16 模拟环境中机器人学习

本章主要介绍模拟环境中机器人学习的一个上手项目，包括在 CoppeliaSim 中设置一个机械臂抓取物体的任务，并用深度强化学习算法柔性 Actor-Critic（Soft Actor-Critic，SAC）去解决它。实验部分展示不同奖励函数的效果，用以验证辅助密集奖励对于解决类似机器人抓取任务的重要性。在本章末尾，我们也对机器人学习应用、模拟到现实的迁移和其他机器人学习项目及模拟器进行简单的讨论。

深度强化学习算法有很多潜在的现实世界应用场景，机器人控制是其中最令人振奋的领域之一。尽管深度强化学习算法已经能够很好地解决绝大多数简单的游戏，像之前介绍的 OpenAI Gym 环境等，我们目前还不能期望深度强化学习方法在机器人控制领域能完全替代传统控制方法，比如反向运动学（Inverse Kinematics）或比例-积分-微分（Proportional-Integral-Derivative, PID）控制等。然而，深度强化学习能够应用于某些具体情形，作为与传统控制相辅相成的方法，尤其是对于高度复杂的系统或者灵活操控任务 (Akkaya et al., 2019; Andrychowicz et al., 2018)。

在绝大多数情况下，机器人控制的动态过程可以用马尔可夫（Markov）过程很好地近似，这使得它成为深度强化学习在模拟和现实中的一个理想的试验场。另外，深度强化学习对于现实世界中机器人控制的巨大潜力也吸引了许多像 DeepMind 和 OpenAI 等高科技公司来投入这个研究领域。近来，OpenAI 甚至通过自动域随机化（Automatic Domain Randomization）技术来解决模拟到现实的迁移（Sim-to-Real Transfer）问题，从而用一个单手五指机械臂解决了 Rubik 魔方，如图 16.1 所示。其他公司也开始研究使用比如在仓储物流中的货物分发任务上使用机械臂，甚至直接让机器人在现实世界中训练 (Korenkevych et al., 2019)。

然而，由于将强化学习算法直接应用于现实世界中有采样效率低和安全性的问题，人们发现直接在现实世界中训练强化学习策略来解决复杂的机器人系统控制或灵活操控任务 (Akkaya et al., 2019) 是有困难的。在模拟环境中训练并在随后将策略迁移到现实世界，或者利用人类专家

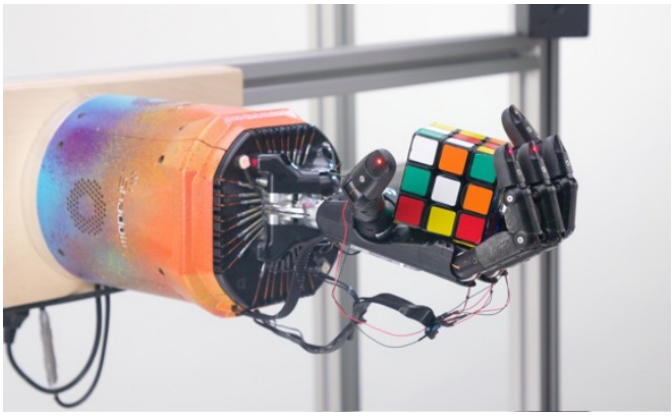

图 16.1 用一个机器手解决 Rubik 魔方的场景。图片改编自文献 (Akkaya et al., 2019)（见彩插）

的示范（Human Expert Demonstrations）来学习，都是更有潜力满足机器人学习的计算性能和安全要求的方式。机器人的模拟器已经发展了数十年，包括 DART、CoppeliaSim（在 3.6.2 版本之前叫作 V-REP）(Rohmer et al., 2013)、MuJoCo、Gazebo 等。在本章最后一小节会有相关讨论。为了便于人们使用深度强化学习控制策略和其他数值操作，这些模拟器多数都有 Python 对应版本。

在模拟环境中学习至少在两个方面有意义。第一，模拟环境可以用作新提出算法或框架的试验地（包括但不限于强化学习领域），尤其是大规模的现实世界应用，比如机器人学习任务。在模拟环境中学习可以作为新方法在应用到现实情景前的验证过程。第二，对于通过模拟到现实迁移的方式解决现实世界问题来说，在模拟中学习是不可或缺的一步，可以减少时间消耗和物理设备磨损。

在这一章，我们将介绍把深度强化学习算法应用到一个模拟环境中简单的机器人物体抓取任务的过程，使用 CoppeliaSim（V-REP）模拟器和它的 Python 封装：PyRep (James et al., 2019a)。我们开源了这个项目的任务描述和深度强化学习算法相关代码[1]，便于读者学习和理解。

由于之前已经介绍了一个将强化学习应用于大规模高维度连续空间的应用，本章的机器人学习任务将更加着重实践中强化学习的其他方面，包括如何构建一个能通过强化学习实现特定任务的模拟环境，如何设计奖励函数来辅助强化学习实现最终的任务目标等，以给读者提供对强化学习更好的理解，不仅限于训练过程，更在于如何设计学习环境。

16.1 机器人模拟

我们第一步要做的是设置一个模拟环境，包括：一个机械臂、与机械臂交互的一个物块。这个模拟环境应当符合现实物理动态规律。然而，这里我们要强调一点，一个真实的模拟不意味着在这个模拟环境中学习到的策略就可以直接在现实世界中取得好的表现。一个“真实的”模拟环

[1]链接见读者服务

境可以通过不同的具体形式实现，而其中只有一种形式可以与实际的现实世界相匹配。举例来说，不同光照条件可以在物体上产生不同的阴影效果，而这些可能看起来都很“真实”，但是只有其中一种是跟现实相同的，而且由于深度神经网络的敏感性，这些外观上的细微差异可能导致现实中做出截然不同的动作。为了解决这类模拟到现实迁移过程的问题，如域随机化（Domain Randomization）、动力学随机化（Dynamics Randomization）等许多方法被提出和应用，我们也将在本章进行相关讨论。

现在有许多机器人的模拟器，包括 CoppeliaSim（V-REP）、MuJoCo、Unity 等。原版 CoppeliaSim（V-REP）软件使用 C++ 和 Lua 语言支持的通用接口，而只有部分函数功能可以通过 Python 实现。然而，对于应用深度强化学习而言，最好使用 Python 接口。幸运的是，我们有 PyRep 软件包来将 CoppeliaSim（V-REP）用于深度机器人学习。在本项目中，我们使用 CoppeliaSim（V-REP）并搭配它的软件包 PyRep 来调用 Python 接口。

我们将在本节展示设置一个机器人学习任务的基本过程。

安装 CoppeliaSim 和 PyRep

CoppeliaSim（V-REP）软件可以在官网[2]下载到，而在本书的写作过程中，我们需要 CoppeliaSim（V-REP）的 3.6.2 版本（可以在网站[3]上找到）来跟 PyRep 兼容。它可以直接通过解压下载的文件来安装。注意高于 CoppeliaSim（V-REP）3.6.2 的版本可能跟这个项目的其他模块不兼容。

安装完 CoppeliaSim（V-REP）之后，我们可以通过以下几步安装我们仓库网站（链接见读者服务）上的一个 PyRep 的分支稳定版本：

```
git clone https://github.com/deep-reinforcement-learning-book/PyRep.git
pip3 install -r requirements.txt
python3 setup.py install --user
# 注意：在以下指令中需要将路径改为用户本机的 VREP 安装位置
export VREP_ROOT=EDIT/ME/PATH/TO/V-REP/INSTALL/DIR
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$VREP_ROOT
export QT_QPA_PLATFORM_PLUGIN_PATH=$VREP_ROOT
source~/.bashrc
```

记得通过上面脚本中的 VREP_ROOT 更改 V-REP 的路径。

Git 克隆本项目

本章的深度强化学习算法应用于机器人学习任务项目可以通过以下命令下载：

[2]链接见读者服务
[3]链接见读者服务

```
git clone https://github.com/deep-reinforcement-learning-book/Chapter16-Robot-Learning
    -in-Simulation.git
```

这个项目包含机器人的部分（机械臂，夹具）和其他我们需要的物体、构建的机器人抓取任务情景、用来训练智能体控制策略的深度强化学习算法等。本项目中的机器人抓取任务情景见图 16.2。我们将在以下几小节中展示如何构建这个包含基本组成部分的场景。

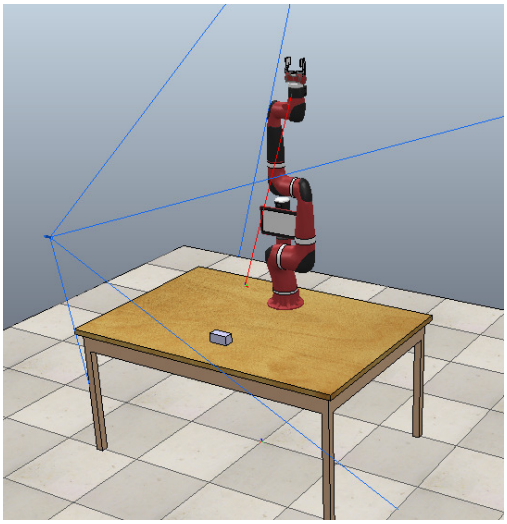

图 16.2 CoppeliaSim (V-REP) 中的抓取（Grasping）任务场景（见彩插）

组装机器人

我们使用名为 *Rethink Sawyer* 的机械臂和一个 *BraxterGripper* 终端夹具。官方 PyRep 软件包提供了多种机械臂和夹具，可以用来组装和构建你想要的任务场景。我们这里提供一个例子，将一个夹具安装到机械臂，如图 16.3 所示。

在我们的 Git 文件下，将`./hands/BaxterGripper.ttm` 和`./arms/Sawyer.ttm` 拖入在 CoppeliaSim（V-REP）中打开的一个新场景。我们选择夹具并同时按 Ctrl 加鼠标左键单击 *Sawyer* 的终端关节（即 `Sawyer_wrist_connector`，它是 CoppeliaSim（V-REP）中的一个力传感器，可以用于连接不同物体），然后单击“组装”按钮，如图 16.4 所示。CoppeliaSim（V-REP）提供了不同种类的连接器，这里的力传感器只是其中一种，且这种连接器在关节受到的真实力大于一个阈值的时候有破碎的可能。另一方面，我们不应该在这里用“组合/合并”（group/merge）选项，这是为了能够独立控制夹具和机械臂。更多关于如何连接和组合不同物体的细节可以查阅 CoppeliaSim（V-REP）的网站。在我们完成以上过程后，所构建场景的层级（Scene hierarchy）应当如图 16.5 所示。

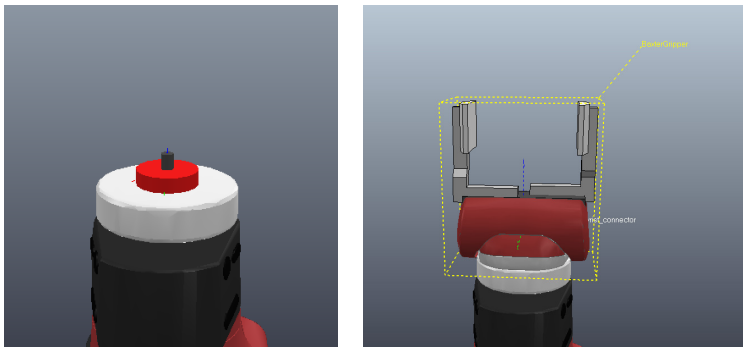

图 16.3　*Sawyer* 机械臂末端（左）和组装的夹具 *BaxterGripper*（右）（见彩插）

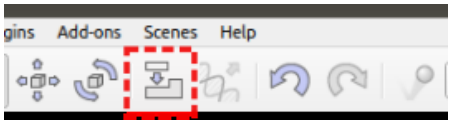

图 16.4　CoppeliaSim (V-REP) 中的“组装”（assemble）按钮

构建学习环境

图 16.2 展示了 CoppeliaSim（V-REP）中一个构建好的场景，相应文件为`./scenes/sawyer_reacher_rl.ttt`。为了构建这个最终场景，我们需要把其他物体添加到当前只包含机械臂和夹具的场景中。

首先，我们通过**添加（Add）-> 简单形状（Primitive shape）-> 长方体（Cuboid）**添加一个目标物体，调整它成为我们想要的尺寸并重命名为“目标”。我们需要双击“目标”前面的图标并选择**公共（Common）-> 可渲染的（Renderable）**来使得物体对视觉传感器可见。

在以上步骤之后，我们需要添加一个可以给我们提供定制场景视野的视觉传感器。这个视觉传感器可以在模拟过程中一直拍摄视野的图像，如果我们使用基于图像的控制，那么这个视觉传感器是必需的（如果不是基于图像的控制，我们可能不需要它）。如果我们在场景中启用这个视觉传感器，我们可以在模拟的每一步返回图像。为了设置这样的场景，单击**添加（Add）-> 视觉传感器（Vision sensor）-> 视角类型（perspective type）**，然后右键单击场景，选择**添加（Add）-> 浮动视野（Floating view）**。这时先单击我们刚刚创建的**视觉传感器**，然后右键单击打开的浮动视野，选择**视图（View）-> 关联视图和已选择的视觉传感器（associate view with selected vision sensor）**。随后，我们手动设置添加的视觉传感器的位置和旋转角度，得到如图 16.6 所示的场景。

下面，我们从项目文件夹`./objects`中拖入物体文件 `table.ttt`。通过单击**物体（Object）/物品移动（item shift）**按键，我们手动设置这个带夹具机械臂的位置和目标长方体的位置，使得它们位于桌子上方，如图 16.7 所示。

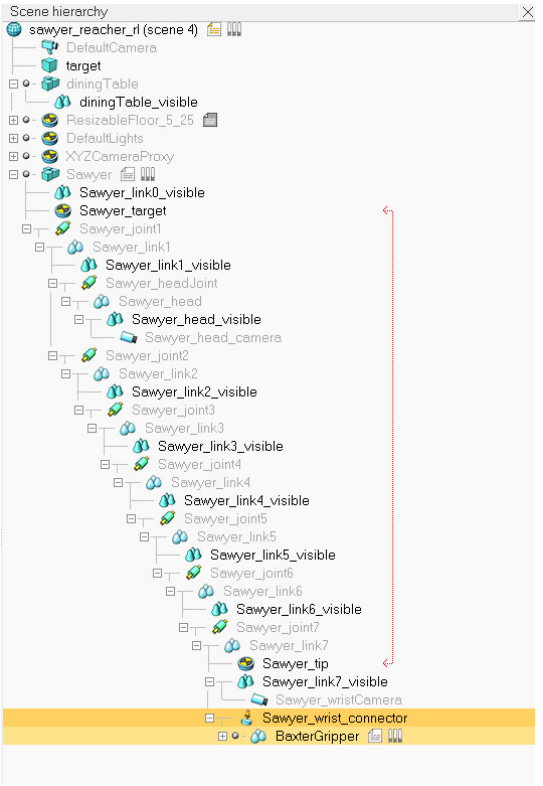

图 16.5 CoppeliaSim (V-REP) 中任务场景的层级，包括 *Sawyer* 机械臂在内的所有物理模型。红色箭头表示用于端点控制模式的反向运动学链。黑色字体表示场景中可见的物体，而灰色字体表示不可见的虚拟物体

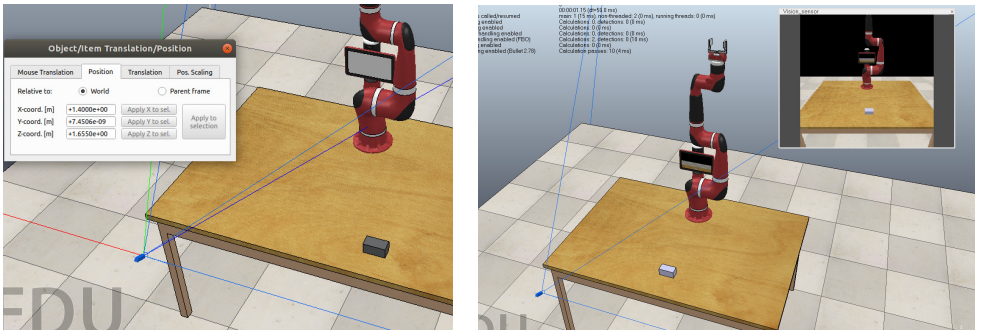

图 16.6 在 CoppeliaSim (V-REP) 中设置视觉传感器。左面的图片设置相机位置；右面图片中右上角的小窗口是由所放置的相机得到的。如果采用基于图像的控制策略并调用相机，那么它可以给每个时间步提供图像观察量（见彩插）

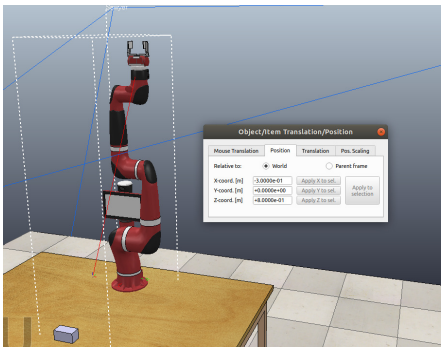

图 16.7　手动改变 CoppeliaSim (V-REP) 中物体的位置（见彩插）

以上是设置环境场景的过程，这个场景给我们提供了任务中可以看到的实体。这些实体的动力学过程将遵从物理模拟器的模拟规则。除此之外，我们还需要给在环境中定义控制流程和奖励函数（Reward Functions），通常包括物体移动的限制条件（主要是运动类的任务）、一个训练片段（Episode）的开启和结束步骤、初始化条件、观察量的形式等。在我们的 Git 文件中，我们提供了一个脚本 `sawyer_grasp_env_boundingbox.py` 用来在场景中实现这些功能。为了便于之后应用强化学习算法进行控制，这个脚本我们采用与 OpenAI Gym 环境相似的应用程序接口（APIs）。我们上面构建的场景本身是静态的，而这个控制脚本可以为它提供控制动力学过程的功能（除了模拟器中实现的物理过程）。对于这个机器人抓取任务，我们使用正向运动学（直接控制关节运动速度）的控制机制来控制机械臂。我们也使用不同配置方式实现了一个通过反向运动学实现控制的（控制机械臂终端位置）的场景。反向运动学控制通常需要求一个描述关节角度和机械臂端点位置关系的雅可比（Jacobian）矩阵的逆，这个功能在 PyRep 中也有支持。更多关于反向运动学设置的细节超出本书范围。我们提供的例子程序中的脚本定义的动力学过程和机器人控制可以支持以上两种控制机制。

注意：实践中，当你尝试构建自己的机器人模型或用不同的组件组装定制机械臂的时候，你需要小心机械臂上不同模块的组装顺序和依赖关系。这与 CoppeliaSim（V-REP）软件对动态和静态组件（比如反向运动学中的 `Sawyer_tip` 是一个静态组件）的一些要求有关。细节参考官方网站[4]。

在 CoppeliaSim（V-REP）中设置好环境场景之后，我们需要用 PyRep 软件包写一个定义环境中动力学过程和奖励函数的控制脚本。我们的仓库中提供了定义环境的代码。下面几小节中我们将介绍项目中用到的函数和模块。

环境脚本中的模块

导入所需软件包并设置下面需要的全局变量。

[4]链接见读者服务

```
from os.path import dirname, join, abspath
from pyrep import PyRep
from pyrep.robots.arms.sawyer import Sawyer
from pyrep.robots.end_effectors.baxter_gripper import BaxterGripper
from pyrep.objects.proximity_sensor import ProximitySensor
from pyrep.objects.vision_sensor import VisionSensor
from pyrep.objects.shape import Shape
from pyrep.objects.dummy import Dummy
from pyrep.const import JointType, JointMode
import numpy as np
import matplotlib.pyplot as plt
import math

POS_MIN, POS_MAX = [0.1, -0.3, 1.], [0.45, 0.3, 1.] # 目标物体有效位置范围
```

所定义机器人抓取任务环境类的整体结构显示如下。这里所有的函数都在类中简写，我们将在后文中展开介绍。

```
class GraspEnv(object):
    # Sawyer 机器人抓取物块
    def __init__(self, headless, control_mode='joint_velocity'):
        # 参数:
        # :headless:  bool, 如果为 True, 没有可视化; 否则有可视化
        # :control_mode:  str, 'end_position' 或'joint_velocity'
        ......

    def _get_state(self):
        # 返回包括关节角度或速度和目标位置的状态
        ......

    def _is_holding(self):
        # 返回抓取目标与否的状态, 为 bool
        ......

    def _move(self, action, bounding_offset=0.15, step_factor=0.2, max_itr=20,
         max_error=0.05, rotation_norm =5.):
        # 对于'end_position' 模式, 用反向运动学根据动作移动末端。反向运动学模式控制是通过设
        # 置末端目标来实现的, 而非使用 solve_ik() 函数, 因为有时 solve_ik() 函数不能正确工作
```

```
    # 模式：闭环比例控制，使用反向运动学

    # 参数：
    # :bounding_offset:  有效目标位置范围外的边界方框所用的偏移量，作为有效且安全的动作
    # 范围
    # :step_factor:  小步长因子，用来乘以当前位置和位置的偏差，即作为控制的比例因子
    # :max_itr:  最大移动迭代次数
    # :max_error:  每次调用时移动距离误差的上边界
    # :rotation_norm:  用来归一化旋转角度值的因子，由于动作对每个维度有相同的值范围，角
    # 度需要额外处理
    ......

def reinit(self):
    # 重新初始化环境，比如可当夹具在探索中破损时调用
    ......

def reset(self, random_target=False):
    # 重置夹具位置和目标位置
    ......

def step(self, action):
    # 根据动作移动机械臂：如果控制模式为'joint_velocity'，则动作是 7 维的关节速度值 +1
    # 维的夹具旋转值；如果控制模式为'end_position'，则动作是 3 维末端（机械臂端点）位置
    # +1 维夹具旋转值
    ......

def shutdown(self):
    # 关闭模拟器
    ......
```

第一步是初始化环境，包括设置共用变量，如 `__init__()` 函数所定义的一样：

```
def __init__(self, headless, control_mode='joint_velocity'):
    # 参数：
    # :headless:  bool，若为 True，则没有可视化；否则有可视化
    # :control mode:  str，'end_position' 或'joint_velocity'

    # 设置公共变量
```

```
self.headless = headless # 若 headless 为 True，则无可视化
self.reward_offset = 10.0 # 抓到物体的奖励值
self.reward_range = self.reward_offset # 奖励值域
self.penalty_offset = 1. # 对不希望发生情形的惩罚值
self.fall_down_offset = 0.1 # 用于判断物体掉落桌面的距离值
self.metadata=[] # gym 环境参数
self.control_mode = control_mode
    # 机械臂控制模式：'end_position' 或'joint_velocity'
```

函数 __init__() 的第二部分是设定和启动场景，并设置场景中物体相应的代理变量：

```
self.pr = PyRep() # 调用 PyRep
if control_mode == 'end_position': # 所有关节都以反向运动学的方式进行的位置控制模式
    SCENE_FILE = join(dirname(abspath(__file__)),
        './scenes/sawyer_reacher_rl_new_ik.ttt') # 使用反向运动学控制的场景
elif control_mode == 'joint_velocity': # 所有关节都以正向运动学的力或力矩方式进行的
                                       # 速度控制模式
    SCENE_FILE = join(dirname(abspath(__file__)),
        './scenes/sawyer_reacher_rl_new.ttt') # 使用正向运动学控制的场景
self.pr.launch(SCENE_FILE, headless=headless) # 启动场景，headless 意味着无可视化
self.pr.start()    # 启动场景
self.agent = Sawyer() # 得到场景中的机械臂
self.gripper = BaxterGripper() # 得到场景中的夹具
self.gripper_left_pad = Shape('BaxterGripper_leftPad') # 夹具手指上的左护垫
self.proximity_sensor = ProximitySensor('BaxterGripper_attachProxSensor')
    # 传感器名称
self.vision_sensor = VisionSensor('Vision_sensor') # 传感器名称
self.table = Shape('diningTable') # 场景中的桌子，用来检查碰撞
if control_mode == 'end_position': # 通过机械臂端点位置来用反向运动学控制机械臂
    self.agent.set_control_loop_enabled(True) # 若为 False，则反向运动学无法工作
    self.action_space = np.zeros(4)
        # 3 自由度的端点位置控制和 1 自由度的夹具旋转控制
elif control_mode == 'joint_velocity':
    # 通过直接设置每个关节速度来用正向运动学控制机械臂
    self.agent.set_control_loop_enabled(False)
    self.action_space = np.zeros(7)
        # 7 自由度速度控制，无须额外控制端点旋转，第 7 个关节控制它
else:
```

```
            raise NotImplementedError
        self.observation_space = np.zeros(17) # 7 个关节的标量位置和标量速度 +3 维目标位置
        self.agent.set_motor_locked_at_zero_velocity(True)
        self.target = Shape('target') # 得到目标物体
        self.agent_ee_tip = self.agent.get_tip()
            # 机械臂末端的一个部分，作为反向运动学控制链的末端来进行控制
        self.tip_target = Dummy('Sawyer_target') # 末端（机械臂的端点）运动的目标位置
        self.tip_pos = self.agent_ee_tip.get_position() # 末端 x, y, z 位置
```

函数 `__init__()` 的第三部分是设置合适的初始机器人姿势和末端位置：

```
        if control_mode == 'end_position':
            initial_pos = [0.3, 0.1, 0.9]
            self.tip_target.set_position(initial_pos) # 设置目标位置
            # 对旋转来说单步控制足以通过设置 reset_dynamics=True 就可以立即设置旋转角
            self.tip_target.set_orientation([0,np.pi,np.pi/2], reset_dynamics=True)
                # 前两个沿着 x 和 y 轴的维度使夹具向下
            self.initial_tip_positions = self.initial_target_positions = initial_pos
        elif control_mode == 'joint_velocity':
            self.initial_joint_positions = [0.0, -1.4, 0.7, 2.5, 3.0, -0.5, 4.1]
                # 一个合适的初始姿态
            self.agent.set_joint_positions(self.initial_joint_positions)
        self.pr.step()
```

如下所示是一个获得观察状态的函数，包括关节位置和速度，以及目标物体的三维空间位置，总共 17 维。

```
    def _get_state(self):
        # 返回包括关节角度或速度和目标位置的状态

        return np.array(self.agent.get_joint_positions() + # list, 维数为 7
                self.agent.get_joint_velocities() + # list, 维数为 7
                self.target.get_position()) # list, 维数为 3
```

一个决定夹具是否抓到物体的函数被定义为 `_is_holding()`，通过夹具护垫上的碰撞检测和近距离传感器来决定物体是否在夹具内。

```
    def _is_holding(self):
        # 返回抓取目标与否的状态，为 bool
```

```
        # 注意碰撞检测不总是准确的，对于连续碰撞帧，可能只有开始的 4～5 帧碰撞可以被检测到
        pad_collide_object = self.gripper_left_pad.check_collision(self.target)
        if pad_collide_object and self.proximity_sensor.is_detected(self.target)==True:
            return True
        else:
            return False
```

函数 _move() 可以在有效范围内通过反向运动学模式操控移动机械臂末端执行器。PyRep 中可以通过在机械臂末端放置一个部件来实现以反向运动学控制末端执行器，具体做法是设置这个末端部件的位置和旋转角。如果调用 pr.step() 函数，那么在 PyRep 中机械臂关节的反向运动学控制可以自动求解。由于单个较大步长的控制可能是不精确的，这里我们将整个动作产生的位移运动分解为一系列小步长运动，并采用一个有最大迭代次数和最大容错值的反馈控制闭环来执行这些小步长动作。

```
    def _move(self, action, bounding_offset=0.15, step_factor=0.2, max_itr=20,
              max_error=0.05, rotation_norm =5.):
        # 对于'end_position' 模式，用反向运动学根据动作移动末端。反向运动学模式控制是通过设
        # 置末端目标来实现的，而非使用 solve_ik() 函数，因为有时 solve_ik() 函数不能正确
        # 工作。
        # 模式：闭环比例控制，使用反向运动学

        # 参数：
        # :bounding_offset:  有效目标位置范围外的边界方框所用的偏移量，作为有效且安全的动作
        # 范围
        # :step_factor:  小步长因子，用来乘以当前位置和目标位置的偏差，即作为控制的比例因子
        # :max_itr:  最大移动迭代次数
        # :max_error:  每次调用时移动距离误差的上界
        # :rotation_norm:  用来归一化旋转角度值的因子，由于动作对每个维度有相同的值范围，
        # 角度需要额外处理

        pos=self.gripper.get_position()

        # 检查状态加动作是否在边界方框内，若在，则正常运动；否则动作不会被执行。该范围为
        # x_min < x < x_max 且 y_min < y < y_max 且 z > z_min
        if pos[0]+action[0]>POS_MIN[0]-bounding_offset and
            pos[0]+action[0]<POS_MAX[0]+bounding_offset \
           and pos[1]+action[1] > POS_MIN[1]-bounding_offset and pos[1]+action[1] <
```

```
            POS_MAX[1]+2*bounding_offset \
        and pos[2]+action[2] > POS_MIN[2]-2*bounding_offset: # z 轴有较大偏移量

        # 物体的 set_orientation() 和 get_orientation() 之间有一个错配情况，
        # set_orientation() 中的（x, y, z）对应 get_orientation() 中的（y, x, -z）
        ori_z=-self.agent_ee_tip.get_orientation()[2]
            # 减号是因为 set_orientation() 和 get_orientation() 之间的错配
        target_pos = np.array(self.agent_ee_tip.get_position())+np.array(action[:3])
        diff=1 # 初始化
        itr=0
        while np.sum(np.abs(diff))>max_error and itr<max_itr:
            itr+=1
            # 通过小步来到达位置
            cur_pos = self.agent_ee_tip.get_position()
            diff=target_pos-cur_pos # 当前位置和目标位置差异，进行闭环控制
            pos = cur_pos+step_factor*diff
                # 根据当前差异迈一小步，防止反向运动学无法求解
            self.tip_target.set_position(pos.tolist())
            self.pr.step() # 每次设置末端目标位置，需调用模拟步来实现

        # 对 z 轴旋转单步即可，但是由于反向运动学求解器的问题，所以还是存在小误差
        ori_z+=rotation_norm*action[3]
            # 归一化旋转值，因为通常在策略中对旋转和位移的动作范围是一样的
        self.tip_target.set_orientation([0, np.pi, ori_z])
            # 使夹具向下并沿 z 轴旋转 ori_z
        self.pr.step() # 模拟步

    else:
        print("Potential Movement Out of the Bounding Box!")
        pass # 如果潜在运动超出了边界方框，动作不会执行
```

这里提供了一个可以重新初始化场景的函数。

```
def reinit(self):
    # 重新初始化环境，比如当夹具在探索中破损时可调用
    self.shutdown() # 首先关掉当前环境
    self.__init__(self.headless) # 以相同的 headless 模式进行初始化
```

如下是一个能够重置场景中目标物体和机械臂的函数。

```
def reset(self, random_target=False):
    # 重置夹具位置和目标位置

    # 设置目标物体
    if random_target: # 随机化
        pos = list(np.random.uniform(POS_MIN, POS_MAX)) # 从合理范围的均匀分布中采样
        self.target.set_position(pos) # 随机位置
    else: # 无随机化
        self.target.set_position(self.initial_target_positions) # 固定位置
    self.target.set_orientation([0,0,0])
    self.pr.step()

    # 把末端位置设置到初始位置
    if self.control_mode == 'end_position': # JointMode.IK
        self.agent.set_control_loop_enabled(True) # 反向运动学模式
        self.tip_target.set_position(self.initial_tip_positions)
            # 由于反向运动学模式或力/力矩模式开启，所以无法直接设置关节位置
        self.pr.step()
        # 避免卡住的情况：由于使用反向运动学来移动，所以机械臂卡住会使得反向运动学难以求解，
        # 从而无法正常重置，因此在预期位置无法到达时需要采用一些随机动作
        itr=0
        max_itr=10
        while np.sum(np.abs(np.array(self.agent_ee_tip.get_position()-
            np.array(self.initial_tip_positions))))>0.1 and itr<max_itr:
            itr+=1
            self.step(np.random.uniform(-0.2,0.2,4)) # 采取随机动作来防止卡住的情况
            self.pr.step()

    elif self.control_mode == 'joint_velocity': # JointMode.FORCE
        self.agent.set_joint_positions(self.initial_joint_positions)
        self.pr.step()

    # 设置可碰撞（collicable）模式，用于碰撞检测
    self.gripper_left_pad.set_collidable(True)
        # 设置夹具护垫为可碰撞的，从而可以检测碰撞
    self.target.set_collidable(True)
```

```
        # 如果夹具没有完全打开，将其完全打开
        if np.sum(self.gripper.get_open_amount())<1.5:
            self.gripper.actuate(1, velocity=0.5)
            self.pr.step()

        return self._get_state() # 返回环境当前状态
```

如其他环境（OpenAI Gym 等）中经常使用的 `step()` 函数，在我们这里的环境中也会用到。这个函数需要相应的动作值作为输入。如果机器人是由 `end_position` 模式使用反向运动学控制的，它需要调用之前定义的 `_move()` 函数来执行动作；如果机器人是由 `joint_velocity` 模式通过正向运动学控制的，那么机械臂上的关节位置可以直接被设定。

```
    def step(self, action):
        # 根据动作移动机械臂：如果控制模式为'joint_velocity'，那么动作是 7 维的关节速度值 +1
        # 维的夹具旋转值；如果控制模式为'end_position'，那么动作是 3 维末端（机械臂端点）位置
        # +1 维夹具旋转值

        # 初始化
        done=False # 片段结束
        reward=0
        hold_flag=False # 是否抓住物体的标签
        if self.control_mode == 'end_position':
            if action is None or action.shape[0]!=4: # 检查动作是否合理
                print('No actions or wrong action dimensions!')
                action = list(np.random.uniform(-0.1, 0.1, 4)) # 随机
            self._move(action)

        elif self.control_mode == 'joint_velocity':
            if action is None or action.shape[0]!=7: # 检查动作是否合理
                print('No actions or wrong action dimensions!')
                action = list(np.random.uniform(-0.1, 0.1, 7)) # 随机
            self.agent.set_joint_target_velocities(action) # 机械臂执行动作
            self.pr.step()

        else:
            raise NotImplementedError
```

除了移动机械臂，奖励函数（Reward Function）、吸收状态（Absorbing State）、结束信号（done）和其他像标记物体被持有状态的信息等也是通过 step() 函数实现的，如下所示。成功抓取物体的奖励是一个正数，而物体掉落桌面的惩罚是一个同样数值大小的负数。这构成了一种稀疏奖励机制，而可能对智能体来说很难学习。所以我们添加了距离上的惩罚项来辅助学习。这个惩罚项的值是末端执行器到目标物体的距离，同时我们也惩罚夹具与桌面的碰撞来避免夹具损坏。这构成了一个密集奖励函数。然而，我们要知道密集奖励函数可能跟最终的任务目标有出入，而我们的目标是让机器人抓取目标物体。由于距离惩罚项正比于夹具和物体中心的距离，它会促使夹具尽可能地接近物体中心，而这可能导致不合适的抓取姿态。更多关于这种奖励函数与强化学习任务目标之间的分歧可以参考第 18 章中的讨论。由于这个原因，我们需要对奖励函数进行修正，比如设定一个位于目标物体上方的位置偏移量（虚拟目标点）来取代目标物体的中心，我们将在随后的几小节中进行相关讨论。

```
        ax, ay, az = self.gripper.get_position()
        if math.isnan(ax): # 捕捉探索中夹具破损的情况
            print('Gripper position is nan.')
            self.reinit()
            done=True
        tx, ty, tz = self.target.get_position()
        sqr_distance = (ax - tx) ** 2 + (ay - ty) ** 2 + (az - tz) ** 2
            # 夹具和目标物体的距离的平方

        # 在夹具与物体足够近且物体被近距离传感器检测到时关闭夹具
        if sqr_distance<0.1 and self.proximity_sensor.is_detected(self.target)== True:
            # 确保抓取之前夹具是打开的
            self.gripper.actuate(1, velocity=0.5)
            self.pr.step()
            self.gripper.actuate(0, velocity=0.5)
                # 如果结束了，关闭夹具，0 是关闭，1 是打开；速度 0.5 可以确保夹具在一帧内关闭
            self.pr.step() # 物理模拟器前进一步

            if self._is_holding():
                reward += self.reward_offset # 抓到物体的额外奖励
                done=True
                hold_flag = True
            else:
                self.gripper.actuate(1, velocity=0.5)
                self.pr.step()
```

```
        elif np.sum(self.gripper.get_open_amount())<1.5: # 如果夹具由于碰撞或其他原因是
            # 关闭的（或未完全打开），打开它；get_open_amount() 返回夹具关节的一些数据值
            self.gripper.actuate(1, velocity=0.5)
            self.pr.step()
        else:
            pass
        # 基本奖励是距离目标的负值
        reward -= np.sqrt(sqr_distance)

        # 物体掉落桌面的情况
        if tz < self.initial_target_positions[2]-self.fall_down_offset:
            done = True
            reward = -self.reward_offset

        # 机械臂与桌面碰撞的惩罚
        if self.gripper_left_pad.check_collision(self.table):
            reward -= self.penalty_offset

        if math.isnan(reward): # 捕捉数值问题
            reward = 0.

        return self._get_state(), reward, done, {'finished': hold_flag}
```

用于关闭环境的函数相对简单：

```
    def shutdown(self):
        # 关闭模拟器
        self.pr.stop()
        self.pr.shutdown()
```

在以下实验中，我们采用如下关于上面抓取任务的基本设定：目标物体的初始位置是固定的；机器人关节位置的初始化选取了可以避免较复杂机器人姿态的方式；机器人是通过正向运动学模式来控制关节速度的；机器人的控制是基于数值状态的，包括关节位置、关节速度和目标位置作为观察量。但是读者可以随意更改这些设置使其更复杂，比如使用反向运动学模式来控制机器人末端位置，使用原始图像进行基于视觉的控制或用它与部分的数值状态相结合，使用更少的信息作为观察量，或者设置任务使其更加困难和复杂，等等。

在项目文件中，Sawyer 抓取任务的环境可以用以下命令来测试：

```
python sawyer_grasp_env_boundingbox.py
```

16.2 强化学习用于机器人任务

上述基于正向运动学控制的机器人学习环境有一个控制关节速度的 7 维连续动作空间，以及一个 17 维连续状态空间，因此相比于之前第 5 章和第 6 章中的例子而言，这是一个相对复杂的环境。并且，机器人模拟系统的复杂性使得采样过程需要耗费相当长的时间。这使得通过单线程或单进程框架在较短时间内训练一个相对好的策略很困难。实践中，我们发现策略学习速度的瓶颈主要在于 CoppeliaSim（V-REP）的模拟过程，如果只用单进程采样，就会使得整个学习过程非常低效。我们需要并行的离线训练框架来改善这个任务的采样速度。

在这个项目中，我们使用并行的柔性 Actor-Critic（Soft Actor-Critic，SAC）算法，使用的是第 13 章的项目中的并行框架。SAC 算法的详细介绍在第 6 章，包括理论和实现方法，所以这里只简短地描述选择 SAC 算法的原因和优点所在。作为一种离线策略（Off-Policy）学习算法，SAC 使用对角高斯（Gaussian）策略来应对高维连续动作空间，并且它在训练中比其他像深度确定性策略梯度（Deep Deterministic Policy Gradient）算法更加稳定并且对参数鲁棒，尤其是采用了对熵因子进行适应性学习的方法 (Haarnoja et al., 2018) 后。它也采用柔性 Q-Learning（Soft Q-Learning）来进行熵正则化，这对像机器人抓取这类难以训练的任务来说可以促进探索。还有，由于 SAC 算法采用离线策略的学习方式，所以它在实践中可以较方便地改成并行版本。

即使采用了并行的采样进程，让机器人基于上面定义的密集奖励探索一个好的物体抓取姿势也很困难，如果只使用稀疏奖励则会难上加难。为了进一步促进学习过程，我们对奖励函数进行启发式增强。首先，目标物体是一个长方体，由于它的长边要长于夹具的开合宽度，所以夹具只能调整方向使其垂直于物体长边来夹取，并且朝向需要向下。因此，我们在奖励函数上添加了一个额外的惩罚项如下：

```
# 对角度的增强奖励：如果夹具与目标物体方向相垂直，这是一个更好的抓取姿态
# 注意夹具的坐标系与目标坐标系有\pi/2的z方向角度差异
desired_orientation = np.concatenate(([np.pi, 0], [self.target.get_orientation()[2]]))
    # 夹具与目标垂直，并且朝下
rotation_penalty = -np.sum(np.abs(np.array(self.agent_ee_tip.get_orientation())-
    desired_orientation))
rotation_norm = 0.02
reward += rotation_norm*rotation_penalty
```

其次，如上所述，将夹具和物体之间距离的负数作为奖励函数的一部分可能会导致非最优的夹取

姿势。所以第二个奖励函数上的修正是通过设置目标物体中心上方一个偏移量的位置作为零惩罚点的，这个对距离项的修改如下所示：

```
# 对目标位置的增强奖励：目标位置相对目标物体有垂直方向上的相对偏移
offset=0.08 # 偏移量
sqr_distance = (ax - tx) ** 2 + (ay - ty) ** 2 + (az - (tz+offset)) ** 2
    # 夹具和目标位置距离的平方
```

通过以上两种对奖励函数的增强，学习效果相比于原始的密集或稀疏奖励的情况得到进一步提升，如图 16.8 所示。

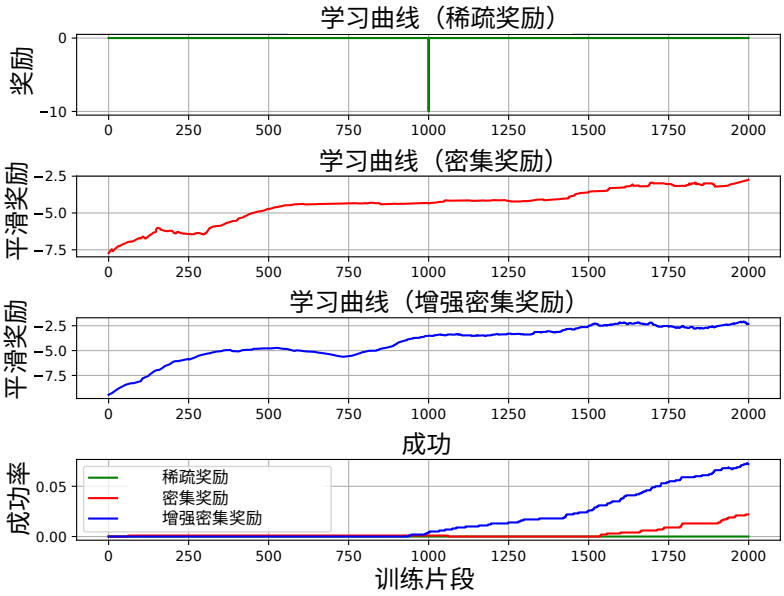

图 16.8　使用 SAC 算法并行训练的Sawyer机器人抓取任务的学习表现，使用不同奖励函数的比较

奖励函数工程是实践中一种有效结合人类先验知识来辅助学习的方式，尽管这可能与科学研究本身的诉求相斥，因为从科研的角度讲，人们往往更加专注于减少奖励函数工程的工作量，以及其他对智能体学习的人为辅助，同时希望实现更加智能和自动化的学习过程。其实，在实践中解决一个任务，类似以上的一些人为辅助设计可能会很有帮助。除了奖励函数工程，从专家示范中学习也是实践中一种有效改善学习效果的方式，如第 8 章所述。

16.2.1 并行训练

CoppeliaSim（V-REP）软件需要每个模拟环境有一个独立的进程。因此，为了加速采样过程，我们设置了多进程而非多线程的方式来并行收集样本。我们的代码库中提供了一个通过 PyTorch 实现的多进程版本的 SAC 算法。其训练和测试过程可以简单地运行如下：

```
# 训练
python sac_learn.py --train
# 测试
python sac_learn.py --test
```

在这个代码中，环境的交互是通过多个进程实现的，每个进程包含一个模拟环境。

16.2.2 学习效果

我们测试了算法在 *Sawyer* 抓取任务上的表现，并在表 16.1 中给出了训练所需的超参数。学习效果如图 16.8 所示，包括三种不同类型的奖励函数。图 16.8 中稀疏奖励函数下的值 -10 是由物体掉落桌面的惩罚造成的。不同的奖励函数给出了不同范围的奖励值，直接对这些奖励值曲线进行比较可能是不公平的。除给出（平滑的）片段奖励外，我们还展示了整个学习过程中的抓取成功率。随着训练进行，我们可以清楚地看到成功事件发生得越来越频繁，这显示出机器人抓取技能的进步。增强的奖励函数对比原始密集奖励函数体现了显著的加速学习的效果，而对于稀疏奖励来说，探索和学习抓取物体几乎是不可能的。

表 16.1 SAC 的超参数

参数	值
优化器	Adam (Kingma et al., 2014)
学习率	3×10^{-4}
奖励折扣（γ）	0.99
工作者（workers）数量	6
隐藏层数（策略）	4
隐藏单元数（策略）	512
隐藏层数（Q 网络）	3
隐藏单元数（Q 网络）	512
批尺寸	128
目标熵值	动作维度的负值
缓存尺寸	1×10^6

如图 16.9 所示，在几千个片段的训练过后，机器人已经能够从一个固定的目标物体位置将其抓到，尽管抓取的姿势不是很完美且成功率还不是很高。在这个例子中，整个训练过程是从头开始的，没有任何的示范或预训练。

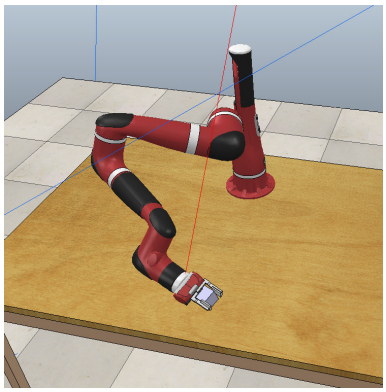

图 16.9　经过训练，Sawyer 在模拟环境中用深度强化学习的策略抓取物体（见彩插）

16.2.3　域随机化

当我们将模拟环境中训练得到的策略用于现实中时，由于现实世界动力学过程和模拟环境中的差异，这个策略往往不能成功。域随机化是改善策略泛化能力的一种方法，尤其是当我们将模拟环境中学习的策略迁移到现实情景中时。

域随机化可以通过随机化环境中的物理参数来实现，包括决定机械臂动力学及其与场景中其他物体动态交互过程的参数。具体来说，随机化物理参数叫作动力学随机化 (Peng et al., 2018)，比如，物体的质量、机械臂上关节的摩擦力、物体和桌面之间的摩擦力等。并且，在基于视觉的控制中，物体颜色、光照条件和物体材质也可以被随机化，这些会影响通过观察机器人图像来对其进行控制的智能体。比如，我们可以用以下命令在 PyRep 中设置物体的颜色：

```
self.target.set_color(np.random.uniform(low=0, high=1, size=3).tolist())
    # 为目标物体颜色设置 [r, g, b]3 通道值
```

其他模拟环境中的物理参数也可以进行相应设置，这里超出了本章范畴。在训练智能体时，我们可以在整个训练过程中对每个片段或者几十个片段进行一次参数重设置。同时，重要的是，要保证对模拟环境中动力学参数和其他特征进行随机化的范围要能够覆盖现实中真实的动力学过程，从而缓解模拟现实间隙。

域随机化只是在模拟到现实迁移中缓解现实间隙的一种可能的方式，以上使用 PyRep 在 CoppeliaSim 中进行视觉特征随机化的步骤也只是一个很简单的例子。关于模拟到现实迁移的详细描述在第 7 章中。

16.2.4 机器人学习基准

在以上小节中，我们展示了如何构建一个机器人抓取任务，并用一个强化学习算法去解决它。近来，文献 (James et al., 2019b) 提出了 RLBench 软件包 (链接见读者服务)，作为一个覆盖 100 个独立的人为设计任务的大规模基准和学习环境。这个软件包专门用于促进基于视觉的机器人操控领域的研究，不仅限于强化学习，而且可以应用于模仿学习、多任务学习、几何计算机视觉和小样本学习。如图 16.10 [5] 所示，RLBench 基于前几节所用的 PyRep，它包含了 100 个基本的机器人操控任务，包括抓取、移动、堆积和其他多样的现实世界中常见的操作，也支持通过简单的设置步骤实现任务定制化，并使得包括强化学习在内的不同的学习方法可以用来解决这个环境中的任务。

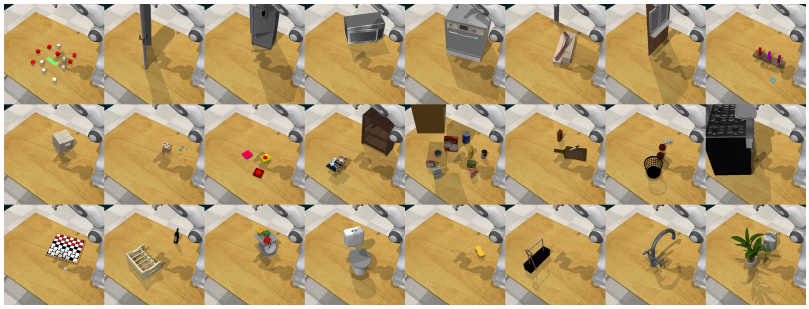

图 16.10 RLBench 中定义的机器人学习任务（见彩插）

前几小节中介绍的机器人抓取任务提供了一个用 CoppeliaSim（V-REP）实现模拟环境中机器人学习的标准框架，这也适用于 RLBench 软件包。它们都包括至少三个基本要素：（1）在 CoppeliaSim（V-REP）中进行任务场景的构建，（2）通过脚本定义环境的模拟过程，包括 `reset()` 和 `step()` 函数，（3）通过脚本提供一个能够学习的智能体，比如用强化学习。RLBench 遵循这种构建流程，但以一种层次化的结构来搭建全体任务。

RLBench 软件包可以通过以下命令来安装（如果你已经安装了 PyRep）：

```
git clone https://github.com/stepjam/RLBench.git
pip3 install -r requirements.txt
python3 setup.py install --user
```

16.2.5 其他模拟器

如图 16.11 所示，有许多不同的机器人学习模拟软件，包括 OpenAI Gym、CoppeliaSim（V-REP/PyRep）(James et al., 2019a; Rohmer et al., 2013)、MuJoCo (Todorov et al., 2012)、Gazebo，

[5] 图像来自 RLBench。

Bullet/PyBullet (Coumans et al., 2016, 2013)、Webots (Michel, 2004)、Unity 3D、NVIDIA Isaac SDK 等。实践中，这些软件包或者平台对于不同的应用有不同的特征。举例来说，OpenAI Gym robotics 环境是一个相对简单的环境，可以快速验证提出的方法；CoppeliaSim 和 Unity 3D 都是基于物理模拟器的，且有着相对较好的渲染效果；MoJoCo 有较为现实和准确的物理引擎，可以用于模拟到现实迁移；Isaac SDK 是一个相对较新的软件（于 2019 年发布），对深度学习算法和应用有较强的支持，以及基于 Unity 3D 的照片级真实的渲染，等等。

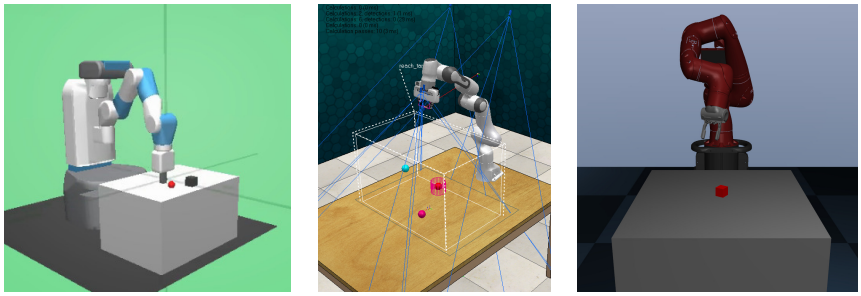

图 16.11　机器人学习任务：（1）OpenAI Gym 中的 FetchPush（左）；（2）使用 PyRep 实现的目标到达任务（中）；RoboSuite 中的 SawyerLift 任务（右）（见彩插）

参考文献

AKKAYA I, ANDRYCHOWICZ M, CHOCIEJ M, et al., 2019. Solving rubik's cube with a robot hand[J]. arXiv preprint arXiv:1910.07113.

ANDRYCHOWICZ M, BAKER B, CHOCIEJ M, et al., 2018. Learning dexterous in-hand manipulation[J]. arXiv preprint arXiv:1808.00177.

COUMANS E, BAI Y, 2016. Pybullet, a python module for physics simulation for games, robotics and machine learning[J]. GitHub repository.

COUMANS E, et al., 2013. Bullet physics library[J]. Open source: bulletphysics. org, 15(49): 5.

HAARNOJA T, ZHOU A, HARTIKAINEN K, et al., 2018. Soft actor-critic algorithms and applications[J]. arXiv preprint arXiv:1812.05905.

JAMES S, FREESE M, DAVISON A J, 2019a. Pyrep: Bringing v-rep to deep robot learning[J]. arXiv preprint arXiv:1906.11176.

JAMES S, MA Z, ARROJO D R, et al., 2019b. Rlbench: The robot learning benchmark & learning environment[J]. arXiv preprint arXiv:1909.12271.

KINGMA D, BA J, 2014. Adam: A method for stochastic optimization[C]//Proceedings of the International Conference on Learning Representations (ICLR).

KORENKEVYCH D, MAHMOOD A R, VASAN G, et al., 2019. Autoregressive policies for continuous control deep reinforcement learning[J]. arXiv preprint arXiv:1903.11524.

MICHEL O, 2004. Cyberbotics ltd. webots: professional mobile robot simulation[J]. International Journal of Advanced Robotic Systems, 1(1): 5.

PENG X B, ANDRYCHOWICZ M, ZAREMBA W, et al., 2018. Sim-to-real transfer of robotic control with dynamics randomization[C]//2018 IEEE International Conference on Robotics and Automation (ICRA). IEEE: 1-8.

ROHMER E, SINGH S P, FREESE M, 2013. V-rep: A versatile and scalable robot simulation framework[C]//2013 IEEE/RSJ International Conference on Intelligent Robots and Systems. IEEE: 1321-1326.

TODOROV E, EREZ T, TASSA Y, 2012. Mujoco: A physics engine for model-based control[C]//IROS.