

华中科技大学

2024

人工智能导论实验报告

基于 PPO 算法训练的机械控制模型

| | |
|-------|--------------------|
| 专 业: | 计算机科学与技术专业 |
| 班 级: | 计算机科学与技术 2307 班 |
| 学 号: | U202315633 |
| 姓 名: | 宁子健 |
| 电 话: | 13134135856 |
| 邮 件: | aurora_nzj@163.com |
| 完成日期: | 2025 年 1 月 8 日 |

摘 要

机械控制作为一项重要的技术领域，涉及到机械的运动控制、感知环境、决策策略等多种方面。随着人工智能技术的发展，强化学习在机器人控制领域的应用取得了巨大成效。本文论述了一种基于策略的强化学习方法——PPO 算法的基本原理，介绍了 PPO 算法的演变过程，采用了 gym 库中的 Acrobot 仿真环境，同时基于 pybullet 库搭建了一个机械臂避障抓取目标物的仿真环境，将 PPO 模型分别应用于离散和连续动作空间的仿真环境进行训练从而实现机械控制，同时分析训练后对机械的控制效果及目标完成情况并进行总结。

关键词：强化学习，策略强化，PPO 算法，机械控制。

Abstract

As an important technical field, mechanical control involves many aspects such as motion control, perception environment, decision-making strategy and so on. With the development of artificial intelligence technology, the application of reinforcement learning in the field of robot control has achieved great results. This paper discusses the basic principle of PPO algorithm, a strategy based reinforcement learning method, introduces the evolution process of PPO algorithm, adopts Acrobot simulation environment in gym library, and builds a simulation environment of a robot arm to avoid obstacles and grasp objects based on pybullet library. The PPO model is applied to the simulation environment of discrete and continuous motion space respectively to realize the mechanical control. At the same time, the control effect of the machine after training and the completion of the target are analyzed and summarized.

目 录

| | |
|------------------------------------|-----------|
| 1 PPO 算法的演变过程 | 1 |
| 1.1 传统策略梯度算法 | 1 |
| 1.2 演员评论家算法 (ACTOR – CRITIC) | 1 |
| 1.3 TRPO 和 PPO 算法 | 2 |
| 1.4 算法演变过程图示 | 2 |
| 2 PPO 算法基本原理 | 3 |
| 2.1 PPO 惩罚与 PPO 截断 | 3 |
| 2.2 广义优势估计 | 4 |
| 2.3 PPO 算法流程 | 4 |
| 3 机械控制的仿真环境 | 6 |
| 3.1 实验设计及环境配置 | 6 |
| 3.2 离散动作的机械控制仿真环境 | 6 |
| 3.3 连续动作的机械控制仿真环境 | 7 |
| 4 模型训练与模型测试 | 8 |
| 4.1 离散动作机械控制模型 | 8 |
| 4.1.1 动作选择 | 8 |
| 4.1.2 模型训练 | 8 |
| 4.1.3 模型测试 | 10 |
| 4.1.4 结果分析 | 11 |
| 4.2 连续动作机械控制模型 | 11 |
| 4.2.1 动作选择 | 11 |
| 4.2.2 模型训练 | 11 |
| 4.2.3 模型测试 | 14 |
| 4.2.4 结果分析 | 14 |
| 5 总结展望和思考 | 15 |

| | |
|----------------------------|-----------|
| 6 附录 | 17 |
| 6.1 PPO_ACTORBOT.PY | 17 |
| 6.2 ACTORBOT_TEST.PY | 25 |
| 6.3 ENV.PY | 27 |
| 6.4 PPO_ROBOTARM.PY | 35 |
| 6.5 ROBOTARM_TEST.PY | 46 |
| 参考文献 | 49 |

1 PPO 算法的演变过程

1.1 传统策略梯度算法

在强化学习中，除了基于值函数的方法，还有一支非常经典的方法就是基于策略（policy-based）的方法。

基于策略的方法最重要的一点是要将策略参数化。首先假定一个随机性目标策略 π_{θ} 且处处可微， θ 即为该策略参数。接着通过神经网络模型对该策略函数建模，通过给定维度状态输入经过神经网络输出执行动作的概率分布。对该模型训练的目标为寻找一个最优策略最大化策略在环境中的期望回报。

目标函数为： $J(\theta) = E_{s_0}[V^{\pi_{\theta}}(s_0)]$

采用梯度上升的方法最大化该目标函数从而得到最优策略。

1.2 演员评论家算法 (Actor – Critic)

强化学习算法中基于值函数的方法只学习了价值函数，而基于策略的方法只学习了策略函数，为达到更好的模型训练效果，提出了同时学习价值函数和策略函数的 Actor – Critic 算法。

将 Actor-Critic 分为两个部分：Actor（策略网络）和 Critic（价值网络）：

- 策略网络通过与环境交互，并在 Critic 价值函数的指导下用策略梯度结合广义优势估计学习一个更好的策略。
- Critic 通过 Actor 与环境交互收集的数据通过一定处理获得时序差分目标，通过梯度下降法更新参数学习一个价值函数，这个价值函数会用于判断在当前状态什么动作是好的，什么动作不是好的，进而帮助 Actor 进行策略更新。

Actor 的参数更新采用策略梯度的更新方式；Critic 价值网络表示为 V_w ，参数为 w ，采用时序差分的学习方式，损失函数定义为： $J(w) = \frac{1}{2}(r + \gamma V_w(s_{t+1}) - V_w(s_t))^2$ ，采用梯度下降的方式更新参数。

1.3 TRPO 和 PPO 算法

策略梯度方法直接优化策略函数，使智能体能够在复杂、高维的环境中获得良好的决策能力。然而，直接优化策略可能会因步长过长导致策略更新过大，出现学习过程不稳定或样本效率低下等问题；为解决这些问题提出了寻找一块信任区域的思想，策略间的距离由 KL 散度来决定，进而根据不等式约束策略空间得到信任区域，在该区域上进行策略的更新使其安全性得到保证，这就是 TRPO 的基本思想。

然而 TRPO 算法也存在一些缺点，计算过程过于复杂且计算量很大，于是在不断探索中就有了对 TRPO 算法的改进版，即 PPO 算法。

1.4 算法演变过程图示

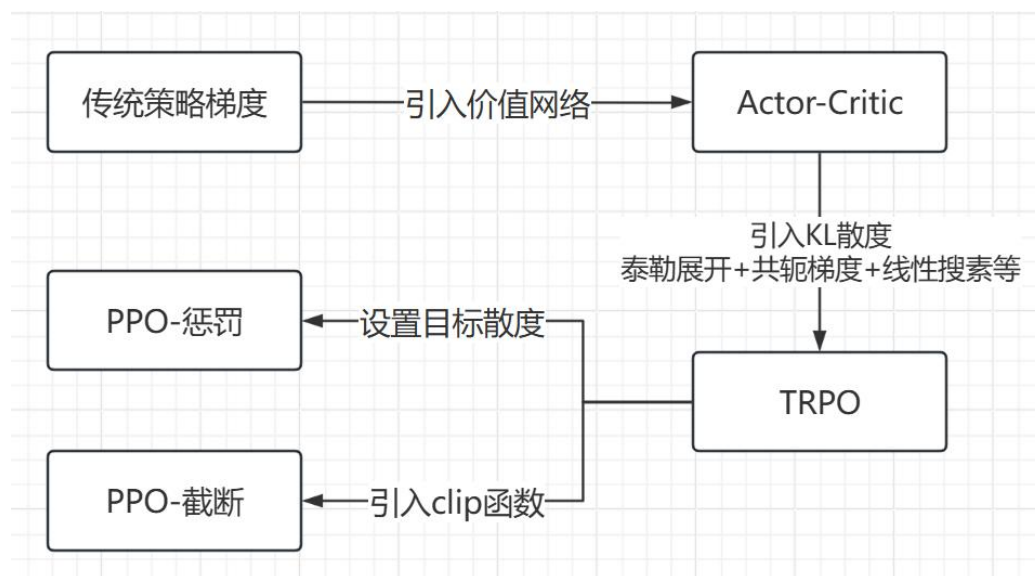


图 1 PPO 算法演变过程

2 PPO 算法基本原理

2.1 PPO 惩罚与 PPO 截断

TRPO 算法中推导出的优化目标和基于 KL 散度的约束条件为：

$$\begin{aligned} \max_{\theta} \quad & E_{s \sim v^{\pi_{\theta_k}}} E_{a \sim \pi_{\theta_k}(\cdot|s)} \left[\frac{\pi_{\theta}(a|s)}{\pi_{\theta_k}(a|s)} A^{\pi_{\theta_k}(s|a)} \right] \\ \text{s.t.} \quad & E_{s \sim v^{\pi_{\theta_k}}} [D_{KL}(\pi_{\theta_k}(\cdot|s), \pi_{\theta}(\cdot|s))] \leq \delta \end{aligned}$$

TRPO 对其求解计算要用到泰勒展开、共轭梯度、线性搜索等复杂方法。而 PPO 算法在不改变优化目标的同时使用了更为简单的两种方法——PPO 惩罚与 PPO 截断。后续项目实战采用 PPO 截断方法，因此仅对 PPO 截断进行简单介绍：

该方法摒弃了 KL 散度确定的信任区域而直接在目标函数中进行范围限制，从而保证更新后的参数与旧的参数差距不会太大，保证了安全性。采用 PPO 截断后的目标函数形式为：

$$\operatorname{argmax}_{\theta} E_{s \sim v^{\pi_{\theta_k}}} E_{a \sim \pi_{\theta_k}(\cdot|s)} [\min(\frac{\pi_{\theta}(a|s)}{\pi_{\theta_k}(a|s)} A^{\pi_{\theta_k}(s|a)}, \operatorname{clip}(\frac{\pi_{\theta}(a|s)}{\pi_{\theta_k}(a|s)}, 1 - \epsilon, 1 + \epsilon) A^{\pi_{\theta_k}(s|a)})]$$

若 $A^{\pi_{\theta_k}(s|a)} > 0$ ，即动作优势大于 0，则该动作价值高于平均值，最大化目标函数即使 $\frac{\pi_{\theta}(a|s)}{\pi_{\theta_k}(a|s)}$ 增大而不超过 $1 + \epsilon$ ，若 $A^{\pi_{\theta_k}(s|a)} < 0$ ，即动作优势小于 0，则该动作价值低于平均值，最大化目标函数即使 $\frac{\pi_{\theta}(a|s)}{\pi_{\theta_k}(a|s)}$ 减小而不超过 $1 - \epsilon$ ，如图示将 $\frac{\pi_{\theta}(a|s)}{\pi_{\theta_k}(a|s)}$ 截断在 $(1 - \epsilon, 1 + \epsilon)$ 之间（ ϵ 为表示截断范围的超参数）。

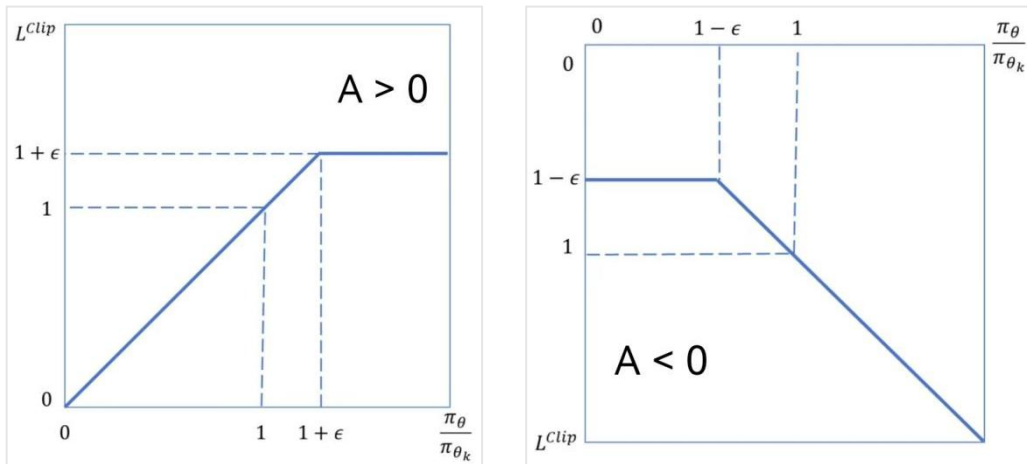


图 2 PPO 截断

2.2 广义优势估计

为估计优势函数 A ，常用的一种方法为广义优势估计 (GAE)

- 首先引入折扣系数 γ , 权值 λ
- 用 $\delta_t = \text{reward}_t + \gamma V(S_{t+1}) - V(S_t)$ 表示时序差分误差，其中 V 为已学习的状态价值函数
- 根据多步时序差分思想有：

$$\begin{aligned} A_t^{(1)} &= \delta_t &= -V(s_t) + r_t + \gamma V(s_{t+1}) \\ A_t^{(2)} &= \delta_t + \gamma \delta_{t+1} &= -V(s_t) + r_t + \gamma r_{t+1} + \gamma^2 V(s_{t+2}) \\ A_t^{(3)} &= \delta_t + \gamma \delta_{t+1} + \gamma^2 \delta_{t+2} &= -V(s_t) + r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \gamma^3 V(s_{t+3}) \\ &\vdots &\vdots \\ A_t^{(k)} &= \sum_{l=0}^{k-1} \gamma^l \delta_{t+l} &= -V(s_t) + r_t + \gamma r_{t+1} + \dots + \gamma^{k-1} r_{t+k-1} + \gamma^k V(s_{t+k}) \end{aligned}$$

- GAE 的方法为将不同步数的优势估计进行指数加权平均：

$$\begin{aligned} A_t^{GAE} &= (1 - \lambda)(A_t^{(1)} + \lambda A_t^{(2)} + \lambda^2 A_t^{(3)} + \dots) \\ &= (1 - \lambda)(\delta_t + \lambda(\delta_t + \gamma \delta_{t+1}) + \lambda^2(\delta_t + \gamma \delta_{t+1} + \gamma^2 \delta_{t+2}) + \dots) \\ &= (1 - \lambda)(\delta_t(1 + \lambda + \lambda^2 + \dots) + \gamma \delta_{t+1}(\lambda + \lambda^2 + \lambda^3 + \dots) + \gamma^2 \delta_{t+2}(\lambda^2 + \lambda^3 + \lambda^4 + \dots) + \dots) \\ &= (1 - \lambda) \left(\delta_t \frac{1}{1 - \lambda} + \gamma \delta_{t+1} \frac{\lambda}{1 - \lambda} + \gamma^2 \delta_{t+2} \frac{\lambda^2}{1 - \lambda} + \dots \right) \\ &= \sum_{l=0}^{\infty} (\gamma \lambda)^l \delta_{t+l} \end{aligned}$$

2.3 PPO 算法流程

首先初始化策略网络 π_θ 和价值网络 V_ϕ 的参数，使用当前策略 π_θ 收集样本数据，通常是包括 (state, action, next_state, reward, done) 的五元组，根据收集到的数据和价值网络 V_ϕ 计算时序差分目标进而计算优势函数 A_t ，根据策略网络 π_θ 和优势函数 A_t 计算损失函数，采用 PPO 截断修正，使用梯度下降法更新策略网络和价值网络的参数以最小化损失函数，当达到设定训练迭代次数时结束训练。算法流程图如下：

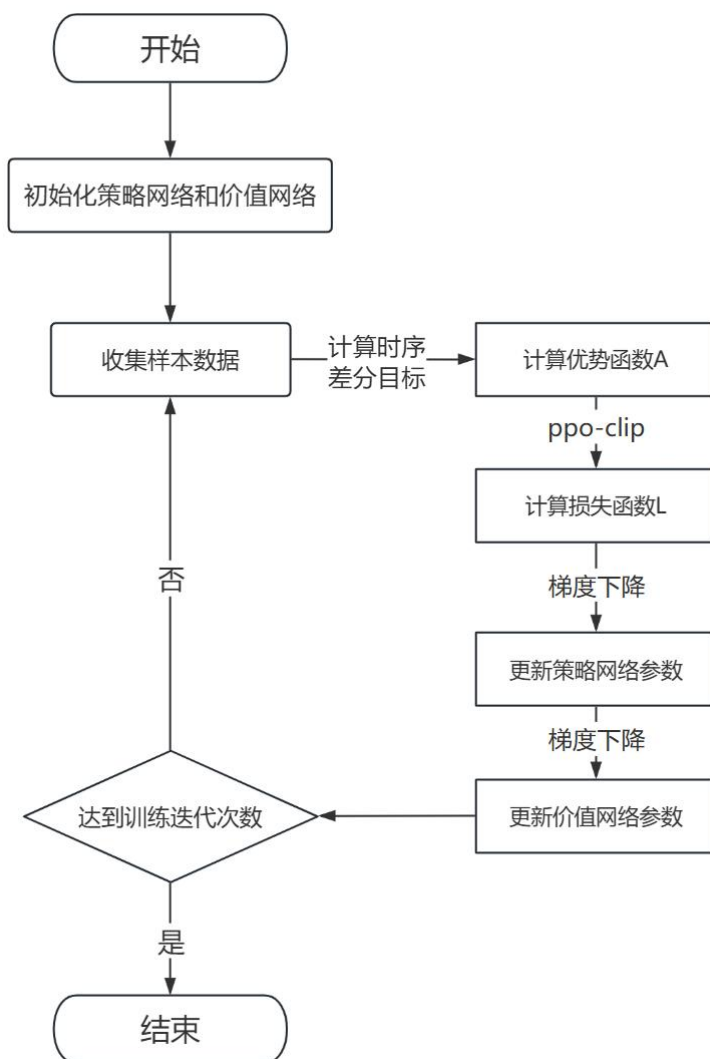


图 3 PPO 算法流程图

3 机械控制的仿真环境

3.1 实验设计及环境配置

- 实验设计：为探究 PPO 算法的适用范围及效果，本实验选取了分别具有离散动作和连续动作的两个不同的机械控制仿真环境，采用强化学习的 PPO 算法对模型进行训练使其更好地完成目标任务，并对训练后的模型进行测试，分析测试结果。
- 环境配置：在 VScode 编译器上采用 python 语言编写算法流程，通过 Pytorch、stable_baseline3 库进行深度强化学习框架的编写和辅助训练，通过 gym 库和 pybullet 库进行仿真环境的搭建，通过 matplotlib、swanlab 库对训练过程进行可视化分析，通过 numpy、math 等基本库进行基本运算操作，采用 cuda_12.6 在 GPU 上进行模型训练。

3.2 离散动作的机械控制仿真环境

实验选取的第一个机械控制仿真环境为 gym 库中提供的 Acrobot 环境，该环境具有离散的动作操作。

- 待训练的智能体为一端被固定住的两个线性连接的链结。
- 目标任务为从竖直向下悬挂的初始状态开始在驱动关节上施加力矩，使线性链的自由端摆动到给定高度以上，若步数超过 500 则任务失败。
- 状态空间为 6 维度，包括两个旋转关节的角度及其角速度信息。
- 动作空间为 3 维度，分别代表对驱动关节施加-1、0、1 的力矩。
- 奖励设置为对未达到目标的步数操作给予-1 作为惩罚。

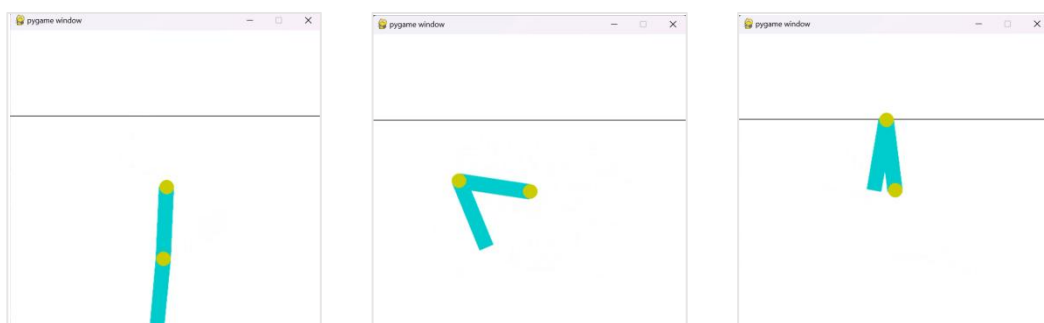


图 4 机械控制仿真环境 1

3.3 连续动作的机械控制仿真环境

实验选取的第二个机械控制仿真环境为基于 `pybullet` 库搭建的机械臂避障抓取目标物环境。该环境具有连续的动作操作。

- 待训练的智能体为具有 6 个可动关节的机械臂。
- 目标任务为在规定步数范围内控制机械臂避开障碍物并实现对目标物的抓取，若步数超过 200 则任务失败，其中障碍物为半径 0.1m 的小球，目标物为圆柱体，当机械臂夹爪的中心离目标中心在 0.05m 以内，即视为抓取成功。
- 状态空间为 12 维度，分别是当前 6 个轴的角度、目标物和障碍物位置坐标。
- 动作空间为 6 维度，分别为六个轴的移动角度（范围：-1 度到 1 度）
- 奖励设置为当抓取成功给予一定的成功奖励，抓取失败或碰到障碍物给予适当的惩罚，抓取过程中根据距离变化量给予奖励或惩罚。

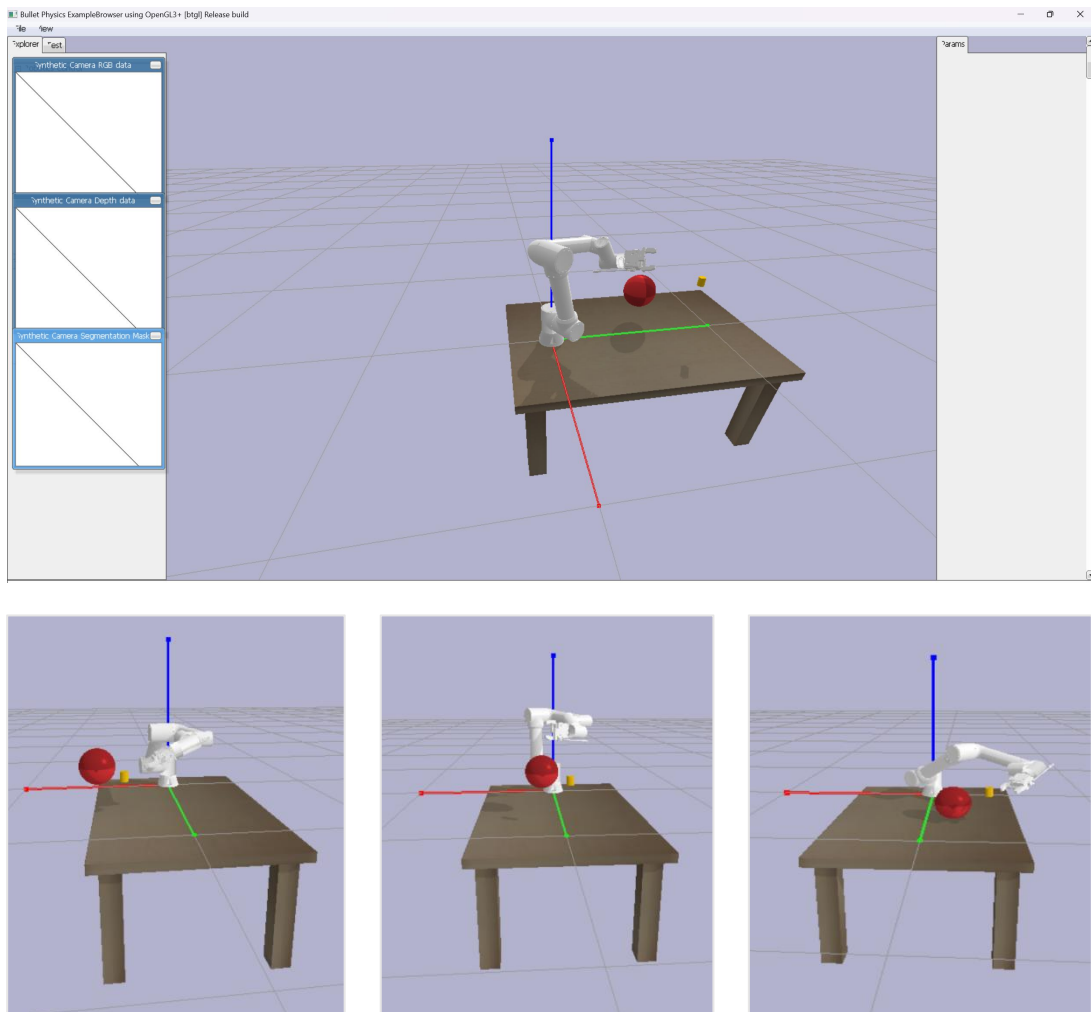


图 5 机械控制仿真环境 2

4 模型训练与模型测试

4.1 离散动作机械控制模型

4.1.1 动作选择

PPO 算法中，在离散动作的环境中，每个动作都是一个离散类别，在策略网络的最终全连接层采用 Softmax 激活函数，将输出转化为概率分布使得和为 1，因此，状态输入通过策略网络输出的值为各个动作的概率分布，然后根据概率的大小进行动作采样。

4.1.2 模型训练

- 使用 pytorch 库进行深度强化学习 PPO 算法的网络搭建和流程编写，选择合适的优化器和损失函数进行训练，使用 matplotlib 库进行图像绘制从而实现训练过程可视化。

1. 网络构建：

- 策略网络：
 - 输入层：6 个神经元，对应状态空间维度。
 - 隐藏层：一个全连接层，每层 64 个神经元，激活函数为 relu。
 - 输出层：3 个神经元，对应动作空间维度，激活函数为 softmax。
- 价值网络：
 - 输入层：6 个神经元，对应状态空间维度。
 - 隐藏层：一个全连接层，每层 64 个神经元，激活函数为 relu。
 - 输出层：1 个神经元，输出状态价值的估计值。

2. 超参数设置：

| 超参数名称 | 设置值 |
|--------------------|------|
| EPISODES（训练迭代回合数） | 5000 |
| CLIP_EPS（裁剪阈值） | 0.2 |
| ACTOR_LR（策略网络学习率） | 1e-4 |
| CRITIC_LR（价值网络学习率） | 5e-3 |

| | |
|-------------------|------|
| LAMBDA (GAE 参数) | 0.95 |
| GAMMA (折扣因子) | 0.98 |
| EPOCHS (每回合更新次数) | 10 |
| STATE_DIM (状态维度) | 6 |
| HIDDEN_DIM (隐藏层) | 64 |
| ACTION_DIM (动作维度) | 3 |

3. 可视化分析

训练过程中分别记录了原始成绩和均值成绩，并在训练结束后对其进行可视化分析，观察训练效果。

- 原始成绩：

原始成绩为每回合训练结束后智能体获得的总奖励值，即所用步数的相反数。如图 6 所示，在训练至 1000 回合时图像已经明显收敛，从最初的 500 步也无法完成任务到训练至 1000 回合时仅用近 100 步就能完成任务，取得了明显的训练成果。但由图像能明显发现，训练取得成效后仍有概率出现较差的情况，每回合并不是完全稳定的。

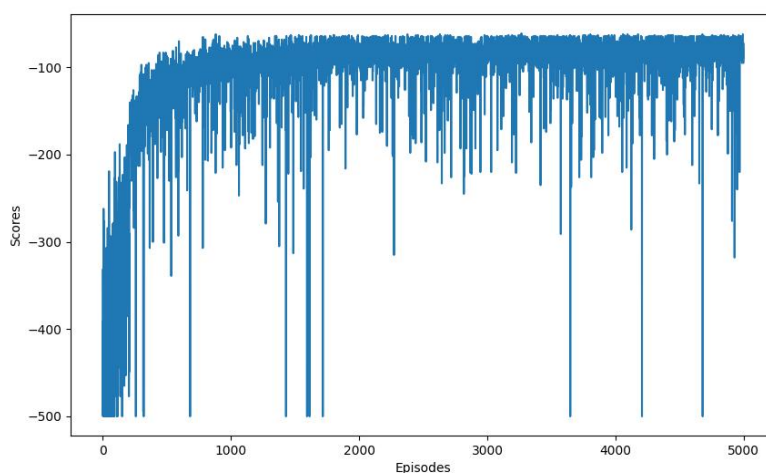


图 6 训练原始成绩

- 均值成绩：

为更加直观的观察训练所带来的变化，我们记录了均值成绩，即每十回合的总成绩的平均值，避免了偶然性和随机性带来图像混乱的问题。如图 7 所示明显观察到了图像的收敛和智能体的进步。

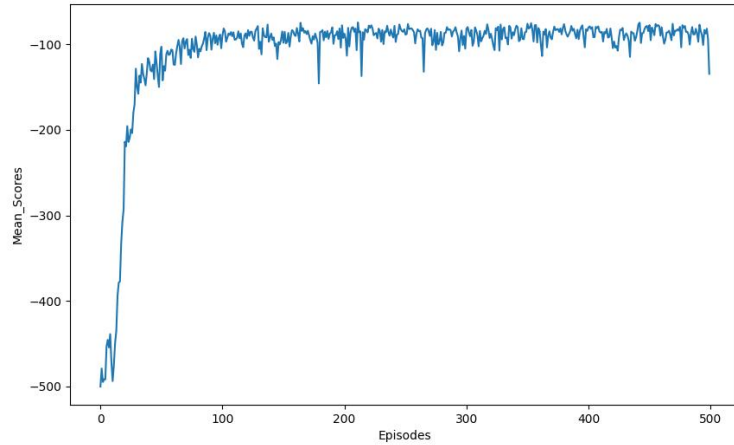


图 7 训练均值成绩

4. 参数保存

模型训练结束后对网络参数进行保存，策略网络参数保存为 Actorbot_actor.pth 文件，价值网络参数保存为 Actorbot_critic.pth 文件，以供测试使用。

4.1.3 模型测试

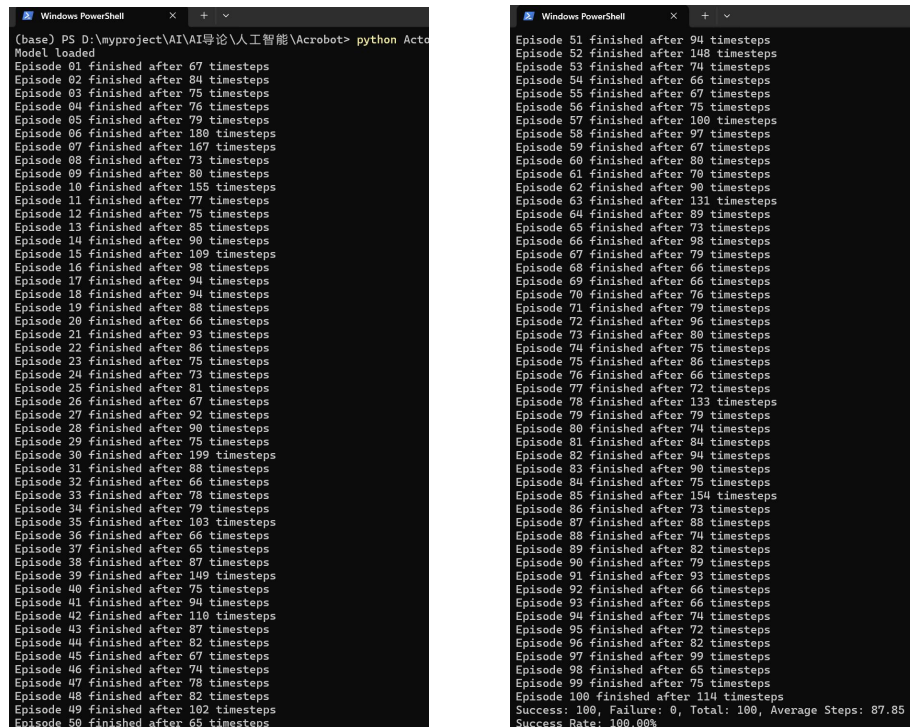


图 8 Acrobot 测试结果

为验证训练所得模型的实际效果，创建新的智能体，导入训练所得参数，随机生成 100 个测试集进行模型测试，输出测试结果。

4.1.4 结果分析

- 在模型测试中，100 个测试集均在规定步数内完成，成功率达到 100%，所用平均步数为 87.85，证明了模型训练达到了不错的效果。

4.2 连续动作机械控制模型

4.2.1 动作选择

PPO 算法中，在连续动作的环境中，动作不再是离散的类别，而是一个区间范围内可以连续变化的数值。在策略网络的最终全连接层采用 Tanh 激活函数，将输出范围限制在[-1,1]之间，输出为动作维度的均值和方差，通过训练均值和方差得到动作服从的高斯分布情况，根据分布采样具体动作。

4.2.2 模型训练

- 该仿真环境较为复杂，训练稳定性差，因目标物距离机械臂有一定距离，所以先使用 100 步的固定动作[-0.3, -0.7, 1, 1, 1, 1]引导机械臂靠近目标物后再开始抓取工作，引入 stable_baseline3 强化学习算法库中的 PPO 框架进行模型训练，并通过 swanlab 库监视训练，实时观察训练的可视化效果。

1. 网络构建：

- 策略网络：
 - 输入层：12 个神经元，对应状态空间维度。
 - 隐藏层：两个全连接层，两层分别有 128 和 256 个神经元。
 - 输出层：6 个神经元，对应动作空间维度。
- 价值网络：
 - 输入层：12 个神经元，对应状态空间维度。
 - 隐藏层：两个全连接层，两层分别有 128 和 256 个神经元。
 - 输出层：1 个神经元，输出状态价值的估计值。

2. 超参数设置：

| 超参数名称 | 设置值 |
|-------------------|--------|
| TIMESTEPS（训练迭代步数） | 500000 |

| | |
|------------------|------|
| CLIP_EPS（裁剪阈值） | 0.1 |
| LR（学习率） | 1e-4 |
| LAMBDA（GAE 参数） | 0.97 |
| GAMMA（折扣因子） | 0.99 |
| EPOCHS（每回合更新次数） | 5 |
| STATE_DIM（状态维度） | 12 |
| HIDDEN_DIM（隐藏层） | 128 |
| | 256 |
| ACTION_DIM（动作维度） | 6 |

3. 可视化分析

训练过程中在 swanlab 监控下给出了抓取所使用的平均步数图像和获得的平均奖励的图像，并给出了训练后的最终回合数据指标。

- 使用步数

由图像观察可知机械臂在训练初期未在规定步数内完成任务，但随着训练的进行不仅能完成任务而且抓取目标物所用步数也随着训练回合的增加而不断减少，不过仍具有一定的随机性和不稳定性，且因初始环境的不同，步数仍存在一定范围的波动。

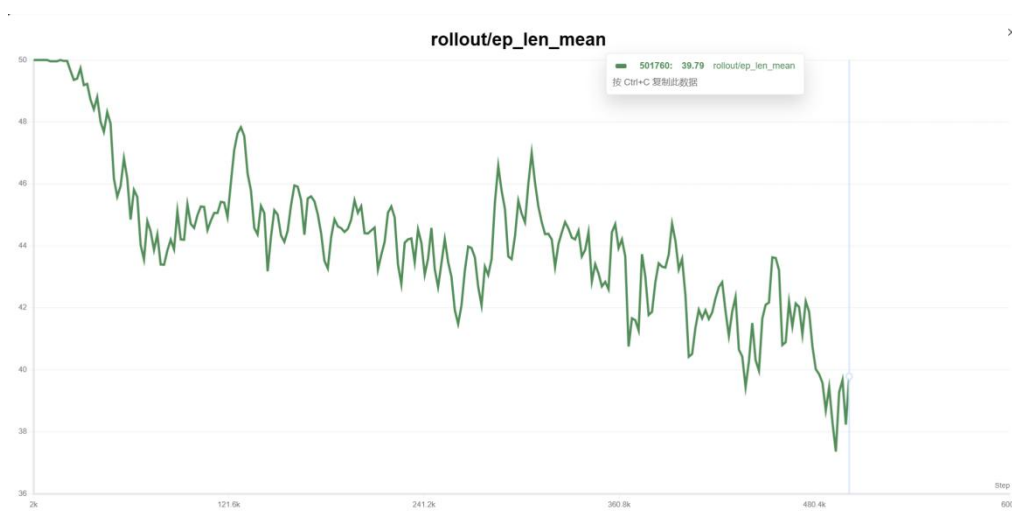


图 9 训练进程中完成任务所需步数变化图像

- 获取奖励

由奖励图像可知智能体获取的奖励值在训练初期迅速得到提高，而后缓慢提高最终趋近于收敛。

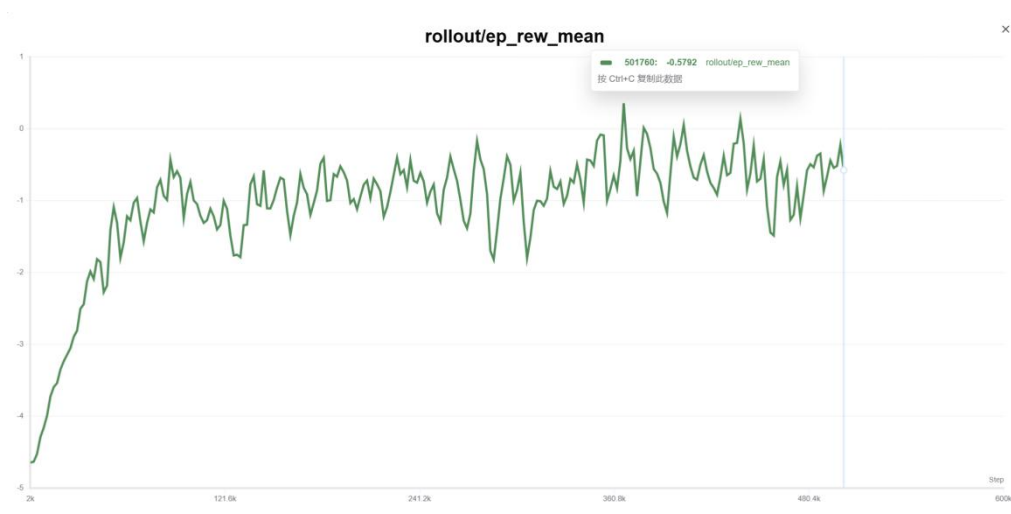


图 10 训练过程中获取奖励值变化图像

- 最终训练回合的指标

由图可知最后一个训练回合机械臂抓取所用的平均步数、获得奖励，各超参数及损失的具体数据。

指标

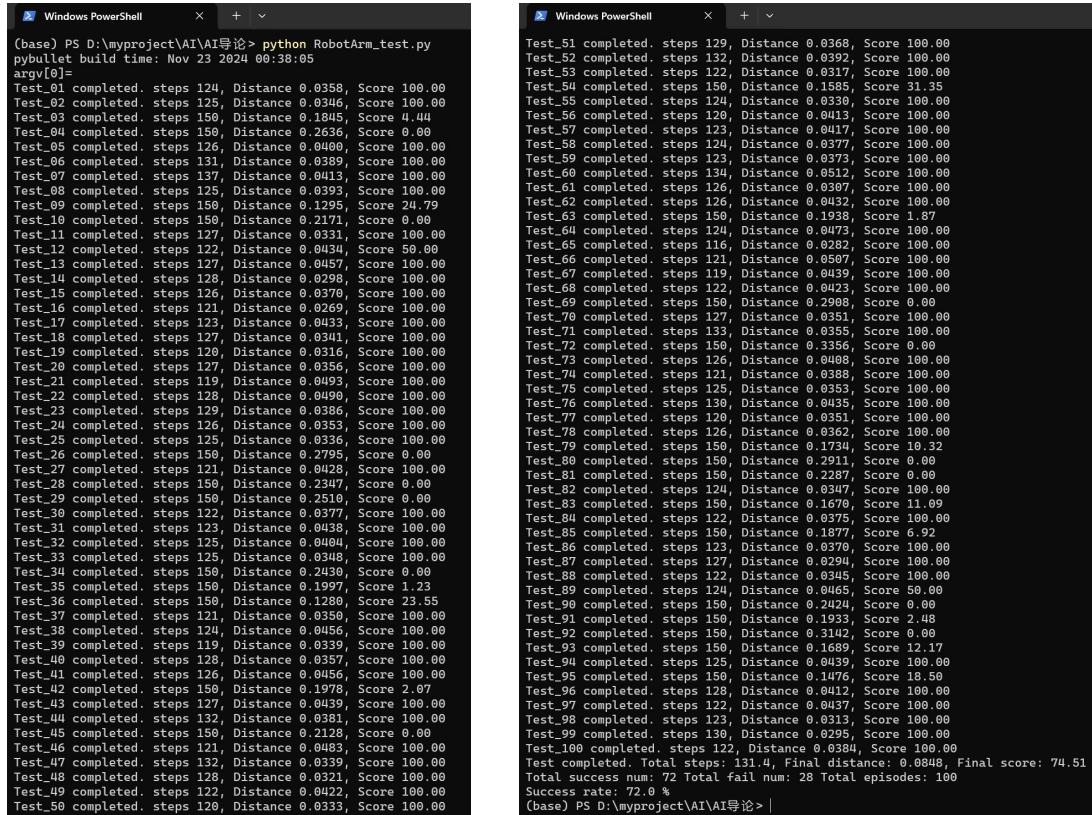
| 键 | 值 |
|----------------------------|------------|
| rollout/ep_len_mean | 39.79 |
| rollout/ep_rew_mean | -0.5792 |
| time/fps | 127 |
| time/iterations | 245 |
| time/time_elapsed | 3936 |
| time/total_timesteps | 5.0176e+5 |
| train/clip_fraction | 0.07255 |
| train/clip_range | 0.1 |
| train/entropy_loss | -7.8838 |
| train/explained_variance | 0.2449 |
| train/learning_rate | 0.0001 |
| train/loss | 1.2583 |
| train/n_updates | 1220 |
| train/policy_gradient_loss | -8.1943e-5 |
| train/std | 0.9026 |
| train/value_loss | 3.5519 |

图 11 最终训练回合指标

4. 参数保存

模型训练结束后对网络参数进行保存，包括策略网络参数，价值网络参数等均保存在 RobotArm.zip 压缩包中，以供测试使用。

4.2.3 模型测试



```
(base) PS D:\myproject\AI\AI导论> python RobotArm_test.py
pybullet build time: Nov 23 2024 00:38:05
argv[0]=
Test_01 completed. steps 124, Distance 0.0358, Score 100.00
Test_02 completed. steps 125, Distance 0.0346, Score 100.00
Test_03 completed. steps 150, Distance 0.1845, Score 4.44
Test_04 completed. steps 150, Distance 0.2636, Score 0.00
Test_05 completed. steps 126, Distance 0.0400, Score 100.00
Test_06 completed. steps 131, Distance 0.0389, Score 100.00
Test_07 completed. steps 137, Distance 0.0413, Score 100.00
Test_08 completed. steps 125, Distance 0.0393, Score 100.00
Test_09 completed. steps 150, Distance 0.1295, Score 74.79
Test_10 completed. steps 150, Distance 0.2171, Score 0.00
Test_11 completed. steps 127, Distance 0.0331, Score 100.00
Test_12 completed. steps 122, Distance 0.0434, Score 50.00
Test_13 completed. steps 127, Distance 0.0457, Score 100.00
Test_14 completed. steps 128, Distance 0.0298, Score 100.00
Test_15 completed. steps 126, Distance 0.0370, Score 100.00
Test_16 completed. steps 121, Distance 0.0269, Score 100.00
Test_17 completed. steps 123, Distance 0.0433, Score 100.00
Test_18 completed. steps 127, Distance 0.0341, Score 100.00
Test_19 completed. steps 128, Distance 0.0316, Score 100.00
Test_20 completed. steps 127, Distance 0.0356, Score 100.00
Test_21 completed. steps 119, Distance 0.0493, Score 100.00
Test_22 completed. steps 128, Distance 0.0490, Score 100.00
Test_23 completed. steps 129, Distance 0.0386, Score 100.00
Test_24 completed. steps 126, Distance 0.0353, Score 100.00
Test_25 completed. steps 125, Distance 0.0336, Score 100.00
Test_26 completed. steps 150, Distance 0.2795, Score 0.00
Test_27 completed. steps 121, Distance 0.0428, Score 100.00
Test_28 completed. steps 150, Distance 0.2347, Score 0.00
Test_29 completed. steps 150, Distance 0.2510, Score 0.00
Test_30 completed. steps 122, Distance 0.0377, Score 100.00
Test_31 completed. steps 122, Distance 0.0438, Score 100.00
Test_32 completed. steps 125, Distance 0.0404, Score 100.00
Test_33 completed. steps 125, Distance 0.0348, Score 100.00
Test_34 completed. steps 150, Distance 0.2430, Score 0.00
Test_35 completed. steps 150, Distance 0.1997, Score 1.23
Test_36 completed. steps 150, Distance 0.1280, Score 23.55
Test_37 completed. steps 121, Distance 0.0350, Score 100.00
Test_38 completed. steps 124, Distance 0.0456, Score 100.00
Test_39 completed. steps 119, Distance 0.0339, Score 100.00
Test_40 completed. steps 128, Distance 0.0357, Score 100.00
Test_41 completed. steps 126, Distance 0.0456, Score 100.00
Test_42 completed. steps 150, Distance 0.1978, Score 2.07
Test_43 completed. steps 127, Distance 0.0439, Score 100.00
Test_44 completed. steps 132, Distance 0.0381, Score 100.00
Test_45 completed. steps 150, Distance 0.2128, Score 0.00
Test_46 completed. steps 121, Distance 0.0483, Score 100.00
Test_47 completed. steps 132, Distance 0.0339, Score 100.00
Test_48 completed. steps 128, Distance 0.0321, Score 100.00
Test_49 completed. steps 122, Distance 0.0422, Score 100.00
Test_50 completed. steps 120, Distance 0.0333, Score 100.00
Test_51 completed. steps 129, Distance 0.0368, Score 100.00
Test_52 completed. steps 132, Distance 0.0392, Score 100.00
Test_53 completed. steps 122, Distance 0.0317, Score 100.00
Test_54 completed. steps 150, Distance 0.1585, Score 31.35
Test_55 completed. steps 124, Distance 0.0330, Score 100.00
Test_56 completed. steps 120, Distance 0.0413, Score 100.00
Test_57 completed. steps 123, Distance 0.0417, Score 100.00
Test_58 completed. steps 124, Distance 0.0377, Score 100.00
Test_59 completed. steps 122, Distance 0.0373, Score 100.00
Test_60 completed. steps 134, Distance 0.0512, Score 100.00
Test_61 completed. steps 126, Distance 0.0307, Score 100.00
Test_62 completed. steps 126, Distance 0.0432, Score 100.00
Test_63 completed. steps 150, Distance 0.1938, Score 1.87
Test_64 completed. steps 124, Distance 0.0473, Score 100.00
Test_65 completed. steps 116, Distance 0.0282, Score 100.00
Test_66 completed. steps 121, Distance 0.0507, Score 100.00
Test_67 completed. steps 119, Distance 0.0439, Score 100.00
Test_68 completed. steps 122, Distance 0.0423, Score 100.00
Test_69 completed. steps 150, Distance 0.2908, Score 0.00
Test_70 completed. steps 127, Distance 0.0351, Score 100.00
Test_71 completed. steps 133, Distance 0.0355, Score 100.00
Test_72 completed. steps 150, Distance 0.3356, Score 0.00
Test_73 completed. steps 126, Distance 0.0408, Score 100.00
Test_74 completed. steps 121, Distance 0.0388, Score 100.00
Test_75 completed. steps 125, Distance 0.0353, Score 100.00
Test_76 completed. steps 130, Distance 0.0435, Score 100.00
Test_77 completed. steps 120, Distance 0.0351, Score 100.00
Test_78 completed. steps 126, Distance 0.0352, Score 100.00
Test_79 completed. steps 150, Distance 0.1734, Score 10.32
Test_80 completed. steps 150, Distance 0.2911, Score 0.00
Test_81 completed. steps 150, Distance 0.2287, Score 0.00
Test_82 completed. steps 124, Distance 0.0347, Score 100.00
Test_83 completed. steps 150, Distance 0.1670, Score 11.09
Test_84 completed. steps 122, Distance 0.0375, Score 100.00
Test_85 completed. steps 150, Distance 0.1877, Score 6.92
Test_86 completed. steps 123, Distance 0.0370, Score 100.00
Test_87 completed. steps 127, Distance 0.0294, Score 100.00
Test_88 completed. steps 122, Distance 0.0345, Score 100.00
Test_89 completed. steps 124, Distance 0.0465, Score 50.00
Test_90 completed. steps 150, Distance 0.2424, Score 0.00
Test_91 completed. steps 150, Distance 0.1933, Score 2.48
Test_92 completed. steps 150, Distance 0.3142, Score 0.00
Test_93 completed. steps 150, Distance 0.1689, Score 12.17
Test_94 completed. steps 125, Distance 0.0439, Score 100.00
Test_95 completed. steps 150, Distance 0.1476, Score 18.50
Test_96 completed. steps 128, Distance 0.0412, Score 100.00
Test_97 completed. steps 122, Distance 0.0437, Score 100.00
Test_98 completed. steps 122, Distance 0.0313, Score 100.00
Test_99 completed. steps 130, Distance 0.0295, Score 100.00
Test_100 completed. steps 122, Distance 0.0384, Score 100.00
Test completed. Total steps: 131.4, Final distance: 0.0848, Final score: 74.51
Total success num: 72 Total fail num: 28 Total episodes: 100
Success rate: 72.0 %
(base) PS D:\myproject\AI\AI导论>
```

图 12 RobotArm 测试结果

为验证训练所得模型的实际效果，创建新的智能体，导入训练所得参数，随机生成 100 个测试集进行模型测试，输出测试结果。

4.2.4 结果分析

测试具体得分计算方式为：

- 抓取成功：100 分
- 超出步数限制时： 根据距离目标 0.05 到 0.2m 的范围分数线性递减，具体公式为 $100 \times (1 - \frac{\text{distance}-0.05}{0.2-0.05})$
- 碰撞惩罚： 触碰到障碍物则评分 $\times 0.5$

在模型测试中，100 个测试样例最终抓取所用的平均步数为 131.4，距离目标物的平均距离为 0.0848m，平均成绩为 74.51 分，有 72 个样例达到了 100 分实现了完美抓取，28 个样例未能达成目标，成功率 72%。

5 总结展望和思考

- 分析总结

在经历一段时间的探索和演变后，PPO 算法作为一种高效且稳定的强化学习算法出现并得到广泛使用，PPO 裁剪机制的引入限制了策略更新的幅度，不仅有效平衡了策略优化过程中的探索和利用，保证了其安全性和稳定性的同时大大减轻了运算的复杂度，优化了训练效率。此外，PPO 算法作为一种 on-policy 的方法具有广泛的应用范围，适用于各种复杂的控制任务，基于策略的方法使得其不仅能在离散动作的环境中训练智能体，在连续动作的环境下同样能取得良好的训练效果。

通过本次实验，我们完成了在两种仿真环境（Acrobot、RobotArm）下使用 PPO 算法对机械控制模型的训练，并取得了一定的成果，实践操作让我们更加深入的了解 PPO 的训练方式和算法原理。

- 不足之处：

尽管最终训练出的智能体能较好的完成目标任务，但我们不难发现其仍存在不足之处，可能源于超参数的设置问题、奖励设置并不够优或者训练回合数不够多以致于未收敛或未稳定等。

在 Acrobot 环境的模型训练中，通过原始成绩我们明显观察到了任务完成的不稳定性，仍有一定概率出现不能很好完成任务的情况，该模型仍有一定上升空间。在 RobotArm 环境的模型训练中，观察步数图像我们可以发现其仍有降低趋势，由此可以判断其可能并未完全收敛，可以通过继续训练达到更好的效果，72%的成功率证明该仿真环境模型仍有很大的提升空间。

- 改进措施：

1. 调整超参数：强化学习的超参数选取对于模型的训练具有很强的影响，可以通过继续改善超参数对比训练结果，根据训练的结果来寻找更优的参数设置。
2. Reward-Shaping：奖励设置是强化学习较为重要的一部分，它决定了训练的走向，为获取更多的奖励智能体很可能采取一些奇特的“刷奖励”方式而偏离最终目标，因此，可以通过 Reward-Shaping 重构奖励，指导智能体更好的完成任务。
3. 增加迭代次数：对于可能未完全收敛的模型训练，可以在已有参数的基础上增

加训练回合数继续监督训练。

- 课程体会

在本学期初,我便有幸接触到了强化学习领域的相关知识,了解了马尔科夫(MDP)问题,并使用了 Qlearning、Sarsa、DQN 等基于价值的方法在 Atari 游戏环境中进行了实践探索。通过在莫益军老师的人工智能导论课堂学习,我对人工智能领域的兴趣愈发浓烈,不仅了解到了 AI 的发展历程,同时看到了其无限魅力和广阔前景。在后续莫老师的课堂介绍中更是再次对强化学习领域有了新的认识,因此在本次实验选取了未尝试过的基于策略的强化学习方法进行模型训练,通过本次实践,我对该领域有了更深层的思考,对其算法原理更加清晰,而且代码实践能力得到了很大的提升,此外我掌握了强化学习的算法库 `stable_baseline`,并且学会了使用能实时监控训练并可视化训练过程的 `swanlab` 库,课程不仅激发了我对人工智能的学习兴趣,而且很大程度的提高了我的技能本领。

6 附录

6.1 ppo_Actorbot.py

```
import numpy as np
import torch
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
import gymnasium as gym
from collections import deque
import matplotlib.pyplot as plt

# Hyperparameters
STATE_DIM = 6
HIDDEN_DIM = 64
ACTION_DIM = 3
ACTOR_LR = 1e-4
CRITIC_LR = 5e-3
LAMBDA = 0.95
EPOCHS = 10
EPS = 0.2
GAMMA = 0.98
EPISODES = 5000
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

class PolicyNet(nn.Module):
```

```
def __init__(self, state_dim, hidden_dim, action_dim):
```

```
    super(PolicyNet, self).__init__()
```

```
    self.fc1 = nn.Linear(state_dim, hidden_dim)
```

```
    self.fc2 = nn.Linear(hidden_dim, action_dim)
```

```
def forward(self, x):
```

```
    x = F.relu(self.fc1(x))
```

```
    return F.softmax(self.fc2(x), dim=-1)
```

```
class ValueNet(nn.Module):
```

```
    def __init__(self, state_dim, hidden_dim):
```

```
        super(ValueNet, self).__init__()
```

```
        self.fc1 = nn.Linear(state_dim, hidden_dim)
```

```
        self.fc2 = nn.Linear(hidden_dim, 1)
```

```
def forward(self, x):
```

```
    x = F.relu(self.fc1(x))
```

```
    return self.fc2(x)
```

```
class PPO:
```

```
    def __init__(
```

```
        self,
```

```
        state_dim=STATE_DIM,
```

```
        hidden_dim=HIDDEN_DIM,
```

```
        action_dim=ACTION_DIM,
```

```
        actor_lr=ACTOR_LR,
```

```
        critic_lr=CRITIC_LR,
```

```

        lmbda=LAMBDA,
        epochs=EPOCHS,
        eps=EPS,
        gamma=GAMMA,
        device=device,
    ):
        self.actor = PolicyNet(state_dim, hidden_dim, action_dim).to(device)
        self.critic = ValueNet(state_dim, hidden_dim).to(device)
        self.actor_optimizer = optim.Adam(self.actor.parameters(), lr=actor_lr)
        self.critic_optimizer = optim.Adam(self.critic.parameters(), lr=critic_lr)
        self.gamma = gamma
        self.lmbda = lmbda
        self.epochs = epochs
        self.eps = eps
        self.device = device

    def take_action(self, state):
        state = torch.tensor(state, dtype=torch.float).to(self.device)
        action_probs = self.actor(state)
        action_dist = torch.distributions.Categorical(action_probs)
        action = action_dist.sample()
        return action.item()

    def cal_advantages(self, td_errors):
        td_errors = td_errors.cpu().detach().numpy().tolist()
        advantages = []
        adv = 0.0
        for td_error in td_errors[::-1]:
            adv = self.gamma * self.lmbda * adv + td_error

```

```

        advantages.append(adv)
    advantages.reverse()
    return torch.tensor(advantages, dtype=torch.float).to(self.device)

def update(self, transition_dict):
    states = torch.tensor(transition_dict["states"], dtype=torch.float).to(
        self.device
    )
    actions = (
        torch.tensor(transition_dict["actions"], dtype=torch.long)
        .view(-1, 1)
        .to(self.device)
    )
    rewards = (
        torch.tensor(transition_dict["rewards"], dtype=torch.float)
        .view(-1, 1)
        .to(self.device)
    )
    next_states = torch.tensor(
        transition_dict["next_states"], dtype=torch.float
    ).to(self.device)
    dones = (
        torch.tensor(transition_dict["dones"], dtype=torch.float)
        .view(-1, 1)
        .to(self.device)
    )

    # 计算优势
    td_targets = rewards + self.gamma * self.critic(next_states) * (1 - dones)

```

```

td_errors = (td_targets - self.critic(states)).detach().squeeze()
advantages = self.cal_advantages(td_errors)

# 更新策略网络
old_log_probs = torch.log(self.actor(states).gather(1, actions)).detach()

for _ in range(self.epochs):
    new_log_probs = torch.log(self.actor(states).gather(1, actions))
    ratios = torch.exp(new_log_probs - old_log_probs)
    surr1 = ratios * advantages
    surr2 = (
        torch.clamp(ratios, 1 - self.eps, 1 + self.eps) * advantages
    ).detach()
    actor_loss = -torch.min(surr1, surr2).mean()
    critic_loss = torch.mean(
        F.mse_loss(self.critic(states), td_targets.detach())
    )

    self.actor_optimizer.zero_grad()
    self.critic_optimizer.zero_grad()
    actor_loss.backward()
    critic_loss.backward()
    self.actor_optimizer.step()
    self.critic_optimizer.step()

def save(self):
    torch.save(self.actor.state_dict(), "Actorbot_actor.pth")
    torch.save(self.critic.state_dict(), "Actorbot_critic.pth")
    print(f"Model saved")

```

```
def load(self, actor_path, critic_path):  
    self.actor.load_state_dict(  
        torch.load(actor_path, map_location=torch.device("cpu"))  
    )  
    self.critic.load_state_dict(  
        torch.load(critic_path, map_location=torch.device("cpu"))  
    )  
    self.actor.eval()  
    self.critic.eval()  
    print(f"Model loaded")
```

```
def Train(ppo):  
    env = gym.make("Acrobot-v1")  
    max_steps = env.spec.max_episode_steps  
    scores = []  
    ave_scores = []  
    scores_window = deque(maxlen=10)  
  
    for i in range(EPIISODES):  
        state = env.reset()[0]  
        score = 0  
        transition_dict = {  
            "states": [],  
            "actions": [],  
            "rewards": [],  
            "next_states": [],  
            "dones": [],
```

```

    }

    done = False
    steps = 0
    while not done and steps < max_steps:
        action = ppo.take_action(state)
        next_state, reward, done, _, _ = env.step(action)
        transition_dict["states"].append(state)
        transition_dict["actions"].append(action)
        transition_dict["rewards"].append(reward)
        transition_dict["next_states"].append(next_state)
        transition_dict["dones"].append(done)
        state = next_state
        score += reward
        steps += 1

    if steps < max_steps:
        print(f"Episode {i} finished after {steps} timesteps")
    else:
        print(f"Episode {i} did not finish")

    scores.append(score)
    scores_window.append(score)
    ppo.update(transition_dict)

    if (i + 1) % 10 == 0:
        print(f"Episode {i}, Average Score: {np.mean(scores_window)}")
        ave_scores.append(np.mean(scores_window))
        ppo.save()

env.close()

```

```
    return scores, ave_scores

def draw(scores, ave_scores):
    plt.figure(figsize=(10, 6))
    plt.plot([i for i in range(len(scores))], scores)
    plt.xlabel("Episodes")
    plt.ylabel("Scores")
    plt.title("Scores of Each Episode")
    # plt.show()
    plt.savefig("scores.png")
    plt.figure(figsize=(10, 6))
    plt.plot([i for i in range(len(ave_scores))], ave_scores)
    plt.xlabel("Episodes")
    plt.ylabel("Average Scores")
    plt.title("Average Scores of 10 Episodes")
    # plt.show()
    plt.savefig("ave_scores.png")

if __name__ == "__main__":
    ppo = PPO()
    scores, ave_scores = Train(ppo)
    draw(scores, ave_scores)
```

6.2 Actorbot_test.py

```
import gymnasium as gym
from ppo_Actorbot import PPO

TEST_NUM = 100

def test(ppo, env, episodes):
    success = 0
    failure = 0
    final_score = 0
    for i in range(episodes):
        state = env.reset()[0]
        done = False
        score = 0
        steps = 0
        while not done and steps < 500:
            action = ppo.take_action(state)
            next_state, reward, done, _, _ = env.step(action)
            state = next_state
            score += reward
            steps += 1
        final_score += steps
        if steps < 500:
            print(f"Episode {(i+1):02d} finished after {steps} timesteps")
            success += 1
        else:
            print(f"Episode {(i+1):02d} did not finish")
            failure += 1
```

```
    print(
        f"Success: {success}, Failure: {failure}, Total: {episodes}, Average Steps:
{final_score / episodes}"
    )
    print(f"Success Rate: {success / episodes*100:.2f}%")

if __name__ == "__main__":
    model = PPO()
    model.load("Actorbot_actor.pth", "Actorbot_critic.pth")
    env = gym.make("Acrobot-v1")
    test(model, env, TEST_NUM)
```

6.3 env.py

```
import numpy as np
import pybullet as p
import pybullet_data
import math
from pybullet_utils import bullet_client
from scipy.spatial.transform import Rotation as R

class Env:
    def __init__(self, is_senior, seed, gui=False):
        self.seed = seed
        self.is_senior = is_senior
        self.step_num = 0
        self.max_steps = 150
        self.p = bullet_client.BulletClient(connection_mode=p.GUI if gui else
p.DIRECT)
        self.p.setGravity(0, 0, -9.81)
        self.p.setAdditionalSearchPath(pybullet_data.getDataPath())
        self.init_env()
        self.distance_last = 0

    def init_env(self):
        np.random.seed(self.seed)
        self.fr5 = self.p.loadURDF(
            "fr5_description/urdf/fr5v6.urdf",
            useFixedBase=True,
            basePosition=[0, 0, 0],
```

```

        baseOrientation=p.getQuaternionFromEuler([0, 0, np.pi]),
        flags=p.URDF_USE_SELF_COLLISION,
    )
    self.table = self.p.loadURDF(
        "table/table.urdf",
        basePosition=[0, 0.5, -0.63],
        baseOrientation=p.getQuaternionFromEuler([0, 0, np.pi / 2]),
    )
    collision_target_id = self.p.createCollisionShape(
        shapeType=p.GEOM_CYLINDER, radius=0.02, height=0.05
    )
    self.target = self.p.createMultiBody(
        baseMass=0,
        baseCollisionShapeIndex=collision_target_id,
        basePosition=[0.5, 0.8, 2],
    )
    collision_obstacle_id = self.p.createCollisionShape(
        shapeType=p.GEOM_SPHERE, radius=0.1
    )
    self.obstacle1 = self.p.createMultiBody(
        baseMass=0,
        baseCollisionShapeIndex=collision_obstacle_id,
        basePosition=[0.5, 0.5, 2],
    )
    self.reset()

def reset(self):
    self.step_num = 0
    self.success_reward = 0

```

```

self.terminated = False
self.obstacle_contact = False
neutral_angle = [
    -49.45849125928217,
    -57.601209583849,
    -138.394013961943,
    -164.0052115563118,
    -49.45849125928217,
    0,
    0,
    0,
]
neutral_angle = [x * math.pi / 180 for x in neutral_angle]
self.p.setJointMotorControlArray(
    self.fr5,
    [1, 2, 3, 4, 5, 6, 8, 9],
    p.POSITION_CONTROL,
    targetPositions=neutral_angle,
)

self.goalx = np.random.uniform(-0.2, 0.2, 1)
self.goaly = np.random.uniform(0.8, 0.9, 1)
self.goalz = np.random.uniform(0.1, 0.3, 1)
self.target_position = [self.goalx[0], self.goaly[0], self.goalz[0]]
self.p.resetBasePositionAndOrientation(
    self.target, self.target_position, [0, 0, 0, 1]
)

self.obstacle1_position = [

```

```

        np.random.uniform(-0.2, 0.2, 1) + self.goalx[0],
        0.6,
        np.random.uniform(0.1, 0.3, 1),
    ]
    self.p.resetBasePositionAndOrientation(
        self.obstacle1, self.obstacle1_position, [0, 0, 0, 1]
    )
    for _ in range(100):
        self.p.stepSimulation()

    for i in range(100):
        self.step([-0.3, -0.7, 1, 1, 1, 1])

    return self.get_observation()

def get_observation(self):
    joint_angles = [
        self.p.getJointState(self.fr5, i)[0] * 180 / np.pi for i in range(1, 7)
    ]
    obs_joint_angles = ((np.array(joint_angles, dtype=np.float32) / 180) + 1) /
2
    target_position =
np.array(self.p.getBasePositionAndOrientation(self.target)[0])
    obstacle1_position = np.array(
        self.p.getBasePositionAndOrientation(self.obstacle1)[0]
    )
    self.observation = (
        np.hstack((obs_joint_angles, target_position, obstacle1_position))
        .flatten()

```

```

        .reshape(1, -1)
    )
    return self.observation

def step(self, action):
    self.step_num += 1
    joint_angles = [self.p.getJointState(self.fr5, i)[0] for i in range(1, 7)]
    action = np.clip(action, -1, 1)
    fr5_joint_angles = np.array(joint_angles) + (np.array(action[:6]) / 180 *
np.pi)

    gripper = np.array([0, 0])
    angle_now = np.hstack([fr5_joint_angles, gripper])
    self.reward()
    self.p.setJointMotorControlArray(
        self.fr5,
        [1, 2, 3, 4, 5, 6, 8, 9],
        p.POSITION_CONTROL,
        targetPositions=angle_now,
    )

    for _ in range(20):
        self.p.stepSimulation()

    return self.get_observation()

def get_dis(self):
    gripper_pos = self.p.getLinkState(self.fr5, 6)[0]
    relative_position = np.array([0, 0, 0.15])
    rotation = R.from_quat(self.p.getLinkState(self.fr5, 7)[1])

```

```

        rotated_relative_position = rotation.apply(relative_position)
        gripper_centre_pos = np.array(gripper_pos) + rotated_relative_position
        target_position = np.array(self.p.getBasePositionAndOrientation(self.target)[0])
        return np.linalg.norm(gripper_centre_pos - target_position)

    def reward(self):
        # 获取与桌子和障碍物的接触点
        table_contact_points = self.p.getContactPoints(bodyA=self.fr5,
        bodyB=self.table)
        obstacle1_contact_points = self.p.getContactPoints(
            bodyA=self.fr5, bodyB=self.obstacle1
        )

        for contact_point in table_contact_points or obstacle1_contact_points:
            link_index = contact_point[3]
            if link_index not in [0, 1]:
                self.obstacle_contact = True

        # 计算奖励
        if self.get_dis() < 0.05 and self.step_num <= self.max_steps:
            self.success_reward = 100
            if self.obstacle_contact:
                self.success_reward *= 0.5
            self.terminated = True

        elif self.step_num >= self.max_steps:
            distance = self.get_dis()
            if 0.05 <= distance <= 0.2:

```

```

        self.success_reward = 100 * (1 - ((distance - 0.05) / 0.15))
    else:
        self.success_reward = 0
    if self.obstacle_contact:
        self.success_reward *= 0.5

    self.terminated = True

def cal_success_reward(self):
    """计算完成度奖励"""
    # 获取与桌子和障碍物的接触点
    table_contact_points = self.p.getContactPoints(bodyA=self.fr5,
bodyB=self.table)
    obstacle1_contact_points = self.p.getContactPoints(
        bodyA=self.fr5, bodyB=self.obstacle1
    )

    for contact_point in table_contact_points or obstacle1_contact_points:
        link_index = contact_point[3]
        if link_index not in [0, 1]:
            self.obstacle_contact = True

    # 计算奖励
    if self.get_dis() < 0.05:
        self.success_reward = 100

    elif self.obstacle_contact or self.step_num >= self.max_steps:
        self.success_reward = -10
        self.terminated = True

```

```
        return self.success_reward

def cal_dis_reward(self, distance):
    """计算距离奖励"""
    if self.step_num == 0:
        distance_reward = 0
    else:
        distance_reward = 100 * (self.distance_last - distance)
    # 保存上一次的距离
    self.distance_last = distance
    return distance_reward

def get_reward(self):
    """获取奖励"""
    distance = self.get_dis()
    # 计算奖励
    success_reward = self.cal_success_reward()
    distance_reward = self.cal_dis_reward(distance)
    total_reward = success_reward + distance_reward

    return total_reward

def close(self):
    self.p.disconnect()
```

6.4 ppo_RobotArm.py

```
import os
from stable_baselines3 import PPO
import numpy as np
import pybullet as p
import pybullet_data
import math
from pybullet_utils import bullet_client
from scipy.spatial.transform import Rotation as R
import gymnasium as gym
from gymnasium import spaces
from stable_baselines3.common.monitor import Monitor
from stable_baselines3.common.vec_env import DummyVecEnv
import swanlab
from swanlab.integration.sb3 import SwanLabCallback

# hyperparameters
STATDIM = 12
ACTIONDIM = 6
HIDENDIM = 128
LR = 1e-4
LAMBDA = 0.97
EPOCHS = 5
EPS = 0.1
GAMMA = 0.99
EPISODES = 5000

class Env(gym.Env):
```

```

def __init__(self, is_senior, seed, gui=False):
    super(Env, self).__init__()
    self.seed = seed
    self.is_senior = is_senior
    self.step_num = 0
    self.max_steps = 150
    self.p = bullet_client.BulletClient(connection_mode=p.GUI if gui else
p.DIRECT)
    self.p.setGravity(0, 0, -9.81)
    self.p.setAdditionalSearchPath(pybullet_data.getDataPath())
    self.distance_last = 0
    self.dis = 0
    self.observation_space = spaces.Box(
        low=0,
        high=1,
        shape=(12,),
        dtype=np.float32,
    )
    self.action_space = spaces.Box(
        low=-1,
        high=1,
        shape=(6,),
        dtype=np.float32,
    )
    self.init_env()

def init_env(self):
    np.random.seed(self.seed)
    self.fr5 = self.p.loadURDF(

```

```

        "fr5_description/urdf/fr5v6.urdf",
        useFixedBase=True,
        basePosition=[0, 0, 0],
        baseOrientation=p.getQuaternionFromEuler([0, 0, np.pi]),
        flags=p.URDF_USE_SELF_COLLISION,
    )
    self.table = self.p.loadURDF(
        "table/table.urdf",
        basePosition=[0, 0.5, -0.63],
        baseOrientation=p.getQuaternionFromEuler([0, 0, np.pi / 2]),
    )
    collision_target_id = self.p.createCollisionShape(
        shapeType=p.GEOM_CYLINDER, radius=0.02, height=0.05
    )
    self.target = self.p.createMultiBody(
        baseMass=0,
        baseCollisionShapeIndex=collision_target_id,
        basePosition=[0.5, 0.8, 2],
    )
    collision_obstacle_id = self.p.createCollisionShape(
        shapeType=p.GEOM_SPHERE, radius=0.1
    )
    self.obstacle1 = self.p.createMultiBody(
        baseMass=0,
        baseCollisionShapeIndex=collision_obstacle_id,
        basePosition=[0.5, 0.5, 2],
    )
    self.reset()

```

```

def reset(self, seed=None, options=None):
    super().reset(seed=seed) # 调用父类的 reset 方法，确保种子被正确
    设置

    if seed is not None:
        np.random.seed(seed)
        self.seed = seed

    self.step_num = 0
    self.success_reward = 0
    self.terminated = False
    self.obstacle_contact = False
    neutral_angle = [
        -49.45849125928217,
        -57.601209583849,
        -138.394013961943,
        -164.0052115563118,
        -49.45849125928217,
        0,
        0,
        0,
    ]
    neutral_angle = [x * math.pi / 180 for x in neutral_angle]
    self.p.setJointMotorControlArray(
        self.fr5,
        [1, 2, 3, 4, 5, 6, 8, 9],
        p.POSITION_CONTROL,
        targetPositions=neutral_angle,
    )
    self.goalx = np.random.uniform(-0.2, 0.2, 1)

```

```

        self.goaly = np.random.uniform(0.8, 0.9, 1)
        self.goalz = np.random.uniform(0.1, 0.3, 1)
        self.target_position = [self.goalx[0], self.goaly[0], self.goalz[0]]
        self.p.resetBasePositionAndOrientation(
            self.target, self.target_position, [0, 0, 0, 1]
        )

        self.obstacle1_position = [
            np.random.uniform(-0.2, 0.2, 1) + self.goalx[0],
            0.6,
            np.random.uniform(0.1, 0.3, 1),
        ]
        self.p.resetBasePositionAndOrientation(
            self.obstacle1, self.obstacle1_position, [0, 0, 0, 1]
        )
        for _ in range(100):
            self.p.stepSimulation()

        for _ in range(100):
            action = [-0.3, -0.7, 1, 1, 1, 1]
            self.step(action)

        return self.get_observation(), {}

def get_observation(self):
    joint_angles = [
        self.p.getJointState(self.fr5, i)[0] * 180 / np.pi for i in range(1, 7)
    ]
    obs_joint_angles = ((np.array(joint_angles, dtype=np.float32) / 180) + 1) /

```

2

```
        target_position =  
np.array(self.p.getBasePositionAndOrientation(self.target)[0])  
        obstacle1_position = np.array(  
            self.p.getBasePositionAndOrientation(self.obstacle1)[0]  
        )  
        self.observation = (  
            np.hstack((obs_joint_angles, target_position, obstacle1_position))  
            .flatten()  
            .reshape(1, -1)  
        )  
        return self.observation  
  
def step(self, action):  
    self.step_num += 1  
    joint_angles = [self.p.getJointState(self.fr5, i)[0] for i in range(1, 7)]  
    action = np.clip(action, -1, 1)  
    fr5_joint_angles = np.array(joint_angles) + (np.array(action[:6]) / 180 *  
np.pi)  
    gripper = np.array([0, 0])  
    angle_now = np.hstack([fr5_joint_angles, gripper])  
    self.reward()  
    self.p.setJointMotorControlArray(  
        self.fr5,  
        [1, 2, 3, 4, 5, 6, 8, 9],  
        p.POSITION_CONTROL,  
        targetPositions=angle_now,  
    )
```

```

        for _ in range(20):
            self.p.stepSimulation()

        observation = self.get_observation()
        reward = self.calculate_reward()
        done = self.terminated
        info = {}
        return observation, reward, done, info, {}

    def get_dis(self):
        gripper_pos = self.p.getLinkState(self.fr5, 6)[0]
        relative_position = np.array([0, 0, 0.15])
        rotation = R.from_quat(self.p.getLinkState(self.fr5, 7)[1])
        rotated_relative_position = rotation.apply(relative_position)
        gripper_centre_pos = np.array(gripper_pos) + rotated_relative_position
        target_position =
np.array(self.p.getBasePositionAndOrientation(self.target)[0])
        return np.linalg.norm(gripper_centre_pos - target_position)

    def reward(self):
        # 获取与桌子和障碍物的接触点
        table_contact_points = self.p.getContactPoints(bodyA=self.fr5,
bodyB=self.table)

        obstacle1_contact_points = self.p.getContactPoints(
            bodyA=self.fr5, bodyB=self.obstacle1
        )

        for contact_point in table_contact_points or obstacle1_contact_points:
            link_index = contact_point[3]

```

```
        if link_index not in [0, 1]:
            self.obstacle_contact = True

# 计算奖励
if self.get_dis() < 0.05 and self.step_num <= self.max_steps:
    self.success_reward = 100
    if self.obstacle_contact:
        if self.is_senior:
            self.success_reward = 20
        elif not self.is_senior:
            self.success_reward = 50
    else:
        return
    self.terminated = True

elif self.step_num >= self.max_steps:
    distance = self.get_dis()
    if 0.05 <= distance <= 0.2:
        self.success_reward = 100 * (1 - ((distance - 0.05) / 0.15))
    else:
        self.success_reward = 0
    if self.obstacle_contact:
        if self.is_senior:
            self.success_reward *= 0.2
        elif not self.is_senior:
            self.success_reward *= 0.5

    self.terminated = True
```

```

def calculate_reward(self):
    # 获取与桌子和障碍物的接触点
    table_contact_points = self.p.getContactPoints(bodyA=self.fr5,
bodyB=self.table)

    obstacle1_contact_points = self.p.getContactPoints(
        bodyA=self.fr5, bodyB=self.obstacle1
    )

    for contact_point in table_contact_points or obstacle1_contact_points:
        link_index = contact_point[3]
        if link_index not in [0, 1] and not self.obstacle_contact:
            self.obstacle_contact = True
            return -2

    # 计算奖励
    if self.get_dis() < 0.05 and self.step_num <= self.max_steps:
        reward = 4
        self.terminated = True
    elif self.step_num >= self.max_steps:
        reward = -10 * self.get_dis()
        self.terminated = True
    elif self.dis - self.get_dis() > 0:
        reward = (
            0.005 * np.exp(-self.step_num / 50)
            + 0.02 * np.exp(-self.get_dis())
            + 0.005
        )
        self.dis = self.get_dis()
    else:
        reward = -0.03

```

```

        return reward

def close(self):
    if self.p is not None:
        self.p.disconnect()
        self.p = None

def Train():
    env = Env(is_senior=False, seed=100, gui=False)

    model = PPO(
        "MlpPolicy",
        env=env,
        device="cpu",
        verbose=1,
        policy_kwargs=dict(
            net_arch=dict(pi=[HIDENDIM, HIDENDIM * 2], vf=[HIDENDIM,
HIDENDIM * 2])),
        ),
        gamma=GAMMA,
        gae_lambda=LAMBDA,
        n_epochs=EPOCHS,
        clip_range=EPS,
        learning_rate=LR,
    )

    model.learn(
        total_timesteps=500000,
        callback=SwanLabCallback(

```

```

        project="PPO",
        experiment_name="MlpPolicy",
        verbose=2,
    ),
)

model.save("RobotArm")
env.close()
swanlab.finish()

class MyCustomAlgorithm:
    def __init__(self):
        self.path = os.path.join(os.path.dirname(__file__), "RobotArm.zip")
        self.model = PPO.load(self.path, device="cpu")
        self.discount = 0.8
        self.pre_action = [-0.3, -0.7, 1, 1, 1, 1]

    def get_action(self, observation, env_step_num):
        if env_step_num < 100:
            action = self.pre_action
        elif env_step_num < 150:
            action = self.model.predict(observation[0], deterministic=True)[0]
            action = [x * self.discount if x != 1 and x != -1 else x for x in action]

        return action

if __name__ == "__main__":
    Train()

```

6.5 RobotArm_test.py

```
import os

from env import Env

from ppo_RobotArm import MyCustomAlgorithm

TEST_NUM = 100

def test(algorithm, test_num):
    env = Env(is_senior=False, seed=100, gui=False)
    done = False
    num_episodes = test_num
    final_score = 0
    total_steps = 0
    total_distance = 0
    total_success_num = 0
    total_fail_num = 0
    for i in range(num_episodes):
        score = 0
        done = False
        env.reset()
        while not done:
            observation = env.get_observation()
            action = algorithm.get_action(observation, env.step_num)
            obs = env.step(action)
            score += env.success_reward
            done = env.terminated

        total_steps += env.step_num
```

```
        total_distance += env.get_dis()
        final_score += score

    if score == 100:
        total_success_num += 1
    else:
        total_fail_num += 1

    print(
        f"Test_{(i+1):02d} completed. steps {env.step_num}, Distance
{env.get_dis():.4f}, Score {score:.2f}"
    )

    final_score /= num_episodes
    avg_distance = total_distance / num_episodes
    avg_steps = total_steps / num_episodes

    print(
        f"Test completed. Total steps: {avg_steps:.1f}, Final distance:
{avg_distance:.4f}, Final score: {final_score:.2f}"
    )

    print(
        "Total success num:",
        total_success_num,
        "Total fail num:",
        total_fail_num,
        "Total episodes:",
        num_episodes,
```

```
    )  
    print("Success rate:", total_success_num / num_episodes * 100, "%")  
    env.close()  
  
if __name__ == "__main__":  
    algorithm = MyCustomAlgorithm()  
    test(algorithm, TEST_NUM)
```

参考文献:

- 【1】 John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, Oleg Klimov
Proximal Policy Optimization Algorithms (<https://arxiv.org/abs/1707.0634>)
