ChatGPT

Simulação do Unitree G1 no Gazebo com ROS2: Repositórios e Abordagem

Diferenças entre *unitree_ros* e *unitree_ros2*

- **unitree_ros (ROS1)** – É o pacote de **simulação ROS** oficial da Unitree. Ele fornece descrições URDF/Xacro completas de **todos os robôs Unitree**, incluindo os modelos quadrúpedes (A1, AlienGo, Go1, etc.) **e** o humanoide **G1 com mãos dexterous (Dex5)** [1]. Esse repositório contém os modelos (malhas, URDFs) com informações de massa, inércia, limites de articulações etc., além de pacotes para integrar ao Gazebo (por ex. `unitree_gazebo`, controladores ROS1) [2]. Em resumo, *unitree_ros* foi feito para ROS1 (Melodic/Noetic) e permite carregar o robô no Gazebo para controle **nível baixo** (torque/posição/velocidade nas juntas) [2]. *Importante:* por limitações, a simulação em Gazebo **não inclui controle de alto nível (locomoção autônoma)** – ou seja, o robô não "anda" sozinho usando o controlador nativo, apenas responde a comandos de junta que você programar [2]. O pacote também inclui ferramentas (*unitree_ros_to_real*) para controlar robôs reais via ROS1.

- **unitree_ros2 (ROS2)** – É o pacote voltado para **controle dos robôs Unitree em ROS2**, utilizando a nova SDK2 da Unitree. Diferentemente do anterior, ele **não fornece simulação ou URDF**; em vez disso, implementa a interface de comunicação com os robôs reais no ambiente ROS2 [3]. O *unitree_ros2* usa o mecanismo DDS (CycloneDDS) para se conectar aos robôs (Go2, B2, H1 **e também G1**) de forma nativa no ROS2 [4]. Em outras palavras, as mensagens ROS2 podem se comunicar diretamente com o hardware do robô Unitree sem necessidade de "bridge" ROS1, pois o *unitree_ros2* aproveita que o robô internamente já usa DDS [4]. As interfaces de mensagem/tópicos seguem o padrão da SDK2 da Unitree, permitindo enviar comandos de movimento, ler estados, etc., no ROS2. **Resumindo:** *unitree_ros2* serve para desenvolver aplicações ROS2 para controlar os robôs físicos (especialmente as novas gerações Go2/B2/H1/G1) [3], mas **não inclui cenário de Gazebo ou modelos URDF** para simulação.

(Observação: A Unitree também disponibiliza o pacote unitree_mujoco – um simulador baseado em MuJoCo com suporte sim2real e terreno variável [5]. No entanto, ele usa modelos no formato do MuJoCo e exigiria conversão para URDF se quisermos usar Gazebo. Como você já notou, o modelo URDF mais completo do G1 (29 DOF com mãos Dex5) está no unitree_ros, então focaremos no Gazebo.)*

Qual repositório usar para simulação no Gazebo (ROS2)?

Para **simular o robô G1 (com mão Inspire de 5 dedos)** no Gazebo utilizando ROS2, a melhor abordagem é **combinar os dois repositórios**: usar o *unitree_ros* como fonte do modelo/descrição do robô, e o

unitree_ros2 para controlar o robô real (e possivelmente integrar comandos ROS2 na simulação). Em particular:

- **Modelo URDF e configuração de Gazebo:** Utilize o conteúdo do repositório *unitree_ros* para obter a descrição URDF/Xacro completa do G1 com as Dex5. Esse pacote já "**suporta G1**" na sua última versão [1] e inclui todos os detalhes do robô. Você poderá aproveitar esse URDF no Gazebo, já que o Gazebo usa URDF/SDF nativamente para representar robôs. Em outras palavras, o *unitree_ros* fornece tudo que é necessário para colocar o G1 no Gazebo (malhas 3D, arquivos URDF, e plugins de controle) [1].

- **Adaptando para ROS2:** Embora o *unitree_ros* seja um pacote ROS1, você **pode usá-lo em um ambiente ROS2** com algumas adaptações. Como não existe (até o momento) um pacote oficial de simulação para G1 em ROS2, será necessário portar ou traduzir partes do *unitree_ros* para ROS2:

- Crie um pacote ROS2 de descrição (**.urdf/.xacro**) do G1 reaproveitando os arquivos do *unitree_ros*. Isso pode ser tão simples quanto copiar os arquivos URDF e garantir que as paths das meshes estejam corretas no seu workspace ROS2.
- Substitua plugins de controle do Gazebo ROS1 por equivalentes do ROS2. Por exemplo, o *unitree_ros* usa o plugin `gazebo_ros_control` (ROS1) para interfacear as juntas; no ROS2, você usará `gazebo_ros2_control` e configurar controladores via `ros2_control` (provavelmente controladores de posição/torque para as juntas do G1). Os controladores PID e parâmetros de física podem precisar ajustes no ROS2, já que houve diferenças (ROS2 control não aplica PIDs da mesma forma automática – pode ser preciso definir *gains* manualmente em um arquivo YAML).
- Implemente launches ROS2 para spawn do modelo no Gazebo. Por exemplo, usando `ros2 launch gazebo_ros spawn_entity_demo.launch.py` ou similares com seu URDF, ou criando um launch que inicia `gazebo` e usa um serviço de spawn. (Há projetos comunitários que seguiram esse caminho para quadrúpedes Unitree no ROS2 [6], provando ser viável carregar o URDF do Unitree no Gazebo 11 junto com *ros2_control*).

- **Por que não usar apenas unitree_ros2?** O *unitree_ros2* sozinho **não resolve a simulação**, pois ele não traz o modelo do robô ou integração com Gazebo. Ele é focado em comunicação com o hardware. Portanto, para *simular* o G1 em Gazebo, você realmente dependerá do URDF e da configuração do *unitree_ros* [1]. Em resumo: **use o unitree_ros (ROS1)** como base do *modelo Gazebo*, mas rode-o no contexto ROS2 (após portar/ajustar como mencionado), e **use o unitree_ros2** para conectar/controlar o robô real via ROS2. Dessa forma, você terá consistência entre o que é simulado e o que é executado no robô físico.

- **Controle na simulação vs. controle no robô real:** Vale destacar que mesmo após portar o modelo para ROS2/Gazebo, a forma de controle diferirá: na simulação, você controlará as juntas do G1 diretamente (ex., publicando comandos de posição, velocidade ou torque nos controladores ROS2 do Gazebo). Já no robô real, via *unitree_ros2*, você enviará comandos de alto nível que o próprio controlador interno do G1 executa (por exemplo, comandos de andar, movimentar braços, etc., conforme a API DDS da Unitree). A simulação **não inclui** o controlador de caminhada do G1 [2], então qualquer locomoção ou equilíbrio terá que ser simulada manualmente (ou seja, você teria que implementar alguma lógica de equilíbrio ou fixar o robô no chão para testes estáticos). Em aplicações práticas, muitos desenvolvedores fazem na simulação apenas testes de percepção,

planejamento e movimentos simples (ex.: movimentar os braços/mãos ou deslocamentos pequenos), e testam a locomoção completa somente no hardware real ou em simuladores de física especializados. Tenha essa limitação em mente: **Gazebo servirá para testar controle de baixo nível e algoritmos de visão/planejamento**, mas o desempenho em caminhar do humanoide real pode não ser totalmente replicado ali [2] .

Configuração do Gazebo e inserção do ambiente 3D (.obj)

Uma vez com o modelo URDF do G1 pronto no ROS2, será preciso escolher e configurar o Gazebo para incluir seu **ambiente de escritório (.obj)**:

- **Versão do Gazebo:** No ROS2 atual (Ubuntu 24.04), você pode optar pelo **Gazebo "clássico"** (ex.: Gazebo 11, que era usado no ROS1) ou pelo **Gazebo Ignition (Fortress, Garden, etc.)**, que é a nova geração. Ambos permitem inserir modelos .OBJ, mas a integração difere:
- *Gazebo clássico* (usado com ROS2 via pacotes `gazebo_ros_pkgs` compatíveis com ROS2 Humble/Iron): Se você portar os plugins do *unitree_ros* para ros2_control, pode usar o Gazebo clássico normalmente. O *unitree_ros* original era testado com Gazebo 8/9 [7] ; porém, Gazebo 11 é compatível e funciona bem em Ubuntu 22.04/ROS2 Humble, e deve rodar também em 24.04 (com possivelmente Gazebo 11 ou 13 via apt). Esse caminho é mais familiar se você já usou Gazebo + ROS1.

- *Ignition Gazebo (Gazebo Fortress/Garden)*: É a versão mais moderna, integrada nativamente com ROS2 através dos pacotes *ros_ign* (ou *ros_gz*). Ela traz melhorias em física e rendering, e pode ser vantajosa para cenários complexos. No entanto, usar Ignition exigiria converter o URDF para SDF ou usar a ferramenta de spawn própria, e adaptar os plugins (por exemplo, em vez de `gazebo_ros2_control`, usar o *ignition-ros2 control bridge* ou similares). É um caminho possível, mas um pouco mais complexo se o modelo tem muitos plugins ROS clássicos.

- **Inserindo o modelo do escritório (.obj):** Independente da versão do Gazebo:

- Você pode converter seu arquivo `.obj` em um modelo estático do Gazebo. Isso envolve criar um diretório de modelo (com um `model.sdf` ou `model.config`) referenciando o mesh .obj. Por exemplo, um SDF simples com um `<visual>` e `<collision>` usando a malha .obj do escritório como objeto estático no mundo. Então inclua esse modelo no arquivo `.world` do Gazebo ou spawn via ROS2.
- Outra abordagem: usar a funcionalidade de **Building Editor** (no Gazebo clássico) ou importar a malha diretamente pelo `.world`. No *unitree_ros*, por exemplo, há um mundo de exemplo com escadas que inclui um modelo estático [8] ; você faria similar para o seu escritório, apontando para o caminho do .obj (assegurando que o Gazebo consiga encontrar o arquivo).
- Certifique-se de que o `.obj` tenha dimensões e posição corretas. Muitas vezes é necessário escalar ou ajustar a origem da malha para que o robô apareça no lugar desejado dentro do escritório.
- **Qual Gazebo é "melhor" para .OBJ?** Não há uma diferença significativa no suporte a meshes: tanto o Gazebo clássico quanto o Ignition suportam malhas .obj ou .dae. A escolha deve se basear mais na compatibilidade com ROS2 e sua preferência. Se você já está adaptando todo o simulador do G1, usar o Gazebo clássico com ros2_control pode ser mais direto inicialmente. Por outro lado, se você preferir usufruir das melhorias do Ignition e está disposto a adaptar a integração, essa versão

também funcionará. Em ambos os casos, você poderá carregar o modelo do escritório sem grandes problemas (basta referenciá-lo corretamente no SDF/world).

Resumo da Solução Adotada

Em suma, para simular **e** controlar o Unitree G1 humanoide com mãos Inspire/Dex5 no ROS2 (Ubuntu 24.04):
1. **Modelo e simulação:** Utilize o repositório *unitree_ros* como fonte do URDF e dos plugins de simulação. Converta/importe esse modelo para seu ambiente ROS2, configurando o Gazebo (clássico ou Ignition) para carregar o G1. Isso garante que você tenha o robô completo (29 DOFs, mãos incluídas) na simulação [1].
2. **Controle na simulação:** Implemente controladores ROS2 (usando ros2_control) para enviar comandos às juntas no Gazebo. Você poderá, por exemplo, comandar movimentos dos braços e pernas publicando em tópicos de comando de posição/torque. Lembre-se das limitações: a estabilidade do robô em pé pode exigir ganhos/PIDs adequados nos controladores para não "escorregar" ou cair na simulação (um problema conhecido se os parâmetros físicos não estão perfeitos). Ajuste a física conforme necessário (coeficientes de fricção, inerências, etc., do URDF, caso veja instabilidades).
3. **Ambiente virtual:** Incorpore o modelo do escritório (.obj) no mundo do Gazebo para testar o robô no cenário 3D realista. Isso pode ser feito via arquivo .world ou spawn do modelo estático, conforme explicado. Assim, você terá a cena completa (robô + escritório).
4. **Controle do robô real em ROS2:** Paralelamente, configure o *unitree_ros2* para comunicar-se com o G1 físico. Siga as instruções da Unitree para instalar a SDK2/CycloneDDS e conectar via Ethernet [4] [9]. Com isso, você poderá enviar comandos ao robô real usando tópicos ROS2 similares (a Unitree define tópicos DDS para comandos de movimento, modos de marcha, estados de sensores, etc.). Teste comandos simples para alternar modos do G1, mover as juntas do braço ou controlar a caminhada (dependendo das APIs disponíveis).
5. **Unificando testes:** Idealmente, desenvolva suas aplicações (por exemplo, algoritmos de navegação, manipulação com as mãos, visão computacional) de forma **agnóstica** em relação ao "simulado vs real". Por exemplo, se o robô publica um tópico `/joint_states` no sim e no real (via bridge do unitree_ros2), use os mesmos nomes de tópico; se comandos de velocidade/caminhada são enviados via `/cmd_vel` no sim (como no pacote de exemplo do Go1) [6], tente usar essa convenção também no real (o unitree_ros2 pode ter um tópico ou serviço para andar). Mantendo interfaces consistentes, você conseguirá alternar entre simulação e hardware facilmente.

Seguindo essa abordagem, você aproveitará o melhor de cada repositório: o **unitree_ros fornece a simulação e o modelo URDF rico do G1** [1], enquanto o **unitree_ros2 permite controlar o robô físico via ROS2** de forma nativa [4]. Essa combinação atende à sua necessidade de usar ROS2 para tudo (simulação **e** real) e de inserir o robô em um ambiente Gazebo realista (seu escritório em 3D). Com a devida configuração, você poderá desenvolver no Gazebo (em ROS2) e depois transferir para o G1 real minimizando retrabalho. Boa sorte na implementação!

Referências: Unitree Robotics open-source repositories [1] [5]; Documentação oficial dos pacotes *unitree_ros* [2] e *unitree_ros2* [4] (GitHub); Exemplo de simulação do Go1 em ROS2/Gazebo [6].

---

[1] [3] [5] Unitree Robotics · GitHub
https://github.com/unitreerobotics

2 7 8 GitHub - unitreerobotics/unitree_ros
https://github.com/unitreerobotics/unitree_ros

4 9 GitHub - unitreerobotics/unitree_ros2
https://github.com/unitreerobotics/unitree_ros2

6 GitHub - Atharva-05/unitree_ros2_sim: A repository of ROS 2 packages for simulation of the Unitree Go1 legged robot in Gazebo.
https://github.com/Atharva-05/unitree_ros2_sim