

# 研究报告

题 目：足式机器人强化学习控制调研

研 究 生 彭道杰

## 摘要

足式机器人相对于常见的轮式和履带式机器人有着更加灵活，环境适应能力强的优势。与启发它们的人类或者动物一样，这类机器人拥有着更强的越野能力，能够适应复杂的地形环境，到达普通轮式或履带式机器人无法到达的地方。然而，这类机器人往往具有更多的自由度，这使得对它们的控制变得具有挑战性。如何让这类足式机器人像它们的动物模仿对象一样能够灵活、优雅地运动成为一个重要的研究课题。在这其中，模仿四足动物的四足机器人（机械狗）是一类被广泛研究的对象。

机械狗是一个多自由度运动系统，其典型自由度数量为 12 个。每条腿有三个自由度分别实现小腿伸缩、大腿伸缩和整腿伸缩。由于它的求解空间巨大而可用的解空间相对较小，因此机械狗的运动控制常常是启发式的——预先定义一些运动，以缩小求解时的搜索空间。经典的基于运动动力学建模和优化问题的控制方式主要有轨迹优化 (*Trajectory Optimization, TO*) 和模型预测控制 (*Model Predictive Control, MPC*)。由于求解需要涉及大量的非线性优化问题，这类控制方法求解算力开销较大往往不能够在线完成，需要上位机的辅助。相比较而言近年来发展起来的基于深度学习的强化学习控制方法在运行期间的算力开销很小。强化学习控制是一种通过构建神经网络并依靠大量训练优化神经网络参数以得到预期的控制效果的控制方法，可以完全不需要动力学建模。其训练过程往往在仿真环境中完成，因此需要对机械狗的物理模型进行建模，这些建模包括：机械狗的躯干及四肢的刚体模型、电机驱动关节的响应模型。

不过由于神经网络的黑盒性，单纯的强化学习控制往往只能针对特定的训练目标而很难与其它模型融合，缺乏跨模式泛化能力。因此经典控制和强化学习控制的融合控制变得十分重要。

**关键词：**机器人控制；轨迹优化；深度学习；强化学习控制；Isaac 仿真

## 目 录

摘要	I
第1章 基于质心动力学 (Centroidal Dynamics) 模型的 TO 控制	1
1.1 机械狗的运动模型 (Motion Model)	1
1.2 分层次优化 (Hierarchical Optimization) 计算	3
1.2.1 基体运动方程	3
1.2.2 地面接触部分运动约束	3
1.2.3 接触力限制	4
1.2.4 扭矩限制	4
1.2.5 目标运动跟随控制	4
1.2.6 接触力最小化	5
1.2.7 根据优化结果计算电机扭矩	5
1.3 质心运动优化 (Motion Optimization)	5
1.3.1 足态保持生成	5
1.3.2 支撑多边形序列生成	6
1.3.3 质心运动优化问题描述	6
1.3.4 规划初始化	7
1.4 质心运动优化 QP 问题的建立	8
1.4.1 成本函数	8
1.4.2 等式约束	10
1.4.3 不等式约束	11
1.4.4 约束松弛	11
第2章 基于直接刚体动力学模型的 TO 控制	13
2.1 基于直接刚体动力学模型的 TO 控制	13
第3章 基于 MPC 的控制	14
3.1 刚体系统的 MPC 问题描述	14
3.1.1 系统模型	14
3.1.2 接触模型	15
3.2 NMPC 方法	15
3.2.1 iLQR-NMPC 算法	16

3.2.2 GNMS-NMPC 算法 .....	17
<b>第 4 章 基于强化学习的机械狗控制</b> .....	<b>18</b>
4.1 强化学习控制 .....	18
4.1.1 强化学习的基本概念 .....	18
4.1.2 基于深度强化学习的控制器的部署和优化流程 .....	18
4.2 单层神经网络直接驱动关节方式 .....	19
4.3 双层神经网络本体感知方式 .....	20
4.3.1 总体概况 .....	20
4.3.2 策略训练 .....	21
4.3.3 地形定义 .....	22
4.3.4 控制架构 .....	22
4.4 双层网络本体和外部感知融合方式 .....	23
4.4.1 问题描述 .....	25
4.4.2 教师策略训练 .....	25
4.4.3 观测和行动 .....	26
4.4.4 策略构架 .....	27
4.4.5 奖励函数 .....	28
4.4.6 课程 .....	31
4.4.7 学生策略训练 .....	31
4.4.8 高度采样随机化 .....	32
4.4.9 信念状态寄存器 .....	33
4.4.10 部署 .....	34
4.5 电机驱动关节点模型 .....	34
<b>第 5 章 强化学习仿真平台</b> .....	<b>35</b>
5.1 关于各种机器人仿真平台 .....	35
5.2 强化学习平台 Isaac .....	35
5.2.1 关于 Isaac Gym .....	35
5.2.2 在 Isaac 上建立仿真 .....	36
5.2.3 张量 API .....	36
5.3 物理模拟器 .....	39
<b>第 6 章 强化学习案例 legged_gym 库</b> .....	<b>41</b>
6.1 legged_gym 工程文件结构 .....	41
6.1.1 legged_gym 工具包 .....	42

6.2 LeggedRobot 类 .....	42
6.2.1 从 Isaac 仿真环境中获取信息 .....	43
6.2.2 LeggedRobot.step 函数 .....	43
6.2.3 self.render 函数 .....	44
6.2.4 self.compute_torques 函数 .....	45
6.2.5 self.post_physics_step 函数 .....	45
6.2.6 self.compute_observations 函数 .....	46
6.2.7 self.compute_reward 函数 .....	46
第 7 章 基于 Isaac 平台的深度学习训练实践 .....	49
7.1 用 legged_gym 例程库训练 ANYmal-c .....	49
7.1.1 训练脚本设置 .....	49
7.1.2 训练过程信息 .....	49
7.1.3 训练结果输出和重演测试 .....	50
第 8 章 几种构想的机器人类型草图 .....	53
8.1 弹跳轮足 + 机械手机器人 .....	53
8.2 轮鞋足式 + 机械手机器人 .....	54
8.3 弹跳轮足 + 飞行器机器人 .....	54
第 9 章 讨论 .....	56
结 论 .....	57
参考文献 .....	58

# 第1章 基于质心动力学(Centroidal Dynamics)模型的TO控制

足式机器人的运动规划是比较困难的，不仅因为它的自由度较多，更因为它的机体运动不能被直接得出，而是要通过四肢状态及四肢与环境的接触产生。

经典控制的基础是对机器人的运动学和动力学建模。经典控制主要方式有轨迹优化(*Trajectory Optimization, TO*)和模型预测控制(*Model Predictive Control, MPC*)两类。常用的轨迹优化机械狗动力模型有两种<sup>[1]p2</sup>：1. 基于质心动力学的模型；2. 基于直接刚体动力学的模型。基于这些模型的优化控制都被描述为非线性优化问题，这些问题的求解算力开销较大往往不能够在线完成，需要上位机的辅助。

## 1.1 机械狗的运动模型(Motion Model)<sup>[2-3]p2</sup>

机器人与环境接触的机械系统的运动模型描述方程可以描述如下：

$$\mathbf{M}(\mathbf{q})\dot{\mathbf{u}} + \mathbf{h}(\mathbf{q}, \mathbf{u}) = \mathbf{S}^T \boldsymbol{\tau} + \mathbf{J}_S^T \boldsymbol{\lambda} \quad (1-1)$$

这其中  $\mathbf{q}$  是一个描述机器人主体及各个节点的广义位置矢量：

$$\mathbf{q} = \begin{bmatrix} {}_I\mathbf{r}_{IB} \\ \mathbf{q}_{IB} \\ \mathbf{q}_j \end{bmatrix} \in SE(3) \times \mathbb{R}^{n_j} \quad (1-2)$$

它里面  ${}_I\mathbf{r}_{IB} \in \mathbb{R}^3$  是机器人主体相对于惯性系的三维位置矢量； $\mathbf{q}_{IB} \in SO(3)$  是机器人主体相对于惯性系的转动描述，用哈密顿单位四元数表示的； $\mathbf{q}_j \in \mathbb{R}^{n_j}$  是一个储存机器人所有节点角度的  $n_j$  维矢量。

这其中  $\mathbf{u}$  是一个描述机器人主体及各个节点的广义速度矢量：

$$\mathbf{u} = \begin{bmatrix} {}_I\mathbf{v}_B \\ {}_B\boldsymbol{\omega}_{IB} \\ \dot{\mathbf{q}}_j \end{bmatrix} \in \mathbb{R}^{n_u} \quad (1-3)$$

它里面的  ${}_I\mathbf{v}_B$  描述了机器人主体相对于惯性系的速度； ${}_B\boldsymbol{\omega}_{IB}$  描述了机器人主体相对于自身的角速度； $\dot{\mathbf{q}}_j$  描述了机器人的各个关节转动的速度。这其中  $\mathbf{M}$  是一

一个关于机器人整体的质量矩阵，它是一个  $n_u \times n_u$  的矩阵：

$$\mathbf{M} \in \mathbb{R}^{n_u \times n_u} \quad (1-4)$$

这个质量矩阵的具体数值跟机器人的机械系统的状态(各个节点的位置  $\mathbf{q}$ )相关，可以通过通用的方式计算出它的表达式。实际计算的时候只需要带入  $\mathbf{q}$  的值，就可以计算出  $\mathbf{M}$  矩阵的各个元素具体数值。这其中  $\mathbf{h}$  是一个跟机器人的机械位置和速度都有关的量，包含了机械系统产生的克里奥利力、离心力和重力的作用，它是一个  $n_u$  维的矢量：

$$\mathbf{h} \in \mathbb{R}^{n_u} \quad (1-5)$$

这其中  $\mathbf{S}$  是一个选择矩阵，可以用来选择整个公式中哪些自由度被激活，它是一个  $n_\tau \times n_u$  的矩阵：

$$\mathbf{S} = \begin{bmatrix} 0_{n_\tau \times (n_u - n_\tau)} & \mathbb{I}_{n_\tau \times n_\tau} \end{bmatrix} \in \mathbb{R}^{n_\tau \times n_u} \quad (1-6)$$

它其中包含的参数  $n_\tau$  表示被激活的自由度数量，如果机器人的所有自由度都被激活，则  $n_\tau = n_j$ 。这其中  $\boldsymbol{\tau}$  是机器人各个关节的电机提供的扭矩，它是一个  $n_j$  维的向量：

$$\boldsymbol{\tau} \in \mathbb{R}^{n_j} \quad (1-7)$$

它的节点成员是否产生作用受到  $\mathbf{S}^T$  矩阵的选择。这其中  $\mathbf{J}_s$  是一些列雅可比矩阵的集合：

$$\mathbf{J}_s = \left[ \mathbf{J}_{c_1}^T \ \dots \ \mathbf{J}_{c_{n_c}}^T \right]^T \in \mathbb{R}^{3n_c \times n_u} \quad (1-8)$$

它是接触点的支撑力  $\boldsymbol{\lambda}$  向节点力转换的矩阵雅可比矩阵，包含了  $n_c$  个雅可比矩阵， $n_c$  为接触地面的肢体个数。

如果足接触建模为点接触的话，在稳定接触的情况下，每个接触点会产生三个相应的约束条件：

$$\begin{cases} {}_I\mathbf{r}_{IC}(t) = const \\ {}_I\dot{\mathbf{r}}_{IC} = \mathbf{J}_C \mathbf{u} = 0 \\ {}_I\ddot{\mathbf{r}}_{IC} = \mathbf{J}_C \dot{\mathbf{u}} + \mathbf{J}_C \mathbf{u} = 0 \end{cases} \quad (1-9)$$

其含义就是在惯性系下看接触点的位置是固定的为定值，并且其速度和加速，也即其一阶导数和二阶导数为零。

## 1.2 分层次优化(Hierarchical Optimization)计算<sup>[2]p2</sup>

整个优化计算过程的最终目的是得到一个扭矩参考目标  $\xi_d$  用来作为输入控制电机,  $\xi_d$  包含两部分信息: 关节点的运动加速度  $\mathbf{u}_d^T$  和接触力  $\boldsymbol{\lambda}_d^T$ 。控制方法采用接触力控制, 优化计算的策略是采取分层次优化的方式。在这一部分我们将从第1.1节中描述的运动模型出发, 拆接触其中的内容进行分层次优化。我们将介绍运动方程、接触部分的运动约束、接触力和扭矩限制、运动跟随、接触力最小化、根据优化结果计算电机扭矩等内容。

### 1.2.1 基体运动方程

通过利用选择矩阵  $\mathbf{S}^T$  诱导的分解, 我们可以将运动和接触力限制在浮动基系统动力学描述的流形上, 有如下式:

$$\begin{bmatrix} \mathbf{M}_{fb} & -\mathbf{J}_{sfb}^T \end{bmatrix} \xi_d = -\mathbf{h}_{fb} \quad (1-10)$$

这里  $XXX_{fb}$ : 下标的意思是浮动的主体-'floating base', 各个参数的具体含义解释如下:

- (1)  $\xi_d$  是一个一个  $n_u + n_c$  行 1 列的向量:  $\xi_d = [\mathbf{u}_d^T \ \boldsymbol{\lambda}_d^T] \in \mathbb{R}^{n_u+n_c}$ , 其中:
  - ①  $\mathbf{u}_d^T$  是目标关节点的加速度;
  - ②  $\boldsymbol{\lambda}_d^T$  是目标接触力;
- (2)  $\mathbf{M}_{fb}$  是复合型惯性矩阵的前六行;
- (3)  $\mathbf{J}_{sfb}^T$  是雅克比阵的前六行, 它将接触力转换到节点的扭矩;
- (4)  $\mathbf{h}_{fb}$  是非线性项的前六行, 包括克里奥利力、离心力和重力项;

### 1.2.2 地面接触部分运动约束

控制器找到的解决方案不应违反(1-9)中定义的接触约束。因此, 我们通过设置在接触点施加空加速度:

$$\begin{bmatrix} \mathbf{J}_s & 0_{3n_c \times 3n_c} \end{bmatrix} \xi_d = -\mathbf{j}_s \mathbf{u} \quad (1-11)$$

它将  $\xi_d$  带进去计算后就得到了:  $\mathbf{J}_s \xi_d = -\mathbf{j}_s \mathbf{u}$  其实就是公式(1-9)的第二个式子  $\mathbf{J}_s \xi_d + \mathbf{j}_s \mathbf{u} = 0$ 。

### 1.2.3 接触力限制

$$\begin{cases} ({_I}\mathbf{h} - {_I}\mathbf{n}_\mu)^T {_I}\boldsymbol{\lambda}_k \leq 0 \\ -({_I}\mathbf{h} + {_I}\mathbf{n}_\mu)^T {_I}\boldsymbol{\lambda}_k \leq 0 \\ ({_I}\mathbf{l} - {_I}\mathbf{n}_\mu)^T {_I}\boldsymbol{\lambda}_k \leq 0 \\ -({_I}\mathbf{l} + {_I}\mathbf{n}_\mu)^T {_I}\boldsymbol{\lambda}_k \leq 0 \end{cases} \quad (1-12)$$

${}_I\mathbf{n}$  是接触面的法向量;  $\mu$  是摩擦系数; 有了这两个再乘上受力  ${}_I\lambda$ , 就可以得出最大静摩擦力  ${}_I\mathbf{n}_\mu^T \boldsymbol{\lambda}$ 。实际在接触点平行于地面方向的两个分力  ${}_I\mathbf{h}$ ,  ${}_I\mathbf{l}$  在正反方向上都不应该比这个值大, 否则就会发生滑动。这就是公式(1-12)约束的由来。

### 1.2.4 扭矩限制

对于扭矩有如下约束:

$$\boldsymbol{\tau}_{min} - \mathbf{h}_j \leq [\mathbf{M}_j \ -\mathbf{J}_{s_j}^T] \leq \boldsymbol{\tau}_{max} - \mathbf{h}_j \quad (1-13)$$

这里的  $\mathbf{h}_j$  就是科里奥利力那一堆东西。而  $[\mathbf{M}_j \ -\mathbf{J}_{s_j}^T]$  就是加速度力  $\mathbf{M}_j \dot{\mathbf{u}}_d^T$  和传递力  $-\mathbf{J}_{s_j}^T \boldsymbol{\lambda}_d^T$  两项的和。它们计算的结果就是电机提供的扭矩力  $\boldsymbol{\tau}$  克服完  $\mathbf{h}(\mathbf{q}, \mathbf{u})$  剩下的力。

### 1.2.5 目标运动跟随控制

为了能跟随浮动主体和摆动腿的目标运动。我们通过实现具有前馈参考加速度和运动相关状态反馈状态的操作空间控制器来约束关节加速度。对于主体的线性运动:

$$[{}_C\mathbf{J}_P \ 0] \boldsymbol{\xi}_d = {}_C \ddot{\mathbf{r}}_{IB}^d + \mathbf{k}_D^{pos}({}_C \dot{\mathbf{r}}_{IB}^d - {}_C \mathbf{v}_B) + \mathbf{k}_P^{pos}({}_C \mathbf{r}_{IB}^d - {}_C \mathbf{r}_B) \quad (1-14)$$

对于主体的角度运动:

$$[{}_C\mathbf{J}_R \ 0] \boldsymbol{\xi}_d = -\mathbf{k}_D^{ang} {}_C \boldsymbol{\omega}_B + \mathbf{k}_P^{ang} (\mathbf{q}_{CB}^d \Xi \mathbf{q}_{CB}) \quad (1-15)$$

具体参数解释如下:

- 雅可比矩阵  ${}_C\mathbf{J}_P$  和  ${}_C\mathbf{J}_R$  是与控制坐标系  $C$  (这是一个与地形局部估计和机器人航向方向对齐的帧) 中表达的基相关的平移和旋转雅可比矩阵。

- $\Xi$  这个算子产生欧拉向量, 表示期望姿态  $q_{CB}^d$  和估计姿态  $q_{CB}$  之间的相对方向。

- 这里面  $\mathbf{k}_P^{pos}, \mathbf{k}_D^{pos}, \mathbf{k}_P^{ang}, \mathbf{k}_D^{ang}$  是用来控制增益的对角正定矩阵。

- 参考的运动  ${}_C\mathbf{r}_{IB}$  和它的导数是运动规划的结果。

### 1.2.6 接触力最小化

可以通过下面的方法将接触力设置为最小值：

$$\begin{bmatrix} 0_{3n_c \times n_u} & \mathbb{I}_{3n_c \times 3n_c} \end{bmatrix} \xi_d = 0 \quad (1-16)$$

这个式子的含义是让每个接触力的目标值都为零。

### 1.2.7 根据优化结果计算电机扭矩

如果给定了一个优化的关节点运动和接触力， $\xi_d = [\dot{\mathbf{u}}_d^T \ \lambda_d^T]^T$  我们可以用以下公式计算各个电机的扭矩：

$$\tau_d = [\mathbf{M}_j \ -\mathbf{J}_{s_j}^T] \xi_d + \mathbf{h}_j$$

其中  $\mathbf{M}_j$ ,  $-\mathbf{J}_{s_j}^T$ ,  $\mathbf{h}_j$  在公式(1-13)已中定义。

因此，所有规划的目的就是给出关节点的运动加速度  $\dot{\mathbf{u}}_d^T$  和接触力  $\lambda_d^T$  目标，也即  $\xi_d^*$ ，从它出发直接计算得到控制各个关节点电机的扭矩目标。

## 1.3 质心运动优化 (Motion Optimization)<sup>[2]p3</sup>

在机械狗的运动模型描述第1.1节部分我们定义了一些关键参数，拥有了对机械狗运动的描述；在分层次优化第1.2节部分我们分解了运动方程的成分并给出了一些硬性约束条件和底层控制方法，知道了如何从关节点的运动加速度  $\dot{\mathbf{u}}_d^T$  的目标值和接触力  $\lambda_d^T$  的目标值这两个变量得到对电机的扭矩  $\tau_d$  的控制目标值。下面就要进行实际的运动优化问题描述和求解，来计算出关节点的运动加速度  $\dot{\mathbf{u}}_d^T$  的目标值和接触力  $\lambda_d^T$  的目标值。

整个机器人的控制遵循一下流程：首先由预定义的方式设定机器人应该遵循的步态第1.3.1节；为了保持机器人的稳定性，这些步态的生成需要满足ZMP约束，也即机械狗的步态是由一些列的支撑多边形序列具体定义的第1.3.2节；在机器人获得稳定支撑的基础上，我们根据外部的指令对机器人的具体运动方式调整以获得指令要求的目标运动，这被描述为一个运动优化问题第1.3.3；在优化前首先要对初始状态进行规划初始化第1.3.4节；优化过程采用质心运动优化的方式进行，这是一个二次优化问题，它的具体建立在第1.4节给出。

### 1.3.1 足态保持生成

外部高级速度命令用于通过调整参考立足点以特定方向和速度驱动运动，这些立足点是针对每个新控制回路的每条腿计算的。根据腿的接触状态，这些是通过两种不同的方式计算的：当腿接触时，命令速度将被投影到计算的立足点位置，

以便平均躯干速度与所需的运动速度相匹配；当腿摆动时，参考立足点以相同的方式计算，但添加了一个速度反馈项，该项用来在机器人受到会导致速度控制误差的外部推力时稳定机器人。相关细节参考文献<sup>[4]</sup>。

这块儿随后要看看，补充到下面...

### 1.3.2 支撑多边形序列生成

我们在相域中 (phase domain) 为每条腿定义升降事件  $\phi_{lo}$  和触下事件  $\phi_{td}$ 。这也定义了所有腿的接触时间表。给定所有腿的触下和升降事件，以及当前脚位置和所需立足点的集合，我们可以计算一系列支持多边形(定义为顶点元组和以秒为单位的时间持续时间)用于运动规划器。

当一个新的运动计划可用时，我们都会执行这样的操作。这样新的解决方案就可以适应接触计划的变化、参考立足点的变化以及高级操作员速度的变化。

为了计算每个多边形，我们从当前阶段  $\phi$  开始，并存储接触的脚的  $x - y$  坐标。我们搜索  $k = 1$  的下一阶段事件  $\phi_k$  以获得第一个多边形  $t_0 = t_s(\phi_k - \phi)$  的持续时间， $t_s$  是以秒为单位的步幅持续时间。这样，我们就通过顶点及其持续时间在几秒钟内完全定义了一个支持多边形。我们通过将  $\phi$  更新为  $\phi_k$  并搜索下一阶段相位事件来不断迭代。由于接触计划是周期性的，我们重复这些步骤，直到相位事件  $\phi_0 + 1$ ，对应于起始相位  $\phi_0$ 。

### 1.3.3 运动优化问题描述<sup>[3]p3</sup>

与文献中类似，每一个坐标方向的质心运动规划都被描述成一个系列的五次样条。比如沿着  $x$  方向的其中第  $i-th$  曲线可以描述为：

$$\begin{aligned} x(t) &= a_{i5}^x t^5 + a_{i4}^x t^4 + a_{i3}^x t^3 + a_{i2}^x t^2 + a_{i1}^x t + a_{i0}^x \\ &= [t^5 \quad t^4 \quad t^3 \quad t^2 \quad t \quad 1] \\ &\quad \cdot [a_{i5} \quad a_{i4} \quad a_{i3} \quad a_{i2} \quad a_{i1} \quad a_{i0}] \\ &= \boldsymbol{\eta}^T(t) \boldsymbol{\alpha}_i^x \end{aligned} \tag{1-17}$$

这其中  $t \in [t, t + \Delta t_i]$  是第  $i$  个曲线段前所有  $(i-1)$  个曲线持续时间长度的总和， $\Delta t_i$  是第  $i$  个曲线持续的时长。基于(1-17)的关于位置的表述，可以很容易地将关于速度和加速度的表述写出来：

$$\dot{x}(t) = \dot{\boldsymbol{\eta}}^T(t) \boldsymbol{\alpha}_i^x \tag{1-18}$$

$$\ddot{x}(t) = \ddot{\boldsymbol{\eta}}^T(t) \boldsymbol{\alpha}_i^x \tag{1-19}$$

这其中：

$$\dot{\boldsymbol{\eta}}^T(t) = \begin{bmatrix} 5t^4 & 4t^3 & 3t^2 & 2t^1 & 1 & 0 \end{bmatrix}^T \quad (1-20)$$

$$\ddot{\boldsymbol{\eta}}^T(t) = \begin{bmatrix} 20t^3 & 12t^2 & 6t^1 & 2 & 0 & 0 \end{bmatrix}^T \quad (1-21)$$

对于质心在  $y, z$  方向上的描述是一样的。每一条质心曲线都由  $x, y, z$  三部分分量  $\boldsymbol{\alpha}_i = [\boldsymbol{\alpha}_i^x \ \boldsymbol{\alpha}_i^y \ \boldsymbol{\alpha}_i^z]^T$  组成，可以将  $n_s$  条质心曲线的  $3n_s$  条曲线参数写到一起，优化的参数矢量可以写成： $\boldsymbol{\alpha} = [\boldsymbol{\alpha}_0^T \ \dots \ \boldsymbol{\alpha}_i^T \ \dots \ \boldsymbol{\alpha}_{n_s}^T]^T$ 。这样一来，质心的位置可以表示为：

$$\mathbf{p}_{CoM}(t) = \mathbf{T}(t)\boldsymbol{\alpha}_i \in \mathbb{R}^3, \quad \mathbf{T}(t) = \begin{bmatrix} \boldsymbol{\eta}^T(t) & 0 & 0 \\ 0 & \boldsymbol{\eta}^T(t) & 0 \\ 0 & 0 & \boldsymbol{\eta}^T(t) \end{bmatrix} \quad (1-22)$$

同样质心的速度和加速度也可以得到了：

$$\dot{\mathbf{p}}_{CoM}(t) = \dot{\mathbf{T}}(t)\boldsymbol{\alpha}_i \in \mathbb{R}^3 \quad (1-23)$$

$$\ddot{\mathbf{p}}_{CoM}(t) = \ddot{\mathbf{T}}(t)\boldsymbol{\alpha}_i \in \mathbb{R}^3 \quad (1-24)$$

这样一来整个优化问题就变为了通过求解二次优化问题 (*Quadratic Problem, QP*) 问题寻找优化的参数  $\boldsymbol{\alpha} = [\boldsymbol{\alpha}_0^T \ \dots \ \boldsymbol{\alpha}_i^T \ \dots \ \boldsymbol{\alpha}_{n_s}^T]^T$ 。接下来将逐步介绍这个优化问题的建立：

$$\min_{\boldsymbol{\alpha}} \quad \frac{1}{2} \boldsymbol{\alpha}^T \mathbf{Q} \boldsymbol{\alpha} + \mathbf{c}^T \boldsymbol{\alpha} \quad (1-25)$$

$$s.t. \quad \mathbf{A}\boldsymbol{\alpha} = \mathbf{b}, \quad \mathbf{D}\boldsymbol{\alpha} \leq \mathbf{f}. \quad (1-26)$$

简单起见，接下来的描述中除特殊说明外都采用  $x$  轴方向来表述代指各条曲线段  $\boldsymbol{\alpha}_i = [\boldsymbol{\alpha}_i^x \ \boldsymbol{\alpha}_i^y \ \boldsymbol{\alpha}_i^z]^T$ 。

### 1.3.4 规划初始化

运动计划使用扩展状态（位置、速度和加速度）进行初始化，该扩展状态用作样条序列初始状态的硬约束。这是通过使用一个 *alpha* 滤波器来融合前一次目标位置  $c\mathbf{r}_{IB}^{des}$  和当前测量  $c\mathbf{r}_{IB}$  到的位置实现的：

$$c\mathbf{r}_{IB}^{ref} = \alpha c\mathbf{r}_{IB}^{des} + (1 - \alpha)c\mathbf{r}_{IB} \quad (1-27)$$

其中  $\alpha = 0.5e^{-\lambda t_c}$ ，其中  $\lambda \in \mathbb{R}$  是一个用来调节前一次目标位置随其已完成时间长度  $t_c$  的权重。类似的过滤器也用于设置初始速度，而加速度使用最后可用的参考

值进行初始化。

## 1.4 质心运动优化QP问题建立

如第1.3.3节所述，质心运动优化问题是可以转化成一个多维二次优化问题，本节将具体描述这问题的建立。一个二次优化问题主要由三个部分的定义：成本函数、等式约束、不等式约束。另外，如果约束太紧可能求解不出结果，因此要进行一定的约束松弛

### 1.4.1 成本函数

在我们的问题描述中，在公式(1-25)中出现的Hessian矩阵 $\mathbf{Q} \in \mathbb{R}^{12n \times 12n}$ 和线性项 $\mathbf{c} \in \mathbb{R}^{12n}$ 都有若干组成部分。我们通过下面式子来惩罚实际位置 $\mathbf{p}(t) = \mathbf{T}(t)\boldsymbol{\alpha}$ 与 $\mathbf{p}_r$ 的偏差：

$$\begin{cases} \frac{1}{2}\|\mathbf{T}\boldsymbol{\alpha} - \mathbf{p}_r\|_W^2 \\ = \frac{1}{2}(\mathbf{T}\boldsymbol{\alpha} - \mathbf{p}_r)^T \mathbf{W}(\mathbf{T}\boldsymbol{\alpha} - \mathbf{p}_r) \\ = \frac{1}{2}\boldsymbol{\alpha}^T \mathbf{T}^T \mathbf{W} \mathbf{T} \boldsymbol{\alpha} - \mathbf{p}_r^T \mathbf{W} \mathbf{T} \boldsymbol{\alpha} + \frac{1}{2}\mathbf{p}_r^T \mathbf{W} \mathbf{p}_r \end{cases} \quad (1-28)$$

最小化公式(1-28)中的最小化平方范数相当于用公式(1-25)的形式求解QP问题：

$$\mathbf{Q} = \mathbf{T}^T \mathbf{W} \mathbf{T} \quad \mathbf{c} = -\mathbf{T}^T \mathbf{W}^T \mathbf{p}_r \quad (1-29)$$

为了惩罚速度参考的偏差，只需在公式(1-28)中使用 $\dot{\mathbf{T}}$ 和 $\dot{\mathbf{p}}_r$ 。可以对加速度偏差进行类似的推理。

一般来说为了使得控制更平滑、节能、稳定，成本函数会包括一些加速度最小化、软最终约束、过大偏差抑制等项目<sup>[2]p5</sup>。

#### 1.4.1.1 加速度最小化

如文献<sup>[5]p</sup>所示，我们可以通过写作来最小化加速度：

$$\mathbf{Q}_k^{acc} = \begin{bmatrix} (400/7)\Delta t_k^7 & 40\Delta t_k^6 & 24\Delta t_k^5 & 10\Delta t_k^4 \\ 40\Delta t_k^6 & 28.8\Delta t_k^5 & 18\Delta t_k^4 & 8\Delta t_k^3 \\ 24\Delta t_k^5 & 18\Delta t_k^4 & 12\Delta t_k^3 & 6\Delta t_k^2 & 0_{4 \times 2} \\ 10\Delta t_k^4 & 8\Delta t_k^3 & 6\Delta t_k^2 & 4\Delta t_k & 0_{4 \times 2} \\ 0_{2 \times 4} & & & & 0_{2 \times 2} \end{bmatrix} \quad (1-30)$$

此时非线性项 $\mathbf{c}_k^{acc} = 0$ 。这里请注意，如果这是添加到成本函数的唯一项，则

不会有与每个样条的  $\alpha_{1k}$  和  $\alpha_{2k}$  系数相关联的成本，从而导致正半定 Hessian 矩阵。当使用诸如 *Active Set one*<sup>[6]P</sup> 之类的方法时，这是有问题的，该方法要求 Hessian 矩阵是正定的。在这种情况下，可以添加一个正则化项：

$$\mathbf{Q}_k^{acc\rho} = \begin{bmatrix} 0_{4 \times 4} & 0_{4 \times 2} \\ 0_{2 \times 4} & \rho \mathbb{I} - 2 \times 2 \end{bmatrix} \quad (1-31)$$

其中  $\rho = 10e^{-8}$ ，线性项为空。

从文献<sup>[6]P</sup>中看得，由于两次求导， $\alpha_{1k}$ ,  $\alpha_{2k}$  将都是零。这样整个成本函数就从(1-30)简化成了(1-31)，这在对矩阵初始化的时候很有意义因为非零项会被初始化成很小的数  $\rho = 10e^{-8}$ ，而恒零项就可以直接赋值为 0。

#### 1.4.1.2 软最终约束

我们将最终位置  $\mathbf{p}_f \in \mathbb{R}^2$  设置为由计划立足点定义的多边形的中心  $\mathbf{p}_f^{ref} \in \mathbb{R}^2$ ，这是如果机器人支持多边形序列的末尾停止，它将支持机器人的多边形。为了最小化这个范数  $\|\mathbf{p}_f - \mathbf{p}_f^{ref}\|_{\mathbf{W}_f}^2 = \|\mathbf{A}_f \mathbf{s}_f - \mathbf{p}_f^{ref}\|_{\mathbf{W}_f}^2$ ，给出如下式子：

$$\mathbf{Q}_f = \mathbf{A}_f^T \mathbf{W}_f \mathbf{A} \quad \mathbf{c}_f = -\mathbf{A}^T \mathbf{W}_f \mathbf{p}_f^{ref} \quad (1-32)$$

其中  $\mathbf{W}_f \in \mathbb{R}^{2 \times 2}$  是一个正权重的对称对角矩阵。为了避免优化器将最终状态放置在远离参考位置的解决方案，我们在位置上添加不等式约束，使其被限制在以参考位置为中心的框中。

#### 1.4.1.3 偏离之前目标的解决方案

由于一旦前一个优化成功，我们就计算一个新的优化，为了避免连续运动计划之间的较大偏差，我们惩罚当前解  $\xi$  得到的位置、速度和加速度与前一个解  $\xi_{i-1}$  产生的偏差。

记  $t_{pre}$  是前一个结果给出后消逝的时间长度，我们通过下式惩罚两者的偏差：

$$\|\mathbf{p}(\bar{t}) - \mathbf{p}_{i-1}(t_{pre} + \bar{t})\|_{\mathbf{W}_f}^2, \bar{t} \in [0, t_f] \quad (1-33)$$

其中  $t_f = \sum_{k=0}^{n-1} f_{fk}$  是之前所有  $n$  段曲线持续时间的总和。我们使用样本时间  $dt$  离散化优化范围  $[0, t_f]$ 。我们采用类似的成本函数来惩罚上一个解决方案获得的速度和加速度的偏差。

#### 1.4.1.4 轨迹正则化

运动计划的持续更新可能会导致躯干相对于参考立足点的运动漂移。这可能是由于控制误差的累积造成的，这改变了解决方案，使运动变得不可行。为了避免这个问题，我们添加一个与参考正则化器路径的偏差比较的成本。这个正则化的路径可以近似表示成曲线段  $\pi(t), \dot{\pi}(t), \ddot{\pi}(t)$ ，这个路径正则化器的样条系数是从最小化问题设置中获得的，使得：

- $\pi(t)$  的初始位置与初始化时支撑多边形的中心位置重合， $\dot{\pi}(t), \ddot{\pi}(t)$  与初始速度和加速度重合；
- $\pi(t), \dot{\pi}(t), \ddot{\pi}(t)$  的结束状态由第1.4.1.2节中定义。
- 加速度最小化
- 通过在样条结点上的状态设置等式约束来平滑地连接样条线

通过对第1.4.1.3节中所做的整个运动进行采样，我们惩罚从路径正则化器产生的运动的偏差。

根据实际策略的不同，成本函数可能会对一些项目进行调整，还可能会包含一些其它的项目，如文献中表述<sup>[3]p4</sup>。

#### 1.4.2 等式约束

等式约束主要是从规划曲线段的连续性上出发的。整体上就是要求曲线段的位置、速度、加速度前后连续。

由于整个运动由一系列样条组成，我们设置运动优化问题以确保它们前后相连接。由于初始状态不能被运动规划器修改，我们设置了一个硬性等式约束，使得第一个样条  $s_0$  的初始状态与计划初始化中的一个集合重合。这个初始化的硬性等式约束可以写作  $\mathbf{p}(0) = \mathbf{p}^r, \dot{\mathbf{p}}(0) = \dot{\mathbf{p}}^r, \ddot{\mathbf{p}}(0) = \ddot{\mathbf{p}}^r$ ，于是有：

$$\begin{bmatrix} \boldsymbol{\eta}(0)^T \\ \dot{\boldsymbol{\eta}}(0)^T \\ \ddot{\boldsymbol{\eta}}(0)^T \end{bmatrix} \boldsymbol{\alpha}_0^x = \begin{bmatrix} \mathbf{p}_x^r \\ \dot{\mathbf{p}}_x^r \\ \ddot{\mathbf{p}}_x^r \end{bmatrix} \quad (1-34)$$

为了保证曲线段  $s_k$  和  $s_{k+1}$  是连续的，引入如下约束：

$$\begin{bmatrix} \boldsymbol{\eta}(t_{fk})^T & -\boldsymbol{\eta}(0)^T \\ \dot{\boldsymbol{\eta}}(t_{fk})^T & -\dot{\boldsymbol{\eta}}(0)^T \end{bmatrix} \begin{bmatrix} \boldsymbol{\alpha}_k^x \\ \boldsymbol{\alpha}_{k+1}^x \end{bmatrix} \quad (1-35)$$

其中  $t_{fk}$  表示以秒为单位的曲线  $s_k$  的持续时间。当连接两个样条的相等连接位于两个不相交的支持多边形之间时，我们不能再保证平滑度或运动，而是必须允许ZMP跳跃，这将导致加速度参考的跳跃。尽管这对控制器有负面影响，但它

确实允许优化器在遍历不相交的支持多边形时找到解决方案。我们使用文献<sup>[1]</sup>中描述的分离轴定理(SAT)来检查两个支撑多边形是否相交。如果两者相交，那我们再多添加一条约束条件：

$$\begin{bmatrix} \ddot{\eta}(t_{fk})^T & -\ddot{\eta}(0)^T \end{bmatrix} \begin{bmatrix} \alpha_k^x \\ \alpha_{k+1}^x \end{bmatrix} = 0 \quad (1-36)$$

### 1.4.3 不等式约束

不等约束的主要内容就是将ZMP约束到支撑多边形内部。

如[11]和[19]所示，通过约束Zero-Moment Point(ZMP)位于支撑多边形内，可以保证运动过程中的平衡。从运动计划调用时使用计划脚的位置测量的脚配置开始，我们通过使用[5]中定义的接触时间表来计算一系列支撑多边形及其持续时间。通过始终假设地面上至少有两只脚，四足机器人的支撑多边形可以是一条线、三角形或四边形。

正如[16]和[19]中所讨论的，ZMP的位置是质心运动的函数。ZMP的x坐标位置定义为：

$$x_{zmp} = x_{com} - \frac{z_{com}\ddot{x}_{com}}{g + \ddot{z}_{com}} \quad (1-37)$$

其中 $g$ 是重力加速度。我们通过计算每个支撑多边形的顶点与中心点的极坐标并比较他们的相位来对其顶点进行顺时针排列。这样一来就可以通过添加下式约束来使得ZMP点满足要求：

$$ax_{zmp} + by_{zmp} + c \geq 0 \quad (1-38)$$

其中， $a, b, c$ 是经过支撑多边形各个边的直线方程的系数。这条针线的法向量是 $\mathbf{n} = [a, b]^T$ ，其方向定义为指向约束多边形内部。

如第1.4.1节所述，我们对运动的最终位置 $\mathbf{p}_f$ 施加了额外的不等式约束。这是通过在 $\mathbf{p}_f$ 上添加如公式(1-37)形式的四个约束来获得的。

### 1.4.4 约束松弛

约束松弛主要涉及三个方面的处理：

- ZMP约束初始化的问题；
- 支撑多边形的缩小和逐步扩大来适应实际情况；
- 去除持续时间过短的约束多边形情况；

如果约束太紧，优化器可能会失败。首先，初始ZMP可能不位于当前支持多边形中。出于这个原因，我们在运动计划的第一个样本上排除了ZMP约束。这样优化器仍然能够找到解决方案，我们的实验表明这适用于真实系统。其次，支撑

多边形安全裕度可能太高。为了抵消这一点，每当优化失败时，我们都会以固定量迭代地减少边距，并以较慢的速度将其增加以确保优化的成功。最后，我们检查与每个多边形相关的持续时间  $t_{fk}$ 。如果它与用于离散化运动并设置约束的样本时间更小或相当，我们从序列中删除支持多边形。

## 第 2 章 基于直接刚体动力学模型的 TO 控制

### 2.1 基于直接刚体动力学模型的 TO 控制<sup>[7-8]p2-6</sup>

基于参考文献阐述基于刚体动力学模型的控制方法...

## 第3章 基于 MPC 的控制

在这部分我将介绍机械狗基于 MPC 的控制。

### 3.1 刚体系统的 MPC 问题描述<sup>[9]p2-4</sup>

特别地，这里介绍 NMPC。NMPC 方法不断迭代求解有限水平的优化控制问题 (*finite-horizon Optimal Control Problem*)，成本函数和非线性动力学系统分别具有如下形式：

$$J(\mathbf{x}(t), \mathbf{u}(t)) = h(\mathbf{x}(t_f)) + \int_{t=0}^{t_f} L(\mathbf{x}(t), \mathbf{u}(t), t) dt \quad (3-1)$$

$$\dot{\mathbf{x}}(t) = f(\mathbf{x}(t), \mathbf{u}(t), t), \quad \mathbf{x}(0) = \mathbf{x}_0 \quad (3-2)$$

其中的状态和控制轨迹是  $\mathbf{x}(t)$  和  $\mathbf{u}(t)$ 。

#### 3.1.1 系统模型

刚体动力学模型如下：

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}) + \mathbf{G}(\mathbf{q}) = \mathbf{J}_c^T \boldsymbol{\lambda}(\mathbf{q}, \dot{\mathbf{q}}) + \mathbf{S}^T \boldsymbol{\tau} \quad (3-3)$$

其中  $\mathbf{M}_{18 \times 18}$  是惯性矩阵， $\mathbf{C}_{18 \times 1}$  是科里奥利力和向心力， $\mathbf{G}_{18 \times 1}$  是重力项。另外这里设定系统是由关节点的力（力矩） $\boldsymbol{\tau}_{12 \times 1}$  驱动的， $\mathbf{S}_{18 \times 12}$  将这些力映射到驱动自由度上。此外，外力  $\boldsymbol{\lambda}$  通过雅克比矩阵  $(\mathbf{J}_c)_{18 \times 12}$  的转换作用到系统上。节点的位置（角度）和速度分别定义为  $\mathbf{q}$  和  $\dot{\mathbf{q}}$ 。考虑到浮动基体的因素，浮动基体的位置和姿态信息也包含在其中。浮动基体的相关信息可以认为是一个在惯性系和基体系之间的无驱动的 6 自由度系统。3D 的转动采用欧拉角参数形式。

整个系统的状态定义如下：

$$\mathbf{x} = \begin{bmatrix} {}_W\mathbf{q}^T & {}_L\dot{\mathbf{q}}^T \end{bmatrix}^T = \begin{bmatrix} {}_W\mathbf{q}_{BW}^T & \mathbf{x}_B^T & \mathbf{q}_J^T & {}_L\dot{\omega}_B^T & \dot{\mathbf{x}}_B^T & \dot{\mathbf{q}}_J^T \end{bmatrix}^T \quad (3-4)$$

其中姿态是在“世界”系  $W$  下表达的，转动是在本地坐标系  $B$  下表达的。由于参考系的不同，转动不是姿势的纯时间导数，而是需要额外的坐标变换  $\mathbf{T}_{WL}$ ，该坐标变换由线速度的纯旋转矩阵以及角速度的稍微复杂的映射矩阵组成。这导致了整个系统的动力学如下：

$$\dot{\mathbf{x}}(t) = \begin{bmatrix} {}_W\mathbf{q}^T & {}_L\dot{\mathbf{q}}^T \end{bmatrix}^T \quad (3-5)$$

$$= \begin{bmatrix} \mathbf{T}_{WL} & {} \\ \mathbf{M}^{-1}(\mathbf{q})(\mathbf{S}^T \boldsymbol{\tau} + \mathbf{J}_c^T \boldsymbol{\lambda}(\mathbf{q}, \dot{\mathbf{q}}) - \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}) - \mathbf{G}(\mathbf{q})) & {} \\ {} & {}^T \mathbf{L} \ddot{\mathbf{q}} \end{bmatrix} \quad (3-6)$$

### 3.1.2 接触模型

我们使用显式接触模型将接触物理添加到动态模型中。因此，接触力成为机器人状态的显式函数：

$$\boldsymbol{\lambda}(\mathbf{q}, \dot{\mathbf{q}}) = \mathbf{g}(\mathbf{x}(t))$$

我们的接触模型由垂直和平行于接触面的线性弹簧和阻尼器组成。对于每个末端执行器，我们在专用接触坐标系 C 中计算接触模型如下：

$${}_c \boldsymbol{\lambda}(\mathbf{q}, \dot{\mathbf{q}}) = -k \exp(\alpha_k {}_c \mathbf{p}_z(\mathbf{q})) - dsig(\alpha_d {}_c \mathbf{p}_z(\mathbf{q})) {}_c \dot{\mathbf{p}}_z(\mathbf{q}, \dot{\mathbf{q}}) \quad (3-7)$$

其中  $d, k$  分别是阻尼器和弹簧的参数。为了获得光滑导数，我们将阻尼项与接触面穿透  $\mathbf{p}(\mathbf{q})$  的法向分量  $\mathbf{p}_z(\mathbf{q})$  的 s 型函数相乘。指数函数和 s 型函数都可以作为平滑单元，它们的“锐度”由  $\alpha_k$  和  $\alpha_d$  控制。最后，将接触力转化到机器人体系中：

$${}_B \boldsymbol{\lambda}(\mathbf{q}, \dot{\mathbf{q}}) = \mathbf{R}_{WB}(\mathbf{q}) {}_C \boldsymbol{\lambda}(\mathbf{q}, \dot{\mathbf{q}}) \quad (3-8)$$

得到的结果随后就传递到正向动力学，在那里它们作用于相应的连杆上。我们对接触模型平滑的特殊选择通过确保接触力永远不会完全消失来支持基于梯度的求解器。因此，求解器甚至可以在接触建立之前对其进行推理。虽然这有点非物理性质，但我们稍后会看到，这并不妨碍硬件上的良好性能。我们强调存在物理上更精确的显式接触模型<sup>[10]p</sup>和隐式接触模型，它们在物理引擎中很流行。然而，后者依赖于基于优化的求解器，无法很好地区分。相比之下，我们的简化模型足够准确地捕捉控制效果，并允许通过使用 Auto-Diff 有效地计算导数，这是快速解决 NMPC 问题的关键<sup>[11]p</sup>。

## 3.2 NMPC 方法

非正式地说，NMPC 是通过以足够高的速率解决非线性最优控制 (*Nonlinear Optimal Control, NLOC*) 问题来实现的。常用的非线性最优控制方法有单次射击 (*Single Shooting*)、多次射击 (*Multiple Shooting*) 或直接配置 (*Direct Collection*)<sup>[12]p</sup>，它们将连续时间最优控制问题离散化，并将其转化为非线性程序 (*Nonlinear Program, NLP*)。这些 NLP 通常使用<sup>[13-14]p</sup> 中提出的通用的现成 NLP 求解器来求解。然而，这种方法并没有充分利用最优控制问题固有的稀疏性结构，并且经常导致较

差的算法运行时间，这对于 MPC 应用来说不够快。为了克服这一问题，我们将问题表述为无约束最优控制问题，并采用优化的自定义求解器，该求解器实现了一系列迭代高斯-牛顿 NLOC 算法 (*iterative Gauss-Newton NLOC algorithms*)<sup>[15]p</sup>。求解器采用一阶方法，将 NLOC 问题局部逼近为线性二次最优控制 (*Linear Quadratic Optimal Control, LQOC*) 问题，采用高斯-牛顿-黑森近似 (*Gauss-Newton Hessian approximation*)。LQOC 采用基于 Riccati(*Riccati-based*) 的求解器求解，该求解器在时间范围上具有线性复杂度。这使得该方法在更大的时间范围内有效。我们的求解器可以被认为是众所周知的 iLQR<sup>[16]p</sup> 和 SLQ<sup>[17]p</sup> 算法的推广，并且涵盖了单次和多次射击。设计了这种形式的时变状态反馈控制器：

$$\mathbf{u}_n(\mathbf{x}) = \mathbf{u}_n^{ff} + \mathbf{K}_n(\mathbf{x}_n - \mathbf{u}_n^{rcf}) \quad (3-9)$$

其中  $\mathbf{u}_n^{ff}$  是前馈控制动作， $\mathbf{K}_n$  是一个线性反馈控制器用来调节状态  $\mathbf{x}_n$  和参考位置  $\mathbf{u}_n^{rcf}$  之间的偏差。在这篇文章<sup>[9]p3</sup>的大多数实验中，我们使用了 iLQR 算法。我们进一步将其与更有效的高斯-牛顿多次射击 (GNMS) 方法进行比较，后者可以作为直接替代。两种算法都使用相同的方法来表述和解决局部 LQOC 问题，但是它们的 MPC 公式有所不同。

### 3.2.1 iLQR-NMPC 算法

---

#### 算法 3-1 Discrete-time iLQR-MPC Algorithm

---

- 1 **Given**
  - 2 - cost function (3-1) and system dynamics (3-2).
  - 3 - receding MPC time horizon N.
  - 4 - stable initial control policy  $\mathbf{u}_n(\mathbf{x})$  of form (3-9)
  - 5 **Repeat Online:**
  - 6 - get state measurement  $\mathbf{x}_{meas}$ .
  - 7 - forward integrate system dynamics (3-2) with  $\mathbf{x}_0 = \mathbf{x}_{meas}$  to obtain state trajectories  $\mathbf{X} = \{\mathbf{x}_0, \mathbf{x}_1, \dots, \mathbf{x}_N\}$ , control trajectories  $\mathbf{U} = \{\mathbf{u}_0, \mathbf{u}_1, \dots, \mathbf{u}_N\}$  and corresponding sensitivities  $\mathbf{A}_n, \mathbf{B}_n$ .
  - 8 (3-1) - quadratize cost function around  $\mathbf{X}$  and  $\mathbf{U}$
  - 9 - solve LQOC problem using a Riccati backward sweep
  - 10 - retrieve control policy  $\mathbf{u}_n^+(\mathbf{x})$  of form (3-9)
  - 11 - **line search** over the control increment  $(\mathbf{u}_n^(\mathbf{x})^+ - \mathbf{u}_n^(\mathbf{x}))$  and update  $\mathbf{X}^+$  by means of a forward simulation of the Nonlinear dynamics (3-2) with  $\mathbf{x}_0 = \mathbf{x}_{meas}$
  - 12 - send policy  $\mathbf{u}_n^+(\mathbf{x})$  and  $\mathbf{X}^+$  to the robot tracking controller
  - 13 - update  $\mathbf{u}_n(\mathbf{x}) \leftarrow \mathbf{u}_n^+(\mathbf{x})$
- 

在算法3-1中对 iLQR-NMPC 算法进行了总结。算法过程中有两个前向积分步骤。一次是在检索状态测量后直接得到标称轨迹，另一次是在更新控制器后进行

直线搜索。对于我们的应用，后者对于获得反馈控制器跟踪的新参考轨迹非常重要。

### 3.2.2 GNMS-NMPC 算法

---

#### 算法 3-2 Discrete-time GNMS-NMPC Algorithm

---

- 1 **Given**
  - 2 - cost function (3-1) and system dynamics (3-2).
  - 3 - receding MPC time horizon  $N$ .
  - 4 - initial state and control trajectories  $\mathbf{X} = \{\mathbf{x}_0, \mathbf{x}_1, \dots, \mathbf{x}_N\}$ ,  $\mathbf{U} = \{\mathbf{u}_0, \mathbf{u}_1, \dots, \mathbf{u}_N\}$
  - 5 **Repeat Online:**
  - 6 *Feedback phase*
  - 7 - get state measurement  $\mathbf{x}_{meas}$ .
  - 8 - forward integrate system dynamics (3-2) with  $\mathbf{x}_0 = \mathbf{x}_{meas}$  on the first multiple-shooting interval, obtain sensitivities  $\mathbf{A}_0, \mathbf{B}_0$ . - quadratize cost function (3-1) around  $\mathbf{X}$  and  $\mathbf{U}$  for first control stage
  - 9 - solve LQOC problem using a Riccati backward sweep
  - 10 - retrieve updated control policy  $\mathbf{u}_n^+(\mathbf{x})$  and updated trajectories  $\mathbf{U}^+, \mathbf{X}^+$ .
  - 11 - send policy  $\mathbf{u}_n^+(\mathbf{x})$  and  $\mathbf{X}^+$  to the root tracking controller
  - 12 *Preparation phase*
  - 13 update:  $\mathbf{u}_n(\mathbf{x}) \leftarrow \mathbf{u}_n^+(\mathbf{x}), \mathbf{X} \leftarrow \mathbf{X}^+, \mathbf{U} \leftarrow \mathbf{U}^+$
  - 14 forward integrate system dynamics (3-2) for the multiple-shooting intervals 1 to  $N$ , obtain sensitivities  $\mathbf{A}_1, \dots, \mathbf{A}_{N-1}, \mathbf{B}_1, \dots, \mathbf{B}_{N-1}$ .
  - 15 - quadratize cost function (3-1) around  $\mathbf{X}, \mathbf{U}$  for multiple-shooting intervals 1 to  $N$ .
- 

相比之下，算法3-2中总结的GNMS-NMPC算法在新控制策略的同时设计了状态参考轨迹。此外，它允许将算法分为反馈阶段 (*feedback phase*) 和准备阶段 (*preparation phase*)[29]，这有助于最小化状态测量和控制策略更新之间的延迟。关于GNMS算法的详细概述，请参考<sup>[15]p</sup>。一个开源参考实现可在这个链接中找到。

## 第4章 基于强化学习的机械狗控制

在这部分我将阐述关于如何使用强化学习来进行运动控制的内容。相比于传统的基于模型的控制方法，该方法通过在模拟环境中训练控制策略，可以更好地适应复杂的非线性系统动力学和环境不确定性，并且能够实时响应用户命令和环境变化。在这里我们主要关注如何使用强化学习来训练机械狗类型的足式和轮式机器人控制器。

### 4.1 强化学习控制

#### 4.1.1 强化学习的基本概念<sup>[18]</sup>

在强化学习中，控制问题被建模为一个马尔可夫决策过程（Markov Decision Process）。MDP 是一个 RL 中常用的用于随机控制过程建模的数学框架，它定义了一个包含状态空间、动作空间、奖励函数和状态转移函数的元组，描述了一个决策过程的基本组成部分。在 MDP 中，每个时间步骤代理从周围环境中观察到某个状态  $s_t \in \mathcal{S}$ ，并输出一个动作  $a_t \in \mathcal{A}$ ，接着环境通过状态转移函数  $p(s_{t+1}|s_t, a_t)$  演化到新的状态  $s_{t+1}$ ，并且根据奖励函数给出相应的奖励  $r_t \in \mathcal{R} : \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{R}$  和对更新后的环境状态的观察结果作为下一周期的  $s_t$ 。代理可以根据  $\theta$  参数化的随机策略  $\pi_\theta(a_t|s_t)$  来采取行动。RL 通过与环境交互来更新参数  $\theta$  以最大化累计折扣奖励（cumulative discounted rewards） $\mathbb{E}[\sum_{t=k}^{\infty} \gamma^t r_t]$ ，其中  $k$  是当前时间步长（timestep）， $\gamma$  是折扣因子（discount factor）。

#### 4.1.2 基于深度强化学习的控制器的部署和优化流程

(1) 在仿真环境中训练控制器：使用深度强化学习算法在仿真环境中训练控制器，使其能够完成所需的任务。在训练过程中，可以使用特权训练方法来提高训练效率和性能。

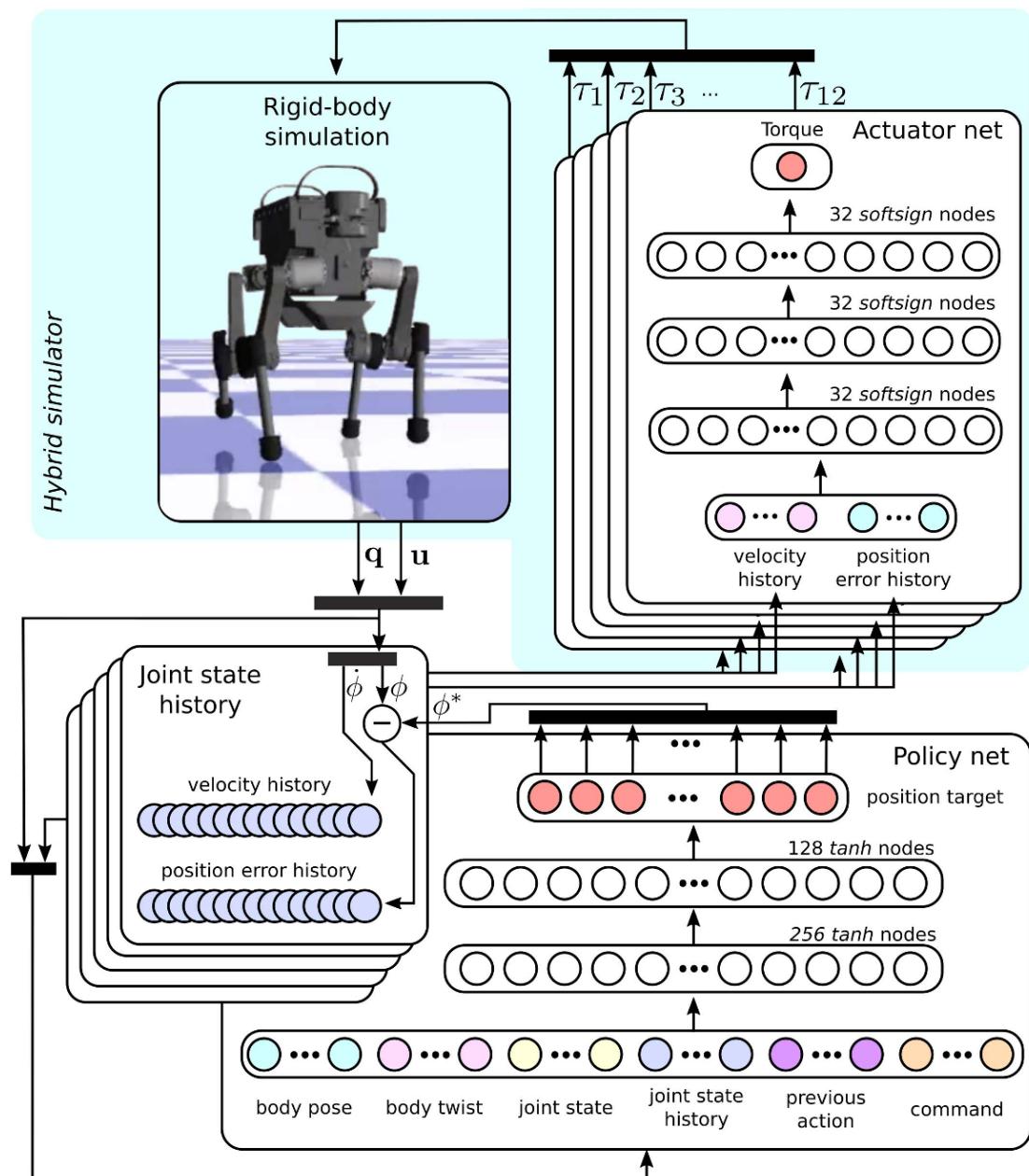
(2) 部署控制器到实际环境中：将训练好的控制器部署到实际环境中，例如机器人或移动设备。在实际环境中，控制器将接收传感器数据并输出动作命令。

(3) 优化控制器：在实际环境中，可以使用在线学习方法来进一步优化控制器的性能。例如，可以使用模型预测控制方法来对控制器进行在线微调。

需要注意的是，在实际环境中，传感器数据可能会受到噪声和不确定性的影响，因此需要设计鲁棒性强的控制器来应对这些问题。此外，还需要考虑实际环

境中的安全性和可靠性问题。

## 4.2 单层神经网络直接驱动关节方式<sup>[19]p3</sup>



**Fig. 5 Training control policies in simulation.** The policy network maps the current observation and the joint state history to the joint target.

图 4-1 The policy network maps the current joint state and position history to the joint target.

如图4-1所示，整个训练循环包含下面几个部分：

- 首先，一个刚体仿真模拟器运用给定的关节节点扭矩和电流状态计算输出机器人下一状态。与此同时关节点的速度和位置误差在有限时间窗口内被缓冲在节点状态历史中以备网络层使用。

- 接着，由两个隐藏层的 MLP 神经网络构成控制策略，它将输入的电流状态和节点状态历史的观测信息映射成节点位置目标。
- 最后，关节点神经网络将关节点状态历史和关节点位置目标映射到 12 个关节点的扭矩值上，然后继续整个循环。

## 4.3 双层神经网络本体感知方式<sup>[20]p8</sup>

### 4.3.1 总体概况

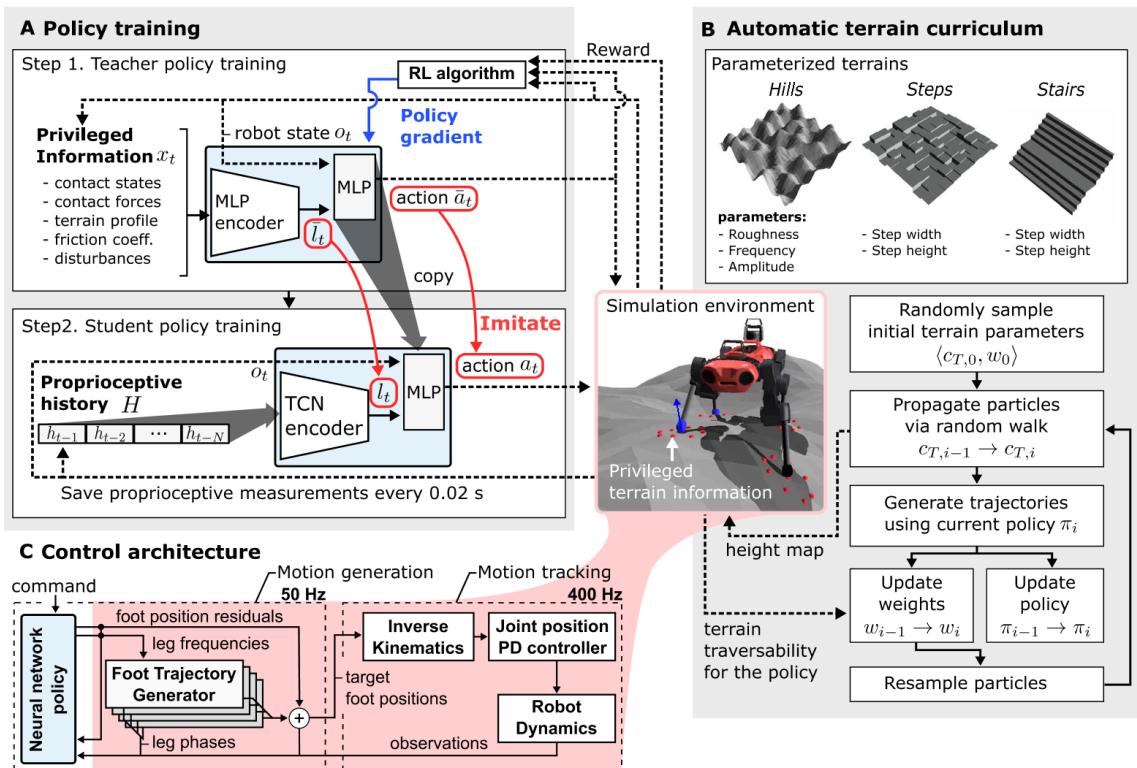


图 4-2 Creating a proprioceptive locomotion controller<sup>[20]p7</sup>.

图4-2是一个只采用本体感知进行机器狗控制的 RL 策略实现架构。它包括训练定义、地形定义、控制架构。

- 策略训练：整个 RL 训练采用特权训练 (*privileged training*)<sup>[21]p</sup> 的理念，训练过程分为两个阶段，如图4-2A 所示。第一阶段用完善的特权信息作为输入训练出一个教师策略 (*teacher policy*)；第二阶段仅用本体感知作为输入训练出一个学生策略 (*student policy*)，学生策略的训练过程通过模仿教师策略实现。实际部署到机器人上的策略是学生策略。

- 地形定义：RL 训练过程是在一定的地形条件下完成的，如图4-2B 所示。对于同样的控制命令，不同的地形条件会产生不同的任务难度。为了能更好地促进训练进步，训练过程中的地形采用自适应地形课程 (*adaptive terrain curriculum*)，它

会从简单地形开始根据训练控制器的性能提升而不断提高地形难度，使得 RL 控制器时钟面对相对于当前状况的中等地形难度。

- 控制架构：整个控制架构采用调节轨迹生成器 (*Policies Modulating Trajectory Generators, PMTG*)<sup>[22]P</sup> 架构来提供运动生成的先验。神经网络策略通过合成残差位置命令来调节腿相和运动原语，如图4-2C 所示。

### 4.3.2 策略训练

#### 4.3.2.1 教师策略

整个控制问题被构建成一个马尔科夫决策过程 (*Markov Decision Process, MDP*)。MDP 是一种用于状态和结果部分随机的离散时间控制过程的数学框架。MDP 被定义为一个状态空间  $\mathcal{S}$ ，动作空间  $\mathcal{A}$ ，一个标量奖励函数  $\mathcal{R}(s_t, s_{t+1})$ ，一个转移概率  $P(s_{t+1}|s_t, a_t)$ 。一个学习代理从策略  $\pi(a_t|s_t)$  中选择一个动作  $a_t$  并从环境中获得奖励  $r_t$ 。RL 的目的是找到一个优化的策略  $\pi^*$  使得无限时间范围内的折扣奖励总和最大。

状态空间  $s_t$  定义为： $\langle o_t, x_t \rangle$ ，其中  $o_t$  是机器人自身的观测信息， $x_t$  是特权信息。

$$s_t = \begin{cases} o_t & \left\{ \begin{array}{l} \text{proprioceptive sensors} \\ \text{state estimator} \end{array} \right. \\ & \left\{ \begin{array}{l} \text{base velocity} \\ \text{orientation} \end{array} \right. \\ x_t & \text{only for simulation} \end{cases} \quad (4-1)$$

动作空间  $\bar{a}_t$  是一个 16 维向量，由腿的频率和交的位置残差构成。

奖励函数  $\mathcal{R}(s_t|s_{t+1})$  定义。

**reward function definition details ...**

如图4-2A 所示，策略网络由两个 MLP 模块构成。第一个 MLP 模块将  $x_t$  信息转化成潜在向量  $\bar{l}_t$ ，由于  $x_t$  中并不含有机器人的状态或指令信息，也就是说他只包含地形信息和接触相关的信息。假设  $\bar{l}_t$  的作用是驱动自适应行为，比如根据地形轮廓改变脚步间隙大小。然后  $\bar{l}_t$  和  $o_t$  再提供给第二个 MLP 网络来计算动作。

训练采用信任区域策略优化 (*Trust Region Policy Optimization, TRPO*)<sup>[23]P</sup>。

#### 4.3.2.2 学生策略

学生策略仅能获取  $o_t$  信息。这里的一个关键假设是潜在特征  $\bar{l}_t$  可以从本体感觉观察的时间序列  $h_t$  中恢复（部分）， $h_t :=$

$o_t\{f_o, \text{joint history}, \text{previous foot position targets}\}$ 。

学生策略采用 TCN<sup>[24]p</sup> 编码器。使用 TCN 架构的原因是它对输入历史长度提供透明的控制，可以容纳长历史，并且已知对超参数设置具有鲁棒性<sup>[24]p</sup>。

学生策略采用监督学习的方式进行训练，损失函数定义为：

$$\mathcal{L} := (\bar{a}_t(o_t, x_t) - a_t(o_t, H))^2 + (\bar{l}_t(o_t, x_t) - l_t(H))^2 \quad (4-2)$$

带  $(\cdot)$  标识的项是来自教师策略的生成。对于每个访问的状态，教师策略计算其嵌入和动作向量  $(\cdot)$ ，这些教师策略的输出随即被用作与相应状态相关的监督信号。

### 4.3.3 地形定义

Adaptive terrain curriculum ...

### 4.3.4 控制架构

整个控制架构结构如图4-2C 所示。它被分为两大类：运动生成、跟随。整个系统的输入只有控制指令和本体感知，输出为关节点位置目标。

运动生成策略是一种基于周期性腿相位的策略，之前的一些工作中通常利用预定的脚接触时间表<sup>[3,25]p7</sup>。为每条腿都定义一个周期的相位变量  $\phi_i \in [0.0, 2\pi]$ 。在每一个时间步长  $t$  里， $\phi_i = (\phi_{i,0} + (f_0 + f_i)t) \pmod{2\pi}$ ，其中  $\phi_{i,0}$  是初始相位， $f_0$  是一般基础频率， $f_i$  是第  $i$  条腿的频率偏移。我们希望腿在  $f_0 + f_i \neq 0$  时表现出周期性运动，并在接触阶段与地面接触。这里基础频率参考之前开发的传统控制<sup>[3]p7</sup> 中鹦鹉步态的值， $f_0 = 1.25\text{Hz}$ 。

target foot position...

我们采用 PMTG 架构来将神经网络集成系统中用来调节控制器的输出。整体实现由四个完全相同的足迹生成器 (*foot trajectory generators, FTGs*) 和一个神经网络策略 (*neural network policy, NNP*) 构成。足迹生成器是一个输出每条腿的脚位置的函数： $F(\phi) : [0.0, 2\pi) \rightarrow \mathbb{R}^3$ 。这个 FTG 在  $f_i$  不为零时驱动垂直方向的踱步运动。

**Addition 4.3.1.**  $F(\phi)$  的定义如下:

$$F(\phi_i) = \begin{cases} (h(-2k^3 + 3k^2) - 0.5)^{H_i Z} & k \in [0, 1] \\ (h(2k^3 - 9k^2 + 12k - 4) - 0.5)^{H_i Z} & k \in [1, 2] \\ -0.5^{H_i Z} & otherwise \end{cases} \quad (4-3)$$

其中  $k = 2(\phi_i - \pi)/\pi$ ,  $h$  是一个表示最大步高的参数。在摆动相位的每一个阶段都是一个连接最高点和最低点的三次 Hermite 样条曲线，在连接点处曲线具有一阶连续性（一阶导数为零）。其它的周期函数，如  $h_i \sin(\phi_i)$  可以用于 FTG。给定一组合适的  $f_0, h, \phi_{i,0}$  参数值，机器人就能稳定地在地面上踱步了。比如，文献<sup>[20]</sup>p<sup>7</sup> 中使用的值： $f_0 = 1.25, h = 0.2, \phi_{i,0}$  从  $U(0, 2\pi)$  中采样得到。

神经网络策略输出  $f_{iS}$  和脚的目标位置残差 ( $\Delta r_{f_i,T}$ )。这样，第  $i$  个脚的目标位置是  $r_{f_i,T} := F(\phi_i) + \Delta r_{f_i,T}$ 。

跟随控制是使用解析逆运动学 (*inverse kinematics, IK*) 和关节点位置控制 (*position control*) 实现的。在  $H_i$  中定义的脚位置首先表述在机器人身体系中，然后用 IK 计算关节点的位置目标。而关节点的位置目标依靠关节点的 PD 控制器来跟随。使用 IK 的好处是可以最大化计算效率同时在仿真到实际 (*sim-to-real*) 的过程中可以复用已有的位置控制驱动器模型<sup>[26-27]</sup>p<sup>7</sup>。

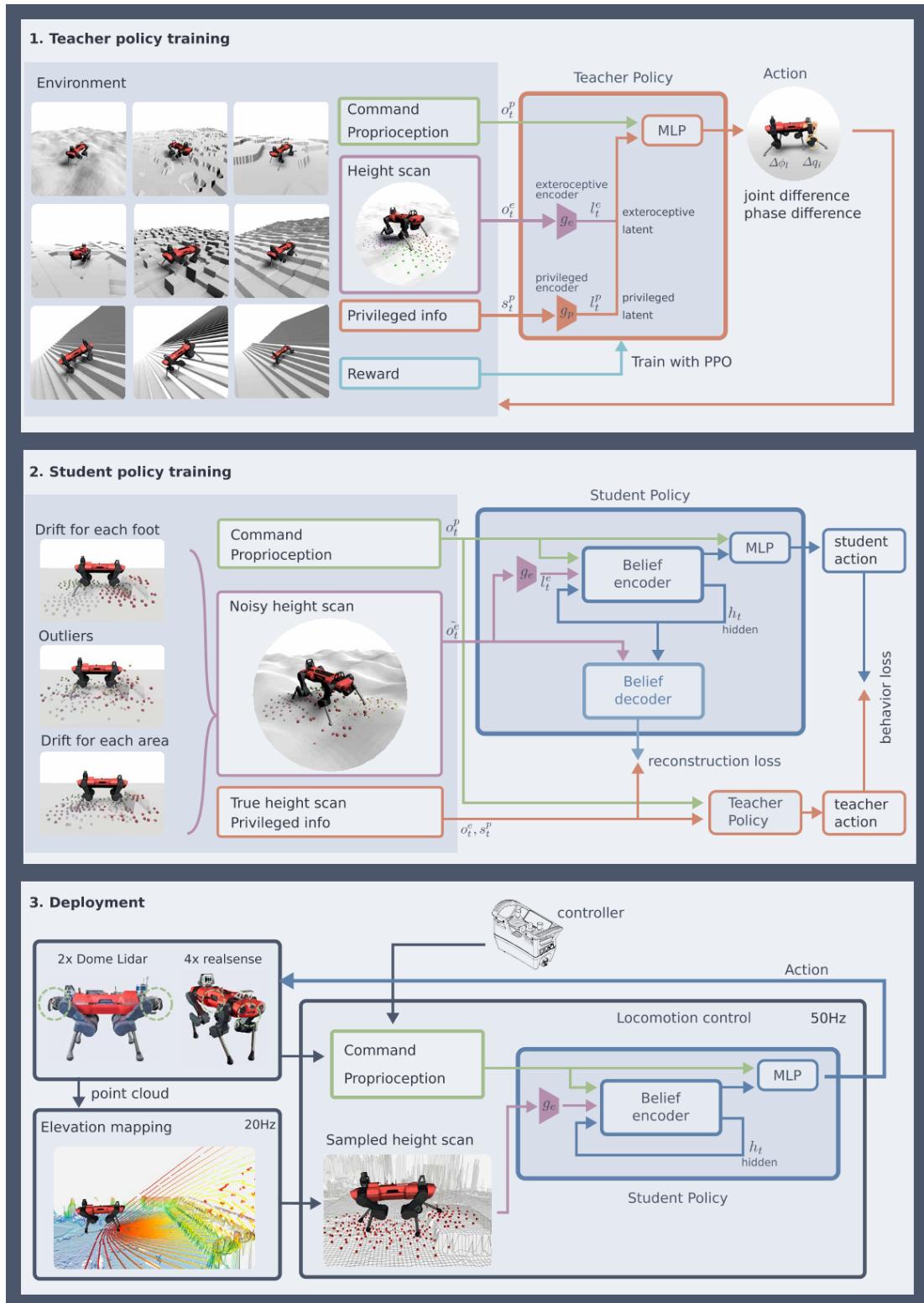
#### 4.4 双层网络本体和外部感知融合方式<sup>[28]</sup>p<sup>7</sup>

整个神经网络的训练是在仿真环境中完成的，然后采用 *zero-shot sim-to-real* 的转换部署到实际的额机器人上。整个方法分为三个阶段，如图4-3所示。

(1) 首先，使用 RL 训练教师策略，以在随机生成的具有随机干扰的地形上遵循随机目标速度。该策略可以访问特权信息，例如无噪声地形测量、地面摩擦和引入的扰动。

(2) 在第二阶段，训练学生策略重现教师策略的动作，而不使用这种特权信息。学生策略构造一个信念状态来使用循环编码器捕获未观察到的信息，并根据该信念状态输出一个动作。在训练期间，我们利用两个损失：行为克隆损失和重建损失。行为克隆损失旨在模仿教师策略。重新构造损失鼓励编码器产生信息丰富的内部表示。

(3) 最后，我们将学习到的学生策略转移到物理机器人上，并将其与机载传感器在现实世界中部署。机器人通过整合来自板传感器的深度数据和从构建的高程图中采样高度读数来构建高程图，以形成策略的外部感知输入。这种外部感知

图 4-3 RL implementation process<sup>[28]p9</sup>.

https://www.bilibili.com/video/BV1Lc4y1G7zU?from=search&seid=13400000000000000000

输入与本体感觉数据相结合，并提供给神经网络，该网络产生执行器命令。板上传感器和自身构建等高图融合

#### 4.4.1 问题描述

我们在离散时间动力学中制定了我们的控制问题，其中环境完全由时间步  $t$  的状态  $s_t$  定义。该策略实施一个动作  $a_t$  并且从环境中获得一个观测结果  $o_t$ ，这个测量结果来自观测模型  $\mathcal{O}(o_t|s_t, a_t)$ 。接着环境以  $P(s_{t+1}|s_t, a_t)$  的概率转移到下一个状态  $s_{t+1}$  并给出一个奖励  $r_{t+1}$ 。

当所有状态都可以被观测的实况下，也即  $o_t = s_t$  时，整个问题可以被看做一个马尔可夫决策过程 *Markov decision process(MDP)*。然而，当存在不可观察性的信息时，例如外力或完整的地形信息，动力学被建模为部分可观察的马尔可夫决策过程 *partially observable Markov decision process(POMDP)*。

RL 的目标是找到一个能够使得未来轨迹的预期折扣奖励最大化的策略  $\pi^*$ ，以使得：

$$\pi^* = \underset{a}{\operatorname{argmax}} E \left[ \sum_{t=0}^{\infty} \gamma^t r_t \right] \quad (4-4)$$

已经开发了许多 RL 算法来解决完全可观察的 MDP 问题，并且易于用于训练。然而，POMDP 问题的情况更具挑战性，因为状态不能完全观察到。这通常通过从历史的观察结果  $o_0, \dots, o_t$  中构建一个信念状态  $b_t$  *belief state* 以尝试构建完全状态的方式来解决。在深度 RL 中，这通常是通过堆叠一系列先前的观察结果<sup>[29]</sup>p 或使用可以压缩过去信息的架构来完成的，例如循环神经网络 (RNN)<sup>[30]</sup>p 或时间卷积网络<sup>[20,24]</sup>p。

从头开始训练一个天真地处理序列数据的复杂的神经网络策略可能很耗时<sup>[20]</sup>p。因此，我们使用特权学习（45），我们首先训练一个具有特权信息的教师策略，然后通过监督学习将教师策略提炼为学生策略<sup>[21]</sup>p。

- (1) 训练环境:
- (2) 地形:
- (3) 域随机化:
- (4) 片段终止条件:

#### 4.4.2 教师策略训练

在训练的第一阶段，我们的目标是找到一个可以访问完美、特权信息的最佳参考控制策略，并使 ANYmal 在随机生成的地形上遵循所需的命令速度。命令需

求随机生成并构成一个向量  $\mathbf{v}_{des} \in \mathbb{R}^3 = (v_x, v_y, w)$ , 其中  $v_x, v_y$  分别表示在机器人自身坐标系下的纵向速度和横向速度,  $w$  表示自转速度。

我们采用近端策略优化 *proximal policy optimization(PPO)*<sup>[31]p</sup> 来训练教师策略。教师策略被建模为一个高斯策略,  $a_t \sim \mathcal{N}(\pi_\theta(o_t = s_t), \sigma I)$ , 其中  $\pi_\theta$  由用  $\theta$  参数化的多层感知器 *multilayer perceptron(MLP)* 实现,  $\sigma$  表示每个动作之间的方差。

#### 4.4.3 观测和行动

教师策略的观察定义为  $o_t^{teacher} = (o_t^p, o_t^e, s_t^p)$ , 其中  $o_t^p$  表示本体感觉观察 *proprioception observation*,  $s_t^e$  表示外部观察 *exteroceptive observation*,  $s_t^p$  表示特权状态 *privileged state*。

- $o_t^p$  包含本体速度、转动、节点位置和速度历史、动作历史、每条腿的相位;
- $o_t^e$  是每只脚周围高度样本的向量, 包括五种不同的半径;
- $s_t^p$  包括接触状态、接触力、接触法线、摩擦系数、大腿和小腿接触状态、施加到身体上外部力和力矩、摆动阶段持续时间;

我们的动作空间受到中心模式生成器的启发<sup>[20]p</sup>。每条腿  $l = 1, 2, 3, 4$  保存一个相位变量  $\phi_l$  并定义了基于相位的标称轨迹。这个标称轨迹是一个脚尖的步进运动, 我们使用逆运动学来计算每个关节执行器  $i = 1, \dots, 12$  的标称关节目标  $q_i(\phi_l)$ 。来自策略的动作是相差  $\Delta\phi_l$  和关节位置目标残差  $\Delta q_i$ 。

**更详细相关内容看相关附件的 S5。**

观测向量定义如4-1表所示。本体感知包括命令、节点、身体信息、腿部相位信息。中央模式生成器 *central pattern generator(CPG)* 的相位信息包含  $\Delta\phi_l, \cos\phi_l, \sin\phi_l$  和每条腿  $l$  的基础频率。对于外部感知我们采用每个脚周围的高度采样来代替局部高程图。圆形采样模式包括每只脚周围的  $\{6, 8, 10, 12, 16\}$  个点, 半径分别为  $\{0.08, 0.16, 0.26, 0.36, 0.48\}$ m。

运动被定义为  $\langle\Delta\phi_l, \Delta q_i\rangle$ , 其中  $\Delta\phi_l$  和  $\Delta q_i$  分别表示每个条腿 ( $l \in \{legs\}$ ) 的相位偏移和节点目标位置残差 ( $i \in \{1, \dots, 12\}$ )。我们有一个标称轨迹  $\mathbf{p}(\phi) : \mathbb{R} \rightarrow \mathbb{R}^3$ , 它将各个  $\phi_l$  映射到目标脚位置, 随着  $\phi$  在  $[0, 2\pi]$  范围内生成周期性步进运动。从动作中, 每条腿  $l$  的节点目标位置用逆运动学  $IK(\cdot)$  和基础相位频率  $\Delta\phi_0$  定义为  $q_{i \in l}^{target} = IK(\mathbf{p}(\phi_l + \Delta\phi_l + \Delta\phi_0)) + \Delta q_{i \in l}$ 。

标称足迹定义如下。如果相位是处于上摆动期间 ( $0 < \phi_l < \pi/2$ ) 则:

$$\mathbf{p}_l(\phi_l) = \langle x_l^n, y_l^n, z_l^n + 0.2 \cdot (-2t_l^3 + 3t_l^2) \rangle, \text{where } t_l = 2/\pi \cdot \phi_l \quad (4-5)$$

$\{x, y, z\}_l^n$  是默认姿态配置处的标称脚位置。三次 Hermite 样条在  $\phi_l = 0$  处连接  $z = z_l^n$  在  $\phi_l = \pi/2$  处连接  $z = z_l^n + 0.2$ 。

在下摆动期间 ( $\pi/2 < \phi_l \leq \pi$ ), 足高计算如下:

$$\mathbf{p}_l(\phi_l) = \langle x_l^n, y_l^n, z_l^n + 0.2 \cdot (2t_l^3 - 3t_l^2 + 1) \rangle, \text{ where } t_l = 2/\pi \cdot \phi_l - 1 \quad (4-6)$$

它与前面的函数对称。在驻立阶段 ( $\pi < \phi_l \leq 2\pi$ ),  $\mathbf{p}_l(\phi_l) = \langle x_l^n, y_l^n, z_l^n \rangle$ 。

表 4-1 Observations. Proprioception is used for both teacher and student training. Exteroception is given in the form of height samples. The privileged information is used only for teacher training.

Observation type	Input	Dimensionality
Proprioception	command	3
	body orientation	3
	body velocity	6
	joint position	12
	joint velocity	12
	joint position history (3 time steps)	36
	joint velocity history (2 time steps)	24
	joint target history (2 time steps)	24
	CPG phase information	13
Exteroception	height samples	208
Privileged info.	contact states	4
	contact forces	12
	contact normals	12
	friction coefficients	4
	thigh and shank contact	8
	external forces and torques	6
	airtime	4

#### 4.4.4 策略构架

我们将教师策略  $\pi_\theta$  建模为一个 MLP。它包括三个 MLP 组成部分：外部感知编码器、特权编码器、主网络，如图4-3所示。

(1) 外部感知编码器  $g_e$  接收来自  $o_t^e$  的信息，然后输出一个小些的潜在表示

$$l_t^e = g_e(o_t^e)$$

(2) 特权编码器  $g_p$  接收来自特权状态  $s_t^p$  的信息，然后输出一个潜在表示

$$l_t^{priv} = g_p(s_t^p)$$

(3)

这些编码器将每个输入压缩为更紧凑的表示，并使得学生策略能更方便地重用一些教师策略组件。

[更详细相关内容看相关附件的 S6。](#)

策略网络由多层 MLP 组成。用基于 MLP 的编码器  $(g_e, g_p)$ ，将高度采样结果首先编码成为一个  $24 \times 4 = 96$  维的潜在向量；将特权信息编码成为一个 24 维的潜在变量。每个编码器有两层分别是  $\{80, 60\}$  和  $\{64, 32\}$  的隐藏单元。高度样本首先分别针对每只脚分别输入到编码器中，然后连接成一个特征向量。然后将这些特征与本体感受观察连接起来，并馈送到另一个具有三个隐藏层  $\{256, 160, 128\}$  的 MLP。所有 MLP 的激活函数为 LeakyReLU (72)。

我们使用具有外部感受门的 GRU 作为信念编码器（图4-4C）。GRU 由 2 个堆叠的层组成，每个层有 50 个隐藏单元。信念编码器和外部感知门  $g_b, g_a$  用于计算  $96 + 24 = 120$  维信念状态  $b_t$  和 96 维注意力向量  $\alpha$ 。每个编码器有两个隐藏层，每个隐藏层有  $\{64, 64\}$  和  $\{64, 64\}$  个隐藏单元。过滤后的外部感受信息  $l_t^e \odot \alpha$  被添加到  $g_b(b_t')$ ，使用零填充来匹配维度差异。

#### 4.4.5 奖励函数

针对速度控制命令的跟随，我们定义正奖励；针对违反约束的情况，我们定义负奖励。指令跟随奖励定义如下：

$$r_{command} = \begin{cases} 1.0, & \text{if } \mathbf{v}_{des} \cdot \mathbf{v} > |\mathbf{v}_{des}| \\ \exp(-(\mathbf{v}_{des} \cdot \mathbf{v})^2), & \text{otherwise} \end{cases} \quad (4-7)$$

其中  $\mathbf{v}_{des} \in \mathbb{R}^2$  是所需的水平速度， $\mathbf{v} \in \mathbb{R}^2$  是身体坐标系下当前身体速度。同样的奖励机制也被应用与转动速度情况。

我们惩罚与期望速度正交的速度分量以及横摇、俯仰和偏航周围的身体速度。此外，我们使用整形奖励进行身体方向、关节扭矩、关节速度、关节加速度和脚滑以及小腿和膝盖碰撞。身体方向奖励用于避免身体的奇怪姿势。联合相关奖励术语用于避免过于激进的运动。脚滑和碰撞奖励术语用于避免它们。我们通过在模拟中查看策略的行为来调整奖励术语。除了遍历性能外，我们还检查了运动的平滑度。

[更详细相关内容看相关附件的 S7。](#)

奖励函数定义为：

$$r = 0.75(r_{lv} + r_{av} + r_{lvo}) + r_b + 0.003r_{fc} + 0.1r_{co} + 0.001r_j + 0.08r_{jc} + 0.003r_s + 1.0 \cdot 10^{-6}r_t + 0.0 \quad (4-8)$$

各项的具体定义如下：

(1) 线速度奖励 *Linear Velocity Reward*( $r_{lv}$ )：这项鼓励策略去跟随需要的水平( $x, y$  plane)速度指令：

$$r_{lv} = \begin{cases} \exp(-|\boldsymbol{v}|^2), & \text{if } |\boldsymbol{v}_{des}| = 0 \\ 1.0, & \text{elseif } \boldsymbol{v}_{des} \cdot \boldsymbol{v} > |\boldsymbol{v}_{des}| \\ \exp(-(\boldsymbol{v}_{des} \cdot \boldsymbol{v} - |\boldsymbol{v}_{des}|)^2), & \text{otherwise} \end{cases} \quad (4-9)$$

其中  $\boldsymbol{v}_{des} \in \mathbb{R}^2$  是目标水平速度， $\boldsymbol{v} \in \mathbb{R}^2$  是身体坐标系下当前身体的速度。

(2) 角速度奖励 *Angular Velocity Reward*( $r_{av}$ )：这一项鼓励策略去跟随需要的转动速度指令：

$$r_{av} = \begin{cases} \exp(-w_z^2), & \text{if } w_{des} = 0 \\ 1.0, & \text{elseif } \boldsymbol{w}_{des} \cdot \boldsymbol{w}_z > w_{des} \\ \exp(-(w_{des} \cdot \boldsymbol{w}_z - |w_{des}|)^2), & \text{otherwise} \end{cases} \quad (4-10)$$

(3) 线性正交速度奖励 *Linear Orthogonal Velocity Reward*( $r_{lvo}$ )：这项惩罚与目标速度指令正交的速度：

$$r_{lvo} = \exp(-3.0|\boldsymbol{v}_o|^2), \text{ where } \boldsymbol{v}_o = \boldsymbol{v} - (\boldsymbol{v}_{des} \cdot \boldsymbol{v})\boldsymbol{v}_{des} \quad (4-11)$$

(4) 身体运动奖励 *Body Motion Reward*( $r_b$ )：这项惩罚身体速度中不符合指令要求的部分：

$$r_{bm} = -1.25v_z^2 - 0.4|\omega_x| - 0.4|\omega_y| \quad (4-12)$$

(5) 脚感奖励 *Foot Clearance Reward*( $r_{fc}$ )：当一条腿处于摆动相位时，比如  $\phi_i \in [0, \pi]$ ，机器人应该将这条腿上对应的脚抬起到比障碍物高的程度。但是，为了防止机器人做出不必要的抬高间隙，我们通过给出惩罚  $r_{fcl}$  来正则化腿轨迹。 $H_{sample,l}$  是第  $l$  只脚的高度采样。这样，间隙成本定义为：

$$f_{fcl} = \begin{cases} -1.0, & \text{if } \max(H_{sample,l}) < -0.2 \\ 0.0, & \text{otherwise} \end{cases} \quad (4-13)$$

$$r_{fc} = \sum_{l=1}^4 r_{fcl} \quad (4-14)$$

注意高度样本是相对于脚高度进行采样，因此  $-0.2$  表示地形高度比脚低  $0.2m$ ；脚比采样的地形高度高  $0.2m$ 。

(6) 小腿和膝关节碰撞奖励 *Shank and Knee Collision Reward*( $r_{co}$ )：我们希望

惩罚脚以外的地形和机器人部件之间的不良接触，以避免硬件损坏：

$$r_{co} = \begin{cases} -c_k, & \text{if shank or knee is in collision} \\ 0.0, & \text{otherwise} \end{cases}$$

这里  $c_k$  是课程因子，它单调增加并收敛到 1。

(7) 节点运动奖励 *Joint Motion Reward*( $r_j$ )：该项惩罚关节速度和加速度以避免振动：

$$r_s = -c_k \sum_{i=1}^{12} (0.01\dot{q}_i^2 + \ddot{q}_i^2) \quad (4-15)$$

其中  $\dot{q}_i, \ddot{q}_i$  分别是节点的速度和加速度。

(8) 节点约束奖励 *Joint Constraint Reward*( $r_{jc}$ )：该项在联合空间中引入了一个软约束。为了避免相反方向的膝关节翻转，我们对超过阈值的惩罚：

$$r_{jc,i} = \begin{cases} -(q_i - q_{i,th})^2, & \text{if } q_i > q_{i,th} \\ 0.0, & \text{otherwise} \end{cases} \quad (4-16)$$

$$r_{jc} = \sum_{i=1}^{12} r_{jc,i} \quad (4-17)$$

$$(4-18)$$

其中  $q_{i,th}$  是第  $i$  个节点的阈值。我们只设置膝关节的阈值。

(9) 目标平滑奖励 *Target Smoothness Reward*( $r_s$ )：通过对目标位置一阶和二阶有限差分导数的大小进行惩罚，使生成的足部轨迹变得更平滑：

$$r_s = c_k \sum_{i=1}^{12} ((q_{i,t}^{des} - q_{i,t-1}^{des})^2 + (q_{i,t}^{des} - 2q_{i,t-1}^{des} + q_{i,t-2}^{des})^2) \quad (4-19)$$

其中  $q_{i,t}^{des}$  是  $t$  步时间里第  $i$  个节点的目标位置。

(10) 扭矩奖励 *Torque Reward*( $r_\tau$ )：我们惩罚节点的扭矩里节省能耗 ( $\tau \propto electric current$ )：

$$r_\tau = -c_k \sum_{i=1}^{12} \tau_i^2 \quad (4-20)$$

其中  $\tau_i$  是执行器网络计算出的第  $i$  个节点的扭矩。

(11) 滑动奖励 *Slip Reward*( $r_{slip}$ )：我们惩罚与地面接触的脚的速度以减少滑

动情况：

$$r_{slip} = -c_k \sum_{l \in \{foot \text{ in contact}\}} v_{f,l^2} \quad (4-21)$$

其中  $v_{f,l^2}$  是与地面接触了的第  $l$  只脚的速度。

#### 4.4.6 课程

随着策略性能的提高，我们使用两个课程来提高难度。一个课程使用自适应方法<sup>[20]</sup>调整地形难度，另一个改变元素，如奖励或使用逻辑函数<sup>[19]</sup>应用干扰。

对于地形课程，粒子滤波更新地形参数，使它们仍然具有挑战性，但在策略训练期间的任何时候都可以实现<sup>[20]</sup>。

第二个课程将域随机化的幅度和一些奖励项（关节速度、关节加速度、方向、滑移和大腿和小腿接触）乘以单调递增且渐近趋势为 1 的因子：

$$c_{k+1} = (c_k)^d$$

其中  $c_k$  是第  $k$  次迭代的课程因子， $d \in (0, 1)$  是收敛率。

#### 4.4.7 学生策略训练

在我们训练好一个可以在特权信息的帮助下穿越各种地形教师策略后，我们就可以将其提炼成一个学生策略，该策略只能访问真实 robot 上可用的信息。我们使用与教师策略相同的训练环境，但在学生高度样本观察中添加额外的噪声： $o_t^{student} = (o_t^p, n(o_t^e))$ ，其中  $n(o_t^e)$  是一个用于高度样本输入的噪声模型。该噪声模型模拟了现场部署过程中经常遇到的外部感觉的不同失败案例，具体如下。

当外部感觉中存在较大的噪声时，它变得不可观察；因此，动力学被认为是 POMDP。此外，由于缺乏直接测量的传感器，特权状态是不可观察的。因此，该策略需要考虑顺序相关性来估计不可观察的状态。我们建议使用循环信念状态编码器来组合外部感知和本体感觉的序列，以估计不可观察的状态作为信念状态。

学生策略由循环信念状态编码器和 MLP 组成，如图4-3所示。我们用  $h_t$  表示循环网络的隐藏状态。信念状态编码器接收  $o_t^{student}, h_t$  并输出一个潜在向量  $b_t$ ，它表示信念状态。我们的目标是将信念状态  $b_t$  与编码所有运动相关信息的教师策略的特征向量 ( $l_t^e, l_t^{priv}$ ) 进行匹配。接下来我们将  $o_t^p$  和  $b_t$  传入 MLP，由它计算出最终的动作。MLP 结构与教师策略相同，这样我们就可以重用教师策略的学习权重来初始化学生网络并加快训练速度。

学生策略的训练通过最小化两个损失以有监督的方式进行训练：行为克隆损失 *behavior cloning loss* 和重建损失 *reconstruction loss*。

- 克隆损失定义为给定相同状态和指令的学生动作和教师动作之间的平方距离。
  - 重建损失定义为无噪声高度样本和特权信息 ( $o_t^e, s_t^p$ ) 及其与信念状态  $b_t$  的重建之间的平方距离。
- 我们通过推出学生策略来生成样本，以提高鲁棒性<sup>[32-33]p</sup>。

#### 4.4.8 高度采样随机化

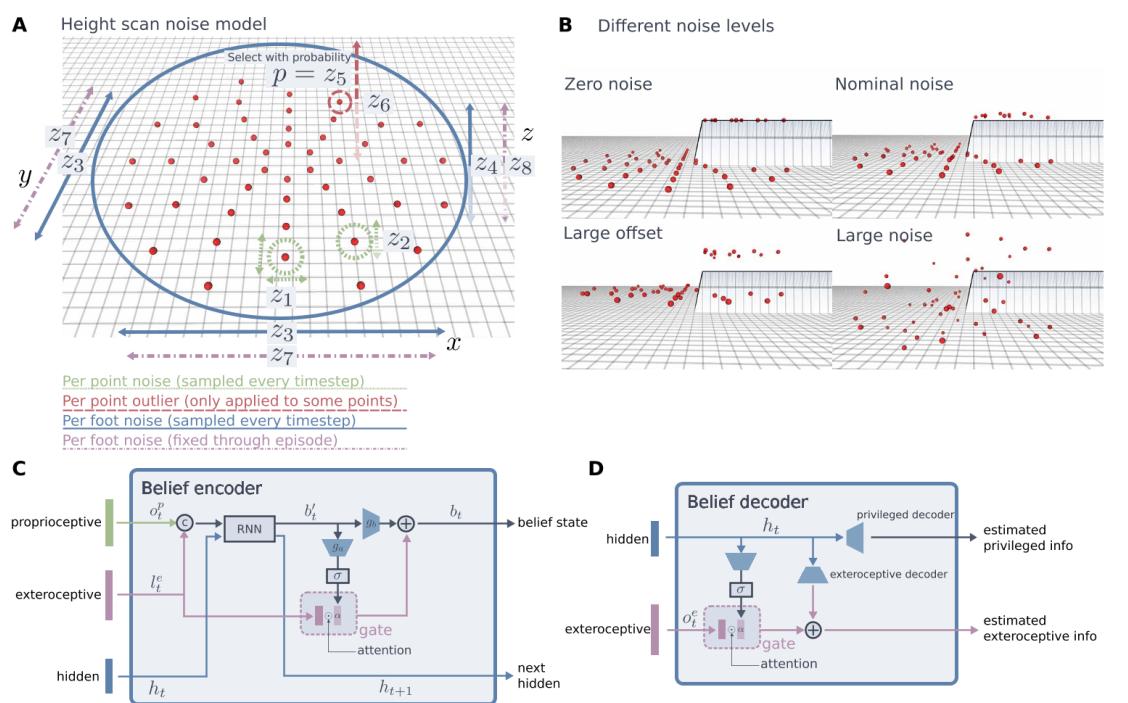


图 4-4 Robust terrain perception<sup>[28]p10</sup>。

在学生训练过程中，我们使用参数化噪声模型  $n(\sigma_t^e | o_t^e, z)$ ,  $z \in \mathbb{R}^{8 \times 4}$  将随机噪声注入到高度样本中。我们在对高度进行采样时应用了两种不同类型的测量噪声，如图4-4A 所示：

- (1) 横向移动扫描点。
- (2) 扰动高度值。

每个噪声值都是从高斯分布中采样的，噪声参数  $z$  定义方差。这两种类型的噪声都应用于三个不同的范围，所有这些都有自己的噪声方差：每个扫描点、每只脚和每一集。每个扫描点和每只脚的噪声值在每个时间步重新采样，而所有扫描点的表观轮廓噪声保持不变。

此外，我们定义了三个具有相关噪声参数  $z$  的映射条件来模拟不断变化的地图质量和误差源，如图4-4B 所示。

- (1) 名义噪声假设常规操作期间具有良好的地图质量。

- (2) 大偏移量噪声来模拟由于姿态估计漂移或可变形地形造成地图偏移。
- (3) 幅度较大的噪声来模拟由于遮挡或映射失败导致完全缺乏地形信息的情况。

这三个映射条件在每个训练集的开头以 60%、30% 和 10% 的比例选择。

最后，我们将每个训练地形划分为单元格，并向高度样本添加附加偏移量，具体取决于它采样的单元格。这模拟了不同地形特征的区域之间的转换，如植被和深度雪。参数向量  $z$  也是学习课程的一部分，其幅度随训练持续时间线性增加。

**更详细相关内容看相关附件的 S8。**

在学生训练期间，我们随机化每只脚周围绘制的高度样本（图4-4A）。我们扰动每个样本的位置，并将噪声添加到测量的高度值中，如下所示。

$$x_p = r_p \cos(\theta_p) + \epsilon_{px} + \epsilon_{fx} + w_x \quad (4-22)$$

$$y_p = r_p \sin(\theta_p) + \epsilon_{py} + \epsilon_{fy} + w_y \quad (4-23)$$

$$h_p = h(x_p, y_p) + \epsilon_{pz} + \epsilon_{fz} + w_z + \epsilon_{outlier} \quad (4-24)$$

其中  $h(x_p, y_p)$  表示在点  $(x_p, y_p)$  处的高度， $r_p$  是  $p$  点的径向距离， $\theta_p$  是  $p$  在脚周围的极坐标中的方位角。 $\epsilon_{px}, \epsilon_{py}, \epsilon_{pz}$  表示每个时间步长里各个点的采样噪声。 $\epsilon_{fx}, \epsilon_{fy}, \epsilon_{fz}$  表示每个时间步长里各个脚的采样噪声。 $w_x, w_y, w_z$  表示每个片段里各个脚的采样噪声。 $\epsilon_{outlier}$  是间歇性添加的大噪声来模拟异常值。

使用参数  $z$  从正态分布中采样每个噪声。 $\epsilon_{px}, \epsilon_{py} \sim \mathcal{N}(0, z_0), \epsilon_{pz} \sim \mathcal{N}(0, z_1), \epsilon_{fx}, \epsilon_{fy} \sim \mathcal{N}(0, z_2), \epsilon_{fz} \sim \mathcal{N}(0, z_3), \epsilon_{outlier} \sim \mathcal{N}(0, z_4)$  概率分别为  $p = z_5, w_x, w_y \sim \mathcal{N}(0, z_6), w_z \sim \mathcal{N}(0, z_7)$ 。

我们为学生训练定义三种情况：*nominal*, *offset*, *noisy*。每种情况的参数  $z$  定义如下：

$$z_{nominal} = \langle 0.004, 0.005, 0.01, 0.04, 0.03, 0.05, 0.1 \rangle \quad (4-25)$$

$$z_{offset} = \langle 0.004, 0.005, 0.01, 0.1c_{sk}, 0.1c_{sk}, 0.02, 0.1 \rangle \quad (4-26)$$

$$z_{noisy} = \langle 0.004, 0.1c_{sk}, 0.1c_{sk}, 0.3c_{sk}, 0.3c_{sk}, 0.3c_{sk}, 0.1 \rangle \quad (4-27)$$

其中  $c_{sk}$  是学生课程因子，会随着训练片段增加而不断线性增加。我们在轨迹的开头和中间随机选择其中一个条件。选择的概率分别是 60%、30% 和 10%。

#### 4.4.9 信念状态寄存器

循环信念状态编码器编码不能直接观察到的状态。为了整合本体感受和外感受数据，我们引入了一个门控编码器，如图 7C 所示，灵感来自门控 RNN 模型 [34-35]p 和多模态信息融合 (4-66)。

信念状态编码器学习使用一个可变的门控因子来控制外部感知信息通过的量。首先，内部感知  $s_t^p$ 、从含噪声观测提取的外部感知  $l_t^e = g_e(\tilde{o}_t^e)$  以及隐藏状态  $s_t$  被 RNN 模型编码成为一个中间信念状态  $b_{t'}^t$ 。它控制最终进入  $b_t$  的外部感知信息量：

$$b_{t'}, h_{t+1} = \text{RNN}(o_t^p, l_t^e, h_t) \quad (4-28)$$

$$\alpha = \sigma(g_a(b_{t'})) \quad (4-29)$$

$$b_t = g_b(b_{t'}) + l_t^e \odot \alpha \quad (4-30)$$

这里  $g_a, g_b$  是全连接的神经网络， $\sigma(\cdot)$  是 sigmoid 函数。

解码器使用相同的门，用于重建特权信息和高度样本（图 7D）。这用于计算重建损失，它鼓励信念状态捕获有关环境的真实信息。我们使用 GR<sup>[34]p</sup> 作为我们的 RNN 架构。

~~门结构有效性的评估见第 S9 节。~~

#### 4.4.10 部署

我们在 PyTorch<sup>[36]p</sup> 中训练策略，并在没有任何微调的情况下部署在机器人 zero-shot 上。我们通过估计机器人的姿态，并相应地从传感器中调节点云读数，构建了一个以机器人为中心的 2.5D 高程图刷新率为 20 赫兹。该策略以 50Hz 运行，并从最新的高程图中映射采样高度；如果查询位置没有可用的地图信息，则填充随机采样的值。

我们开发了一个高程映射管道，用于在图形处理单元上快速地形映射，以并行化点云处理。我们遵循与 Fankhauser 等人<sup>[37]p</sup> 使用的类似方法，以卡尔曼滤波的方式更新地图，另外按漂移补偿 *drift compensation* 和光线投射 *ray casting* 以获得更吻合的地图。这种快速映射实现对于保持快速处理速率和跟上我们的控制器实现的快速运动速度至关重要。

### 4.5 电机驱动关节点模型<sup>[4]p4</sup>

~~基于参考文献阐述电机驱动关节点模型描述...~~

## 第 5 章 强化学习仿真平台

**Addition 5.0.1.** 两个参考的 RL 深度学习案例库：

*isaacgym* 库

*legged\_gym* 库

### 5.1 关于各种机器人仿真平台

目前比较流行的机器人仿真平台有 MuJoCo<sup>[38]p</sup>、PyBullet[7]、DART<sup>[39]p</sup>、Drake[9]、V-Rep<sup>[40]p</sup>等，这些物理引擎需要大量的 CPU 集群来解决具有挑战性的 RL 任务，它们的应用因此面临着瓶颈。例如，在这篇文献<sup>[41]p</sup>中，几乎 30,000 个 CPU 内核（920 个工人机器，每个 32 个内核）用于训练机器人以使用 RL 解决 Rubik 的 Cube 任务。在类似的任务中，这篇文献<sup>[42]p</sup>使用了一组 384 个系统，具有 6144 个 CPU 内核，加上 8 个 NVIDIA V100 GPU，并且需要 30 小时的训练 RL 才能收敛。这其中的主要原因是 CPU 计算并行能力的限制以及在有 GPU 参与的情况下 CPU-GPU 之间通信速度的限制。GPU 在图形计算和仿真方面有这很大的优势，因此我们希望能够将仿真和深度学习策略同时在 GPU 上运行，仅在 CPU 上定义实验流程和获取结果反馈。这在英伟达提供的 *Isaac Gym* 平台得到了实现。

### 5.2 关于强化学习平台 Isaac<sup>[43]p4-10</sup>

*Isaac Gym* 提供了一个高性能的学习平台，可以直接在 GPU 上训练各种机器人任务的策略。物理模拟和神经网络策略训练都在 GPU 上进行，并通过直接将物理缓冲区的数据以 PyTorch 张量的形式进行通信，避免了与 CPU 配合带来的通信瓶颈。与使用基于 CPU 的模拟器结合 GPU 的神经网络的传统 RL 训练相比，在单个 GPU 上分解复杂机器人任务的快速训练时间能够提高 2 – 3 个数量级<sup>[43]p1</sup>。

#### 5.2.1 关于 Isaac Gym

*Isaac Gym* 是一个端到端 (*end-to-end*) 的高性能机器人仿真平台。它运行端到端的 GPU 加速训练管道，这使研究人员能够克服上述限制，并在连续控制任务中实现 2 – 3 个数量级的训练加速。*Isaac Gym* 利用 NVIDIA PhysX[13] 来提供 GPU 加速的模拟后端，使其能够以仅使用高度并行性实现的速率收集机器人 RL 所需

的经验数据。它提供了一个基于 PyTorch 张量的 API 来在本地访问 GPU 上的物理模拟结果。观察张量可以用作策略网络的输入，得到的动作结果张量可以直接反馈到物理系统中。

图 5-1 Overview of the Isaac's workflow<sup>[43]p5</sup>. 张量 API 提供从 Python 代码到 PhysX 后台的接口，同时可以直接获得和设置 GPU 上仿真器的状态值，也提供了与已存在机器人模型的交互功能。

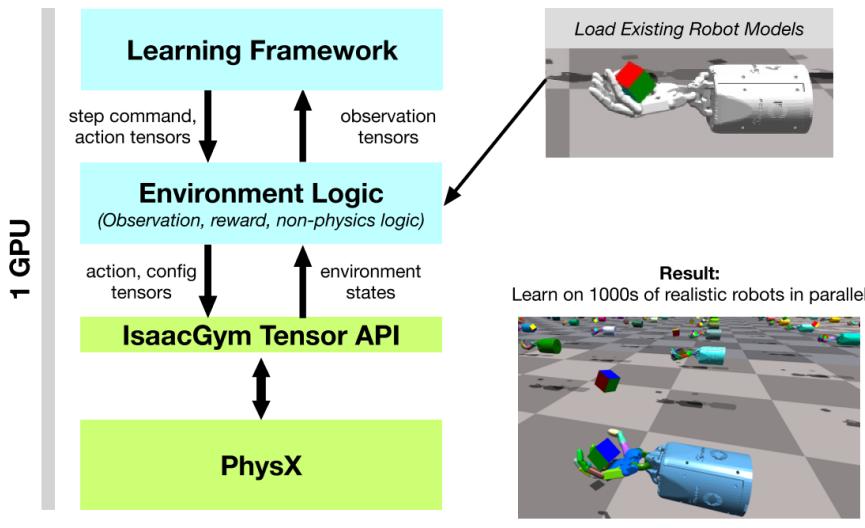


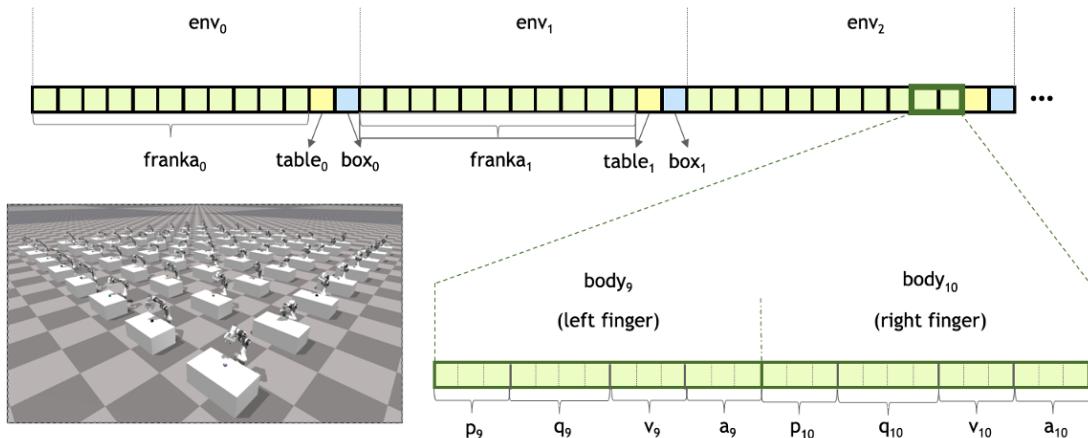
图5-1展示了整个 *Isaac Gym* 系统的工作架构，*Isaac Gym* 提供了一个直接 API，用于创建和填充带有机器人和对象的场景，支持常见的 URDF 和 MJCF 文件格式来加载数据。每个环境可以根据需要重复多次，同时保留副本之间变化的能力（例如通过域随机化<sup>[44]p</sup>）。各个环境同时模拟，与其它环境互不影响。*Isaac Gym* 还包括基本的近端策略优化 (PPO) 实现和一个简单的 RL 任务系统，但用户可以根据需要替代任务系统或 RL 算法。

## 5.2.2 在 Isaac 上建立仿真

### 5.2.3 张量 API

*Isaac Gym* 在物理引擎上提供数据抽象层。这允许我们使用共享的前端 API 支持多个物理引擎，不仅限于 *PhysX*。

用户不是直接调用物理引擎函数，而是可以访问平面缓冲区中的所有物理数据。这种面向数据的方法使我们能够消除许多开销，这是由于在用户代码中循环超过数万个单个模拟参与者造成的。物理状态暴露于 Python 用户作为全局张量。例如，所有刚体状态都可以在单个刚体状态张量中找到。图5-2显示了一个典型的 *Isaac Gym* 场景，由相同环境的各种副本组成，模拟并行运行的不同变体以及与它相关的相应张量。控制输入也可以使用全局张量应用。例如，将力应用于模拟中

图 5-2 View of tensors Isaac<sup>[43]p8</sup>.

的所有刚体可以使用单个函数调用完成，该函数调用取包含所有力的张量。用户可以创建全局张量的自定义视图或切片以适应他们的需求。当将多个环境实例打包到模拟中时，可以使用环境索引作为维度之一来创建数据的自定义视图。通过在多个环境中并行运行 GPU 内核，这使得向量化观察和奖励计算变得容易。

### 5.2.3.1 Python 接口

*Isaac Gym* 的核心是使用 *C++* 和 *CUDA* 实现的。它完全独立于机器学习中常用的任何 *Python* 框架。为了使 *Python* 用户可以轻松访问数据，*Isaac Gym* 提供实用程序，可以将原始数据缓冲区“包装”为 *PyTorch* 等常见机器学习框架中的张量对象。张量包装实用程序使得在没有任何复制开销的情况下使用 *Python* 共享本地 CPU 或 GPU 缓冲区成为可能。*Isaac Gym* 的一个强大功能是通过简单地打乱标志在 CPU 或 GPU 上运行相同代码的能力。*Python* 用户不需要编写自定义 *C++* 或 *CUDA* 内核来计算观察、奖励或动作。当物理状态和控制张量被包装为 *PyTorch* 张量时，用户可以利用 *TorchScript JIT* 将他们的 *Python* 函数编译为低级脚本，可以快速编排训练管道。

### 5.2.3.2 物理状态张量

物理状态张量用于获得运行模拟的状态快照。*Isaac Gym* 允许使用最大和最小坐标与模拟交互。物理状态包括刚体的运动学状态和自由度 (DOF)。刚体状态由位置、方向(四元数)、线速度和角速度组成。自由度状态包括位置和速度。在下面的代码片段展示了如何通过 API 访问它们。

转动自由度使用弧度为单位，直线自由度使用米作为单位。其它提供的状态数据包括接触力、刚体力传感器和自由度力传感器。为了支持操作空间控制和逆运

动学应用, *Isaac Gym* 还提供了可用于关节行为体的雅可比矩阵 (*Jacobian Matrix*) 和广义质量矩阵 (*Mass Matrix*)。可用的状态张量列在表5-1中

表 5-1 Physics state tensors.  $N_A$  is the total number of actors,  $N_B$  is the total number of rigid bodies (including articulation links),  $N_D$  is the total number of degrees of freedom, and  $N_F$  is the total number of rigid body force sensors.

Tensor	Description	Shape	Usage
Actor root state	State of all actor root bodies (position, orientation, linear and angular velocity).	$(N_A, 13)$	Get/Set
DOF state	State of all degrees of freedom (position, velocity).	$(N_D, 2)$	Get/Set
Rigid body state	State of rigid bodies (position, orientation, linear and angular velocity).	$(N_B, 13)$	Get
DOF forces	Net forces experienced at each degree of freedom.	$N_D$	Get
Rigid body forces	Rigid body forces and torques experienced at force sensor locations.	$(N_F, 6)$	Get
Net contact forces	Net forces experienced by each rigid body.	$(N_B, 3)$	Get
Jacobian matrix	Jacobian matrices for a homogeneous group of actors.	Variable	Get
Mass matrix	Generalized mass matrices for a homogeneous group of actors.	Variable	Get

### 5.2.3.3 物理控制张量

物理模拟输入包括力、力矩和 PD 控制, 如位置和速度目标。力和力矩可以应用于刚体和自由度。PD 目标应用于配置为使用位置或速度驱动器的自由度。用户可以使用单独的 API 配置刚度和阻尼等驱动参数。表5-2列出了可用的控制张量。控制张量通常在 PyTorch 等更高级别的框架中创建, 但可以使用张量包装实用程序与 Isaac Gym 有效地共享。

表 5-2 Physics control tensors.  $N_B$  is the total number of rigid bodies (including articulation links),  $N_D$  is the total number of degrees of freedom.

Tensor	Description	Shape	Applied to
DOF actuation forces	Torques or linear forces to be applied to degrees of freedom.	$N_D$	All actors or indexed subset
DOF position targets	PD position targets for degrees of freedom.	$N_D$	All actors or indexed subset
DOF velocity targets	PD velocity targets for all degrees of freedom.	$N_D$	All actors or indexed subset
Rigid body forces	Forces to be applied to rigid bodies.	$(N_B, 6)$	All rigid bodies
Rigid body torques	Torques to be applied to rigid bodies.	$(N_B, 3)$	All rigid bodies

**Addition 5.2.1.** 关于如何使用 Python 接口访问和获取这些物理状态张量 (表 5-1) 和物理控制张量 (表 5-2) 将在下面的章节中介绍。

### 5.3 物理模拟器

机器人模拟使用减少坐标关节的物理引擎PhysX。任何单个刚体都可以用最大坐标刚体或单连杆简化坐标关节来模拟。具有单连杆的关节和刚体的是等效的和可互换的。表 5-3 描述了向用户公开的用于调优模拟器的各种参数。

表 5-3 Parameters exposed to users to tune the simulator.

Parameter	Description
Delta time (dt)	Control time-step size
Gravity	Control the gravity in the scene
Collision filtering	Filters Collisions between shapes
Position iteration	Biased (velocity + position error correcting) solver iterations
Velocity iterations	Unbiased (velocity error only correcting) solver iterations
Max bias coefficient	Limits the magnitude of position error bias Friction
Restitution	Controls bounce
Static/dynamic friction	Static and dynamic friction coefficients
Bounce threshold	Relative normal velocity limit below which restitution is ignored
Rest offset	Distance at which shapes are held separated. Default is 0 but can be increased to hold objects at gap. Useful for thin objects.
Friction offset threshold	Distance at which shapes are held separated. Default is 0 but can be increased to hold objects at gap. Useful for thin objects.
Friction offset threshold	Distance at which friction anchors are discarded (static friction depends on friction anchor caching)
Solver offset slop	An epsilon value used to correct for round-off errors in contact gen. Corrects small skew effects with rolling spheres or capsules.
Friction correlation distance	Distance at which contacts are merged into a single friction constraint
Max force	Per-body and per-contact force limits
Drive stiffness	Positional error correction coefficient of a PD controller
Drive damping	Velocity error correction coefficient of a PD controller
Joint friction Per-joint friction term. Simulates dry friction in a joint.	
Joint armature	Per-joint armature term - simulates motor inertia.
Body/link Damping	World-space linear/angular damping on each body/link
Max velocity	Linear/angular limit per-body

## 第6章 强化学习案例 legged\_gym 库

### 6.1 legged\_gym 工程文件结构<sup>[45]p1</sup>

`legged_gym`是ETH机器人研究团队开发的基于英伟达 `isaac_gym` 平台的强化学习 (*Reinforcement Learning, RL*) 案例库，提供了包括 A1、ANYmal-b、ANYmal-c 等型号的机器人的 RL 训练模型。

图 6-1 `legged_gym` 整体工程文件结构

```
pdj@pdj:~/legged_gym_env/legged_gym$ tree -L 2
.
├── legged_gym
│   ├── envs
│   ├── __init__.py
│   ├── __pycache__
│   ├── scripts
│   ├── tests
│   └── utils
├── legged_gym.egg-info
│   ├── dependency_links.txt
│   ├── PKG-INFO
│   ├── requires.txt
│   ├── SOURCES.txt
│   └── top_level.txt
└── LICENSE
└── licenses
    ├── assets
    └── dependencies
        └── logs
            ├── a1_pdj
            ├── anymal_cpdj
            ├── flat_anymal_c
            ├── rough_a1
            ├── rough_anymal_b
            ├── rough_anymal_c
            └── rough_cassie
        └── Motion Model.md
    └── README.md
    └── resources
        └── actuator_nets
            └── robots
        └── setup.py

21 directories, 10 files
pdj@pdj:~/legged_gym_env/legged_gym$
```

0. 用于实现深度学习的核心代码

2. RL训练输出的中间结果和策略文件

1. 辅助RL实现的机器人和驱动电机模型资源

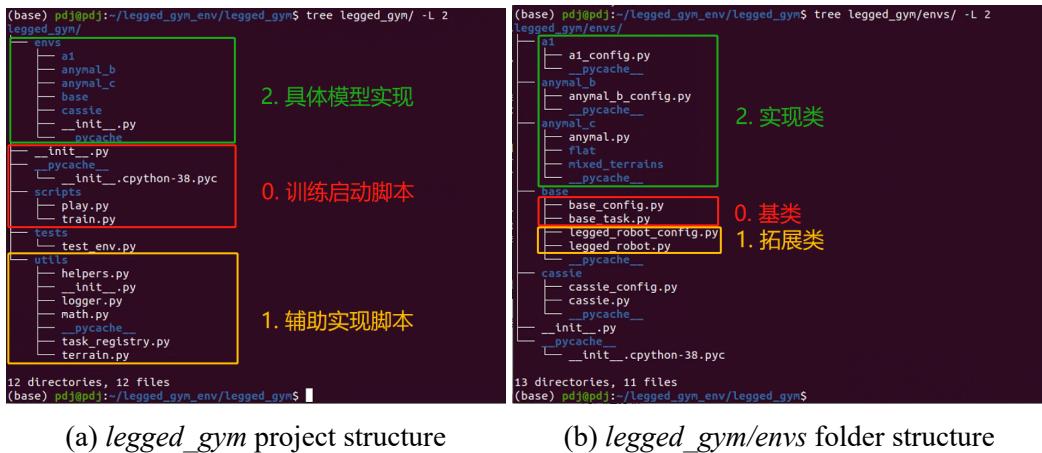
整个例程库的文件结构如图6-1所示，图中标出了主要的三个文件：0. 用于实现深度学习的核心代码；1. 辅助实现 RL 的机器人电机模型和机器人物理模型；2. RL 训练输出的中间结果和最终的策略文件输出，其余的是一些基本的软件环境建立、开源协议等。这其中最重要的就是用于实现深度学习的核心代码部分（图6-1-0.），下面将介绍它的内容。

### 6.1.1 legged\_gym 工具包

legged\_gym 文件是整个 *legged\_gym* 工程中的核心部分，它的内部文件结构如图6-2(a)所示，主要包括三个部分：

- (1) 训练的启动脚本；
- (2) 辅助实现的脚本；
- (3) 具体的模型实现文件；

其中最值得关注的是第 2. 部分关于具体环境模型实现的源码。它的文件路为：*legged\_gym/legged\_gym/envs/\**，内部的具体文件结构如图6-2(b)所示，它包含了各种机器人模型的 RL 训练设置。我们重点关注机械狗类型的实现类（a1, anymal\_b, anymal\_c 6-2(b)-2.），它这些实现都是基于 *legged\_robot\*.py* 中定义的腿式机器人拓展类（6-2(b)-1.）实现，而它的基类是（6-2(b)-0.）类。



实际上整个 RL 训练还涉及到一个由 ETH 团队实现的用于深度学习的辅助库 *rsl\_rl*，关于它的具体内容等待后续另做介绍。在接下来的内容中，我将会具体讲解关于腿式机器人的拓展类：*LeggedRobot* 类。以它为基础来对整个机器人的 RL 训练相关内容建立一个基本的认知。

## 6.2 LeggedRobot 类

在 *legged\_gym* 中用于足式机器人的仿真核心类就是 *LeggedRobot*，它位于 *..../legged\_gym/legged\_gym/envs/legged\_robot.py* 文件中。*LeggedRobot* 的父类是 *BaseTask*，它主要定义了整个 RL 训练过程的基本流程和一些待实现的方法模板，比较简单，这里不做过多介绍。实际上主要的方法实现位于 *LeggedRobot* 中，下面将介绍它内部核心实现函数。

### 6.2.1 从 Isaac 仿真环境中获取信息

*legged\_gym* 是在英伟达的 Isaac 仿真平台上实现训练的。关于 Isaac 平台的介绍在第5.2节中进行了介绍，它能够实现仿真和 RL 训练都在 GPU 平台上运行，拥有者很高效的学习效率。

图 6-3 *function: \_init\_buffers* 这个函数处理了来自仿真环境中的数据；初始化了计算会用到的原始信息和将会生成的相关信息的内存空间；设置了关节点的位置偏移和 PD 增益。

```

177     return noise_vec
178
179
180     #-----
181     def __init__(self):
182         """ Initialize torch tensors which will contain simulation states and processed quantities
183
184         # get gym GPU state tensors
185         actor_root_state = self.gym.acquire_actor_root_state_tensor(self.sim)
186         dof_state_tensor = self.gym.acquire_dof_state_tensor(self.sim)
187         net_contact_force_tensor = self.gym.acquire_net_contact_force_tensor(self.sim)
188         self.gym.refresh_dof_state_tensor(self.sim)
189         self.gym.refresh_actor_root_state_tensor(self.sim)
190         self.gym.refresh_net_contact_force_tensor(self.sim)
191
192         # create some wrapper tensors for different slices
193         self.root_states = gymtorch.wrap_tensor(actor_root_state)
194         self.dof_state = gymtorch.wrap_tensor(dof_state_tensor)
195         self.dof_pos = self.dof_state.view(self.num_envs, self.num_dof, 2)[..., 0]
196         self.dof_vel = self.dof_state.view(self.num_envs, self.num_dof, 2)[..., 1]
197         self.base_quat = self.root_states[:, 3:7]
198
199         self.contact_forces = gymtorch.wrap_tensor(net_contact_force_tensor).view(self.num_envs, -1, 3) # shape: num_envs, num_bodies, xyz axis
200
201         # initialize some data used later on 3. 直接获取到的张量是集成度很高的量，使用时需要提取目标值；
202         self.common_step_counter = 0
203         self.extras = {}
204         self.noise_scale_vec = self.get_noise_scale_vec(self.cfg)
205         self.gravity_vec = to_torch(get_axis_params(-1., self.up_axis_idx), device=self.device).repeat((self.num_envs, 1))
206         self.forward_vec = to_torch([1., 0., 0.], device=self.device).repeat((self.num_envs, 1))
207         self.torques = torch.zeros(self.num_envs, self.num_actions, dtype=torch.float, device=self.device, requires_grad=False)
208         self.p_gains = torch.zeros(self.num_actions, dtype=torch.float, device=self.device, requires_grad=False)
209         self.d_gains = torch.zeros(self.num_actions, dtype=torch.float, device=self.device, requires_grad=False)
210         self.actions = torch.zeros(self.num_envs, self.num_actions, dtype=torch.float, device=self.device, requires_grad=False)
211         self.last_actions = torch.zeros(self.num_envs, self.num_actions, dtype=torch.float, device=self.device, requires_grad=False)
212         self.last_dof vel = torch.zeros_like(self.dof vel)
213         self.last root vel = torch.zeros_like(self.root_states[:, 7:13]) 3.1 比如这里挑选7, 8, 9, 10, 11, 12列本体姿态；
214         self.commands = torch.zeros(self.num_envs, self.cfg.commands.num_commands, dtype=torch.float, device=self.device, requires_grad=False)
215         self.commands_scale = torch.tensor([self.obs_scales.lin vel, self.obs_scales.lin vel, self.obs_scales.ang vel], device=self.device)
216         self.feet_air_time = torch.zeros(self.num_envs, self.feet_indices.shape[0], dtype=torch.float, device=self.device, requires_grad=False)
217         self.last_contacts = torch.zeros(self.num_envs, len(self.feet_indices), dtype=torch.bool, device=self.device, requires_grad=False)
218         self.base lin vel = quat_rotate_inverse(self.base quat, self.root_states[:, 7:10])
219         self.base ang vel = quat_rotate_inverse(self.base quat, self.root_states[:, 10:13]) 3.2 具体来说7, 8, 9表示的是线速度；10, 11, 12表示的是转动速度；
220         self.projected_gravity = quat_rotate_inverse(self.base quat, self.gravity_vec)
221         if self.cfg.terrain.measure_heights:
222             self.height_points = self._init_height_points()
223         self.measured_heights = 0
224
225         # init positions offsets and PD gains
226
227     #-----
```

Isaac 仿真平台提供 Python 接口来访问 GPU 中的数据，数据的交互是通过 PyTorch 张量实现的。如图6-3所示，这是在内存初始化函数中将 isaac 仿真平台中的实例的状态张量进行初始化（6-3-0.），并通过包裹成 PyTorch 张量的形式进行使用（6-3-1.）。由于 Isaac 仿真平台提供的张量是集成度很高的格式，再具体使用的时候还需要对数据进行有目的选择6-3-2.。这些数据的结构在节中也进行了简单的介绍，更具体的实现可以参考：[IsaacGymEnvs](#)、[nvidia-isaac-gym](#)。

### 6.2.2 LeggedRobot.step 函数

在整个 LeggedRobot 类中 *step* 函数是一个提纲挈领性的函数，它调用其余各个部分函数组件实现仿真的循环。外部函数对 LeggedRobot 类的激活就是通过调用其中的 *step* 函数实现的。在 *legged\_gym* 工程中 *step* 函数的调用出现在这几个地

方：

- (1) legged\_gym/legged\_gym/envs/base/base\_task.py 中的 *BaseTask* 类；
- (2) legged\_gym/legged\_gym/scripts/play.py 中的 *play* 函数；
- (3) legged\_gym/legged\_gym/scripts/test\_env.py 中的 *test\_env* 函数；

图 6-4 LeggedRobot.step 函数

```

78     def step(self, actions):
79         """ Apply actions, simulate, call self.post_physics_step()
80
81         Args:
82             actions (torch.Tensor): Tensor of shape (num_envs, num_actions_per_env)
83
84             clip_actions = self.cfg.normalization.clip_actions
85             self.actions = torch.clip(actions, -clip_actions, clip_actions).to(self.device)
86             # step physics and render each frame
87             self.render()
88             for _ in range(self.cfg.control.decimation):
89                 self.torques = self._compute_torques(self.actions).view(self.torques.shape)
90                 self.gym.set_dof_actuation_force_tensor(self.sim, gymtorch.unwrap_tensor(self.torques))
91                 self.gym.simulate(self.sim)
92                 if self.device == 'cpu':
93                     self.gym.fetch_results(self.sim, True)
94                     self.gym.refresh_dof_state_tensor(self.sim)
95             self.post_physics_step()
96
97             # return clipped obs, clipped states (None), rewards, dones and infos
98             clip_obs = self.cfg.normalization.clip_observations
99             self.obs_buf = torch.clip(self.obs_buf, -clip_obs, clip_obs)
100            if self.privileged_obs_buf is not None:
101                self.privileged_obs_buf = torch.clip(self.privileged_obs_buf, -clip_obs, clip_obs)
102            return self.obs_buf, self.privileged_obs_buf, self.rew_buf, self.reset_buf, self.extras
103
104

```

如图所示，*step* 函数的功能主要通过调用一下几个函数实现：

- (1) *self.render*
- (2) *self.compute\_torques*
- (3) *self.gym.set\_dof\_actuation\_force\_tensor*
- (4) *self.gym.simulate*
- (5) *self.gym.fetch\_results*
- (6) *self.gym.refresh\_dof\_state\_tensor*
- (7) *self.post\_physics\_step*

这其中比较关于仿真环境的 *self.gym.xxx* 我们暂时不讨论，下面主要关注这三个：*self.render*; *self.compute\_torques*; *self.post\_physics\_step*。

### 6.2.3 self.render 函数

如图6-5所示，*self.render* 函数在 *LeggedRobot* 的父类 *BaseTask* 中定义和实现，主要的作用就是处理 CPU 端监测、界面和键盘交互的一些操作，比如关闭窗口等。

图 6-5 LeggedRobot.render 函数

```

119     def render(self, sync_frame_time=True):
120         if self.viewer:
121             # check for window closed
122             if self.gym.query_viewer_has_closed(self.viewer):
123                 sys.exit()
124
125             # check for keyboard events
126             for evt in self.gym.query_viewer_action_events(self.viewer):
127                 if evt.action == "QUIT" and evt.value > 0:
128                     sys.exit()
129                 elif evt.action == "toggle_viewer_sync" and evt.value > 0:
130                     self.enable_viewer_sync = not self.enable_viewer_sync
131
132             # fetch results
133             if self.device != 'cpu':
134                 self.gym.fetch_results(self.sim, True)
135
136             # step graphics
137             if self.enable_viewer_sync:
138                 self.gym.step_graphics(self.sim)
139                 self.gym.draw_viewer(self.viewer, self.sim, True)
140                 if sync_frame_time:
141                     self.gym.sync_frame_time(self.sim)
142                 else:
143                     self.gym.poll_viewer_events(self.viewer)

```

#### 6.2.4 self.compute\_torques 函数

如图6-6所示，*self.\_compute\_torques* 函数的作用是根据输入的控制动作计算机器人各个关节点的扭矩，目标的的控制可能是基于位置的、基于速度的或者直接基于缩放后的扭矩的。采用的控制方法是 PD 控制。计算后的扭矩结果会被发送到仿真环境中。

图 6-6 LeggedRobot.\_compute\_torques 函数

```

352     def _compute_torques(self, actions):
353         """ Compute torques from actions.
354         Actions can be interpreted as position or velocity targets given to a PD controller, or directly as scaled torques.
355         [NOTE]: torques must have the same dimension as the number of DOFs, even if some DOFs are not actuated.
356
357         Args:
358             actions (torch.Tensor): Actions
359
360         Returns:
361             [torch.Tensor]: Torques sent to the simulation
362         """
363
364         #pd controller
365         actions_scaled = actions * self.cfg.control.action_scale
366         control_type = self.cfg.control.control_type
367         if control_type=="P":
368             torques = self.p_gains*(actions_scaled + self.default_dof_pos - self.dof_pos) - self.d_gains*self.dof_vel
369         elif control_type=="V":
370             torques = self.p_gains*(actions_scaled - self.dof_vel) - self.d_gains*(self.dof_vel - self.last_dof_vel)/self.sim_params.dt
371         elif control_type=="T":
372             torques = actions_scaled
373         else:
374             raise NameError(f"Unknown controller type: {control_type}")
375         return torch.clip(torques, -self.torque_limits, self.torque_limits)
376

```

#### 6.2.5 self.post\_physics\_step 函数

如图6-7所示，*self.post\_physics\_step* 函数的主要作用包括监测终止条件、计算观测和奖励等。相关的函数包括：

- (1) *self.\_post\_physics\_step\_callback*
- (2) *self.check\_termination*
- (3) *self.compute\_reward*
- (4) *self.reset\_idx*
- (5) *self.compute\_observations*
- (6) *self.\_draw\_debug\_vis*

图 6-7 LeggedRobot.post\_physics\_step 函数

```

104
105     def post_physics_step(self):
106         """ check terminations, compute observations and rewards
107         calls self._post_physics_step_callback() for common computations
108         calls self._draw_debug_vis() if needed
109
110         self.gym.refresh_actor_root_state_tensor(self.sim)
111         self.gym.refresh_net_contact_force_tensor(self.sim)
112
113         self.episode_length_buf += 1
114         self.common_step_counter += 1
115
116         # prepare quantities
117         self.base_quat[:] = self.root_states[:, 3:7]
118         self.base_lin_vel[:] = quat_rotate_inverse(self.base_quat, self.root_states[:, 7:10])
119         self.base_ang_vel[:] = quat_rotate_inverse(self.base_quat, self.root_states[:, 10:13])
120         self.projected_gravity[:] = quat_rotate_inverse(self.base_quat, self.gravity_vec)
121
122         self._post_physics_step_callback()
123
124         # compute observations, rewards, resets, ...
125         self.check_termination()
126         self.compute_reward()
127         env_ids = self.reset_buf.nonzero(as_tuple=False).flatten()
128         self.reset_idx(env_ids)
129         self.compute_observations() # in some cases a simulation step might be required to refresh some obs (for example body positions)
130
131         self.last_actions[:] = self.actions[:]
132         self.last_dof_vel[:] = self.dof_vel[:]
133         self.last_root_vel[:] = self.root_states[:, 7:13]
134
135         if self.viewer and self.enable_viewer_sync and self.debug_viz:
136             self._draw_debug_vis()
137

```

这些被调用的函数中最值得关注的是第（3）条：*self.compute\_reward* 和 *self.compute\_observations*。*self.compute\_reward* 就是 RL 问题中的奖励计算函数，包含着十几条具体的奖励计算定义。*self.compute\_observations* 就是 RL 问题中的观测状态计算函数。这两个函数的具体定义将在接下来的两节中介绍。

### 6.2.6 self.compute\_observations 函数

如图6-8所示，*self.compute\_observations* 函数的作用包括计算本体感知、添加外部感知、添加为感知噪声。

### 6.2.7 self.compute\_reward 函数

如图6-9所示，*self.compute\_reward* 函数会从一系列的奖励函数中计算总和。这一系列的奖励函数都定义都预先用 *self.\_prepare\_reward\_function* 进行了打包，因此在 *self.compute\_reward* 函数中只需要遍历访问和累加即可。这些函数的具体计算定义也在 *LeggedRobot* 类中，由于数量过多就不一一展示了，大致如图6-10所示。

这些奖励的计算过程会考虑到他们的系数，可以根据不同机器人的情况方便

图 6-8 LeggedRobot.compute\_observations 函数

```
208
209     def compute_observations(self):
210         """ Computes observations
211         """
212         self.obs_buf = torch.cat((self.base_lin_vel * self.obs_scales.lin_vel,
213                                 self.base_ang_vel * self.obs_scales.ang_vel,
214                                 self.projected_gravity,
215                                 self.commands[:, :3] * self.commands_scale,
216                                 (self.dof_pos - self.default_dof_pos) * self.obs_scales.dof_pos,
217                                 self.dof_vel * self.obs_scales.dof_vel,
218                                 self.actions
219                                 ), dim=-1)
220
221         # add perceptive inputs if not blind
222         if self.cfg.terrain.measure_heights:
223             heights = torch.clip(self.root_states[:, 2].unsqueeze(1) - 0.5 - self.measured_heights, -1, 1.) * self.obs_scales.height_measurements
224             self.obs_buf = torch.cat((self.obs_buf, heights), dim=-1)
225
226         # add noise if needed
227         if self.add_noise:
228             self.obs_buf += (2 * torch.rand_like(self.obs_buf) - 1) * self.noise_scale_vec
```

图 6-9 LeggedRobot.compute\_reward 函数

```
189
190     def compute_reward(self):
191         """ Compute rewards
192             Calls each reward function which had a non-zero scale (processed in self._prepare_reward_function())
193             adds each terms to the episode sums and to the total reward
194         """
195
196         self.rew_buf[:] = 0.
197         for i in range(len(self.reward_functions)):
198             name = self.reward_names[i]
199             rew = self.reward_functions[i]() * self.reward_scales[name]
200             self.rew_buf += rew
201             self.episode_sums[name] += rew
202         if self.cfg.rewards.only_positive_rewards:
203             self.rew_buf[:] = torch.clip(self.rew_buf[:], min=0.)
204         # add termination reward after clipping
205         if "termination" in self.reward_scales:
206             rew = self._reward_termination() * self.reward_scales["termination"]
207             self.rew_buf += rew
208             self.episode_sums["termination"] += rew
```

图 6-10 LeggedRobot.xxx 奖励函数

```
814  
815     #----- reward functions -----  
816 >     def _reward_lin_vel_z(self): ...  
817  
818 >     def _reward_ang_vel_xy(self): ...  
819  
820 >     def _reward_orientation(self): ...  
821  
822 >     def _reward_base_height(self): ...  
823  
824 >     def _reward_torques(self): ...  
825  
826 >     def _reward_dof_vel(self): ...  
827  
828 >     def _reward_dof_acc(self): ...  
829  
830 >     def _reward_action_rate(self): ...  
831  
832 >     def _reward_collision(self): ...  
833  
834 >     def _reward_termination(self): ...  
835  
836 >     def _reward_dof_pos_limits(self): ...  
837  
838 >     def _reward_dof_vel_limits(self): ...  
839  
840 >     def _reward_torque_limits(self): ...  
841  
842 >     def _reward_tracking_lin_vel(self): ...  
843  
844 >     def _reward_tracking_ang_vel(self): ...  
845  
846 >     def _reward_feet_air_time(self): ...  
847  
848 >     def _reward_stumble(self): ...  
849  
850 >     def _reward_stand_still(self): ...  
851  
852 >     def _reward_feet_contact_forces(self): ...  
853
```

地调节各个奖励的贡献。这些系数初始定义在 *LeggedRobot* 类相应的配置类 *LeggedRobotCfg* 类里面，如图6-11所示。

图 6-11 LeggedRobotCfg.rewards 函数

```
129
130     class rewards:
131         class scales:
132             termination = -0.0
133             tracking_lin_vel = 1.0
134             tracking_ang_vel = 0.5
135             lin_vel_z = -2.0
136             ang_vel_xy = -0.05
137             orientation = -0.
138             torques = -0.00001
139             dof_vel = -0.
140             dof_acc = -2.5e-7
141             base_height = -0.
142             feet_air_time = 1.0
143             collision = -1.
144             feet_stumble = -0.0
145             action_rate = -0.01
146             stand_still = -0.
147             # 15 rewards
148
149     only_positive_rewards = True # if true negative total rewards are clipped at zero (avoids early termination problems)
150     tracking_sigma = 0.25 # tracking reward = exp(-error^2/sigma)
151     soft_dof_pos_limit = 1. # percentage of urdf limits, values above this limit are penalized
152     soft_dof_vel_limit = 1.
153     soft_torque_limit = 1.
154     base_height_target = 1. # p: parameters for inner computing
155     max_contact_force = 100. # forces above this value are penalized
156
```

## 第7章 基于 Isaac 平台的深度学习训练实践

关于强化学习控制的整体理念流程在第4节中已经进行了介绍。对于具体的实现平台和代码实现案例在第5.2节和第6节中也进行了说明。本节将给出一些实际进行 RL 训练的案例，以建立对实际 RL 训练的整体流程性认识。

### 7.1 用 legged\_gym 例程库训练 ANYmal-c

前几节所述的控制方法基本都是以 ETH 的足式机器人展开的，其中很多都是以 ETH 的 ANYmal-c 型号机器人为基础平台实现的。因此这里选择使用该型号的机器人进行实际的训练演示。

#### 7.1.1 训练脚本设置

首先需要按照官网的说明配置好相应的 python、Isaac、CUDA、PyTorch 等基本环境。然后将 *legged\_gym* 的源码下载到本地安装。完成基本的配置后启动训练的操作是比较简单的。如下面代码7-1所示，以我配置的环境为例。启动并进入到训练所需的 conda 环境（`conda`, `conda_leg38` 是自定义的命令行别名），然后运行 `train.py` 脚本并选择相应的训练目标即可。

```

1 aconda # 激活 conda 环境
2 conda_leg38 # 激活深度学习 Python 子环境
3 # 转移到训练脚本目录
4 cd ~/legged_gym_env/legged_gym/legged_gym/scripts
5 # 启动训练脚本，其中--task=xxx 用来选择要开启的训练任务
6 python train.py --task=anymal_cpdj

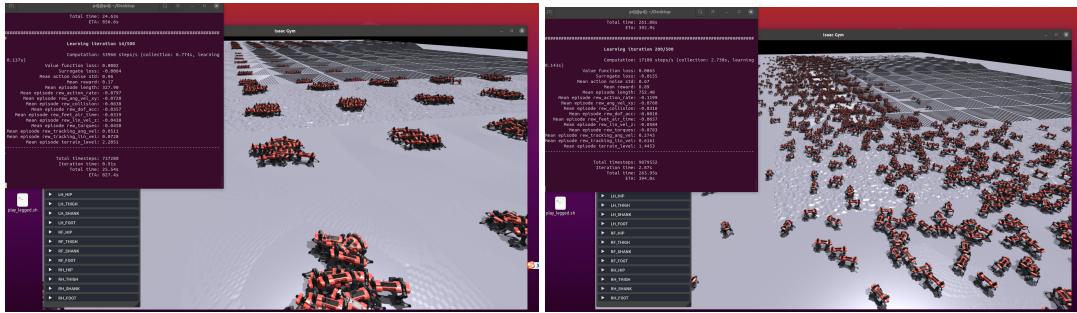
```

代码 7-1 自定义的 RL 训练.sh 快捷启动脚本

#### 7.1.2 训练过程信息

启动训练后会打开一个监视器来观察仿真空间，同时再终端会输出学习过程的一阶基本信息。图片7-1(a)到7-1(d)展示了整个训练过程的几个节点。整个训练方式采用基于游戏的模式进行<sup>[45]p3</sup>，初始时候所有的实例被初始化到不同的地形环境中并行地进行训练，随着训练的进行，表现好的会升级并被安排难度更高的任务，最终训练的结果是整个地图中所有的实例成一定的分布（各个位置都有一

定的分布以保证最终的策略可以较好地处理所有的地形情况和任务难度)。

(a) *legged\_gym* 第 14/500 次迭代。(b) *legged\_gym* 第 200/500 次迭代。(c) *legged\_gym* 第 350/500 次迭代。(d) *legged\_gym* 第 495/500 次迭代。

### 7.1.3 训练结果输出和重演测试

完成训练后启动训练的结果重演测试也是比较简单的。如下面代码7-2所示，以我配置的环境为例。启动并进入到训练所需的 conda 环境 (conda, conda\_leg38 是自定义的命令行别名)，然后运行 *play.py* 脚本并选择相应的训练目标即可。

```

1 aconda # 激活conda环境
2 conda_leg38 # 激活深度学习Python子环境
3 # 转移到训练脚本目录
4 cd ~/legged_gym_env/legged_gym/legged_gym/scripts
5 # 启动训练脚本，其中--task=xxx用来选择要开启的训练任务
6 python play.py --task=anymal_cpdj

```

代码 7-2 自定义的 RL 重演.sh 快捷启动脚本

如图7-2所示，用训练好的策略控制机器人在仿真环境中重演测试。可以看到机器人基本上已经能够无失误地应对各种地形环境，完成诸如上下楼梯、光滑地面行走、崎岖地面行走等任务了。

不过，在面对较难任务时他们可能会犹豫、停顿甚至失败跌倒，如图7-3所示。这些问题可以通过设计更优的奖励机制进行训练、结合更多传感信息进行训练和部署、与传统控制结合起来控制等方式来进行解决。比如前面第4.4节介绍的，在这篇文献<sup>[28]</sup>p4中介绍了使用本体感知和外部感知融合，并使用经典的肢体动作生

图7-2 *legged\_gym*采用训练好的策略控制机器人在仿真环境重演测试：整体上训练得到的策略已经能完成对不同地形的适应。

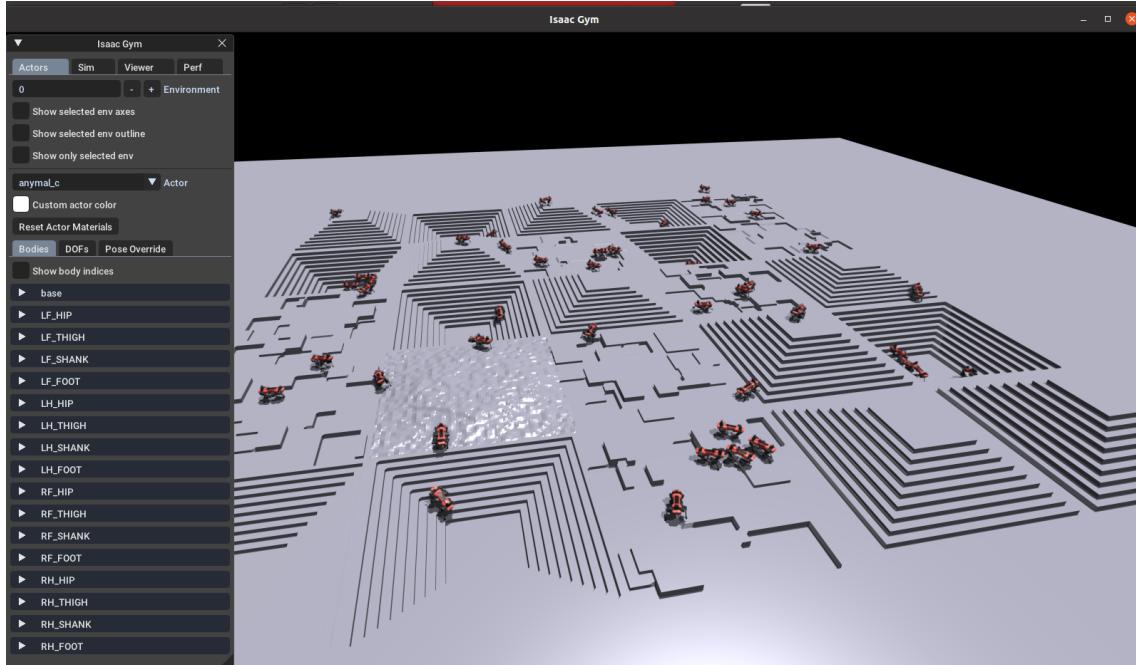
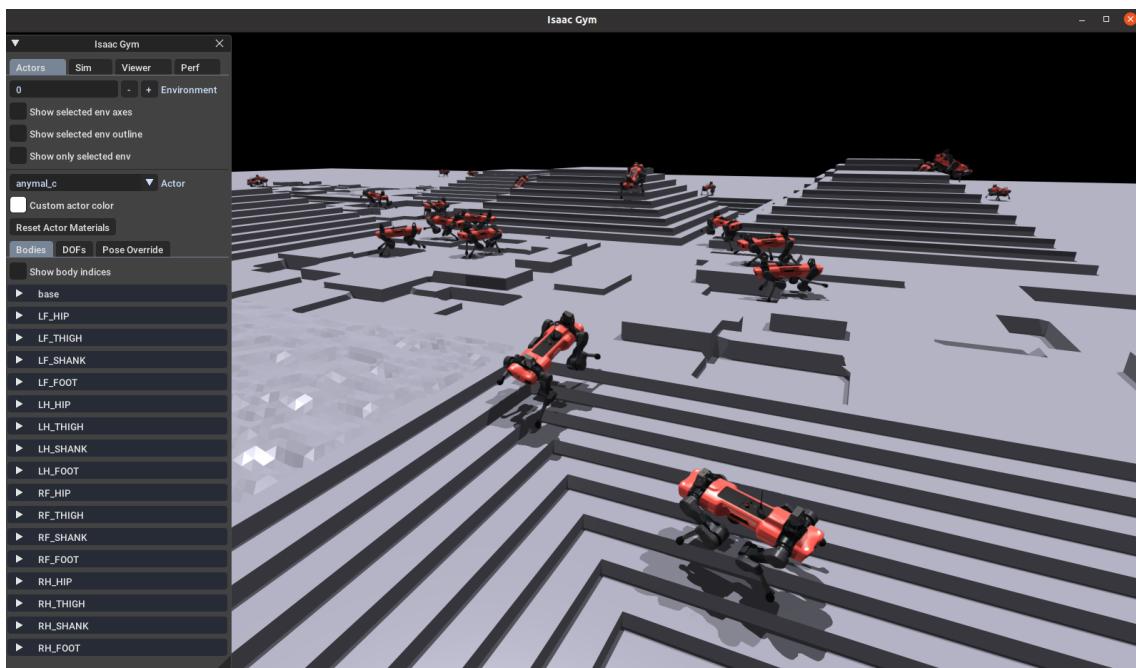


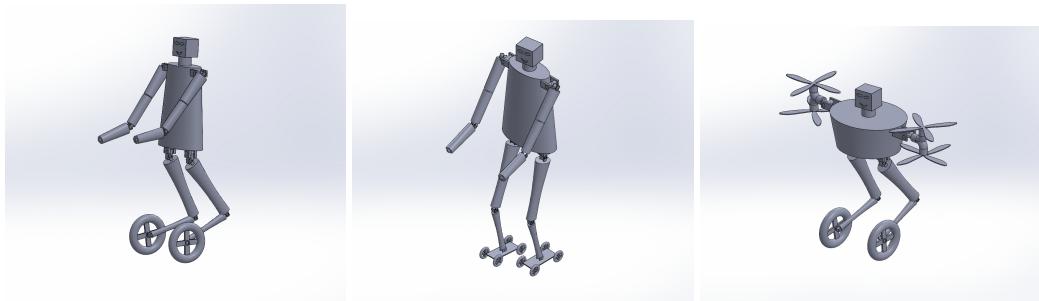
图7-3 *legged\_gym*采用训练好的策略控制机器人在仿真环境重演测试：在完成某些地形情况下的任务时可能会出现犹豫、停顿甚至失败跌倒。



成器进行训练的方法，取得了很好的效果。

在完成重演测试的同时还会将 PyTorch 的策略结果转化成可用于 C++ 实现的形式，输出路径为：`..legged_gym/logs/anymal_cpdj/exported/policies/policy1.pt`，输出格式为：`.pt` 文件。这为后续的仿真到实际 (*sim-to-real*) 做了准备。

## 第8章 几种构想的机器人类型草图



(a) Wheeled humanoid

(b) Roller humanoid

(c) wheeled flying

图 8-1 三种构想机器人的示意图汇总

如图8-1所示，这里给出几种近来构想的机器人结构草图和简单介绍，可以作为随后研究的机器人备选平台参考。

### 8.1 弹跳轮足 + 机械手机器人

如图8-2所示，该类型的机器人由类似于 Ascento<sup>[46]p1</sup> 类型机器人的双轮足，在此基础上添加两个机械臂构成类人形机器人。该类型的机器人可以轮式地行走和跳跃，同时可以用机械臂模仿双手进行各种操作。

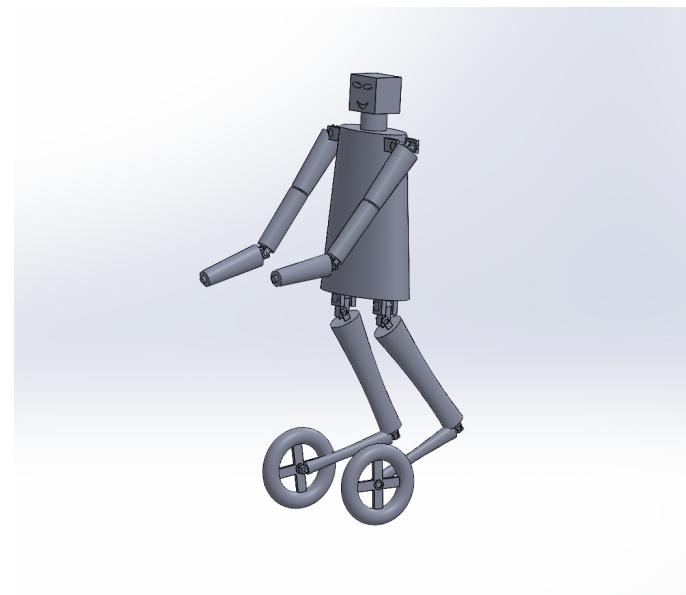


图 8-2 Schematic view of wheeled-humanoid robot

该类型机器人的控制基础是倒立摆模型和机械臂运动学，它相对于现有双轮

机器人，如 Ascento<sup>[46]p1</sup> 等，新的控制问题难点是：

- (1) 在机械手运动的同时保持双轮足的稳定，比如从空手到搬起重物的过程；
- (2) 如何让手臂配合轮足实现更加灵巧和优雅的动作；

## 8.2 轮鞋足式 + 机械手机器人

如图8-3所示，该类型的机器人由类似一般人形机器人的基本结构，在此基础上为每个脚添加四个轮子

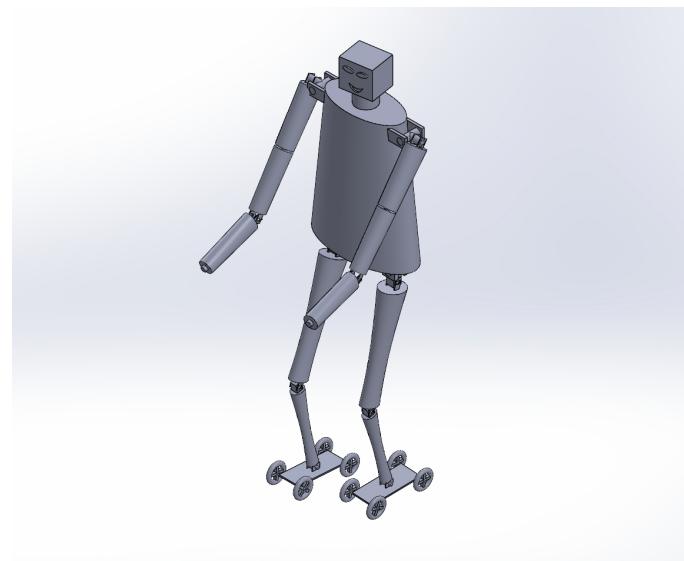


图 8-3 Schematic view of roller humanoid.

该类型机器人的控制基础是 ZMP 约束和机械臂运动学，它相对于现有人形机器人，如<sup>[47,48]p</sup>等，新的控制问题难点是：

- (1) 如何实现步态和轮式之间的运动切换；
- (2) 如何实现步态和轮式的联合运动平衡；
- (3) 在上面两点的基础上，如何结合机械臂实现更加灵巧、优雅、节能的动作；

## 8.3 弹跳轮足 + 飞行器机器人

如图8-4所示，该类型的机器人由类似于 Ascento<sup>[46]p1</sup> 类型机器人的双轮足，在此基础上之上添加一对旋翼使其具备飞行能力。

该类型机器人的控制基础是倒立摆模型和飞行器控制，它相对于现有双轮机器人，如 Ascento<sup>[46]p1</sup> 等，新的控制问题难点是：

- (1) 飞行器的矢量控制；

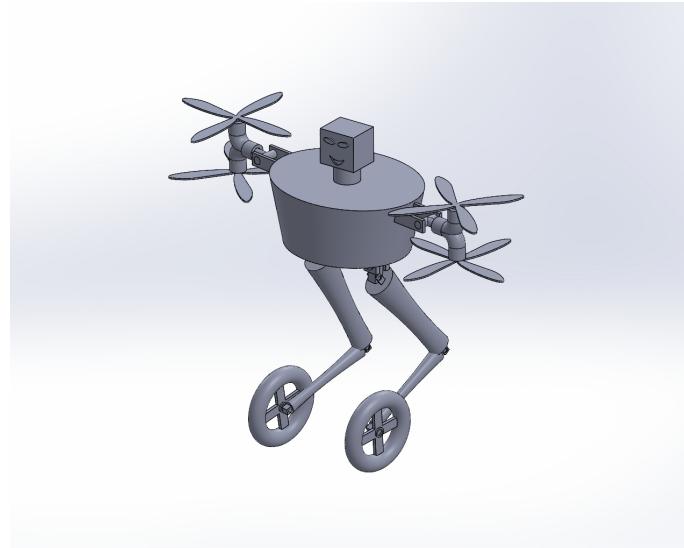


图 8-4 Schematic view of wheeled-flying robot.

- (2) 如何优化设计使得旋翼提供的升力可以满足飞行控制
- (3) 续航安全问题;
- (4) 如何让旋翼配合轮足实现更加灵巧、优雅、节能、稳定地动作;

## 第9章 讨论

整体的感觉是这个领域的研究在没有人指导的情况下一个人突破是比较费劲的，如果没有硬件平台要自己构建的话会更加费劲和耗时。就当前的了解的情况来看，经典学习的问题求解和控制到机器人上的部署是比较有挑战的；而 RL 虽然仿真这步起来看似容易，但它前期的训练和优化策略设置、基于实际机器人建立准确的仿真物理模型和后期的策略的仿真到实际 (*sim-to-real*) 部署和进一步优化也都很有挑战性。如果能到成熟的组里学习一段时间或找到比较懂的人指导一下，实际上手项目来先建立一个整体的研究方法和流程感受是比较好的。有了基本的方法和流程概念后，再进行独立研究和设计新类型的机器人会更有效率。

另外，对于接下来的研究，实现强化学习到实际机器人 ANYmal 上的部署还欠缺的有：

- (1) 一套 ANYmal 机器人硬件；
- (2) 可用的例程代码；
- (3) 电机、控制手柄、各类传感器的驱动和软件接口。
  - 在有直接可用例程的情况下，可以采用例程的架构和电机接口直接部署如第4.2节所述的神经网络策略。
  - 在没有直接可用例程有类似参考例程的情况下，可以花一些时间阅读相关代码，逐步将电机、控制手柄、各类传感器等驱动重构到 ANYmal 上，然后在此基础上设计控制结构实现策略部署。
  - 在没有任何软件接口的情况下至少应该有各种器件相应的手册。会消耗比较长的时间来完成驱动和信息格式的整理和构建。完成后设计控制结构进行策略部署。

(4) 对 Isaac 仿真环境的深入学习，一方面提高对它的理解深度和使用能力，另一方面也需跟进了解其他仿真平台的发展；

- (5) 对 PyTorch 框架的深入学习，加深对算法的掌握，积累策略设计的经验；

对于后续的研究，在上面的基础上需要进一步学习和尝试将经典策略应用到机器人上，以更加深入的理解经典控制方法。与此同时加强对强化学习理论的理解和仿真环境的使用能力，在接下来的研究中构思经典-强化学习融合的控制策略。在此基础上，考虑设计和实现新结构的机器人。

## 结 论

对于经典控制的实现，涉及到复杂的运动模型和动力学建模，难点在于选择优化目标构建优化策略和优化问题的求解。经典控制的优点在于控制过程清晰，可以方便地进行迭代优化和泛化部署；缺点在于建模和实际机器调试复杂，运行时算力开销大。对于强化学习控制的实现，涉及到机器人的仿真模型建模和神经网络设计，难点在于设计合适的优化目标和训练策略以使得训练过程能较快收敛且结果达到预期目标。强化学习控制的优点在于模型设计和调试应用简单迅速，运行时算力开销小；缺点在于控制过程不清晰，导致模型的泛化能力较弱。

经典控制与深度学习控制各有优缺点，在实践应用中两者的有机结合有助于实现更好的控制。在面对一种机器人设计控制策略时，如何分配经典控制和深度学习控制是一个十分重要的问题。两者的融合也是未来的重要发展方向之一。

## 参考文献

- [1] WINKLER A W, BELLICOSO C D, HUTTER M, et al. Gait and Trajectory Optimization for Legged Systems Through Phase-Based End-Effector Parameterization[J/OL]. IEEE Robotics and Automation Letters, 2018: 1560–1567. <http://dx.doi.org/10.1109/lra.2018.2798285>.
- [2] BELLICOSO C, JENELTEN F, FANKHAUSER P, et al. Dynamic locomotion and whole-body control for quadrupedal robots[C/OL]//2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). 2017. <http://dx.doi.org/10.1109/iros.2017.8206174>.
- [3] BELLICOSO C D, JENELTEN F, GEHRING C, et al. Dynamic Locomotion Through Online Nonlinear Motion Optimization for Quadrupedal Robots[J/OL]. IEEE Robotics and Automation Letters, 2018: 2261–2268. <http://dx.doi.org/10.1109/lra.2018.2794620>.
- [4] GEHRING C, COROS S, HUTTER M, et al. An Optimization-based Approach to Controlling Agile Motions for a Quadruped Robot[J/OL]. IEEE Robotics and Automation, 2016. DOI: [10.3929/ethz-a-010644954](https://doi.org/10.3929/ethz-a-010644954).
- [5] KALAKRISHNAN M, BUCHLI J, PASTOR P, et al. Fast, robust quadruped locomotion over challenging terrain[C/OL]//2010 IEEE International Conference on Robotics and Automation. 2010. <http://dx.doi.org/10.1109/robot.2010.5509805>.
- [6] GOLDFARB D, IDNANI A. A numerically stable dual method for solving strictly convex quadratic programs[J/OL]. Mathematical Programming, 1983, 27(1): 1–33. <http://dx.doi.org/10.1007/bf02591962>.
- [7] DARIO BELLICOSO C, GEHRING C, HWANGBO J, et al. Perception-less terrain adaptation through whole body control and hierarchical optimization[C/OL]//2016 IEEE-RAS 16th International Conference on Humanoid Robots (Humanoids). 2016. <http://dx.doi.org/10.1109/humanoids.2016.7803330>.
- [8] PARDO D, NEUNERT M, WINKLER A, et al. Hybrid direct collocation and control in the constraint-consistent subspace for dynamic legged robot locomotion[C/OL]//Robotics: Science and Systems XIII. 2017. <http://dx.doi.org/10.15607/rss.2017.xiii.042>.
- [9] NEUNERT M, STAUBLE M, GIFTTHALER M, et al. Whole-Body Nonlinear Model Predictive Control Through Contacts for Quadrupeds[J/OL]. IEEE Robotics and Automation Letters, 2018: 1458–1465. <http://dx.doi.org/10.1109/lra.2018.2800124>.
- [10] AZAD M, FEATHERSTONE R. A New Nonlinear Model of Contact Normal Force[J/OL]. IEEE Transactions on Robotics, 2014, 30(3): 736–739. <http://dx.doi.org/10.1109/tro.2013.2293833>.
- [11] GIFTTHALER M, NEUNERT M, STÄUBLE M, et al. Automatic Differentiation of Rigid Body Dynamics for Optimal Control and Estimation[J/OL]. Advanced Robotics, 2017, 31(22): 1225–1237. <http://dx.doi.org/10.1080/01691864.2017.1395361>.
- [12] DIEHL M, BOCK H, DIEDAM H, et al. Fast Direct Multiple Shooting Algorithms for Optimal Robot Control[Z]. 2005.

- 
- [13] POSA M, KUINDERSMA S, TEDRAKE R. Optimization and stabilization of trajectories for constrained dynamical systems[C/OL]//2016 IEEE International Conference on Robotics and Automation (ICRA). 2016. <http://dx.doi.org/10.1109/icra.2016.7487270>.
  - [14] PARDO D, MOLLER L, NEUNERT M, et al. Evaluating direct transcription and nonlinear optimization methods for robot motion planning[J/OL]. IEEE Robotics and Automation Letters, 2016, 1(2): 946–953. <http://dx.doi.org/10.1109/lra.2016.2527062>.
  - [15] GIFTTHALER M, NEUNERT M, STÄUBLE M, et al. A Family of Iterative Gauss-Newton Shooting Methods for Nonlinear Optimal Control[A]. 2017.
  - [16] TODOROV E, LI W. A generalized iterative LQG method for locally-optimal feedback control of constrained nonlinear stochastic systems[C/OL]//Proceedings of the 2005, American Control Conference, 2005. 2005. <http://dx.doi.org/10.1109/acc.2005.1469949>.
  - [17] SIDERIS A, BOBROW J. An efficient sequential linear quadratic algorithm for solving nonlinear optimal control problems[C/OL]//Proceedings of the 2005, American Control Conference, 2005. 2005. <http://dx.doi.org/10.1109/acc.2005.1470308>.
  - [18] LEE J, BJELOVIC M, HUTTER M. Control of Wheeled-Legged Quadrupeds Using Deep Reinforcement Learning[M/OL]. 2023: 119–127. [http://dx.doi.org/10.1007/978-3-031-15226-9\\_14](http://dx.doi.org/10.1007/978-3-031-15226-9_14).
  - [19] HWANGBO J, LEE J, DOSOVITSKIY A, et al. Learning agile and dynamic motor skills for legged robots[J/OL]. Science Robotics, 2019. <http://dx.doi.org/10.1126/scirobotics.aau5872>.
  - [20] LEE J, HWANGBO J, WELLHAUSEN L, et al. Learning Quadrupedal Locomotion over Challenging Terrain[J/OL]. Science Robotics, 2020. <http://dx.doi.org/10.1126/scirobotics.abc5986>.
  - [21] CHEN D, ZHOU B, KOLTUN V, et al. Learning by Cheating[J]. Conference on Robot Learning, Conference on Robot Learning, 2019.
  - [22] ISCEN A, CALUWAERTS K, TAN J, et al. Policies Modulating Trajectory Generators[J]. Conference on Robot Learning, Conference on Robot Learning, 2018.
  - [23] SCHULMAN J, LEVINE S, MORITZ P, et al. Trust Region Policy Optimization[A]. 2015.
  - [24] BAI S, KOLTER J, KOLTUN V. An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling[A]. 2018.
  - [25] BARASUOL V, BUCHLI J, SEMINI C, et al. A reactive controller framework for quadrupedal locomotion on challenging terrain[C/OL]//2013 IEEE International Conference on Robotics and Automation. 2013. <http://dx.doi.org/10.1109/icra.2013.6630926>.
  - [26] LEE J, HWANGBO J, HUTTER M. Robust Recovery Controller for a Quadrupedal Robot using Deep Reinforcement Learning.[A]. 2019.
  - [27] HWANGBO J, BELLICO C, FANKHAUSER P, et al. Probabilistic foot contact estimation by fusing information from dynamics and differential/forward kinematics[J]. Intelligent Robots and Systems, Intelligent Robots and Systems, 2016.
  - [28] MIKI T, LEE J, HWANGBO J, et al. Learning robust perceptive locomotion for quadrupedal robots in the wild[J/OL]. Science Robotics, 2022(VOL. 7, NO. 62). DOI: [DOI: 10.1126/scirobotics.abk2822](http://dx.doi.org/10.1126/scirobotics.abk2822).

- 
- [29] MNIH V, KAVUKCUOGLU K, SILVER D, et al. Playing Atari with Deep Reinforcement Learning[A]. 2013.
  - [30] ZHU P, LI X, POUPART P, et al. On Improving Deep Reinforcement Learning for POMDPs [A]. 2017.
  - [31] SCHULMAN J, WOLSKI F, DHARIWAL P, et al. Proximal Policy Optimization Algorithms [A]. 2017.
  - [32] ROSS S, GORDON G, BAGNELL J. A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning[A]. 2010.
  - [33] CZARNECKI W, PASCANU R, OSINDERO S, et al. Distilling Policy Distillation[A]. 2019.
  - [34] CHO K, VAN MERRIENBOER B, GULCEHRE C, et al. Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation[C/OL]//Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP). 2014. <http://dx.doi.org/10.3115/v1/d14-1179>.
  - [35] HOCHREITER S, SCHMIDHUBER J. Long short-term memory[J/OL]. Neural Computation, 1997: 1735–1780. <http://dx.doi.org/10.1162/neco.1997.9.8.1735>.
  - [36] PASZKE A, GROSS S, MASSA F, et al. PyTorch: An Imperative Style, High-Performance Deep Learning Library: Article No.: 721[M/OL]. 2019: Pages 8026-8037. DOI: <https://dl.acm.org/doi/10.5555/3454287.3455008>.
  - [37] FANKHAUSER P, BLOESCH M, HUTTER M. Probabilistic Terrain Mapping for Mobile Robots With Uncertain Localization[J/OL]. IEEE Robotics and Automation Letters, 2018: 3019–3026. <http://dx.doi.org/10.1109/lra.2018.2849506>.
  - [38] TODOROV E, EREZ T, TASSA Y. MuJoCo: A physics engine for model-based control [C/OL]//2012 IEEE/RSJ International Conference on Intelligent Robots and Systems. 2012. <http://dx.doi.org/10.1109/iros.2012.6386109>.
  - [39] LEE J, X. GREY M, HA S, et al. DART: Dynamic Animation and Robotics Toolkit[J/OL]. The Journal of Open Source Software, 2018: 500. <http://dx.doi.org/10.21105/joss.00500>.
  - [40] ROHMER E, SINGH S P N, FREESE M. V-REP: A versatile and scalable robot simulation framework[C/OL]//2013 IEEE/RSJ International Conference on Intelligent Robots and Systems. 2013. <http://dx.doi.org/10.1109/iros.2013.6696520>.
  - [41] OPENAI O, AKKAYA I, ANDRYCHOWICZ M, et al. Solving Rubik’s Cube with a Robot Hand.[J]. Cornell University - arXiv,Cornell University - arXiv, 2019.
  - [42] ANDRYCHOWICZ O M, BAKER B, CHOCIEJ M, et al. Learning Dexterous In-Hand Manipulation[J/OL]. The International Journal of Robotics Research, 2020: 3 – 20. <http://dx.doi.org/10.1177/0278364919887447>.
  - [43] MAKOVICHUK V, WAWRZYNIAK L, GUO Y, et al. Isaac Gym: High Performance GPU-Based Physics Simulation For Robot Learning[A]. 2021.
  - [44] TOBIN J, FONG R, RAY A, et al. Domain randomization for transferring deep neural networks from simulation to the real world[C/OL]//2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). 2017. <http://dx.doi.org/10.1109/iros.2017.8202133>.

- [45] RUDIN N, HOELLER D, REIST P, et al. Learning to Walk in Minutes Using Massively Parallel Deep Reinforcement Learning[A]. 2021.
- [46] KLEMM V, MORRA A, SALZMANN C, et al. Ascento: A Two-Wheeled Jumping Robot [C/OL]//2019 International Conference on Robotics and Automation (ICRA). 2019. <http://dx.doi.org/10.1109/icra.2019.8793792>.
- [47] HERZOG A, ROTELLA N, SCHAAAL S, et al. Trajectory generation for multi-contact momentum control[C/OL]//2015 IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids). 2015. <http://dx.doi.org/10.1109/humanoids.2015.7363464>.
- [48] DIETRICH A, BUSSMANN K, PETIT F, et al. Whole-body impedance control of wheeled mobile manipulators Stability analysis and experiments on the humanoid robot Rollin' Justin [J/OL]. Springer Science+Business, 2015, Media New York 2015. DOI: [10.1007/s10514-015-9438-z](https://doi.org/10.1007/s10514-015-9438-z).