

2 实践：训练 G1 人形机器人行走控制强化学习模型

作业目标

在机器人控制领域，强化学习技术应用前景广阔。本次作业聚焦于利用强化学习算法实现对 G1 机器人的行走控制。完成作业过程中，大家能熟悉强化学习原理和应用流程，掌握在 Isaac Gym 仿真平台训练机器人控制算法，探索在 Mujoco 中验证模型效果，提升在机器人研发与人工智能领域的能力。

系统要求


- 操作系统：推荐使用 Ubuntu 20.04
- 显卡：Nvidia 显卡（显存>8GB、RTX系列显卡）
- 驱动版本：建议使用 525 或更高版本

硬件准备

由于 Isaac_gym 仿真平台需要 `CUDA`，本文建议硬件需要配置 NVIDIA 显卡（显存>8GB、RTX系列显卡），并安装相应的显卡驱动。建议系统使用 `Ubuntu20`，显卡驱动 525 版本。

| | | | | | | | | | |
|---------------------------------------|------------------|------------|---------------|----------------------------|------------------------------|--------|----------------------|------------|-----|
| +-----+-----+-----+-----+-----+-----+ | | | | | | | | | |
| NVIDIA-SMI | | 525.125.06 | | Driver Version: 525.125.06 | | | CUDA Version: 12.0 | | |
| +-----+-----+-----+-----+-----+-----+ | | | | | | | | | |
| GPU | Name | | Persistence-M | | Bus-Id | Disp.A | Volatile Uncorr. ECC | | |
| Fan | Temp | Perf | Pwr:Usage/Cap | | Memory-Usage | | GPU-Util | Compute M. | |
| | | | | | | | MIG M. | | |
| +-----+-----+-----+-----+-----+-----+ | | | | | | | | | |
| 0 | NVIDIA RTX A4000 | | Off | | 00000000:01:00.0 | On | | | Off |
| 41% | 49C | P8 | 19W / 140W | | 710MiB / 16376MiB | | 34% | Default | |
| | | | | | | | | | N/A |
| +-----+-----+-----+-----+-----+-----+ | | | | | | | | | |
| +-----+-----+-----+-----+-----+-----+ | | | | | | | | | |
| Processes: | | | | | | | | | |
| GPU | GI | CI | PID | Type | Process name | | | GPU Memory | |
| | | ID | ID | | | | | Usage | |
| +-----+-----+-----+-----+-----+-----+ | | | | | | | | | |
| 0 | N/A | N/A | 1033 | G | /usr/lib/xorg/Xorg | | | 110MiB | |
| 0 | N/A | N/A | 1963 | G | /usr/lib/xorg/Xorg | | | 249MiB | |
| 0 | N/A | N/A | 2091 | G | /usr/bin/gnome-shell | | | 107MiB | |
| 0 | N/A | N/A | 28273 | G | ...RendererForSitePerProcess | | | 25MiB | |
| 0 | N/A | N/A | 31494 | G | ...044935437964763312,131072 | | | 29MiB | |
| 0 | N/A | N/A | 32849 | G | gnome-control-center | | | 3MiB | |
| 0 | N/A | N/A | 33003 | G | /usr/lib/firefox/firefox | | | 172MiB | |
| +-----+-----+-----+-----+-----+-----+ | | | | | | | | | |

安装配置

 安装和配置步骤请参考：
https://github.com/unitreerobotics/unitree_rl_gym/blob/main/doc/setup_zh.md，该开源项目由宇树官方维护，不定时更新，建议大家按照上述链接安装环境。

1 创建虚拟环境

建议在虚拟环境中运行训练或部署程序，推荐使用 Conda 创建虚拟环境。如果您的系统中已经安装了 Conda，可以跳过步骤 1.1。

1.1 下载并安装 MiniConda

MiniConda 是 Conda 的轻量级发行版，适用于创建和管理虚拟环境。使用以下命令下载并安装：

代码块

```
1  mkdir -p ~/miniconda3
2  wget https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh -O
   ~/miniconda3/miniconda.sh
```

```
3  bash ~/miniconda3/miniconda.sh -b -u -p ~/miniconda3
4  rm ~/miniconda3/miniconda.sh
```

安装完成后，初始化 Conda：

代码块

```
1  ~/miniconda3/bin/conda init --all
2  source ~/.bashrc
```

1.2 创建新环境

使用以下命令创建虚拟环境：

代码块

```
1  conda create -n unitree-rl python=3.8
```

1.3 激活虚拟环境

代码块

```
1  conda activate unitree-rl
```

2 安装依赖

2.1 安装 PyTorch

PyTorch 是一个神经网络计算框架，用于模型训练和推理。使用以下命令安装：

代码块

```
1  conda install pytorch==2.3.1 torchvision==0.18.1 torchaudio==2.3.1 pytorch-
    cuda=12.1 -c pytorch -c nvidia
```

2.2 安装 Isaac Gym

Isaac Gym 是 Nvidia 提供的刚体仿真和训练框架。

2.2.1 下载

从 Nvidia 官网下载 <https://developer.nvidia.com/isaac-gym>。

2.2.2 安装

解压后进入 `isaacgym/python` 文件夹，执行以下命令安装：

代码块

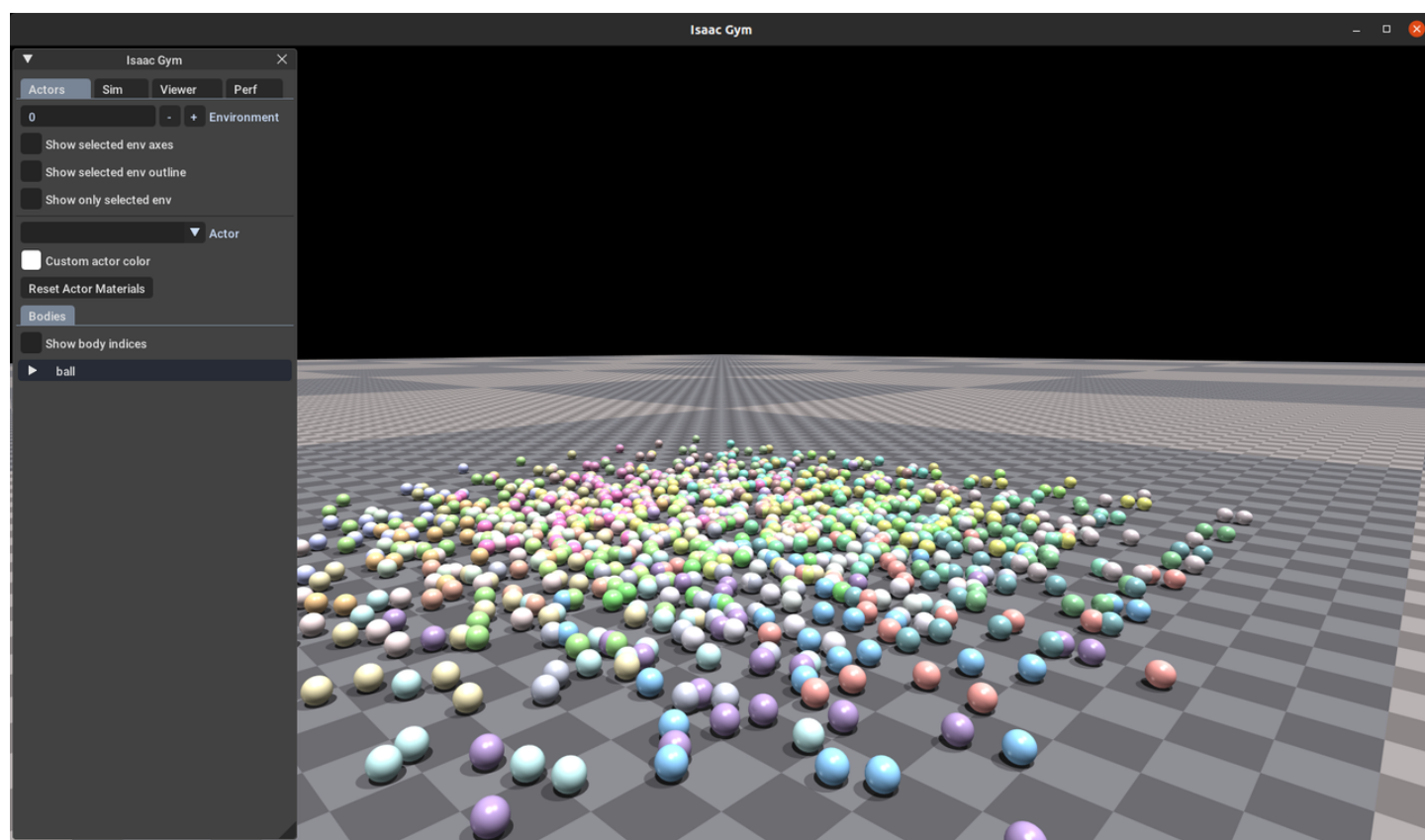
```
1 cd isaacgym/python
2 pip install -e .
```

2.2.3 验证安装

运行以下命令，若弹出窗口并显示 1080 个球下落，则安装成功：

代码块

```
1 cd examples
2 python 1080_balls_of_solitude.py
```



如有问题，可参考 `isaacgym/docs/index.html` 中的官方文档。

2.3 安装 rsl_rl

`rsl_rl` 是一个强化学习算法库。

2.3.1 下载

通过 Git 克隆仓库：

代码块

```
1 git clone https://github.com/leggedrobotics/rsl_rl.git
```

2.3.2 切换分支

切换到 v1.0.2 分支：

代码块

```
1 cd rsl_rl
2 git checkout v1.0.2
```

2.3.3 安装

代码块

```
1 pip install -e .
```

2.4 安装 unitree_rl_gym

2.4.1 下载

通过 Git 克隆仓库：

代码块

```
1 git clone https://github.com/unitreerobotics/unitree_rl_gym.git
```

2.4.2 安装

进入目录并安装：

代码块

```
1 cd unitree_rl_gym
2 pip install -e .
```

2.5 安装 unitree_sdk2py（可选）

`unitree_sdk2py` 是用于与真实机器人通信的库。如果需要将训练的模型部署到物理机器人上运行，可以安装此库。

2.5.1 下载

通过 Git 克隆仓库：

代码块

```
1 git clone https://github.com/unitreerobotics/unitree_sdk2_python.git
```

2.5.2 安装

进入目录并安装：

代码块

```
1 cd unitree_sdk2_python
2 pip install -e .
```

总结

按照上述步骤完成后，您已经准备好在虚拟环境中运行相关程序。若遇到问题，请参考各组件的官方文档或检查依赖安装是否正确。

模型训练 Train

强化学习实现运动控制的基本流程为：

`Train` → `Play` → `Sim2Sim` → `Sim2Real`

- Train: 通过 Gym 仿真环境，让机器人与环境互动，找到最满足奖励设计的策略。通常不推荐实时查看效果，以免降低训练效率。
- Play: 通过 Play 命令查看训练后的策略效果，确保策略符合预期。
- Sim2Sim: 将 Gym 训练完成的策略部署到其他仿真器，避免策略小众于 Gym 特性。
- Sim2Real: 将策略部署到实物机器人，实现运动控制。

训练命令

运行以下命令进行训练：

代码块

```
1 python legged_gym/scripts/train.py --task=xxx
```

参数说明

- `--task` : 必选参数, 值可选(go2, g1, h1, h1_2)
- `--headless` : 默认启动图形界面, 设为 true 时不渲染图形界面 (效率更高)
- `--resume` : 从日志中选择 checkpoint 继续训练
- `--experiment_name` : 运行/加载的 experiment 名称
- `--run_name` : 运行/加载的 run 名称
- `--load_run` : 加载运行的名称, 默认加载最后一次运行
- `--checkpoint` : checkpoint 编号, 默认加载最新一次文件
- `--num_envs` : 并行训练的环境个数
- `--seed` : 随机种子
- `--max_iterations` : 训练的最大迭代次数
- `--sim_device` : 仿真计算设备, 指定 CPU 为 `--sim_device=cpu`
- `--rl_device` : 强化学习计算设备, 指定 CPU 为 `--rl_device=cpu`

默认保存训练结果:

```
logs/<experiment_name>/<date_time>_<run_name>/model_<iteration>.pt
```

效果演示 Play

如果想要在 Gym 中查看训练效果, 可以运行以下命令:

代码块

```
1 python legged_gym/scripts/play.py --task=xxx
```

参数说明

- Play 启动参数与 Train 相同。
- 默认加载实验文件夹上次运行的最后一个模型。
- 可通过 `load_run` 和 `checkpoint` 指定其他模型。

导出网络

Play 会导出 Actor 网络, 保存于 `logs/{experiment_name}/exported/policies` 中:

- 需要注意: 官方 play.py 代码中默认导出模型的名字都为 `policy_lstm_1.pt`, 你可以自己命名更具辨识度的网络名称

- 如果你后续想在Gazebo中部署强化学习模型，则要导出 ONNX 格式模型，此时需要修改 play.py 文件，参考如下：

代码块

```
1
2  import os
3  import sys
4  from legged_gym import LEGGED_GYM_ROOT_DIR
5
6  import isaacgym
7  from legged_gym.envs import *
8  from legged_gym.utils import get_args, export_policy_as_jit, task_registry,
  Logger
9
10 import numpy as np
11 import torch
12 import copy
13
14
15 def export_policy_as_onnx(actor_critic, path, obs_shape):
16     """Export policy as ONNX format compatible with MNN"""
17     os.makedirs(path, exist_ok=True)
18
19     if hasattr(actor_critic, 'memory_a'):
20         # For LSTM models, we need to handle the recurrent structure
21         print("Warning: ONNX export for LSTM models is not fully supported
yet")
22         return
23
24     # Create a dummy input for tracing
25     dummy_input = torch.randn(1, obs_shape, dtype=torch.float32)
26
27     # Get the actor network
28     model = copy.deepcopy(actor_critic.actor).to('cpu')
29     model.eval()
30
31     # Export to ONNX with MNN-compatible settings
32     onnx_path = os.path.join(path, 'policy_1.onnx')
33     torch.onnx.export(
34         model,
35         dummy_input,
36         onnx_path,
37         export_params=True,
38         opset_version=9, # Use opset 9 for better MNN compatibility
39         do_constant_folding=True,
40         input_names=['input'],
```



```

41     output_names=['output'],
42     dynamic_axes={
43         'input': {0: 'batch_size'},
44         'output': {0: 'batch_size'}
45     },
46     verbose=False,
47     keep_initializers_as_inputs=False, # Important for MNN
48     operator_export_type=torch.onnx.OperatorExportTypes.ONNX # Use ONNX
operators only
49 )
50 print(f'Exported policy as ONNX (MNN-compatible) to: {onnx_path}')
51
52 # Optional: Print model info for verification
53 try:
54     import onnx
55     onnx_model = onnx.load(onnx_path)
56     print('ONNX model info:')
57     print(f' - IR version: {onnx_model.ir_version}')
58     print(f' - Opset version: {onnx_model.opset_import[0].version}')
59
60     # Print input shapes
61     input_shapes = {}
62     for input_info in onnx_model.graph.input:
63         shape = [dim.dim_value for dim in
input_info.type.tensor_type.shape.dim]
64         input_shapes[input_info.name] = shape
65     print(f' - Input shapes: {input_shapes}')
66
67     # Print output shapes
68     output_shapes = {}
69     for output_info in onnx_model.graph.output:
70         shape = [dim.dim_value for dim in
output_info.type.tensor_type.shape.dim]
71         output_shapes[output_info.name] = shape
72     print(f' - Output shapes: {output_shapes}')
73
74 except ImportError:
75     print("Note: onnx package not available for model verification")
76 except Exception as e:
77     print(f"Warning: Could not verify ONNX model: {e}")
78
79
80 def play(args):
81     env_cfg, train_cfg = task_registry.get_cfgs(name=args.task)
82     # override some parameters for testing
83     env_cfg.env.num_envs = min(env_cfg.env.num_envs, 100)
84     env_cfg.terrain.num_rows = 5

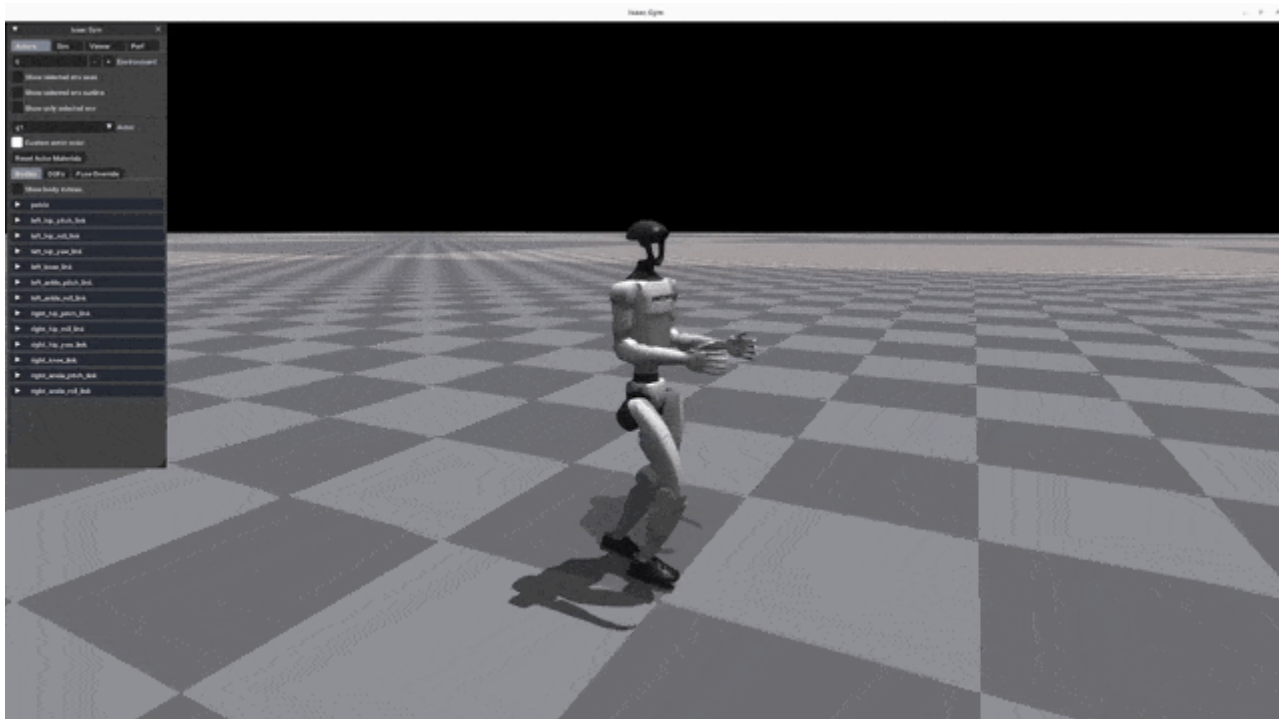
```

```

85     env_cfg.terrain.num_cols = 5
86     env_cfg.terrain.curriculum = False
87     env_cfg.noise.add_noise = False
88     env_cfg.domain_rand.randomize_friction = False
89     env_cfg.domain_rand.push_robots = False
90
91     env_cfg.env.test = True
92
93     # prepare environment
94     env, _ = task_registry.make_env(name=args.task, args=args, env_cfg=env_cfg)
95     obs = env.get_observations()
96     # load policy
97     train_cfg.runner.resume = True
98     ppo_runner, train_cfg = task_registry.make_alg_runner(env=env,
99     name=args.task, args=args, train_cfg=train_cfg)
100     policy = ppo_runner.get_inference_policy(device=env.device)
101
102     # export policy as a jit module (used to run it from C++)
103     if EXPORT_POLICY:
104         path = os.path.join(LEGGED_GYM_ROOT_DIR, 'logs',
105         train_cfg.runner.experiment_name, 'exported', 'policies')
106         export_policy_as_jit(ppo_runner.alg.actor_critic, path)
107         print('Exported policy as jit script to: ', path)
108
109     # export policy as ONNX
110     if EXPORT_ONNX:
111         export_policy_as_onnx(ppo_runner.alg.actor_critic, path,
112         obs.shape[-1])
113
114     for i in range(10*int(env.max_episode_length)):
115         actions = policy(obs.detach())
116         obs, _, rews, dones, infos = env.step(actions.detach())
117
118 if __name__ == '__main__':
119     EXPORT_POLICY = True
120     EXPORT_ONNX = True
121     RECORD_FRAMES = False
122     MOVE_CAMERA = False
123     args = get_args()
124     play(args)

```

Play 效果



模型验证 Sim2Sim (Mujoco)

支持在 Mujoco 仿真器中运行 Sim2Sim:

代码块

```
1 python deploy/deploy_mujoco/deploy_mujoco.py {config_name}
```

参数说明

- `config_name`: 配置文件, 默认查询路径为 `deploy/deploy_mujoco/configs/`

示例代码

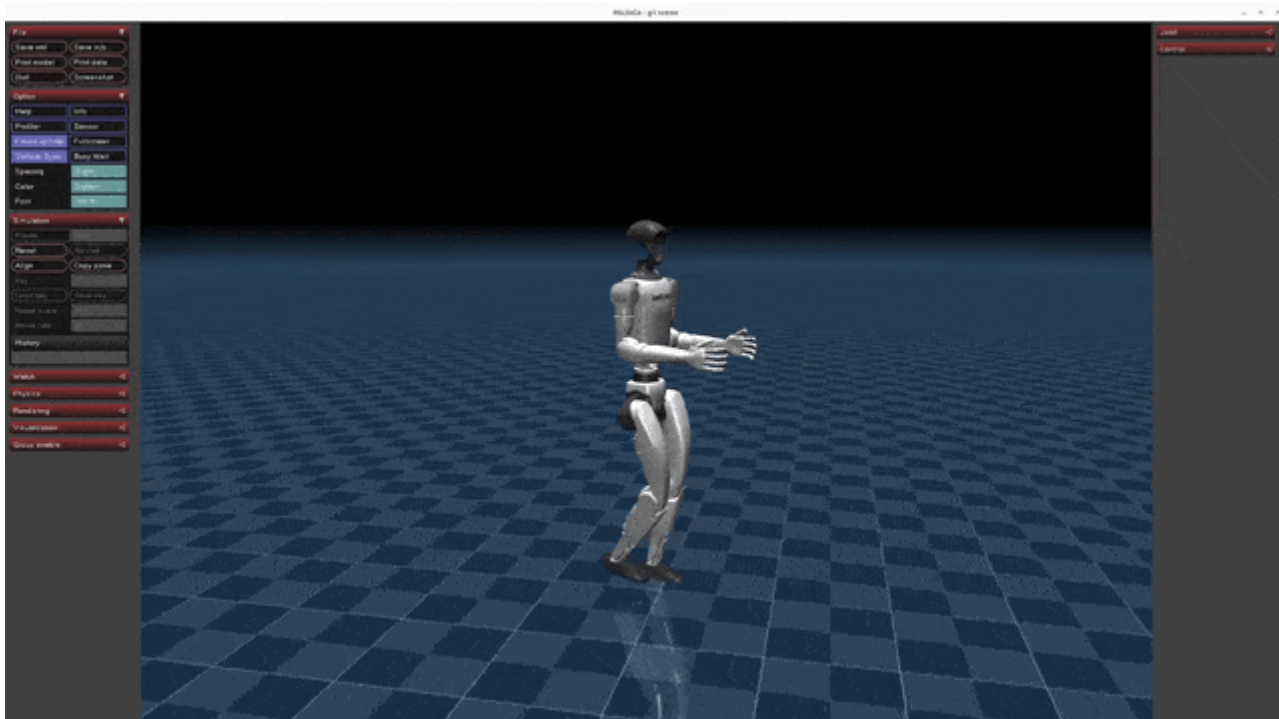
代码块

```
1 python deploy/deploy_mujoco/deploy_mujoco.py g1.yaml
```

替换模型

默认模型位于 `deploy/pre_train/{robot}/motion.pt`; 自己训练模型保存于 `logs/g1/exported/policies/policy_lstm_1.pt`, 只需替换 yaml 配置文件中 `policy_path`。

运行效果



参考资料

参考宇树开源项目：https://github.com/unitreerobotics/unitree_rl_gym/tree/main

作业任务

1. 环境搭建与验证

- 按官方指南配置 Ubuntu 20.04 环境，安装 NVIDIA 驱动 (≥ 525 ，推荐 535)、Conda 虚拟环境及依赖库 (PyTorch、Isaac Gym、rsl_rl、unitree_rl_gym 等)。
- 验证环境：运行 Isaac Gym 示例 `1080_balls_of_solitude.py` 和 `play.py` 预训练模型，确认仿真正常。

2. 模型训练与监控

- 默认参数训练：用 `python legged_gym/scripts/train.py --task=g1` 训练 G1，通过 TensorBoard 记录奖励值等指标曲线。
- 修改 Reward 参数训练：调整奖励函数参数，重新训练并保存日志，对比与默认训练的指标差异。

3. 效果验证与分析

- Isaac Gym 验证：用 `play.py` 分别运行两个模型，观察行走流畅性、稳定性，记录异常（如摔倒）。

- Mujoco 验证：替换 `g1.yaml` 中模型路径，运行 `deploy_mujoco.py`，对比跨仿真平台的性能差异。
- 分析总结：结合指标曲线和仿真效果，说明 Reward 参数对训练效率和机器人行为的影响。

4. 模型部署至 Gazebo

- 将play保存的.onnx网络转换为.mnn，方式如下：

代码块

```
1 cd src/unitree_guide/thirdParty/MNN
2 mkdir build && cd build
3 cmake .. -DMNN_BUILD_CONVERTER=ON
4 make -j4
5
6 #执行
7 ./MNNConvert -f ONNX --modelFile model.onnx --MNNModel model.mnn --bizCode MNN
```

- 将训练好的Policy（神经网络模型）部署到 Gazebo，解决部署中的问题。
- 验证行走效果，对比与其他仿真环境的差异。

作业提交

- 操作文档：记录关键指令、问题及解决方案。
- 分析报告：含训练指标对比表、曲线分析、多平台性能差异总结。
- 模型文件：默认参数和修改参数的训练模型。
- 演示视频：G1 在 Mujoco 和 Gazebo 中运行的清晰视频（ ≥ 30 秒）。