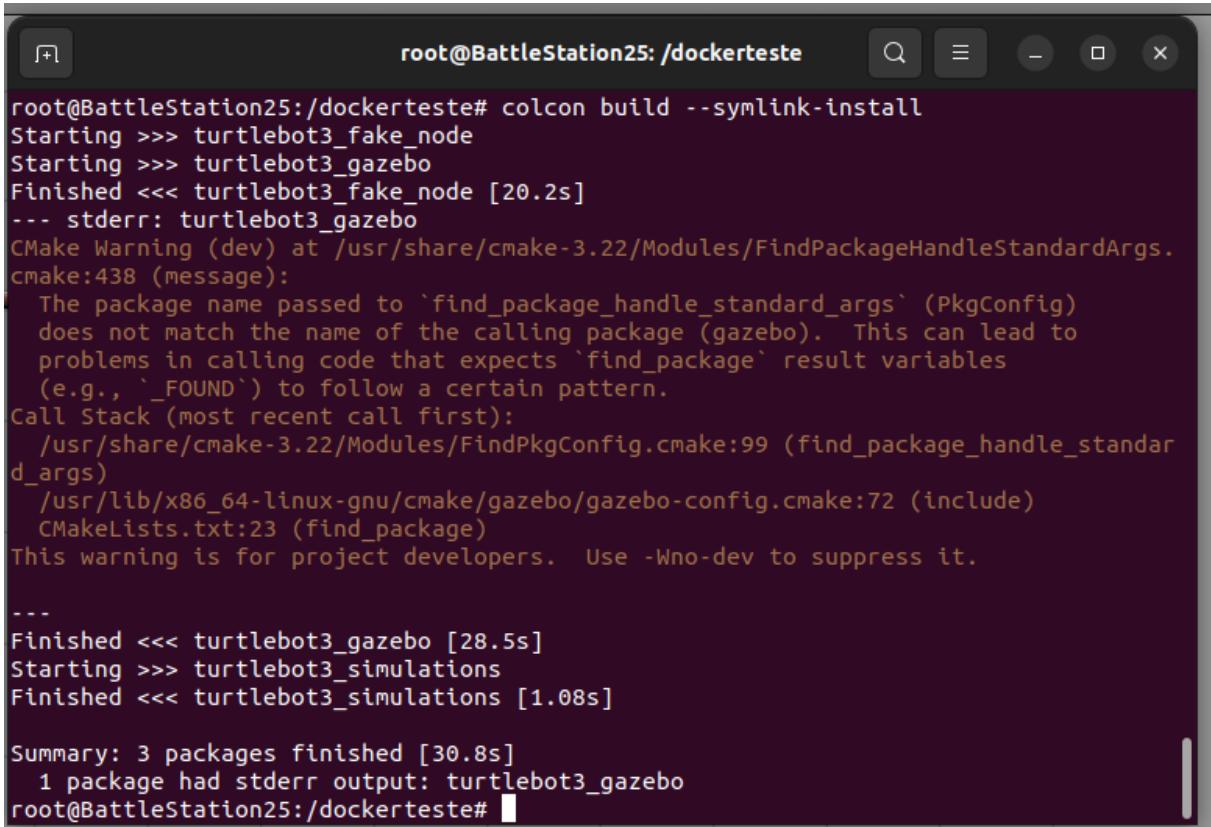


Figura 11 – ROS2 Humble dentro do Docker adquire os novos tópicos



```

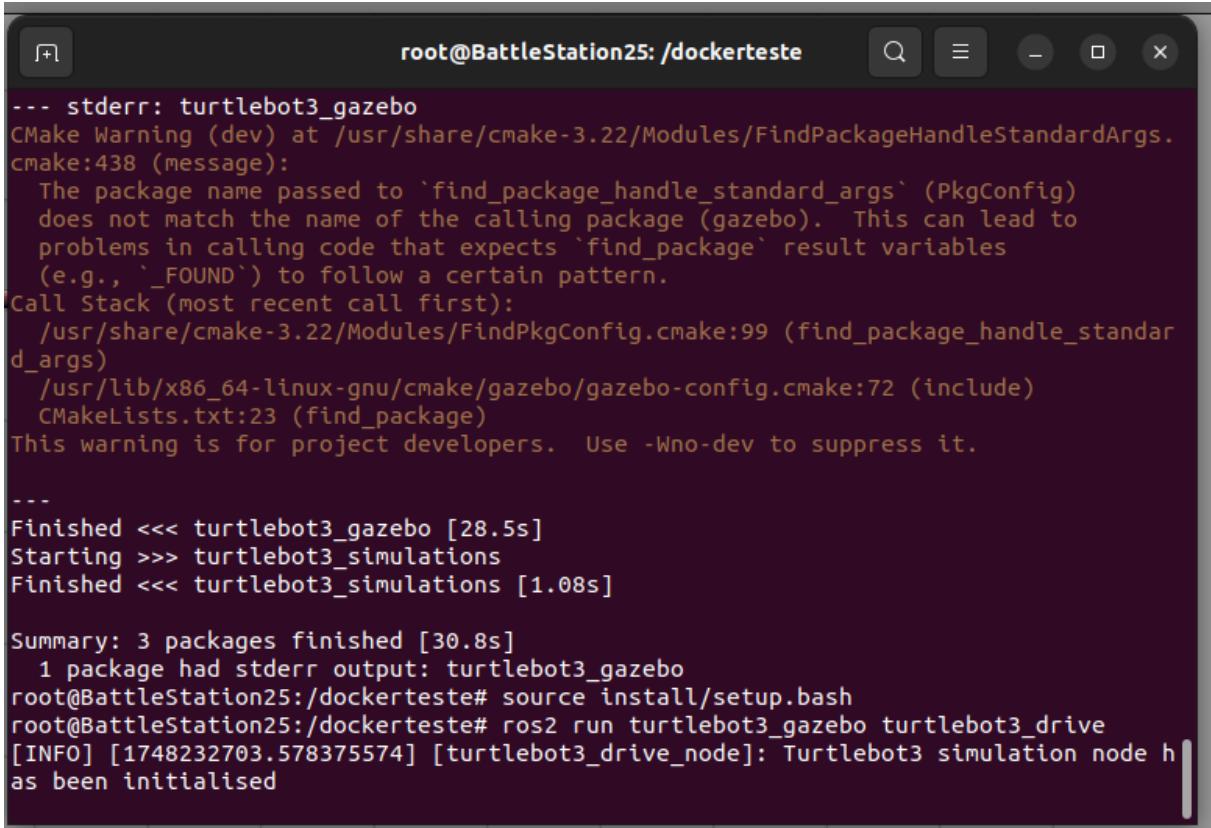
root@BattleStation25:/dockerteste# colcon build --symlink-install
Starting >>> turtlebot3_fake_node
Starting >>> turtlebot3_gazebo
Finished <<< turtlebot3_fake_node [20.2s]
--- stderr: turtlebot3_gazebo
CMake Warning (dev) at /usr/share/cmake-3.22/Modules/FindPackageHandleStandardArgs.cmake:438 (message):
  The package name passed to `find_package_handle_standard_args` (PkgConfig)
  does not match the name of the calling package (gazebo). This can lead to
  problems in calling code that expects `find_package` result variables
  (e.g., `_FOUND`) to follow a certain pattern.
Call Stack (most recent call first):
  /usr/share/cmake-3.22/Modules/FindPkgConfig.cmake:99 (find_package_handle_standard_args)
  /usr/lib/x86_64-linux-gnu/cmake/gazebo/gazebo-config.cmake:72 (include)
  CMakeLists.txt:23 (find_package)
This warning is for project developers. Use -Wno-dev to suppress it.

---
Finished <<< turtlebot3_gazebo [28.5s]
Starting >>> turtlebot3_simulations
Finished <<< turtlebot3_simulations [1.08s]

Summary: 3 packages finished [30.8s]
  1 package had stderr output: turtlebot3_gazebo
root@BattleStation25:/dockerteste#

```

Figura 12 – ROS2 Humble dentro do Docker builda projetos



```

--- stderr: turtlebot3_gazebo
CMake Warning (dev) at /usr/share/cmake-3.22/Modules/FindPackageHandleStandardArgs.cmake:438 (message):
  The package name passed to `find_package_handle_standard_args` (PkgConfig)
  does not match the name of the calling package (gazebo). This can lead to
  problems in calling code that expects `find_package` result variables
  (e.g., `_FOUND`) to follow a certain pattern.
Call Stack (most recent call first):
  /usr/share/cmake-3.22/Modules/FindPkgConfig.cmake:99 (find_package_handle_standard_args)
  /usr/lib/x86_64-linux-gnu/cmake/gazebo/gazebo-config.cmake:72 (include)
  CMakeLists.txt:23 (find_package)
This warning is for project developers. Use -Wno-dev to suppress it.

---
Finished <<< turtlebot3_gazebo [28.5s]
Starting >>> turtlebot3_simulations
Finished <<< turtlebot3_simulations [1.08s]

Summary: 3 packages finished [30.8s]
  1 package had stderr output: turtlebot3_gazebo
root@BattleStation25:/dockerteste# source install/setup.bash
root@BattleStation25:/dockerteste# ros2 run turtlebot3_gazebo turtlebot3_drive
[INFO] [1748232703.578375574] [turtlebot3_drive_node]: Turtlebot3 simulation node has been initialised

```

Figura 13 – ROS2 Humble dentro do Docker roda projetos para movimentação do robô dentro da simulação Gazebo presente no host

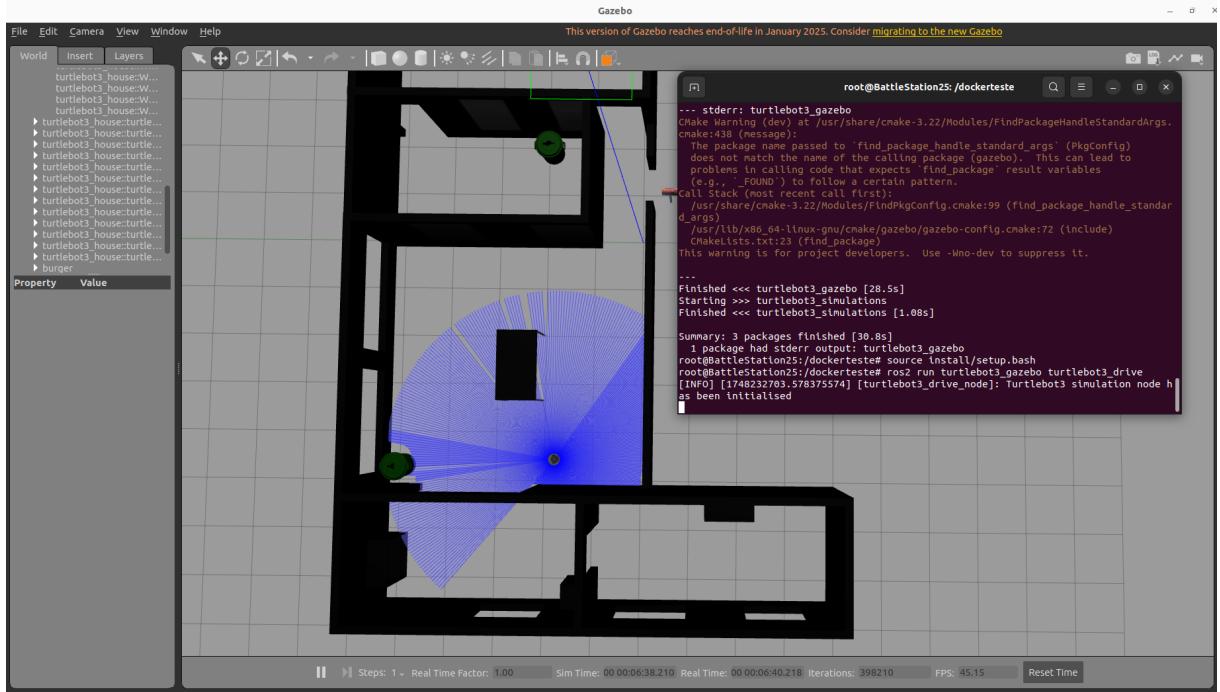


Figura 14 – Turtlebot3 Burger é movimentado pelo ROS2 Humble presente dentro do conteiner Docker

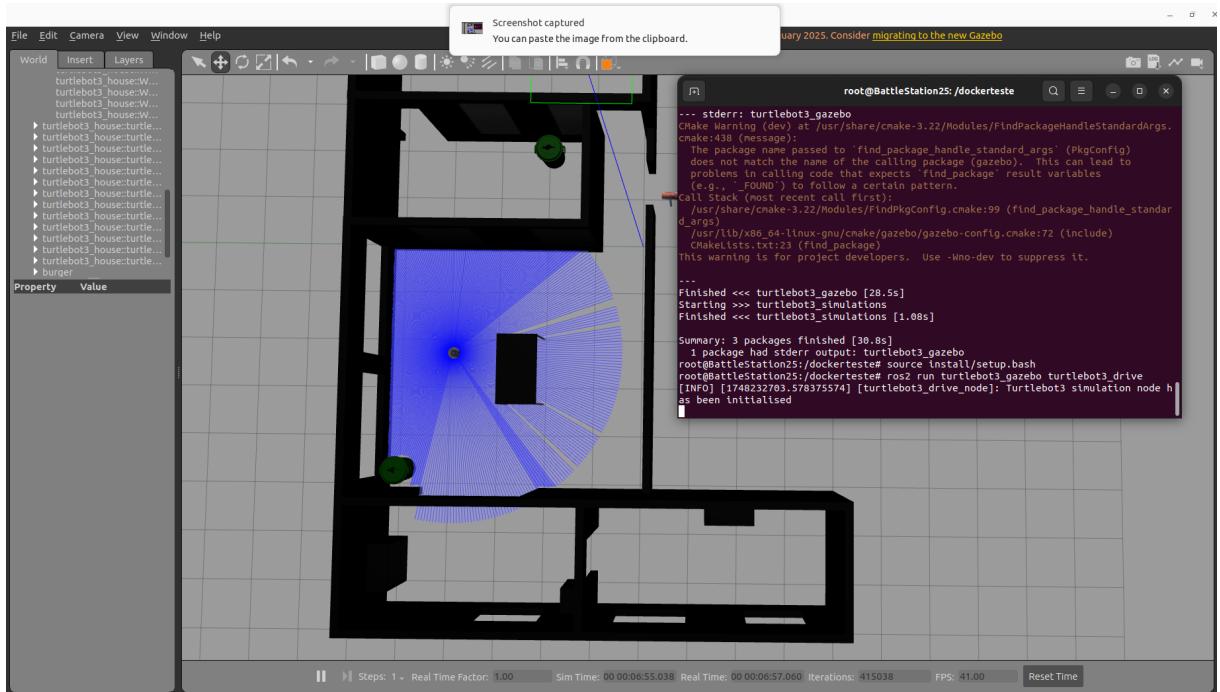


Figura 15 – Turtlebot3 Burger é movimentado pelo ROS2 Humble presente dentro do conteiner Docker

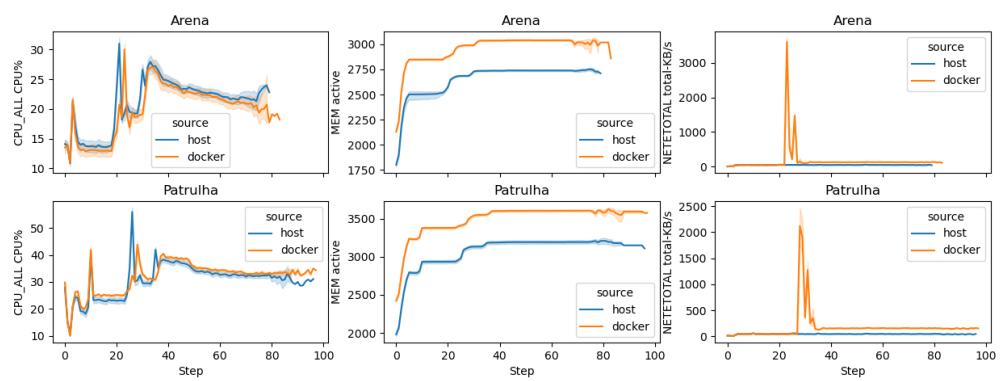


Figura 16 – Resultados dos testes representados em gráficos.

6 DISCUSSÃO E CONCLUSÃO

Questão a ser respondida pela Conclusão/Discussões Finais: O que tudo isso significa?

Esta seção mostra como o trabalho avançou o estado atual do conhecimento dentro da área de pesquisa.

Você deve fornecer uma justificativa clara para o seu trabalho na conclusão. Além disso, você pode sugerir experimentos futuros e apontar os que estão em andamento.

6.1 TRABALHOS FUTUROS

Devido a problemas com relação ao uso da sala K4-04 e seus equipamentos e à falta de tempo, os testes físicos não foram realizados, isso acabou por alterar o tema do projeto que envolvia a comparação do desempenho do robô simulado com e sem contêiner, com o robô físico com e sem contêiner. Foi decidido que os testes não seriam realizados mas que estariam disponíveis para serem usados como um trabalho futuro para algum próximo aluno que possa se interessar pelo tema e decidir continuar o desenvolvimento dos testes.

Como trabalhos futuros, seria a obtenção e comparação de dados de testes reais e das simulações feitas com as ferramentas criadas por este projeto, facilitando a obtenção de dados da parte simulada.

6.2 AGRADECIMENTOS

Agradecemos ao professor Fagner Pimentel pelo fornecimento do arquivo "K404.sdf" que foi de suma importância para o desenvolvimento do projeto.

REFERÊNCIAS

- DOCKER, I. **Docker**. 2025. Accessed: 2025-05-25. Disponível em: <<https://docs.docker.com/get-started/docker-overview/>>.
- OPEN; SOURCE, R. F. I. **ROS**. 2018. Accessed: 2025-05-24. Disponível em: <<https://wiki.ros.org/ROS/Introduction>>.
- ROBOTIS. **TurtleBot3 e-Manual: Overview**. 2025. Accessed: 2025-05-24. Disponível em: <<https://emanual.robotis.com/docs/en/platform/turtlebot3/overview/>>.
- ROBOTIS-GIT. **TurtleBot3 GitHub Repository**. 2025. Accessed: 2025-05-24. Disponível em: <<https://github.com/ROBOTIS-GIT/turtlebot3>>.
- SOBIERAJ, M.; KOTYŃSKI, D. Docker performance evaluation across operating systems. **Applied Sciences**, v. 14, n. 15, 2024. ISSN 2076-3417. Disponível em: <<https://www.mdpi.com/2076-3417/14/15/6672>>.
- WEN, L. et al. **A Containerized Microservice Architecture for a ROS 2 Autonomous Driving Software: An End-to-End Latency Evaluation**. 2024. Disponível em: <<https://arxiv.org/abs/2404.12683>>.
- _____. Bare-metal vs. hypervisors and containers: Performance evaluation of virtualization technologies for software-defined vehicles. In: . [s.n.], 2023. Disponível em: <https://www.researchgate.net/publication/372496789_Bare-Metal_vs_Hypervisors_and_Containers_Performance_Evaluation_of_Virtualization_Technologies_for_Software-Defined_Vehicles>.