ChatGPT

Treinamento de um Salto Mortal no Unitree G1 via Imitação de Movimento

Visão Geral do Desafio

Treinar um robô **Unitree G1** (humanoide) para realizar um **salto mortal para trás (backflip)** envolve **imitação de movimento**: usar dados de movimento humano capturados (MoCap) como referência para que a política de RL aprenda a reproduzir o salto. Dividiremos o processo em etapas claras:

1. **Encontrar e baixar sequências MoCap de backflip** em formatos adequados (C3D, BVH ou SMPL).
2. **Retargeting (remapeamento) do movimento humano** para o esqueleto do robô G1 (12 graus de liberdade nas pernas).
3. **Conversão do movimento retargeted em *buffers* de referência** no formato esperado pelo `unitree_rl_gym` – contendo tempo, ângulos articulares (`q_ref`), posições dos pés (`p_foot_ref`), velocidade do centro de massa (`v_com_ref`), orientação do tronco (`quat_ref`) e indicadores de contato de pés.
4. **Integração no ambiente de treino (Isaac Gym)**, incluindo configurar recompensas de imitação e permitir acionar o salto com a tecla **espaço**, iniciando e finalizando em uma postura equilibrada de pé.
5. **Validação em MuJoCo e ROS2** – opcionalmente testar a trajetória/política em outra simulação e implementar no robô real via ROS2.

Cada etapa é explicada passo a passo abaixo, com definições de termos técnicos e links diretos para dados, ferramentas e referências relevantes.

1. Seleção e Download de *Datasets* Públicos de Backflip (MoCap)

Para obter sequências de salto mortal, podemos recorrer a *datasets* públicos de movimentos humanos capturados por MoCap. Priorize conjuntos que já disponibilizam os dados nos formatos desejados:

- **CMU Graphics Lab Motion Capture Database** – Um dos maiores acervos gratuitos de MoCap. Contém milhares de sequências, incluindo diversas acrobacias e **salto(s) mortal(is)**. Por exemplo, o *subject* 87 tem movimentos *87_03* e *87_04* rotulados como *"Backflip"*, e o *subject* 88 inclui *88_01 Backflip* e até combinações como *cartwheel into backflip* [1] [2].
- *Download:* A base pode ser acessada no site oficial da CMU [3], que permite buscar por categoria ou número de sujeito e baixar arquivos em vários formatos. Cada movimento geralmente tem: um arquivo **ASF/AMC** (esqueleto e animação), um **C3D** (posições 3D dos marcadores brutos) e até um vídeo de referência. Você pode baixar, por exemplo, o arquivo `87_03.c3d` e `87_03.amc` correspondentes ao backflip do sujeito 87 (o site lista links diretos para `.c3d` e `.amc`).
- *BVH:* Para conveniência, a comunidade CGSpeed disponibilizou uma conversão completa do acervo CMU para **BVH** (formato Biovision) compatível com softwares 3D. Todas as sequências já convertidas podem ser obtidas em pacotes .zip no site do CGSpeed – o qual lista cada movimento e seu arquivo

BVH correspondente [4]. Por exemplo, você encontrará os BVHs como `87_03.bvh` para o backflip. Essa é uma forma rápida de obter os dados em formato de hierarquia de articulações sem precisar processar AMC.

- *Formato SMPL:* Alternativamente, o dataset CMU (e muitos outros) foi unificado no formato paramétrico **SMPL** através do projeto **AMASS** (Archive of Motion Capture as Surface Shapes). O AMASS consolidou 15 conjuntos de MoCap em um formato comum baseado no modelo SMPL [5]. Isso significa que você pode baixar sequências do CMU já como parâmetros SMPL (vetores de pose e shape) – útil caso opte por um pipeline baseado em SMPL. *(Explicação: SMPL é um modelo estatístico de corpo humano que representa pose com ~72 parâmetros e shape com ~10; a partir desses parâmetros é possível obter as posições articulares do esqueleto padrão do SMPL).* O site do AMASS [5] fornece acesso aos dados unificados. Sequências de backflip do CMU aparecerão dentro do AMASS com identificadores correspondentes.

- **SFU Motion Capture Database (Simon Fraser Univ.)** – Conjunto gratuito que disponibiliza todos os movimentos diretamente em **BVH** *e* **C3D** (além de outros formatos) [6]. Embora focado em dança, locomoção e artes marciais, contém movimentos úteis como **saltos, rolamentos e estrelas (cartwheels)** que são similares em dinâmica a um backflip. Por exemplo, há um movimento de *"Cartwheel"* (estrela) e *"Parkour roll"* em BVH/C3D [7] [8]. Esses podem servir para treinar fases iniciais (impulsão, giro) ou para enriquecimento de dados. O site permite visualizar e baixar cada sequência (como `0007_Cartwheel001.bvh` e `.c3d`). *Observação:* Não identifiquei um backflip explícito no SFU, mas as sequências de **rolamento e mortal lateral** podem complementar seu conjunto de referência.

- **HDM05 Motion Capture Dataset (MPI)** – Base abrangente (~3 horas de movimentos) com mais de 70 classes de movimento gravadas sistematicamente [9]. Disponível em **C3D** e **ASF/AMC**. Inclui muitas ações atléticas (corrida, pulos, chutes, exercícios), embora não haja "backflip" nominal nas classes documentadas. Ainda assim, pode conter *saltos altos* ou *mortal para frente* em meio a sequências de ginástica. Você pode baixar todos os C3D já segmentados em *clips* curtos [10]. Esse dataset é útil se precisar de movimentos adicionais para treinar equilíbrio ou transições (por exemplo, **pulos em altura** como *jumpDown* [11]). Além disso, o HDM05 fornece um *parser* em MATLAB para ler C3D/AMC [12], o que pode ajudar na conversão caso use MATLAB; caso contrário, existem bibliotecas Python (ver abaixo).

- **LAAS Parkour Dataset** – Conjunto menor (65 sequências) focado em técnicas de **parkour** altamente dinâmicas (saltos sobre obstáculos, *vaults*, muscle-up, etc) capturado pelo LAAS-CNRS. Disponível em **C3D** com vídeos sincronizados [13] [14]. Embora não tenha flips explícitos, inclui movimentos de explosão de potência e **aterrissagens complexas** que podem ajudar a validar contatos e equilíbrio. Destaca-se por fornecer dados de **força de contato** medidos por placas de força e sensores – todas essas informações estão salvas nos arquivos C3D [15]. Se desejar entender distribuição de contatos ou comparar dinâmica de aterrissagem, esse dataset pode ser consultado. *(Por exemplo, um "Kong vault" – salto por cima de um obstáculo com apoio das mãos – tem fase aérea e aterrissagem dura, com dados de força no C3D.)*

Como baixar e usar os dados: Após escolher o dataset e baixar os arquivos (BVH, C3D, etc.), você precisará convertê-los em trajetórias de articulações ou posições para o robô. Algumas dicas e ferramentas úteis:

- Para **BVH**: softwares de animação (Blender, MotionBuilder) conseguem importar BVH e visualizar o movimento. Você pode utilizá-los para conferir a qualidade da captura. Também é possível exportar do Blender em outros formatos ou aplicar retargeting usando um rig do robô (mais sobre isso adiante).
- Para **C3D**: ferramentas como **Blender (com addon)** ou **MATLAB** podem ler arquivos .c3d [16]. O MoCap Toolbox do MATLAB é uma opção; em Python, bibliotecas como `pyomeca` ou `ezc3d` conseguem extrair as posições dos marcadores. Note que dados C3D requerem que você tenha o modelo de esqueleto para interpretar as marcas – geralmente você usará o arquivo ASF/AMC do mesmo sujeito para obter a estrutura óssea e então mapear marcadores para articulações.
- Uma rota alternativa é usar o formato AMC/ASF: o Kaggle disponibiliza um pacote do CMU Mocap que inclui os **arquivos .ASF (esqueleto)** e **.AMC (movimento)** além dos .c3d [17]. Com eles, você pode usar um parser – por exemplo, o **AMCParser** em Python (link no Kaggle) – para obter diretamente as rotações articulares por quadro. Esse parser (CalciferZh/AMCParser) facilita converter AMC em um formato numérico manipulável.
- **Conversão para SMPL**: se você preferir trabalhar com SMPL, obtenha os parâmetros SMPL do movimento (via AMASS ou reconstrução a partir de vídeo, como veremos). Depois, será necessário aplicar o modelo SMPL para recuperar posições articulares 3D ou ângulos equivalentes. Lembre-se que o SMPL fornece um esqueleto padrão humano; teremos que adaptá-lo ao robô no próximo passo.

Agora, com os dados de movimento humano em mãos, passamos ao **retargeting**, ou seja, transferir esse movimento para o robô Unitree G1.

2. *Retargeting*: Mapeando o Movimento Humano para o Robô Unitree G1 (12 DOF)

Retargeting é o processo de pegar a animação de um esqueleto humano e adaptá-la para outro esqueleto (neste caso, o do robô G1), preservando ao máximo a semelhança do movimento. O Unitree G1 humanoide possui **12 DoF nas pernas** (6 em cada perna: normalmente 3 no quadril [eixos yaw, pitch, roll], 1 joelho, 2 tornozelo) e possivelmente 1 DoF de cintura. Vamos supor que focaremos nas pernas/tronco, já que os braços podem ficar junto ao corpo durante o salto. A seguir, as etapas e considerações para o retargeting:

- **a) Preparar o modelo do robô para retargeting:** Certifique-se de ter o modelo cinemático do G1 disponível. No `unitree_rl_gym`, deve haver uma URDF ou descrição do robô G1 (ângulos de junta neutros, limites, etc.). Isso é importante para podermos calcular cinematica inversa e representar as posturas. Conheça o **sistema de coordenadas** do robô – por exemplo, se o eixo *z* aponta para cima, e onde é a origem (provavelmente no centro do tronco ou cintura do robô). O mesmo vale para os dados humanos: tenha clareza de qual pé é qual, qual direção é "frente", etc., para podermos alinhar ambos.

- **b) Alinhar posições iniciais e escala:** Normalmente, os dados de MoCap humano virão numa certa escala (um humano adulto, ~1.7m de altura). O G1 tem ~1.32m de altura em pé [18], então talvez seja necessário **escalar ligeiramente** as posições ou reduzir amplitudes de movimento para caber nos

limites do robô. Comece alinhando a pose inicial: por exemplo, tanto o humano quanto o robô iniciando em pé, braços ao lado (T-pose ou posição neutra). Aplique rotações ou offsets de posição se necessário para colocar o robô exatamente na mesma postura que o humano no quadro inicial. Isso cuida do componente estático do retargeting (a chamada etapa de *shape alignment*). Ferramentas de retargeting frequentemente primeiro fazem o *matching* das poses de repouso entre as duas armaduras [19].

- **c) Mapeamento de articulações (junta-a-junta):** Defina quais juntas humanas correspondem às juntas do robô:

- **Quadril**: O humano tem 3 rotações no quadril; o robô G1 também (três motores no quadril por perna). No modelo CMU, por exemplo, pode haver eixos rotacionais diferentes; teremos que mapear para yaw/pitch/roll do robô. Poderá ser necessário permutar eixos ou adicionar offsets.
- **Joelho**: mapear direto (1 DoF).
- **Tornozelo**: humanos têm tornozelo com 2 DoF principais (flexão plantar/dorsal e inversão/everse). O G1 parece ter também 2 (pitch e roll no tornozelo). Mapeie-os adequadamente.
- **Pés**: alguns dados incluem posição do pé ou dedos, mas podemos tratar o pé do robô como um ponto ou placa de contato equivalente ao pé humano.
- **Tronco/Cintura**: O G1 tem 1 DoF na cintura (flexão de tronco) – se o modelo humano tiver coluna com vários segmentos, talvez agreguemos tudo isso em um único articulador de tronco no robô. Se o robô tiver esse grau de liberdade, poderíamos usar a inclinação do tronco humano (por ex., ângulo lombar) para controlar esse motor.

Dica: Nem sempre o mapeamento é 1:1. Podemos ignorar pequenas articulações humanas (como dedos, coluna superior) se o robô não as tiver. Concentre-se em replicar os **movimentos das pernas e orientação do tronco**, que são críticos para o salto mortal.

- **d) Cinemática Inversa para posições dos pés:** Uma estratégia robusta de retargeting é via **cinemática inversa (IK)**. Ou seja, em vez de tentar copiar diretamente cada ângulo articular humano (o que pode não fazer sentido se as proporções diferem), vamos **seguir a trajetória dos pés e do tronco** do humano, resolvendo os ângulos do robô que realizam essa mesma trajetória. Por exemplo:
- Extraia do MoCap a **posição 3D de cada pé do humano ao longo do tempo** e marque quando estão em contato com o solo (pode deduzir: quando a velocidade do pé ≈ 0 e ele está na altura mínima, ou usar dados de força se disponíveis como no dataset de parkour [15]). Teremos, por exemplo, que no backflip humano ambos os pés estão no chão no frame inicial, depois sobem (decolagem) e ficam sem contato no ar, e finalmente voltam ao solo na aterrissagem.
- Extraia também a **posição do centro de massa (CoM)** do humano ou, mais acessível, a posição do quadril/pélvis ao longo do tempo, bem como a **orientação do tronco** (por exemplo, a rotação da pelve ou tórax). No backflip, a orientação do tronco humano faz uma rotação completa para trás em torno do eixo lateral.
- Agora, para cada quadro no tempo, imponha essas como metas para o robô: queremos que o **tronco do robô** (sua base) tenha a mesma orientação e posição relativa que o tronco do humano, e que os **pés do robô** alcancem posições comparáveis às do humano. Por exemplo, no momento do salto, os pés do humano se estendem e deixam o chão – para o robô, isso significará todos os dois pés (no caso humanoide) decolando também. No pico do salto, o humano talvez flexione as pernas no ar; o robô deveria fazer similar (dobrando joelhos) para reunir as pernas durante a cambalhota.

Com IK, você pode especificar onde cada pé deve estar e a pose do tronco, e resolver os ângulos das juntas das pernas que satisfazem isso.

- Use uma **biblioteca de robótica** se possível para a IK – por exemplo, o *Pinocchio* (biblioteca de dinâmica do LAAS) ou o *RBDL*, ou até funções do MuJoCo or Isaac Gym (Genesis) se expostas. Em Python, uma opção simples é usar algoritmos como *Levenberg-Marquardt* sobre as equações de posição. Como ponto de partida, coloque o robô na pose inicial idêntica ao humano e teste a IK para as posições dos pés no primeiro frame (que deve ser trivial – pés no chão próximos da posição inicial).

- Garanta continuidade: o backflip é rápido, então idealmente sua solução IK deve produzir uma sequência suave de ângulos. Você pode resolver IK em cada quadro de forma independente e depois suavizar, ou incorporar velociadades no processo. Alguns pipelines fazem *retargeting* resolvendo uma otimização global que minimiza erros de posição dos pés e diferenças de pose durante todo o movimento.

- **e) Montar os dados de referência:** Ao final do retargeting, você deve obter:

- Uma sequência temporal (digamos, dt = 1/60s por quadro, ou conforme o dataset).
- Para cada tempo *t*: um conjunto de **ângulos articulares do robô (** `q_ref` **)** resultantes.
- As **posições dos pés (** `p_foot_ref` **)** – que você já usou na IK – representadas no referencial do robô (ex: coordenadas dos pés em relação ao quadro do tronco).
- A **velocidade do CoM (** `v_com_ref` **)** – pode ser calculada diferenciando a posição do CoM do robô (ou do tronco) frame a frame. Como referência, você pode usar a velocidade do quadril do humano também. O CoM real do robô incluiria pernas, mas considerar o tronco como massa dominante é aceitável se distribuição for fixa.
- A **orientação do tronco (** `quat_ref` **)** – provavelmente você terá isso diretamente do moCap (orientação da pelve humana). Converta para quaternion (x,y,z,w) ou outro formato consistente com o que o ambiente espera. Certifique-se de que está no mesmo sistema de coordenadas do robô (pode ser necessário rotacionar 90 graus em algum eixo se o modelo do humano e do robô diferirem em orientação inicial).
- Os **indicadores de contato (** `contacts` **)** – para cada pé do robô (no humanoide, pé esquerdo e direito), um flag binário indicando se está no chão (1) ou no ar (0) naquele frame. Isso você determina pelas fases do salto: no início, ambos 1 (de pé no chão), durante o salto ambos 0 (voo livre), na aterrissagem voltam a 1. *(Esses contatos podem ser deduzidos da altura do pé em relação ao solo – p. ex., contato = 1 quando a coordenada z do pé <= 0 no humano, ou usando os dados de força se disponíveis* [20] *.)*

Este processo de retargeting pode ser complexo, mas é crucial para gerar uma trajetória viável para o robô. Em pesquisas recentes, pipelines similares têm sido usados para transferir movimentos humanos para humanoides. Por exemplo, o framework **ASAP (Aligning Simulation and Real Physics)** da CMU/NVIDIA demonstra: eles capturaram movimentos humanos (de vídeo, reconstruídos em SMPL), treinaram uma política para segui-los em simulação, e **retargetaram esses movimentos para o Unitree G1** antes de implantar no robô real [21] . Eles utilizam um retargeting em duas etapas – alinhamento de formato e movimento – para garantir correspondência precisa [19] . Ou seja, validar que nossas etapas acima (alinhar pose inicial, escalar, IK para pés/tronco) seguem as melhores práticas.

Dica: se você não deseja programar toda a IK manualmente, outra abordagem é usar ferramentas de animação: - No **Blender**, você pode importar o BVH do humano e ter (ou criar) um rig do Unitree G1 (por

exemplo, a URDF pode ser convertida em um esqueleto via addons). Depois, usar técnicas de *retargeting* de animação no Blender: basicamente, alinhar os bones correspondentes e transferir a animação do bone humano para o bone do robô, talvez ajustando pesos ou controlando pelo IK do Blender. Em seguida, você pode exportar a animação resultante das juntas do robô. Isso requer alguma experiência em animação 3D, mas Blender é gratuito e potente para esse tipo de ajuste visual. - Ferramentas comerciais como Autodesk **MotionBuilder** também facilitam retargeting: é praticamente feito para mapear mocap em rigs de personagens. Se tiver acesso, você poderia configurar o esqueleto do G1 no MotionBuilder e usar a funcionalidade de *Mocap to Rig* [22] para aplicar os dados .C3D ou .BVH nos controladores do G1.

Uma vez concluído o retargeting, você terá a trajetória de referência pronta – essencialmente uma série temporal de estados alvo para o robô realizar o backflip. Agora precisamos fornecer isso ao algoritmo de treinamento.

3. Conversão para Buffers de Referência Compatíveis com `unitree_rl_gym`

O `unitree_rl_gym` (baseado no NVIDIA Isaac Gym) espera os **dados de referência do movimento** em um formato específico, geralmente como arrays numpy ou arquivos `.npz` contendo as chaves `{t, q_ref, p_foot_ref, v_com_ref, quat_ref, contacts}`. Vamos montar esse buffer:

- Organize os arrays conforme as definições:
- `t`: vetor de tempo (por exemplo, `t = [0, Δt, 2Δt, ..., T_final]` onde Δt é o passo de tempo do seu movimento, tipicamente 1/60 s ou 1/120 s dependendo do dataset).
- `q_ref`: matriz de ângulos articulares do robô ao longo do tempo. Dimensão será [N_frames x N_juntas]. Para o G1, N_juntas = 12 (pernas). Inclua também a junta de cintura se ela for usada (ficaria 13). Certifique-se da ordem das juntas correspondendo à ordem esperada pelo código do `unitree_rl_gym` – provavelmente algo como `[leg1_joint1, leg1_joint2, ..., leg2_joint6]`.
- `p_foot_ref`: posições dos pés ao longo do tempo. Pode ser um array [N_frames x 2 x 3], por exemplo, com posição 3D de pé esquerdo e direito em cada quadro (no referencial do tronco do robô). Se o robô tiver 4 "patas" (no caso de quadrupede, aplicaria), mas aqui para humanoide são 2 pés.
- `v_com_ref`: vetor velocidade do centro de massa em cada tempo [N_frames x 3] (componentes vx, vy, vz). Se você não calculou o CoM exato, usar a velocidade do tronco/pelve do robô é suficiente como aproximação, já que as pernas são relativamente leves comparado ao tronco + cabeça.
- `quat_ref`: orientação do tronco como quaternion em cada tempo [N_frames x 4]. Alternativamente, alguns ambientes usam [N_frames x 3] de ângulos de Euler – mas provavelmente quaternion (w,x,y,z ou x,y,z,w). Use a convenção do ambiente – por exemplo, no Isaac Gym normalmente usam `(w,x,y,z)` quaternions. Essa orientação deve corresponder à base do robô (no G1, a base fixada no tronco/peito).

- `contacts`: array [N_frames x N_feet] de flags 0/1. Para humanoide, N_feet = 2. Ex.: `contacts[t, 0] = 1` se pé esquerdo no chão no instante t.

- **Geração do arquivo:** O Python pode ser usado para salvar esses arrays. Exemplo usando numpy:

```
import numpy as np
np.savez('backflip_ref.npz', t=t, q_ref=q_ref, p_foot_ref=p_foot_ref,
         v_com_ref=v_com_ref, quat_ref=quat_ref, contacts=contacts)
```

Isso criará um arquivo `backflip_ref.npz` contendo todos os buffers.

- **Validação rápida:** Pode ser útil verificar se os dados fazem sentido antes do treino. Por exemplo, plotar os ângulos `q_ref` ao longo do tempo para ver se são suaves e dentro dos limites do robô. Ou animar o esqueleto do robô seguindo `q_ref` para ver se realmente parece um backflip. Se você tem o Isaac Gym já configurado com a classe de robô G1, pode escrever um pequeno script de visualização usando as APIs do gym (ou até usar MuJoCo just for visualization). Isso previne alimentar dados incorretos ao RL.

- **Utilização de múltiplas sequências:** Se encontrar mais de uma captura de backflip (por exemplo, vários sujeitos ou variações como *backflip com giro*), você pode incluir várias no treinamento para maior robustez. Nesse caso, poderia concatenar as sequências no buffer com algum separador ou fornecer várias trajectórias e amostrar uma por episódio. Algumas implementações criam um **dataset de motion clips** e escolhem aleatoriamente qual imitar por episódio, para generalizar. Mas inicialmente, usar uma única sequência explícita está OK para focar no movimento desejado.

- **Ferramentas de conversão**: Lembre-se de aproveitar ferramentas existentes:

- O *parser* em MATLAB do HDM05 já citado pode ingerir C3D/ASF e dar ângulos, servindo de referência caso precise conferir valores [12] .
- O Kaggle/AMCParser em Python para AMC também pode exportar ângulos que você reorganiza em `q_ref`.
- Se você reconstruiu via SMPL, terá as rotações de cada articulação em ângulos de eixo, que precisam ser convertedos para o sistema do robô. Nesse caso, usar a etapa de IK mencionada já gera `q_ref` diretamente no domínio do robô.

Em resumo, ao final desta etapa você terá um arquivo (ou arrays na memória) prontos para serem utilizados pelo ambiente de RL, definindo exatamente a trajetória alvo que o robô deve aprender a seguir.

4. Configurando o Treinamento no Isaac Gym (unitree_rl_gym) e Disparo do Salto por Comando

Com os dados de referência prontos, vamos integrar isso ao pipeline de *Reinforcement Learning* no Isaac Gym:

- **a) Configurar o ambiente de imitação:** O `unitree_rl_gym` é baseado no *Legged Gym* da Unitree/NVIDIA, e já suporta simulação do G1. Precisamos habilitar a **imitação de movimento** (“motion tracking”) no ambiente. Isso normalmente envolve:
- **Carregar os dados de referência:** modificar/estender a tarefa (no código, provavelmente dentro de `legged_gym/envs/g1/` ou similar [23]) para ler o arquivo `backflip_ref.npz`. Pode-se criar uma nova classe de ambiente, digamos `BackflipTask`, derivada da de locomoção, mas com

recompensas de imitar pose. Por exemplo, definir um *curriculum* de fase: estado inclui um fase temporal $0\to1$ que percorre a trajetória [24] , e em cada *step* compara o estado do robô com o estado de referência naquele fase.

- **Estado do agente:** inclua no vetor de observação tudo o que for necessário para o controle de pose. Tipicamente, observações podem ser: orientações atuais das juntas (para calcular erro vs `q_ref`), velocidade do tronco vs `v_com_ref`, orientação atual vs `quat_ref`, talvez posição dos pés vs `p_foot_ref`. Muitas implementações de imitação fornecem diretamente ao agente o **erro** ou o **estado referência próximo** para guiar. Outra abordagem (DeepMimic style) é fornecer ao agente *o próximo passo da referência* como parte da observação, so it knows what pose to aim for. Verifique a estrutura esperada pelo `unitree_rl_gym` – possivelmente já há suporte a um **"reference trajectory"** para movimentos (já que o repositório menciona suporte a controladores e múltiplos robôs, talvez tenham incluído exemplos).
- **Função de recompensa:** defina recompensas que penalizam a diferença entre o estado atual do robô e a referência naquele instante. Por exemplo:
    - $r_{\text{pose}} = -|q_{\text{robô}} - q_{\text{ref}}|^2$ (erro quadrático dos ângulos),
    - $r_{\text{feet}} = -\sum_{\text{pés}}|p_{\text{robô}} - p_{\text{ref}}|^2$ (erro de posição dos pés),
    - $r_{\text{com}} = -|\dot{v}*{com} - *dot{v}*|^2$ (diferença de aceleracão do CoM, ou use velocidades linear/angulares),}^{ref
    - $r_{\text{orient}} = -\text{quatDiff}(quat, quat_ref)$ (distância entre a orientação atual e desejada).
      Além disso, pode adicionar pequenas recompensas de esforço (para evitar ações muito bruscas) e estabilidade na aterrissagem (por exemplo, bonus se contacts = 1 nos frames finais coincidir com pés no chão efetivamente). O objetivo é que a política aprenda a **seguir a sequência de referência do início ao fim**, recebendo recompensas maiores quanto mais próxima estiver do movimento alvo.

- **Episódio e resete:** configure o episódio para durar exatamente o tempo do backflip (desde o início em pé até aterrissagem). Por exemplo, se a sequência tem 2 segundos, então o episódio poderia ter ~2 segundos simulado (120 steps se 60Hz). No final, resetamos o ambiente. No início de cada episódio, coloque o robô na posição inicial equilibrada (você pode usar o primeiro frame do `q_ref` diretamente para inicializar as articulações). Assim garantimos consistência.

    - *Observação:* Um detalhe importante é preparar a política para recuperar o equilíbrio no fim. Às vezes, após o fim da referência, pode ser útil deixar a simulação rodar alguns segundos para ver se o robô permanece de pé – e dar recompensa adicional por não cair. Isso encoraja o controle a garantir equilíbrio pós-aterrissagem. Pode-se incluir no final do episódio alguns passos extras com recompensa por manter tronco ereto e pés no chão.

- **b) Treinamento por RL:** Com o ambiente configurado, use o algoritmo de sua escolha (PPO, por exemplo, já implementado no legged_gym) para treinar a política. Inicialmente, a política começará falhando (o robô cairá ou não conseguirá decolar corretamente), mas com o tempo e as recompensas de imitação, ele deverá aprender a reproduzir o arco do movimento. A **fase aérea** é crítica – você pode precisar de muitas iterações e talvez curriculum:

- Uma técnica é **facilitar o aprendizado inicial**: por exemplo, começar treinando apenas a fase de decolagem e pouso (sem exigir rotação completa), depois gradualmente aumentar a exigência até o flip completo. Contudo, como temos a referência completa, pode deixar a recompensa guiar todo o

movimento de uma vez. As heurísticas de *curriculum* podem ser: aumentar gradualmente a velocidade exigida ou amplitude.
- Assegure-se de usar um número suficiente de ambientes paralelos (Isaac Gym permite milhares) para explorar variações. Talvez randomizar levemente a massa do robô, ou a altura inicial, para robustez (domain randomization leve).

- Monitore métricas: erro de pose médio, altura máxima atingida, etc. Assim verá progressos.

- **c) Execução e disparo com tecla espaço:** Após treinar e obter uma política capaz de fazer o backflip em simulação, você vai integrá-la para uso interativo. A ideia é que o robô normalmente fique de pé equilibrado, e ao apertar **Espaço** a animação do salto seja iniciada. Existem algumas maneiras de implementar isso:

- **Modo de reprodução direta da referência:** Uma forma simples (embora não "aprendizado", mas para validação) é programar que ao apertar espaço, o robô siga a *trajetória referencial pré-gravada*. Ou seja, aplicar os comandos de torque/posição seguindo `q_ref` ao longo do tempo. Isso testaria se a trajetória em si funciona no simulador (seria como um *playback* open-loop). No entanto, o mais interessante é usar a **política aprendida** para executar o movimento, o que dará mais robustez (ela pode ajustar se algo sair do ideal).

- **Uso da política treinada:** Se você integrou o backflip como um ambiente separado, você pode escrever um script de *play/inferência* (muitas libs têm um `play.py`). Esse script carrega a política treinada e aplica no simulador. Para permitir o controle por tecla:

    - Capture a tecla *space* (por exemplo, usando uma biblioteca de interface gráfica/pygame, ou se estiver em um ambiente gráfico do Isaac Sim, usar os eventos de teclado da GUI). No contexto do Isaac Gym (preview 3 - se é o que está usando), talvez você precise envolver com alguma janela OpenGL custom. Outro caminho é rodar a simulação passo a passo manualmente e, quando a tecla for detectada, acionar uma mudança de estado.
    - Quando *space* for pressionado, **resete o ambiente de backflip** e comece a simulação do episódio do salto. Ou, se o robô já estava na posição inicial dentro do sim, possivelmente você pode diretamente iniciar a referência do momento inicial. Dependendo de como estruturou, possivelmente mais fácil é: ao apertar espaço, chame `env.reset()` para garantir estado inicial perfeito, e depois libere a simulação para rodar por N steps equivalentes à duração do flip, enquanto aplica a política a cada passo. Você pode também engajar a política somente quando a tecla é pressionada – antes disso, manter o robô parado.
    - **Retorno à posição equilibrada:** Idealmente, sua política já termina com o robô em pé. Se assim for, após terminar o episódio, você pode simplesmente voltar ao modo de espera (robô parado). Caso queira que o robô fique parado antes do comando sem drift, você poderia usar uma **política separada de equilíbrio** (por exemplo, um controlador PD para manter postura ou até uma outra política RL treinada para balance, se houver). Uma solução pragmática: depois que o backflip policy termina, congele a simulação ou ative um controlador de postura nos motores para segurar o robô em pé.
    - **Prevenindo execução acidental:** Você pode configurar para que a tecla espaço só funcione se o robô estiver atualmente em estado "pronto". Exemplo: definir uma variável `ready = True` inicialmente. Ao apertar espaço, se `ready`, então inicia o flip (set `ready=False` durante a execução para ignorar entradas adicionais), e após concluir e estabilizar, define

`ready=True` novamente para aceitar o próximo comando. Assim evita múltiplos flips em sequência sem preparar.

- **Integração com *Unitree_rl_gym* framework:** Como referência, o repositório Unitree RL Gym foi concebido para suportar *vários robôs e ambientes*, incluindo possivelmente o G1. Ele inclusive menciona compatibilidade com **Isaac Gym e MuJoCo** [25]. Veja se já existe um exemplo de *keyboard control*: algumas demos usam teclas para, por exemplo, alternar entre modos de andar. Se não houver, você terá que inserir manualmente a leitura do teclado no loop principal de teste.

- **d) Ajustes de última hora:** Fazer um robô simulado dar um backflip é desafiador, então talvez seja necessário alguns truques:

- *Assistência gravitacional:* No mundo real, robôs usam forte potência para saltar. Se a política estiver tendo dificuldade para decolar, você pode diminuir ligeiramente a gravidade na simulação durante o treino (ex: 0.9g) para facilitar, depois ir aumentando para 1.0g conforme melhora. Cuidado para não viciar muito – mas é uma forma de curriculum.
- *Contato de pés:* Garanta que o solver de física do Isaac Gym está lidando bem com o choque na aterrissagem. Pode ser preciso amortecer um pouco (parâmetros de restituição, damping das juntas). Verifique no simulador se o robô não está quicando ou atravessando o chão – ajuste parâmetros de contato se necessário.
- *Reset se cair:* Configure para terminar o episódio (com penalidade) se o robô cai antes de completar, para que ele aprenda a não se espatifar. Um critério: se a altura do tronco cair abaixo de um limite ou se uma orientação exceder certo ângulo (indicando queda), termina episódio.

Depois de tudo configurado e com a política realizando o salto corretamente no simulador quando comando, você já alcançou o objetivo principal. O robô virtual deve partir de posição estática equilibrada, executar o backflip e retornar à posição de pé. O vídeo do Unitree G1 realizando flips e artes marciais sugere que abordagens assim de **imitação a partir de MoCap e RL** são efetivamente usadas para dar essas habilidades ao robô (há menção a *"video mimic learning"* no contexto do G1) [26] [27].

5. Validação em MuJoCo e Integração ROS2 para o Robô Real

Por fim, é prudente validar o movimento e preparar a transição para o robô físico:

- **Teste em MuJoCo (Simulação Alternativa):** Já que o `unitree_rl_gym` suporta MuJoCo, você pode exportar sua política e referência para rodar no MuJoCo e comparar comportamentos. MuJoCo possui diferenças de modelagem física (contatos, inércias) – é um *sim2sim* test. Isso ajuda a revelar se o movimento é robusto fora do ambiente exato do Isaac Gym. O paper ASAP, por exemplo, avalia políticas transferindo de IsaacGym para IsaacSim e MuJoCo (Genesis) antes de ir pro real [28] [29].
- Para testar, crie um modelo MJCF do G1 (talvez a Unitree forneça, ou use a URDF com Mujoco). Aplique a sequência de ações ou a política em MuJoCo e veja se ainda consegue completar o flip. Ajustes de PD podem ser necessários, mas se as referências estiverem boas, a política deve generalizar.

- **Simulação sem aprendizado (reprodução):** você pode simplesmente reproduzir o `q_ref` no MuJoCo com controle de posição PD para ver se a física do MuJoCo aceita o movimento sem o robô cair. Isso verifica se há alguma sutileza na dinâmica (por ex, se no Isaac Gym tinha alguma estabilidade artificial).

- **Implementação no robô real via ROS2:** Com a confiança de simulação, passar para o hardware real requer uso da interface de controle do G1:

- A Unitree fornece um SDK (SDK2) e pacotes ROS2 para seus robôs (até agora mencionam Go2, B1/B2, H1 – G1 possivelmente similar ao H1) [30]. Verifique se há atualizações no repositório oficial para suporte ao G1 [31]. Provavelmente existe um nó ROS2 que recebe mensagens de posição/velocidade de junta para o G1.
- **Crie um nó ROS2** que carrega a política treinada (a rede neural) e, ao receber um comando (por exemplo, uma mensagem std_msgs/Bool "do_backflip" ou mesmo capturando tecla), ele obtém o estado atual do robô (ângulos e velocidades via tópico do SDK) como observação e executa a política passo a passo, mandando torque/comandos de posição para os atuadores do G1 em alta frequência. É importante garantir um loop de controle rápido e *realtime* para sincronizar com o robô (talvez rodando a 200Hz ou mais, dependendo da latência).
- **Safety:** Treinar no sim não garante sucesso no real de primeira. Considere medidas de segurança: use cabos de proteção nas primeiras tentativas físicas, limite o torque ou reduza a velocidade em 10-20% inicialmente (pode escalar o tempo do movimento para ver o comportamento). Conforme ASAP discutiu, sempre há discrepâncias dinâmicas – os autores usaram um "Delta Action Model" para compensar diferenças [32]. Sem entrar nesse mérito, esteja preparado para iterar: talvez refinar a política com dados reais (finetune), ou ajustar parâmetros do modelo do robô para melhor correspondência.
- **Teste incremental:** Você pode primeiro testar sub-fases do movimento no real. Por exemplo, treine o robô para pular e cair de pé sem tentar rotacionar 360°. Depois acrescente a rotação. Isso pode evitar tombos graves inicialmente.

- **Verificação de contatos:** Use os sensores do G1 (IMU, talvez sensores de esforço nas juntas) para detectar se a aterrissagem ocorreu corretamente. Um sucesso será o robô aterrissar e estabilizar sem cair. Se ele conseguir, parabéns – o método de imitação funcionou!

- **Controlador de equilíbrio pós-salto:** Em ROS2, também combine a política de flip com um controlador de equilíbrio básico para após o movimento. Por exemplo, ao término, troque para modo de torque zero nas juntas (robô passivo) ou algum controle ativo para manter postura ereta (talvez um simples P em torno da posição reta das pernas). Isso pode segurar o robô caso a política não contemple bem os segundos após a aterrissagem.

- **Consideração final sobre ROS2:** O pacote `unitree_ros2` permite comunicar com o robô usando DDS diretamente [30]. Você poderia publicar um trajeto de referência inteiro como mensagem de JointTrajectory para o controlador – porém, dado que o movimento é muito rápido e necessita de controle fino, é melhor usar sua política aprendida em tempo real enviando comandos low-level (velocidade/corrente). Verifique na documentação do G1 qual nível de controle é aberto – possivelmente há interface para position or velocity control mode em cada articulação via API. Use a que se assemelhe ao que foi usado em sim (se na sim você controlou torques diretamente, talvez usar controle de torque no real, se disponível, resulte na maior fidelidade).

Em suma, a validação no MuJoCo e no robô real garantirá que o salto não é apenas um *feito de simulação*. Com o pipeline acima – do dataset ao retargeting, RL no Isaac Gym, e depois transferência – você segue um caminho já trilhado em pesquisa de ponta para ensinar robôs a executar movimentos altamente dinâmicos

imitando humanos [21]. Boa parte do trabalho estará em ajustar detalhes para o seu caso específico, mas essas etapas fornecem um **guia estruturado**.

Referências e Links Úteis

- **CMU Graphics Lab MoCap Database** – acervo de movimentos (incl. acrobacias) gratuito. Página oficial: [3]. Ver lista de backflips nos sujeitos 87-90 [1] [2].
- **CGSpeed CMU BVH Conversion** – download dos BVHs já convertidos do dataset CMU [4] (Google Sites).
- **SFU MoCap Database** – dataset com BVH e C3D de diversos movimentos (saltos, parkour) [6].
- **MPI HDM05 Dataset** – movimentos em C3D/ASF (exercícios, pulos) [9]. Página de download e [documentação PDF] [33].
- **LAAS Parkour Dataset** – movimentos de parkour com C3D + vídeos + forças [13] [15] (inclui link direto para baixar .tar.gz com C3D).
- **Kaggle Mocap (CMU)** – dataset espelho contendo arquivos C3D e AMC do CMU, e link para AMCParser [17].
- **CalciferZh AMCParser** – script Python para ler arquivos .ASF/.AMC (GitHub link na página do Kaggle acima).
- **Unitree unitree_rl_gym (GitHub)** – repositório oficial do ambiente RL (Isaac Gym) para robôs Unitree (Go1, A1, B1, G1 etc) [25]. Útil para ver exemplos de configuração.
- **ASAP paper (He et al. 2023)** – Framework alinhando sim-real para humanoides ágeis (usa G1). Descrição do pipeline de imitação do humano para G1 [21] [19]. Código: https://github.com/LeCAR-Lab/ASAP (contém possivelmente scripts de retargeting SMPL->G1).
- **Generalized Animal Imitator (2023)** – Exemplo de trabalho onde um quadrúpede aprende múltiplas habilidades (incl. backflip) imitando movimentos [34] [35]. Mostra a viabilidade de um único controlador aprender flips.
- **Unitree ROS2 SDK2** – repositório para integrar robôs Unitree ao ROS2 (CycloneDDS) [30]. Útil para controlar o G1 via ROS2.

Com esses recursos e passos, você terá um **guia completo** para implementar o salto mortal: desde obter os dados certos até colocar o robô para girar no ar tanto na simulação quanto, com devidos cuidados, no mundo real. Boa sorte no projeto, e bons saltos! [21] [25]

---

[1] [2] [4] cgspeed - BVH Conversion Release - motions list
https://sites.google.com/a/cgspeed.com/cgspeed/motion-capture/the-motionbuilder-friendly-bvh-conversion-release-of-cmus-motion-capture-database/bvh-conversion-release-motions-list

[3] Carnegie Mellon University - CMU Graphics Lab - motion capture ...
https://mocap.cs.cmu.edu/

[5] AMASS
https://amass.is.tue.mpg.de/

[6] [7] [8] SFU MOCAP
https://mocap.cs.sfu.ca/

[9] [12] [33] Motion Database HDM05
https://resources.mpi-inf.mpg.de/HDM05/

10 11 Motion Database HDM05
https://resources.mpi-inf.mpg.de/HDM05/cuts/index.html

13 14 15 20 main
https://gepettoweb.laas.fr/parkour/

16 c3d Mocap to iClone - Newb Question - Reallusion Forum
https://forum.reallusion.com/PrintTopic413507.aspx

17 CMU Mocap - Kaggle
https://www.kaggle.com/datasets/kmader/cmu-mocap

18 Humanoid robot G1_Humanoid Robot Functions_Humanoid Robot Price | Unitree Robotics
https://www.unitree.com/g1/

19 21 24 28 29 32 ASAP: Aligning Simulation and Real-World Physics for Learning Agile Humanoid Whole-Body Skills
https://arxiv.org/html/2502.01143v1

22 Retargeting Mocap Data from C3D or BVH Files to a Rig - Autodesk
https://download.autodesk.com/global/docs/Softimage2014/en_us/userguide/files/mocap_RetargetingBiovisionorAcclaimMocapDatatoaRigMocaptoRig.htm

23 unitree_rl_gym/legged_gym/envs/g1/g1_config.py at main - GitHub
https://github.com/unitreerobotics/unitree_rl_gym/blob/main/legged_gym/envs/g1/g1_config.py

25 STEMfinity | Unitree G1-Comp | Unitree
https://stemfinity.com/products/unitree-g1-comp?srsltid=AfmBOopxn8K66Eogk-WEx72g6BYuy2Xjv-twKnq3lnc-TWjkhslCitZc

26 Unitree (@UnitreeRobotics) / X
https://x.com/unitreerobotics?lang=en

27 World's First Side-Flipping Humanoid Robot: Unitree G1 - YouTube
https://www.youtube.com/watch?v=29xLWhqME2Q&pp=0gcJCfwAo7VqN5tD

30 GitHub - unitreerobotics/unitree_ros2
https://github.com/unitreerobotics/unitree_ros2

31 Official Open Source - Unitree Robotics
https://www.unitree.com/opensource

34 35 Generalized Animal Imitator: Agile Locomotion with Versatile Motion Prior
https://rchalyang.github.io/VIM/