中国科学院大学
University of Chinese Academy of Sciences

演化计算课程报告

Acrobot 机器人起摆控制若干算法研究

姓名　涂崧峻 [1], 张曜程 [*1], 徐凯旋 [*1], 鲍伟 [*2]

院所　中科院自动化所 [1], 中科院高能物理所 [2]

2023 年 10 月 24 日

目录

1 问题描述

本文研究二阶体操机器人 Acrobot 的起摆问题。Acrobot 是一种典型的双连杆欠驱动机器人系统结构，其特点是被驱动关节少于被驱动自由度。该系统由在两个对应关节上旋转的两个连杆组成，其中第二个连杆受外扭矩驱动为主动连杆，另一个连杆为被动连杆。Acrobot 机器人的典型结构如图1所示。

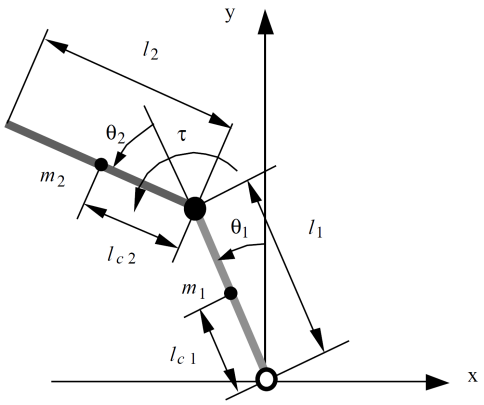

图 1: Acrobot 的结构示意图

本实验主要完成机器人的上移控制，即是将两个连杆从底部平衡位置旋转到顶部平衡位置。由于系统的非线性程度高，很难使用单一的控制器完成从底部平衡位置到顶部平衡位置的控制任务。一种控制方法是设置一个顶部能量吸引域，在吸引域外采用非线性控制方法（如模糊控制，Bang-Bang 控制等）方法，在吸引域内将系统线性化，用 LQR 控制器达到顶部的稳定平衡。非线性控制器往往参数较多，且对参数敏感，有必要对其进行参数的搜索和整定；而 LQR 控制器的求解方法简单，只需求解代数 Riccati 方程就可以实现线性系统的镇定。因此，本文主要对吸引域外的非线性控制器的参数整定进行研究。

本文的代码开源，详见：

https://github.com/TU2021/UCAS_Project_EvolutionaryComputation

2 环境与优化算法工具箱

2.1 Gym 中的 Acrobot 经典控制环境修改

Gym 环境一般用于强化学习的智能体训练。用于我们的控制算法需要进行一些修改。参考 [1]，修改后的 gym 中的 Acrobot 环境如图2所示。

首先，Gym 中的 Acrobot-v1 环境为离散动作控制环境，固定动作输入为离散的三个状态 $-1, 0, 1$，我们将其修改为连续输入的状态，输入动作 (即力矩) 为 $[-U_{max}, +U_{max}]$。

第二，对两个连杆的质量，长度等进行修改，修改成我们需要的参数。

第三，对 episode 终止的条件进行修改。在控制算法中，我们仅仅需要环境的动力学即可，因此回合结束的条件应修改成满足能量吸引域的条件。

最后，加入能量计算的内置函数，用于计算系统的动能和势能；有约束的情况对两个连杆的角度加入约束。

图 2: 修改后的 gym 中的 Acrobot 环境示意图

2.2 Geatpy 进化算法工具箱

Geatpy 是一个由华南农业大学、暨南大学、华南理工大学等优秀硕士以及一批优秀校友、优秀本科生组成的团队研究的一个高性能实用型进化算法工具箱，提供了许多已实现的进化算法各项操作的函数，如初始化种群、选择、交叉、变异、多目标优化参考点生成、非支配排序、多目标优化 GD、IGD、HV 等指标的计算等等 [2]。Geatpy 在 Python 上提供简便易用的面向对象的进化算法框架。通过继承问题类，可以轻松把实际问题与进化算法相结合。Geatpy 还提供多种进化算法模板类，涵盖遗传算法、差分进化算法、进化策略等，可以通过调用算法模板，轻松解决单目标优化、多目标优化、组合优化、约束优化等问题。

本次实验使用 Geatpy2.7 的模板，通过调用算法模板，实现遗传算法的设计。

2.3 Stable Baseline3 强化学习算法工具箱

Stable Baselines3 是一个用于强化学习的 Python 库，它是 OpenAI Gym 的扩展，用于训练和评估强化学习智能体。Stable Baselines3 是 Stable Baselines 库的第三个主要版本，该工具箱提供了可以直接调用的 RL 算法模型，如 A2C、DDPG、DQN、HER、PPO、SAC、TD3，方便用户训练和比较不同算法的性能，被广泛应用于各种强化学习任务训练中。

本次实验使用 Stable Baselines3 2.1.0 的模板，通过调用算法模板，实现 Soft Actor-Critic(SAC) 算法的设计。

3 考虑关节无约束的模糊控制器参数搜索

3.1 模糊控制方案简述

我们使用模糊神经网络 (FNN) 实现对机器人吸引域外的控制，我们的控制方案和机器人结构参数都与文献 [3] 中相同，故不再详细描述控制方案，仅简述模糊控制器的基本思路，以及对实现过程进行详细描述。文献 [3] 使用了一个五层的 FNN，第一层输入了角度信息 $\theta_1, \theta_1+\theta_2$ 和角速度信息 $\omega_1, \omega_1+\omega_2$，第二层进行对输入语义的标准化，保证输入的角度在 $-\pi$ 到 π 的范围内，第三层进行隶属度的计算，这里引入 4 个待优化的隶属度参数 $k_i, i=1,2,3,4$:

$$O_{ij}^3 = (1+sign(j)\tanh(k_i O_i^2)/2,\ i=1,2,3,4, j=\pm 1 \tag{1}$$

第四层为模糊规则计算，根据前四层的输出 $O^i, i=1,2,3,4$，可以设置 16 个模糊规则，对应 16 个控制器输出参数，由于对称性可以简化成 8 个待优化的参数 $\eta_m, m=1,...,8$，最后，整个控制器输出可以写成:

$$O^5 = \sum_{m=1}^{16} \eta_m \prod_{i=1}^{4} O_{ij}^3,\ j=\pm 1, m=1,...,16 \tag{2}$$

我们用整个非线性过程的能量函数作为待优化目标，同时也设置为适应度函数，表示为:

$$L = fitness = \frac{1}{\sum_{t=1}^{N} e(t)},\ e(t) = |E - E_{top}| \tag{3}$$

其中 $e(t)$ 为当前系统能量与最顶部稳定时的能量之差。进入吸引域的条件为:

$$|E - E_{top}| < \epsilon \ \text{ and } \ |E_p - E_{top}| < \epsilon \tag{4}$$

其中 E_p 为系统动能，上式表示进入线性吸引域的条件为系统总能量之差与目标位置能量足够小，且动能也应足够小。

3.2 遗传算法求解参数搜索问题

3.2.1 算法流程与实现

我们用 Geatpy 中的增强精英保留的遗传算法 (SEGA)[4] 解决该问题。SEGA 的具体流程如下:

1) 根据编码规则初始化 N 个个体的种群。
2) 若满足停止条件则停止，否则继续执行。
3) 对当前种群进行统计分析，比如记录其最优个体、平均适应度等。
4) 独立地从当前种群中选取 N 个母体。
5) 独立地对这 N 个母体进行交叉操作。

6) 独立地对这 N 个交叉后的个体进行变异。

7) 将父代种群和交叉变异得到的种群进行合并，得到规模为 2N 的种群。

8) 从合并的种群中根据选择算法选择出 N 个个体，得到新一代种群。

9) 回到第 2 步。

借助于 Geatpy 库，SEGA 算法的运行主函数如下：

```
def run(self, prophetPop=None):  # prophetPop 为先知种群（即包含先验知识的种群）
    # ========================== 初始化配置 ==========================
    population = self.population
    NIND = population.sizes
    self.initialization()  # 初始化算法类的一些动态参数
    # ========================== 准备进化 ==========================
    population.initChrom(NIND)  # 初始化种群染色体矩阵
    # 插入先验知识
    if prophetPop is not None:
        population = (prophetPop + population)[:NIND]  # 插入先知种群
    self.call_aimFunc(population)  # 计算种群的目标函数值
    # 计算适应度
    population.FitnV = ea.scaling(population.ObjV, population.CV, self.problem.maxormins)
    # ========================== 开始进化 ==========================
    while not self.terminated(population):
        # 选择
        offspring = population[ea.selecting(self.selFunc, population.FitnV, NIND)]
        # 进行进化操作
        offspring.Chrom = self.recOper.do(offspring.Chrom)  # 重组
        offspring.Chrom = self.mutOper.do(offspring.Encoding, offspring.Chrom, offspring.Field)
        self.call_aimFunc(offspring)  # 计算目标函数值
        population = population + offspring  # 父子合并
        # 计算适应度
        population.FitnV = ea.scaling(population.ObjV, population.CV, self.problem.maxormins)
        # 采用基于适应度排序的直接复制选择生成新一代种群
        population = population[ea.selecting('dup', population.FitnV, NIND)]
    return self.finishing(population)  # 调用 finishing 完成后续工作并返回结果
```

3.2.2 算法参数设置

对于 Acrobot 机器人起摆问题，我们选取二进制编码，种群数为 100，最大迭代次数为 100，交叉概率 $p_c = 0.9$，变异概率 $p_m = 0.1$；仿真器中，选择仿真步长 $\Delta t = 0.02s$，仿真时长为 $10s$，当满足吸引域条件时，自动结束仿真，取吸引域为 $\epsilon = 0.05E_{top}$；参数搜索范围为 $\eta_i \in [-8, 8]$，$k_i \in [0.5, 8]$。

3.2.3 求解结果

对于关节角度不受限的情况，即两个连杆都可以沿着平面 360 度旋转。运行 SEGA 的最优函数值与平均值的变化随着迭代次数的增加的演化情况如图3所示，系统中各部分的参数变化如图4所示。最终得出的参数为：

$$\eta = [-4.7654, -7.0819, 4.5027, 3.6559, -6.1141, -5.1590, 2.3375, 6.5966] \tag{5}$$

$$k = [4.6261, 6.5382, 7.5485, 7.9454] \tag{6}$$

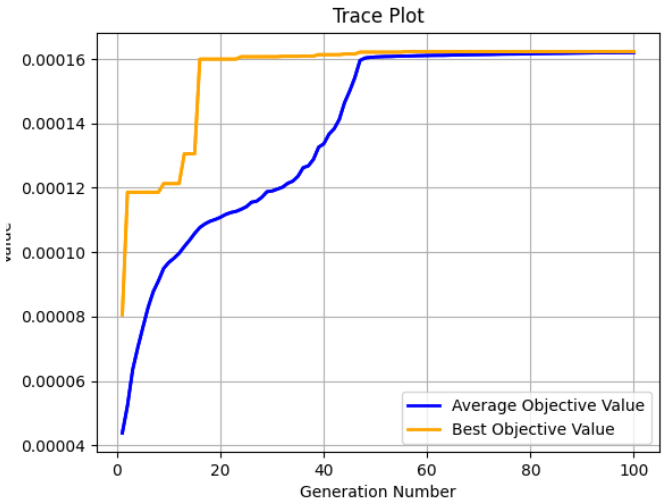

图 3: SEGA 搜索的最优函数值与平均值的变化：训练了 100 个种群，到最后种群中的个体能基本趋向最优解。

最终得出的优化目标最大值为：

$$L = \frac{1}{\sum_{t=1}^{N} e(t)} = 1.62345 \times 10^{-4},\ N = 237(t = 4.74s) \tag{7}$$

3.3 粒子群算法求解参数搜索问题

在本节，我们将使用自适应粒子群算法 (APSO)[5] 优化模糊控制器的参数，并取得了可观的结果。

3.3.1 算法流程与实现

首先，简要回顾一下基本 PSO 算法 [6] 的具体流程：

1) 初始化：粒子个数 m，迭代次数 m，第 i 个粒子在 D 维的搜索空间中的位置 x_i，第 i 个粒子在 D 维的搜索空间中的飞翔速度 v_i，惯性权重因子 ω，速度区间 $[v_{min}, v_{max}]$。

2) 若满足停止条件则停止，否则继续执行。

3) 计算群落下的所有粒子的适应度。

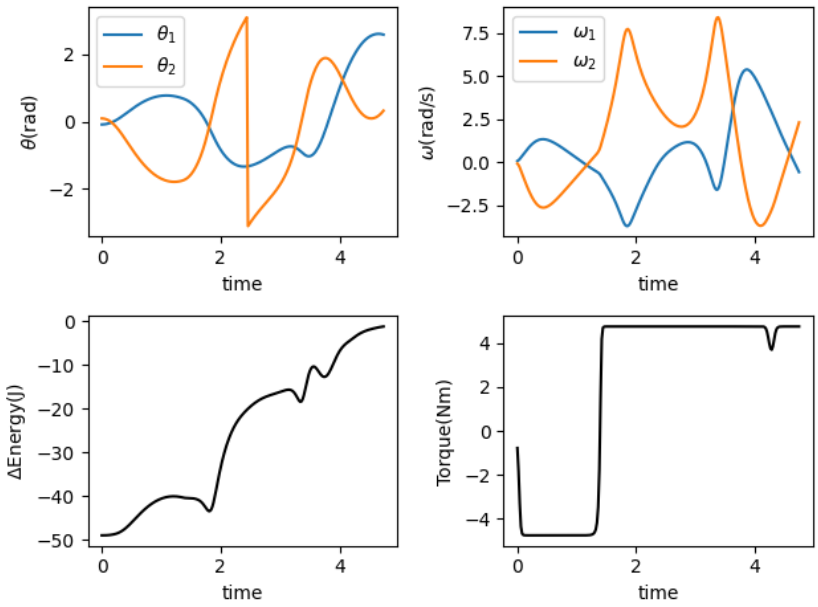

图 4: 最优解下的 Acrobot 在吸引域外部分的状态变化：左上图表示两个关节角度的变化 (突变的原因是进行了 $-\pi$ 到 π 角度的标准化)；右上图表示两个关节角速度的变化；左下角表示系统能量与目标能力的差值随时间变化图，其总体基本是单调的，说明了我们的控制算法能有效使得系统能量稳步上升；右下是模糊控制器输出的力矩随时间的变化，系统输出的力矩可以控制在 $\pm 5N \cdot m$ 内。

4) 适应度比较：

将第 i 个粒子适应度与前一时刻适应度比较，得出最优适应度 p_{best} 及其位置 p_{id}；整个粒子群的适应度比较，得出最优适应度 p_{best} 和最优位置 p_{id}

5) 更新粒子位置和速度：

$$\begin{aligned} v_{id} &= \omega v_{id} + c_1 r_1 (p_{id} - x_{id}) + c_2 r_2 (p_{gd} - x_{id}) \\ x_{id} &= x_{id} + v_{id} \end{aligned} \tag{8}$$

6) 回到第 2 步。

基于此，自适应 PSO 算法通过一个非线性动态自适应调节惯性权重因子 ω 控制粒子速度，具体公式如下所示：

$$w = \begin{cases} w_{\min} + \frac{(w_{\max} - w_{\min})(f - f_{\min})}{f_{\text{avg}} - f_{\min}}, & f \le f_{\text{avg}} \\ w_{\max} & f > f_{\text{avg}} \end{cases} \tag{9}$$

当目标值优于平均目标值的粒子，小的惯性权重因子，使该粒子得以保护；目标值差于平均目标值的粒子，大的惯性权重因子，使该粒子更快地趋于更好的空间。这更适合于优化参数较为敏感的目标，如果只使用基本的 PSO 算法，速度常常达到边界 (最大或最小)，这极大限制了搜索目标的精度，并且算法难以收敛。

我们手动实现了自适应粒子群算法，并未使用已有的 python 库函数，部分核心代码如下：

```python
class Particle:
    def __init__(self, num_vars):
        mean1 = 0  # 第一个均值
        std_dev1 = 3  # 第一个方差
        mean2 = 4  # 第二个均值
        std_dev2 = 3 # 第二个方差
        self.position = np.zeros(num_vars)  # 创建零向量用于存储初始化后的数值
        self.position[:8] = np.random.normal(mean1, std_dev1, 8)
        self.position[num_vars-4:] = np.random.normal(mean2, std_dev2, 4)
        self.velocity = np.random.normal(0.4, 0.2, num_vars)
        self.personal_best_position = None
        self.personal_best_fitness = np.inf
        self.inertia_weight = None  # 添加粒子的惯性权重变量
        self.cognitive_weight = None  # 添加粒子的认知权重变量
        self.social_weight = None  # 添加粒子的社会权重变量

class PSO:
    def __init__(self, problem, num_particles, max_iterations):
        self.problem = problem
        self.num_particles = num_particles
        self.max_iterations = max_iterations
        self.max_velocity = [1.2] * 8 + [0.6] * 4
        self.min_velocity = [-1.2] * 8 + [-0.6] * 4

    def optimize(self):
        num_vars = len(self.problem.lb)
        particles = [Particle(num_vars) for _ in range(self.num_particles)]
        global_best_position = None
        global_best_fitness = np.inf

        # 为可视化设置
        fitness_history = []
        fitness_average_history = []
        fitness_variance_history = []
        for iteration in range(self.max_iterations):
            fitness_average = []
            for particle in particles:
                fitness = self.problem.evaluate(particle.position)
                fitness_average.append(-fitness)
                if fitness < particle.personal_best_fitness:
                    particle.personal_best_position = particle.position
                    particle.personal_best_fitness = fitness

                if fitness < global_best_fitness:
                    global_best_position = particle.position
```

```
            global_best_fitness = fitness

        # 计算自适应权重
        if particle.personal_best_fitness != np.inf:
            particle.inertia_weight = 1 / (np.abs(particle.personal_best_fitness)*20000)
        if global_best_fitness != np.inf:
            particle.cognitive_weight = 1 / (np.abs(global_best_fitness)*15000)
        particle.social_weight = 1 / (np.abs(fitness) * 15000)
```

3.3.2 算法参数设置

对于该模糊控制参数优化而言，粒子群初始化参数是相对敏感的，需要手动不断尝试，具体参数如表1所示。**我们并未对位置进行统一的初始化，因为其涉及到两个不同范围的参数 η 和 k，分开初始化更利于粒子群算法优化。我们选择在搜索空间的局中位置进行高斯分布的初始化 (0 和 4)，并选择相对较大的方差，有利于参数分布的更广，否则算法极易收敛到局部最优或是无法收敛。此外速度初始化不易较大，否则会容易震荡发散**。速度区间在 APSO 算法中相对不那么重要，因为自适应机制会限制速度的更新。

表 1: **PSO 算法参数列表**

Hyperparameter	**Value**
粒子数 m	30
迭代次数 n	200
位置初始化	$x_\eta \sim N(0,3)$
	$x_k \sim N(4,3)$
速度初始化	$v \sim N(0.4,0.2)$
惯性权重因子 ω	$\propto 1/\text{fitness}$
学习因子 c_1, c_2	$\propto 1/\text{fitness}$
速度区间	$v_\eta \in [-1.2, 1.2]$
	$v_k \in [-0.6, 0.6]$

3.3.3 求解结果

对于关节不受限的情况，基本 PSO 算法和自适应 PSO 算法训练过程如图5所示。最终得出的优化目标最大值为：

$$L = \frac{1}{\sum_{t=1}^{N} e(t)} = 1.6199 \times 10^{-4},\ N = 236(t = 4.72s) \tag{10}$$

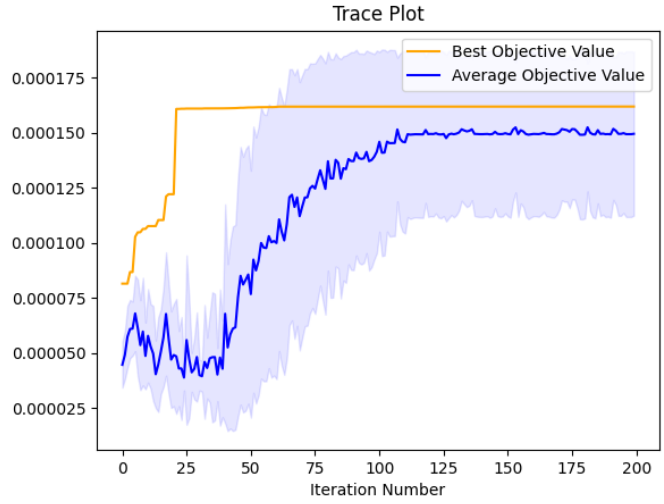

图 5: PSO 搜索的最优适应度与平均值的变化，阴影部分代表平均值的方差，算法最终收敛。

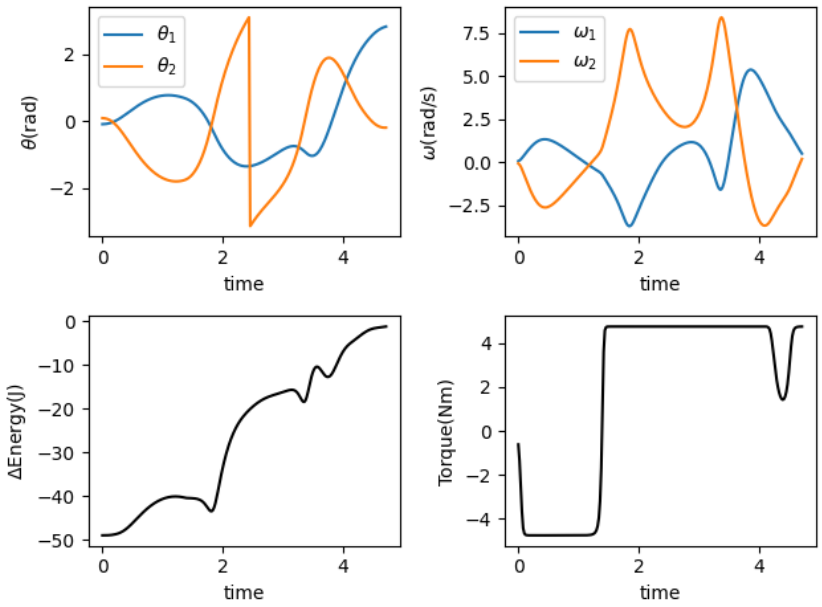

图 6: 最优解下的 Acrobot 在吸引域外部分的状态变化大体与 SEGA 搜索相同，略有不同的是右下图力矩变化在 $t = 4.72s$ 时，力矩减小的更为明显。从图中可以看出，PSO 与 SEGA 搜索到了相同的摆动方式，以最优的方式完成了目标。

4 考虑关节无约束的强化学习控制

4.1 强化学习控制方案简述

我们使用了强化学习算法作为对比算法来对机器人在吸引域外进行控制，在该方法中我们将整个控制过程表征为了马尔科夫链的形式 $\{\mathcal{S}, \mathcal{A}, \rho_0, T(s_{t+1}|s_t, a_t), r(s_t, a_t), \gamma\}$。

其中 ρ_0 为初始状态分布，$\mathcal{S}$ 为状态空间，$\mathcal{A}$ 为动作空间，$T(s_{t+1}|s_t,a_t)$ 为环境动态，$r(s_t,a_t)$ 为奖励函数，γ 为折现因子。强化学习的目标是找到一个策略 $\pi(a_t|s_t):\mathcal{S}\times\mathcal{A}\to[0,1]$ 来最大化整个过程的预期回报：

$$\max_{\pi}\mathbb{E}[\sum_{t=0}^{\infty}\gamma^t r(s_t,a_t)|s_0\sim\rho_0(\cdot),a_t\sim\pi(\cdot|s_t),s_{t+1}\sim T(\cdot|s_t,a_t)] \tag{11}$$

在该方法中我们对奖励函数进行设计使得我们所期望得到的目标控制过程（决策过程）对应的是最大的预期回报。因此，在该 Acrobot 机器人起摆问题中我们使用了系统每个时刻的总能量和动能对奖励函数进行设计，表示为：

$$R(s_t,a_t)=\begin{cases}0, & s_{t+1}\ is\ terminal\\ \frac{-|E-E_{top}|^2-|E_p-E_{top}|^2}{1000}, & else\end{cases} \tag{12}$$

上式表示：线性系统在每一时刻 t 的总能量之差与目标位置能量越小而且系统的动能越小时，所对应的奖励值越大。当系统进入吸引域时，能够对应最大的奖励值。

4.2 SAC 算法求解控制问题

4.2.1 算法流程与实现

面对连续控制问题，可以使用经典的软演员-评论员算法 (Soft Actor-Critic, SAC [7]) 进行求解。借助 Stable-Baseline3 强化学习库，该算法的主要训练流程如下：

1) 初始化状态价值函数网络 $V(s_t)$，目标价值函数网络 $V'(s_t)$，2 个软动作价值函数网络 $Q1(s_t,a_t)$，$Q2(s_t,a_t)$，策略网络 $\pi(a_t|s_t)$。

2) 若满足停止条件则停止，否则继续执行。

3) 与环境交互来收集数据，$\mathcal{D}\leftarrow\{s_t,a_t,r_t,s_{t+1}\}\cup\mathcal{D}$。

4) 更新价值函数网络 V 的权重。

5) 分别更新软动作价值函数网络 $Q1$ 和 $Q2$ 的权重。

6) 更新策略网络 P 的权重。

7) 使用软更新来更新目标价值函数网络 V' 的权重。

8) 回到第 2 步。

其中具体的训练流程可以参考论文 [7], 本文不再详述。在推理时，针对每个状态 s_t，使用策略网络 $\pi(a_t|s_t)$ 推理出动作 a_t 即可。

4.2.2 算法参数设置

训练和推理过程中所对应的部分参数如表 2所示：

表 2: **SAC 算法参数列表**

Hyperparameter	**Value**
Architecture	
所有网络 $(V, V', Q1, Q2, \pi)$ 的隐藏层维度	256
所有网络的隐藏层数	2
所有网络的激活函数	ReLU
Hyperparameter	
所有网络的优化器	Adam
所有网络的训练批大小	256
折扣因子 γ	0.99
所有网络的学习率	3e-4
缓冲池大小	1e6
软更新系数	5e-3
训练总步数	1e6

4.2.3 求解结果

训练过程中的平均成功率和回报值如图7所示，控制过程中机器人的各项性能参数如图8所示。基于 SAC 的控制算法可以完成优于 GA 和 PSO 的模糊控制器的性能，其摆起时两个关节的转动幅度更小，但控制信号波动较为剧烈。

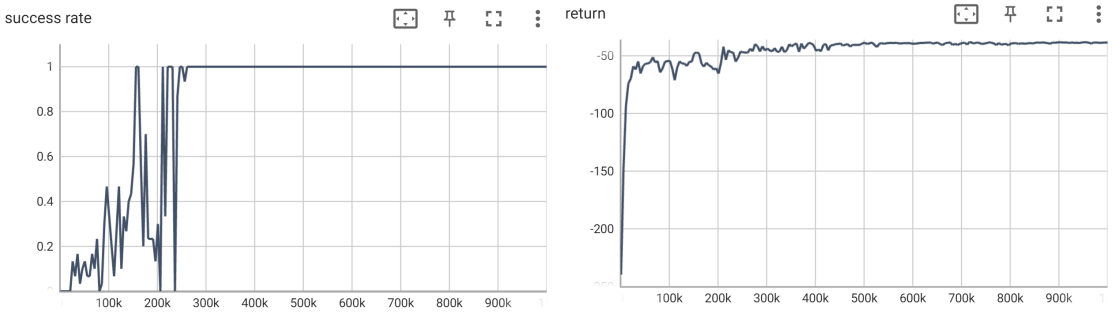

图 7: SAC 算法在训练过程中的成功率 (success rate) 和平均回报值 (return) 的变化，每次测试结果由 30 个环境得出。可以看到随着训练步数的增加，大概到 220k 步时，成功率到达最高值 100%，并保持该值不变, 而平均回报值在训练到 400k 步时收敛到-39 左右。

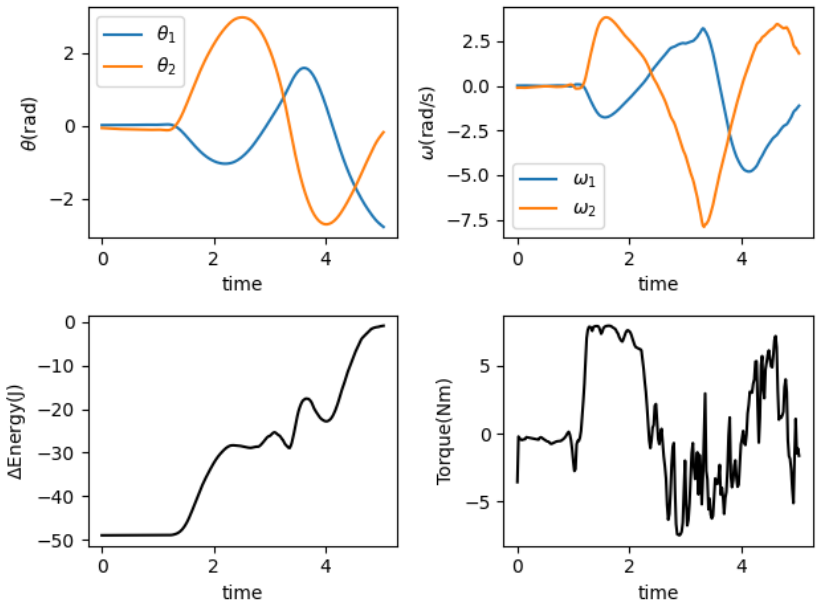

图 8: SAC 算法收敛时 Acrobot 在吸引域外部分的状态变化：左上图表示两个关节角度的变化；右上图表示两个关节角速度的变化；左下角表示系统能量与目标能力的差值随时间变化图，其总体基本是单调的，说明了我们的控制算法能有效使得系统能量稳步上升；右下是控制器输出的力矩随时间的变化，系统输出的力矩范围相比 SEGA 优化下的模糊控制器的输出范围更大，同时波动程度较为剧烈。

5 考虑关节角度约束的 Acrobot 起摆控制

除了对关节角度无约束条件下 Acrobot 机器人起摆问题进行求解之外，我们在 Acrobot 机器人第二个关节的变化范围为 $[-\frac{2\pi}{3}, \frac{2\pi}{3}]$ 的约束下对起摆问题进行了求解。我们研究了基于 SEGA 的模糊控制和 SAC 的两种控制器设计，经过调试后，我们并没有得到使得 Acrobot 到达理想状态的结果。SEGA 在最大时间范围内无法搜索到稳定解，而 SAC 搜索到了一个稳定停留的位置。问题无法求解的原因可能是力矩只施加在两个关节的节点上，本身难以在两关节夹角受约束的情况获得最优解；也可能是设计的模糊控制器本身较为粗糙，无论如何优化参数都难以做到进入平衡域。

我们绘制出 SEGA 对于关节约束下的求解结果如图9和图10所示，**SEGA 在指定时间内没有达到平衡态，机器人在平面空间中不断旋转**。

SAC 训练过程中的平均成功率和回报值如图11所示。最终关节稳定时的角度如图12所示，由于关节的角度受到到限制，**机器人最后未能成功摆动到吸收域之内，而是稳定在了吸收域之外**。

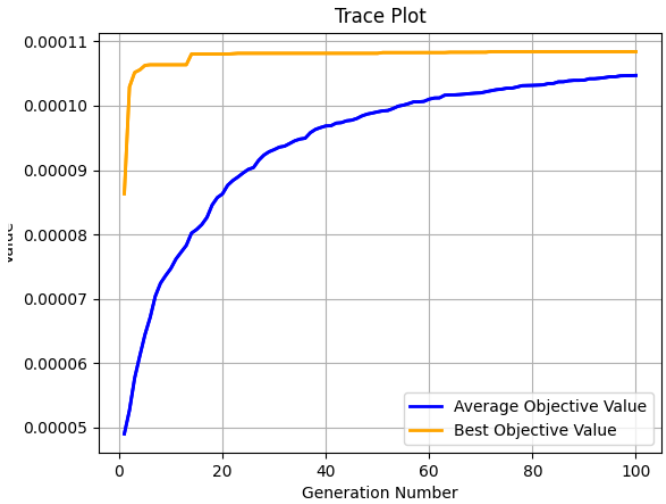

图 9: 关节约束下 SEGA 搜索的最优函数值与平均值的变化。

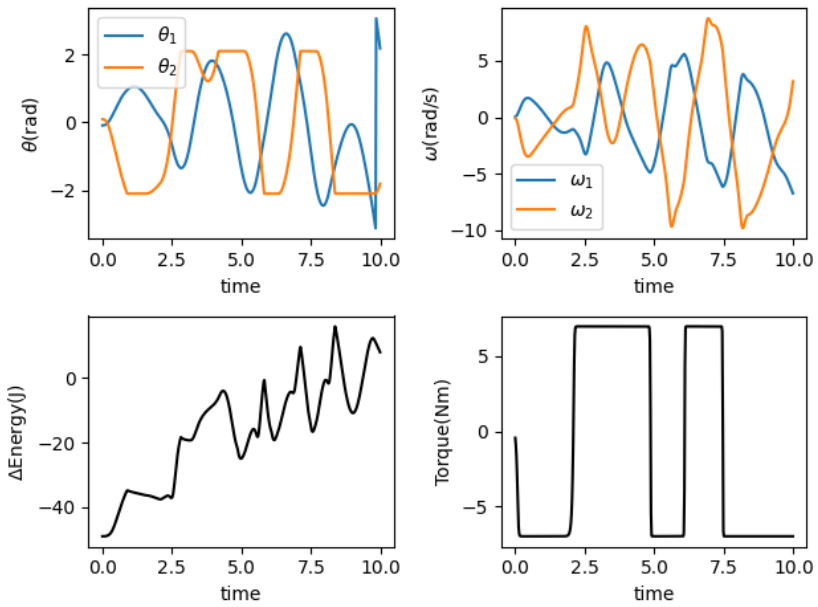

图 10: 关节约束下最优解下的 Acrobot 在吸引域外部分的状态变化：SEGA 在指定时间内没有达到平衡态，机器人在平面空间中不断旋转。

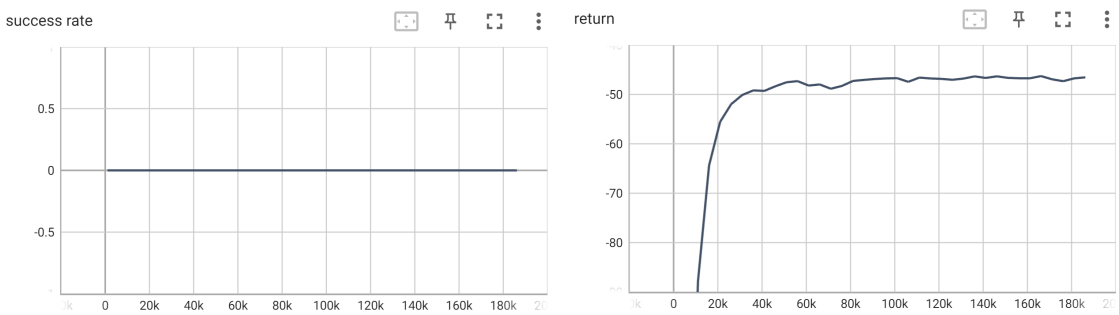

图 11: SAC 算法在训练过程中的成功率 (success rate) 和平均回报值 (return) 的变化，每次测试结果由 30 个环境得出。可以平均回报值在训练到 400k 步时收敛到-48 左右，但是成功率却保持为 0 不变，这说明前文提出的奖励函数并不适合于角度受限时的情况

图 12: 关节角度受限下，机器人所达到的稳定位置

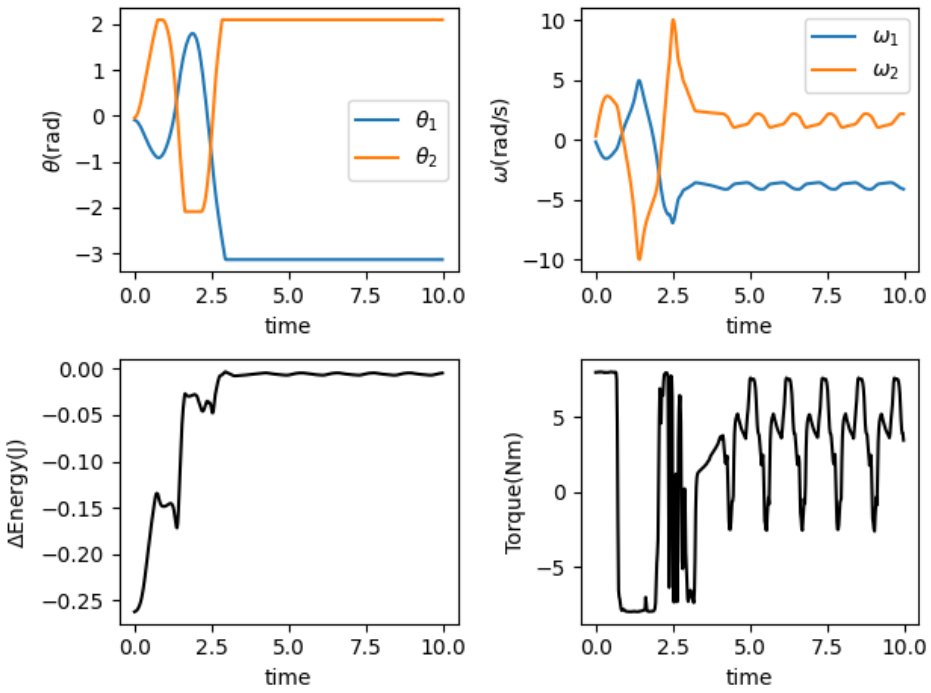

图 13: SAC 算法收敛时 Acrobot 机器人在吸引域外部分的状态变化，虽然总体的能量在不断减少，但是最终没有到达指定的位置。

6 总结

本文研究了 Acrobot 机器人的起摆问题。首先在关节角度无约束的情况下，我们分别使用了模糊控制器和强化学习方法求解该问题：我们分别用 SEGA 和 APSO 算法优化模糊控制器参数，得到了相似的结果；我们用 SAC 直接求解求解连续控制问题，得到的摆动过程的效率要优于演化算法。最后，我们考虑关节约束的情况，演化算法求解出的模糊控制器和强化学习控制器虽然都未完成控制任务，但表现出了完全不同的结果。下一步的工作可以对约束下的 Acrobot 机器人的起摆问题进行研究和讨论。

参考文献

[1] Gym documentation for acrobot. https://gymnasium.farama.org/environments/classic_control/acrobot/.

[2] Geatpy 进化算法工具箱. http://geatpy.com/.

[3] Dongbin Zhao and Jianqiang Yi. Ga-based control to swing up an acrobot with limited torque. *Transactions of the Institute of Measurement and Control*, 28(1):3–13, 2006.

[4] John H Holland. Genetic algorithms. *Scientific american*, 267(1):66–73, 1992.

[5] Zhi-Hui Zhan, Jun Zhang, Yun Li, and Henry Shu-Hung Chung. Adaptive particle swarm optimization. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 39(6):1362–1381, 2009.

[6] James Kennedy and Russell C Eberhart. A discrete binary version of the particle swarm algorithm. In *1997 IEEE International conference on systems, man, and cybernetics. Computational cybernetics and simulation*, volume 5, pages 4104–4108. IEEE, 1997.

[7] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pages 1861–1870. PMLR, 2018.