实验目标：在物理仿真环境下，通过操控髋、膝、踝关节及其腹部姿态，为一个七连杆双足机器人生成一个能够在复杂环境中，保持平衡并独立行走的步态模型。

实验工具：物理引擎、强化学习算法工具包gym、机器学习框架tensorflow

实验方案：

1. 参考游戏物理引擎的设计方法，编写一个较为简单的，具备计算运动、旋转功能的物理仿真环境，作为实验所需的模拟环境。
2. 构造一个七连杆双足机器人模型，通过改变髋(3x2)、膝(1x2)、踝(2x2)关节及其腹部(3)共15个角度的大小来使机器人运动；设计强化学习的策略模型奖励与价值函数，输入为15个角度大小及机器人位姿(视条件而定，多做尝试)，输出为15个角度的变化量，这一阶段的训练目标是让机器人能够从坐、躺等姿势转换到站立姿势，并保持平稳站立，且能对抗一定外力干扰(推动后保持平衡站立)。
3. 在平衡能力达到要求后，开始为机器人设定目标位置，使其在以两足行走的方式移动到目标位置，整个过程中保持立直姿态，同时在移动速度与能量消耗两方面做出权衡，达到速度快、能耗低的要求。
4. 修改机器人模型，或自己设计机器人模型，将各部位长度、关节位置等作为变量，结合前一阶段中的学习方法，对比不同模型的平衡能力与移动能力，选取3到5个性能较好的模型进行进一步的学习，训练在负重条件下、复杂环境中的平衡、移动能力。

1 强化学习基本原理

1.1 有限马尔科夫决策过程 (*Finite Markov Decision Processes*)

强化学习是一种机器学习算法，该算法使用一个代理，代理通过获取环境状态、做出反应动作、得到环境反馈等一系列行为来进行学习，最终达到既定目标。
有限马尔科夫决策过程是序贯决策 (*sequential decision making*) 的一种经典形式，即行为不仅影响到立即奖励，还会对后续的状态造成影响，导致未来所获奖励的变化。
有限马尔科夫决策过程是对强化学习问题的最理想的表达。

强化学习有两个基本的组成部分：

1. 行动代理 (*agent*)
   Agent即是实验中使用的机器人模型，通过程序控制其关节角度变化来与环境进行交互。其具有行为策略policy作为决策核心，接收仿真环境的state和reward等信息作为输入，给出下一步行为动作action，
2. 环境 (*environment*)
   Environment负责对agent给的行为做出反应，给出反馈信号，并拥有其本身的状

态数据可供agent获取。

在agent与environment交互的过程中，会产生三个信号：

1. 行为 (*action*)
   Action是agent根据环境状态做出的行为决策，其决定了agent的下一步动作，该行为会对环境造成影响并获得反馈。
2. 奖励 (*reward*)
   Reward是某次action后由environment的变化情况得到的标量信号，其表示了这一次action的好坏，是算法学习所需要用到的核心信号。
3. 状态 (*state*)
   State是environment固有的状态信息，会随着agent做出的action而产生变化，是agent行为决策的依据。

强化学习还有三个重要的概念： 策略 (*policy*)：Agent根据state做出action的依据称为行为策略policy，policy可以视作一个从state到所有action的概率的映射。

回报 (*return*)：由于policy的最终目标是使得agent获得的总reward最高，而不是单次action获得的reward最高。Return所表示的就是在较长的一段时间里，获得的总reward之和。其定义如下：

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{T-t-1} R_T = \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$$

其中 $\gamma \in [0,1]$ 叫做折扣率(*discounting rate*)，它表示代理考虑奖励的范围。
γ 越小，表示代理越短视，当 $\gamma = 0$ 时，代理只会考虑即刻得到的奖励，而不考虑未来可能获得奖励的情况。反之，γ 越大，表示代理考虑得越长远，当 $\gamma = 1$ 时，表示代理对所有奖励一视同仁，代理追求的就是总奖励之和。

值函数 (*value function*)：值函数表明在给定策略 π 下，一个 state (或者 state-action pair) 后所能得到的回报的期望值：

$$v_\pi(s) \doteq \mathbb{E}[G_t|S_t = s] = \mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s]$$

$$q_\pi(s,a) \doteq \mathbb{E}_\pi[G_t|S_t = s, A_t = a] = \mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s, A_t = a]$$

agent 和 environment 交互的过程是持续的：agent 根据 state 选择 action；environment 受到 action 影响给出新的 state，同时产生一个 reward 信号；agent 根据新的 state 再次做出决策，选择下一个 action，并且 agent 会用 reward 来更新它的行为策略。

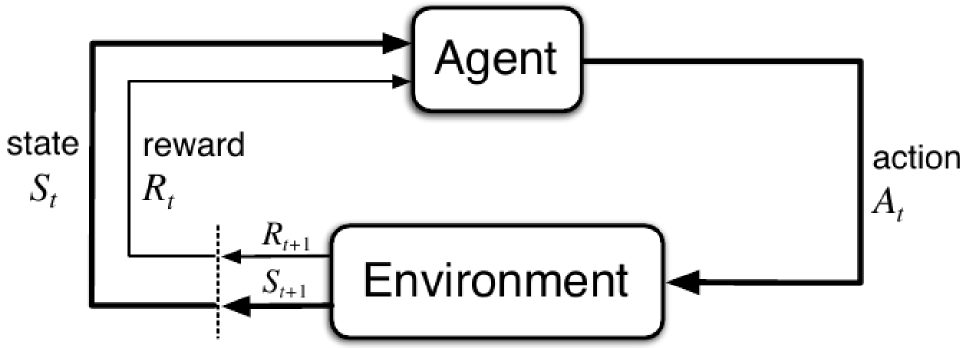

Figure 3.1: The agent–environment interaction in a Markov decision process.

在一个 *finite* MDPs 中，states，actions 和 rewards 取值的集合($\mathcal{S}, \mathcal{A}, and\ \mathcal{R}$)中的元素个数都是有限的。
在这种情况下，随机变量 R_t 和 S_t 有明确定义的离散概率分布，且仅依赖于前一个 state 和 action.
即对于这些随机变量，出现在时间 t 的具体值，$s' \in \mathcal{S}$ 和 $r \in \mathcal{R}$，有一个概率：

$$p(s', r \mid s, a) \doteq \Pr\{S_t = s', R_t = r \mid S_{t-1} = s, A_{t-1} = a\}$$

条件概率在此处表明对于所有的 $s \in \mathcal{S},\ a \in \mathcal{A}(s).$，其概率和为 1:

$$\sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r \mid s, a) = 1, \quad \text{for all } s \in \mathcal{S},\ a \in \mathcal{A}(s).$$

四参函数 p 完整地表述了一个有限马尔科夫决策的动态过程。利用它能够计算关于 environment 的一切其它信息。
比如状态转移概率(*state-transition probabilities*):

$$p(s' \mid s, a) \doteq \Pr\{S_t = s' \mid S_{t-1} = s, A_{t-1} = a\} = \sum_{r \in \mathcal{R}} p(s', r \mid s, a).$$

又如一对 state-action 的期望奖励:

$$r(s, a) \doteq \mathbb{E}[R_t \mid S_{t-1} = s, A_{t-1} = a] = \sum_{r \in \mathcal{R}} r \sum_{s' \in \mathcal{S}} p(s', r \mid s, a)$$

还有一组 state-action-(next-state) 的期望奖励:

$$r(s, a, s') \doteq \mathbb{E}[R_t \mid S_{t-1} = s, A_{t-1} = a, S_t = s'] = \sum_{r \in \mathcal{R}} r \frac{p(s', r \mid s, a)}{p(s' \mid s, a)}.$$

1.2 贝尔曼方程 (*Bellman Equation*)

解决强化学习问题，就是要找到一个能够获得最大累计奖励的策略 policy；
对于 finite MDPs，能够精确地定义一个最优 policy
value functions 对 policies 定义部分排序

如果一个策略 π 在所有的状态 states 下，其期望回报 return 均大于或等于另一个策略 π' ，那么就称 π 优于 π'
用数学语言表达就是： $\pi \geq \pi' \; if \; and \; only \; if \; v_\pi(s) \geq v_{\pi'}(s) \; for \; all \; s \in \mathcal{S}$
总是会有至少一个策略会优于或等于所有其它的策略，称之为最优策略(*optimal policy*)，将它或它们统一定义为 π_* ，它们共享相同的 state-value function ，称为最优 状态-值 函数(*optimal state-value function*)，写作 v_* ; 同理有最优 状态-动作-值 函数 $q_*(s,a)$ ，它们的定义如下：

$$
\begin{aligned}
v_*(s) &\doteq \max_\pi v_\pi(s), \\
q_*(s,a) &\doteq \max_\pi q_\pi(s,a), \\
&= \mathbb{E}\Big[R_{t+1} + \gamma v_*(S_{t+1}) \mid S_t = s, A_t = a\Big].
\end{aligned}
$$

贝尔曼最优方程(*Bellman optimality equation*)表现出：在最优策略下，一个状态的 value 必须等于该状态下的最优 action 的期望回报：

$$
\begin{aligned}
v_*(s) &= \max_{a \in \mathcal{A}(s)} q_{\pi_*}(s,a) \\
&= \max_a \mathbb{E}_{\pi_*}\Big[G_t \mid S_t = s, A_t = a\Big] \\
&= \max_a \mathbb{E}_{\pi_*}\Big[R_{t+1} + \gamma G_{t+1} \mid S_t = s, A_t = a\Big] \\
&= \max_a \mathbb{E}\Big[R_{t+1} + \gamma v_*(S_{t+1}) \mid S_t = s, A_t = a\Big] \\
&= \max_a \sum_{s',r} p(s',r \mid s,a)\Big[r + \gamma v_*(s')\Big]
\end{aligned}
$$

状态-动作 值函数的贝尔曼最优方程：

$$
\begin{aligned}
q_*(s,a) &= \mathbb{E}_{\pi_*}\Big[R_{t+1} + \gamma \max_a{}' q_*(S_{t+1},a') \mid S_t = s, A_t = a\Big] \\
&= \sum_{s',r} p(s',r \mid s,a)\Big[r + \gamma \max_{a'} q_*(s',a')\Big]
\end{aligned}
$$

对于有限马尔科夫决策过程，$v_*(s)$ 的贝尔曼方程有独立于策略的唯一解
贝尔曼方程实际对每个状态都是一个方程组，那么 n 个状态就有关于 n 个未知数的 n 个等式
如果环境动态 p 已知，那么原则上就能够用解非线性方程组的方法解出关于 v_* 的方程组，继而解出关于 q_* 的方程组

一旦有了 v_* ， 就很容易得到一个最优策略
对于每个状态 s ， 会有一到多个拥有最大值的 action ， 只要简单地只选择这些 actions 就能得到一个最优策略，即一个简单的贪婪策略就是最优策略，仅仅是做了一个单步搜索
v_* 的妙处就在于，它考虑了未来可能的长期回报

如果有了 q_* ， 问题会变得更加简单，它连单步搜索都不需要做，对任何状态 s ， 都能轻易的找到一个使得 $q_*(s，a)$ 最大的 action ， 它本身就包含了单步搜索的结果

1.3 动态规划 (*Dynamic Programming*)

动态规划(*dynamic programming, DP*)指的是一类算法，该类算法用于在已知环境的完全模型的情况(比如 MDP)下，计算出最优策略
DP 在 RL 中应用有限，因为其要求有环境的完整模型，以及其计算量消耗巨大，但它依然是很重要的理论基础，有助于理解后续的方法。

首先，假设环境是 finite MDP ，即状态、动作、奖励的空间是有限的。
对于连续空间的问题，可以量化其三个空间，然后使用 finite-state DP.

DP 的关键在于利用 值函数 来组织构造对优等策略的搜索。
我们已知，如果有了满足贝尔曼最优方程的 v_* 或 q_* ，就能轻易地得到最优策略。
DP 会把贝尔曼方程转化为逼近所求 值函数 的更新规则。

1.3.1 策略估计 (***Policy Evaluation*** **or** ***Prediction***)

策略估计是用于为任意的给定策略计算其对应的 state-value function v_π
首先给出 v_π 的定义：

$$\begin{aligned}
v_\pi(s) &\doteq \mathbb{E}_\pi\left[G_t \mid S_t = s\right] \\
&= \mathbb{E}_\pi\left[R_{t+1} + \gamma G_{t+1} \mid S_t = s\right] \\
&= \mathbb{E}_\pi\left[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s\right] \\
&= \sum_a \pi(a \mid s) \sum_{s',r} p(s', r \mid s, a)\Big[r + \gamma v_\pi(s')\Big], \qquad \text{for all } s \in \mathcal{S},
\end{aligned}$$

如果环境动态完全已知(p 已知)，那么上式就是 $|\mathcal{S}|$ 个未知数的 $|\mathcal{S}|$ 个非线性方程组。
原则上，它的解是简单易算的，我们使用迭代法来计算它。
考虑一组估计值 $v_0, v_1, v_2, \ldots,$ 每一个都是从 $\mathcal{S}_+$ 到 $\mathbb{R}$ 的映射，初始估计值 v_0 随机选取，后续的估计值就能用上式作为更新规则来获得。

$$\begin{aligned}
v_{k+1}(s) &\doteq \mathbb{E}_\pi\left[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s\right] \\
&= \sum_a \pi(a \mid s) \sum_{s',r} p(s', r \mid s, a)\Big[r + \gamma v_k(s')\Big],
\end{aligned}$$

在此更新规则中，$v_k = v_\pi$ 是固定的，而 v_π 的贝尔曼方程保证了该例中的相等性。
实际上，在相同条件下，序列 $\{v_k\}$ 在 $k \to \infty$ 时会收敛于 v_π.
该算法称为迭代策略估计(*iterative policy evaluation*)

1.3.2 策略改进 (*Policy Improvement*)

计算 值函数 是为了找到更好的 policy.
现在已经知道，在状态 s 下使用当前策略有多好—$v_\pi(s)$ ，但改变策略有可能得到更好的结果。
于是可以试着去选择这个 $a \neq \pi(s)$.

$$
\begin{aligned}
q_\pi(s,a) &\doteq \mathbb{E}_\pi\left[G_t \mid S_t = s, A_t = a\right] \\
&= \sum_{s',r} p(s',r \mid s,a)\Big[r + \gamma v_\pi(s)\Big].
\end{aligned}
$$

关键在于这个值与 $v_\pi(s)$ 的大小关系：
policy improvement theorem.： $q_\pi(s,\pi'(s)) > v_\pi(s). \quad \text{for all } s \in \mathcal{S}$
表明策略 π' 优于策略 π： $v_{\pi'}(s) > v_\pi(s). \quad \text{for all } s \in \mathcal{S}$

照下式更新策略，即可使得新的策略不差于原始的策略，这就是 *policy improvement*：

$$
\begin{aligned}
\pi'(s) &\doteq \arg\max_a q_\pi(s,a) \\
&= \arg\max_a \mathbb{E}\left[R_{t+1} + \gamma v_\pi(S_{t+1} \mid S_t = s, A_t = a)\right] \\
&= \arg\max_a \sum_{s',r} p(s',r \mid s,a)\left[r + \gamma v_\pi(s')\right],
\end{aligned}
$$

当新的策略与旧的策略相同时，就说明其收敛；且根据贝尔曼最优方程，其必然是最优策略。

$$
\begin{aligned}
v_{\pi'}(s) &= \max_a \mathbb{E}\left[R_{t+1} + \gamma v_{\pi'}(S_{t+1}) \mid S_t = s, A_t = a\right] \\
&= \max_a \sum_{s',r} p(s',r \mid s,a)\left[r + \gamma v_{\pi'}(s')\right].
\end{aligned}
$$

1.3.3 策略迭代 (*Policy Iteration*)

在用 v_π 优化原策略得到新策略 π' 后，就可以用新策略来计算新的值函数 $v_{\pi'}$,然后，继续优化：

$$
\pi_0 \xrightarrow{E} v_{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} v_{\pi_1} \xrightarrow{E} \pi_2 \xrightarrow{E} \cdots \xrightarrow{I} \pi_* \xrightarrow{E} v_*,
$$

其中 $\xrightarrow{E}$ 表示 policy evaluation，而 $\xrightarrow{I}$ 表示 policy improvement.
finite MDP 有 finite policies，从而该过程在 有限次迭代 后会收敛于 最优策略 和 最优值函数。