机器学习工程师纳米学位毕业项目

使用强化学习训练四轴飞行器学会飞行

乔虹

2019 年 1 月 12 日

目录

1 定义

1.1 项目概述

项目将采用强化学习算法训练四轴飞行器，在模拟环境中完成四个飞行任务：起飞，悬停，降落和组合。

四轴飞行器或四轴飞行器无人机在个人和专业应用领域都变得越来越热门。它易于操控，并广泛应用于各个领域。大多数四轴飞行器有四个提供推力的发动机，通过接受一个推力大小和俯仰/偏航/翻滚控件简化飞行控件。传统的方法是通过诸如 PID 控制器的外在媒介手动控制，但如果使用机器学习算法，让四轴飞行器自己学会飞行，则是一件非常有趣也有价值的研究。就像自动驾驶一样，具备自主巡航能力的无人机将会极大的提升工作效率，也会扩展出更多的应用领域。

如果把该任务进行抽象，会发现这是一个典型的连续空间中的强化学习问题。因此我在本项目中采用用强化学习算法，更具体的，是采用深度强化学习中的确定性策略梯度算法（DDPG）。

1.2 问题描述

在该项目中，需要通过 ROS 平台控制一架模拟四轴飞行器，并使用强化学习算法训练四轴飞行器完成四个任务：起飞(Takeoff)，悬停(Hover)，降落(Landing)，和整套动作(Combine)。我们可以把飞行器看作智能体，飞行器在模拟器中飞行就是智能体跟环境的交互，而飞行器的动作包括 3 个方向的线性推力和 3 个方向的扭矩，而这里关键的奖励则需要根据任务的要求去设计。因此，这是一个典型的连续空间中的强化学习问题，而且动作空间也是连续的，具体来说，状态向量有 7 维，而动作向量有 6 维。第一个起飞任务已经设置好。后面的任务都需要先进行定义，设计一个合适的方案，包括奖励机制和目标，状态表示和奖励函数等。对每个任务，都需要选择合适的强化学习算法，然后定义合适的智能体类完成它们。每个任务都要记录下阶段累积奖励，以评估智能体的学习性能。

1.3 评价指标

该项目的评估标准主要是考察飞行器对每个任务的完成度：第一个任务—起飞，要求使飞行器平稳起飞，达到目标高度（地面以上 10 个单元）；第二个任务—悬停，要求飞行器在起飞后在指定高度悬停一段时间；第三个任务—降落，要求飞行器从指定高度平稳且缓慢

的着落；最后，要设计一个端对端任务，完成连贯的从起飞，到悬停，再到降落三个动作。

2 分析

2.1 数据研究

该项目的输入是四轴飞行器 7 维状态向量，包括三个位置向量(pose.position.x, pose.position.y, pose.position.z)，表示飞行器在三维空间中的坐标，和四个方向向量(pose.orientation.x,pose.orientation.y,pose.orientation.z,pose.orientation.w)，分别表示沿 x,y,z 坐标的飞行方向和旋转方向。

在具体的任务中，这些状态向量并不是都需要的，可以通过限制状态空间简化任务，避免不必要的干扰。比如起飞任务就只需要考虑位置，忽略方向，在设计奖励时，主要是根据飞行高度来判断是否起飞，因此奖励函数只需要 pose.position.z 就可以；对于悬停任务，也是主要考虑高度，对于悬停和降落任务，首要考虑的也是高度 pose.position.z，此外还需要入一个特征量-瞬时速度，其方向与飞行方向一致，大小等于高度差/一个时间步的时间长度通过奖励函数限制速度和大小和方向，从而更好的控制飞行姿态。

因此在不同的任务中，需要根据具体情况限制状态空间，选择不同的维度子集。

2.2 算法和方法

2.2.1 算法

项目采用深度强化学习的算法设计模型，深度强化学习是传统强化学习算法在连续空间的扩展，该领域的里程碑是 DeepMind 发表在 nature 的文章 Human-level control through deep reinforcement learning(Volodymyr Mnih, et al. 2015) [1]。这篇文章提出的 DQN 算法结合了深度学习，在视频游戏类任务中表现出色。但由于飞行器的动作空间也是连续的，这个任务并不适合采用 DQN 算法（DQN 只适用于离散动作空间）。DQN 本质上是 value-based 算法，主要考虑值函数的更新，智能体在交互过程中需要通过诸如 Epsilon 贪婪算法的手段间接的选择动作输出。后来又发展出了更加直接的 policy-based 方法，典型的就是策略梯度算法[2]，这类算法不考虑值函数，直接输出策略函数，代表性的有随机策略梯度，确定性策略梯度。在此基础上，又发展出了结合 value-based 和 policy-based 的方法，行动者-评论者（actor-critic），这种算法有两个模型组成，policy 网络是 actor（行动者），输出动作。value 网络是 critic（评价者），用来评价 actor 网络所选动作的好坏。目前最具代表性的 actor-critic

方法是 google 提出的 DDPG（深度确定性策略梯度）算法 [3]。DDPG 中，actor 网络的输入是 state，输出 action，以 DNN 进行函数拟合，对于连续动作 NN 输出层可以用 tanh 或 sigmod，离散动作以 softmax 作为输出层则达到概率输出的效果。critic 网络输入为 state 和 action，输出为 Q 值。

本项目采用的主要是 DDPG 算法，DDPG 基于行动者-评论者方法，结合了 value-based 方法和 policy-based 方法。该算法非常适合连续状态空间和连续动作空间的强化学习，所以符合本项目的需求。DDPG 的主要结构包含两个网络，一个策略网络（Actor），一个价值网络（Critic）。策略网络输出动作，价值网络评判动作。两者都有自己的更新信息，策略网络通过梯度计算公式进行更新，而价值网络根据目标值进行更新。同时，DDPG 沿用了 DQN 的成功经验，即采用了样本池和固定目标值网络这两项技术。算法把智能体和环境的交互分成两部分，训练和学习，在训练阶段，智能体不断跟环境交互并获得反馈，使用一个缓存区存储学习经验；在学习阶段，批量从缓存区中取样，更新模型参数。

2.2.2 技术

本项目使用 ROS（机器人操作系统）作为智能体和模拟之间的主要沟通机制。ROS（Robot Operating System） 是一个在计算机上对机器人进行操作的一个开源系统。ROS 系统通常由大量节点组成，其中任何一个节点均可以通过发布/订阅的方式与其他节点进行通信。举例来说，机器人上的一个位置传感器如雷达单元就可以作为 ROS 的一个节点，雷达单元可以以信息流的方式发布雷达获得的信息，发布的信息可以被其他节点如导航单元、路径规划单元获得。ROS 作为一个灵活的操作系统，其上的节点具有很大的随意性，它们可以位于不同的计算机上，甚至可以位于不同的网络上。举例来说，我们可以使用一个 Arduino 作为一个节点发布信息，使用一台笔记本电脑作为一个节点订阅上述信息以及使用一台手机作为一个节点驱动电机等。上述灵活性使得 ROS 可以适应很多不同场合的应用，另外，ROS 还是一个开源的系统，由专门的社区维护。

Actor 和 Critic 模型采用了的 keras 框架进行构建。Keras 是一个高层神经网络 API，Keras 由纯 Python 编写而成并基 Tensorflow、Theano 以及 CNTK 后端，支持 GPU 和 CPU。Keras 可以基于两个 Backend，一个是 Theano，一个是 Tensorflow，这里采用的是 Tensorflow 的后端。Keras 的功能十分强大，具有用户友好，高度模块化，易扩展性的特点，并能很好的与 python 进行协作。

2.3 基准测试

这里选择项目自带的随机策略搜索（RandomSearch）模型作为基准模型，与我自己定义的 DDPG 模型做对比。RandomSearch 模型是一个非常简单的随机模型，在每个时间步，智能体都会随机选择动作，因此选择每种动作组合的概率都是均等的，这就导致智能体无法进行学习，不能从环境反馈的奖励中学到经验，也就无法提升性能。在前期我已经试验过使用 RandomSearch 训练智能体进行 takeoff 任务，效果十分糟糕，飞行器会毫无章法的翻滚旋转，无法起飞，记录下累积奖励后发现，阶段奖励值一直维持在很低的水平，无法提升。用这个模型跟我的 DDPG 模型对比，可以清楚的反应出 DDPG 模型的优势。如果 DDPG 模型训练的智能体可以很好的完成任务，并获得上升的奖励曲线，就足以说明模型的成功。

3 具体方法

3.1 数据处理

3.2 实施

项目使用一个简单的模拟环境，该环境由 Unity 3D 游戏引擎提供支持。智能体将通过 ROS（机器人操作系统）与该模拟环境互动。在虚拟机中安装并配置好 ROS 后，在主机安装模拟器，然后通过虚拟机运行 ROS，将它与主机的模拟器连接。然后通过执行 ROS 命令建立节点，启动 ROS，并指定运行的任务和智能体，然后打开模拟器就可以运行项目。

该项目的主要工作就是为每个任务编写任务程序（第一个任务 takeoff.py 已经事先定义好）和智能体 agent 程序。takeoff.py 可以作为模板，后三个任务的程序都可以在此基础上修改，具体来说包括：在__init__() 中定义状态（观察）空间和行动空间，并初始化其他需要的变量来使任务顺利运行；修改 reset() 方法，改变初始状态；最核心的部分是 update() 方法，在这里定义任务的状态，通过设计奖励函数计算每个时间步的奖励，并检查阶段是否完成。然后要把 state，reward 和 done 值传递给智能体的 step() 方法，并接受智能体返回的动作向量 action,将它转换为一条 ROS Wrench 信息，这将作为控制命令返回到模拟器，并且会影响四轴飞行器的姿态、方向和速度等。接下来是根据项目提供的智能体模板编写 agent 程序，智能体的核心就是要定义 step () 方法，它接收 task 传来的 state，reward 和 done，并通过算法模型生成下个时间步的动作向量 action 并将其返回给任务程序，我在实现过程中根据算法流程和具体需要定义了其他的方法，会在后面详述。

因为原本的 7 维状态空间和 6 维动作空间结合以后非常大，这样可能会导致智能体策略或值函数模型变得更加复杂。而很多因素和手头的任务都有可能不相关，为了简化任务，防止不必要的干扰，我通过在智能体中设定的几个方法限制了状态和动作空间。对每个任务，我都只保留了状态空间中的高度 z，和动作空间中的 force_z。在悬停和降落任务中，由于要考虑到飞行方向和速度问题，我又加入了额外的特征量，关于这点会在后面的任务实现部分详细描述。

（注：任务实现部分用文字描述了奖励函数的设计，具体的代码片段见项目 notebook 文件“RL-Quadcopter.ipynb”或“RL-Quadcopter.html”）

3.2.1 DDPG 智能体模型设计

我设计的 DDPG 智能体模型由五个部分组成：DDPG 主模型，Actor 模型，Critic 模型，经验缓存区 ReplayBuffer，OU 噪点生成器。

每个时间步，Actor 选择当前状态下采取的动作，并把相应的经验 (s,a,s,r) 放入缓存；每隔一段时间，当缓存中有足够的经验时，从中批量取出样本用于训练 Critic 模型；Critic 模型负责生成策略梯度，用以训练 Actor 模型，更新参数。这里还有一个关键的地方是设置两套 Actor 和 Critic 模型的副本，一个本地副本 local_actor 和 local_critic，一个目标副本 target_actor 和 target_critic，拆分被更新的参数和生成目标值的参数。目标副本作为“固定 Q 目标”，本地副本用于实时更新和输出。在用多个经验进行训练后，将新学习的权重（来自本地模型）复制到目标模型中，进行软更新。

Actor 模型和 Critic 模型都采用 MLP 网络结构，使用 keras 构建。Actor 模型用于将状态映射到动作，它的损失函数损失使用动作值（Q 值）梯度进行定义，这些梯度需要使用评论者模型计算，并在训练时提供梯度。因此指定为在训练函数中使用的“输入”的一部分。Critic 模型需要将（状态、动作）对映射到它们的 Q 值，因此模型输入分为 states 和 actions 两个层级，这两个层级首先可以通过单独的“路径”（迷你子网络）处理，最终通过使用 Keras 中的 Add 层级类型来合并。该模型的最终输出是任何给定（状态、动作）对的 Q 值。然后，还需要计算此 Q 值相对于相应动作向量的梯度，以用于训练行动者模型。

在 DDPG 主模型的 __init__() 方法中，设定了模型的超参数和初始值，指定主模型调用的 Actor 类和 Critic 类（包括两个副本），设定经验回放缓存区 ReplayBuffer 和随机探索 Ornstein‐Uhlenbeck 的噪点；act () 方法通过调用 Actor 模型，把当前状态（向量）作为输入，并加入随机噪点后，返回根据当前策略所选择的动作（向量）；step()方法首接收 task 传来的

state，调用 act()，生成新的 action 返回给任务；然后把经验元组，包括上个时间步的状态 s(t)和动作 a(t)，当前状态 s(t+1)，获得的奖励 reward 和是否完成 done，存入 ReplayBuffer 中，当 ReplayBuffer 有足够的经验时，从中批量取样，调用 learn()方法进行学习；learn()方法则是算法的核心部分，首先要计算得到 Q_target，$Q_{target} = R_t + Q(s_{t+1} + a_{t+1})\gamma$，其中 R_t 为当前回报值，γ 为折扣率，$Q(S_{t+1} + a_{t+1})$ 为过去经验，通过 target_actor 和 target_critic 得到。然后用 Q_target 做标签，训练 local_critic 模型，更新参数；而 local_critic 负责生成策略参数梯度 action_gradients，用 action_gradients 训练 local_actor 模型，最终更新 local_actor 的参数。因此经过一段时间的经验学习，local_actor 和 local_crtic 都得到不断更新，使得 actor 模型能输出更好的 action，而 crtic 模型也能做出更好的评判（生成更准确的 Q 值）。最后调用 soft-update()方法，缓慢更新 target 模型的参数。另外还需要一些辅助的方法，把 task 传来的原始 state 向量降维成低维向量用于模型训练的方法，还有生成一维 action 后（只包括 z_force）将其还原回完整 6 维向量的方法，以及用于软更新的 soft-update()。

3.2.2 任务 1：起飞（takeoff）

takeoff 任务需要实现飞行器的顺利起飞，达到目标高度。因为该只跟高度相关，并且只需要 z 轴的线性推力，为了简化任务和避免出现不必要的旋转和歪斜，这里限制了状态和动作空间。只给飞行器施加了 z 方向的推力，忽略了水平推力和扭矩，因此智能体的运动被限制在 z 轴的单一维度上，也不会有翻转动作，原有的 7 维状态向量中只需要保留 z 高度。

该任务比较简单，也容易实现。直接采用项目自带的任务程序，无需自己设计奖励函数，DDPG 模型参数设置如下：gamma 取 0.99，soft-update 参数取 0.05；Actor 模型采用 3 hidden layers 的结构，激活函数都用 relu；Critic 的两个子网络都用 2 hidden layers，另外 Actor 和 Critic 模型的优化器都用了 Adam。

3.2.3 任务 2：悬停 （hover）

hover 任务中需要实现智能体在指定高度悬停一段时间。这个任务中，我改变了状态表示和奖励函数。状态表示采用了三维向量，分别是高度 z，瞬时速度，和 z 轴线性加速度。跟起飞任务一样，只考虑了原有状态向量中的高度 z，但为了更好的控制飞行方向和速度，又增加了两个特征量：瞬时速度和 z 轴线性加速度。瞬时速度是通过每个时间步的位移除以时间长度计算的，它的正负反映了飞行方向。z 轴线性加速度是飞行器原本自带的特征量，

能够准确反映在某一瞬间飞行器的飞行方向和加速度。在奖励函数的设计中，通过对这两个特征量的限制，可以防止飞行器出现持续加速上升或下落的情况。

为了维持飞行器稳定的悬停，奖励函数从三个方面施加控制，首先是高度偏离惩罚，位置与目标高度的距离越大惩罚越大；然后是速度惩罚，速度和加速度的绝对值越大惩罚越大（目标速度为 0）；还要施加正奖励，当飞行器达到目标高度时施加逐次增大的奖励，以鼓励它向目标靠近。这里我把加速度惩罚作为必须项，速度惩罚只有在目标高度才施加，这是为了在早期探索阶段给飞行器更大的自由度，在飞离了目标高度的情况下让它还可以通过调整姿态再飞回来，如果把速度条件限制太严格可能会扼杀这种探索，导致飞行器很难靠近目标。

因为原有 takeoff 使用的 agent 模型参数和设定无法完成任务，经过反复试验和探索后，对 DDPG 模型做了改进：

1) 改进了经验存放和取样的机制，采用优先级存取

2) gamma 维持 0.99,软更新参数设为 0.05；

3) 为了更好的促进智能体的随机探索行为，我调大了 OUNoise 模型的方差

4) 对 Actor 和 Critic 的网络结构做了调整，actor 最后一层 hidden-layer 的激活函数改为 tanh，Adam 优化器的学习率调至 0.0001；Critic 的优化器采用 Adadelta，加入 L2 正则化。

3.2.4 任务 3: 降落 （landing）

降落任务需要使飞行器从指定高度（z=10）缓慢降落到地面。

该任务的状态表示采用跟 hover 任务一样的三维特征量，初始条件改为从目标高度开始，即在 reset 中设置初始高度为 10.0。重新设计了奖励函数，为了实现缓慢降落，飞行方向必须始终向下，并且速度需要维持在一个合适的较小的值左右。按照 7 秒钟完成从目标高度降落到地面的要求，大致估算了合适的瞬时速度，然后考察每个时间步实际速度跟该目标速度的差距，对其施加惩罚；同时惩罚错误的飞行方向（向上飞）。同时也通过降落总时间进行控制，如果到达地面的时间小于规定时间，要给以惩罚。另一方面，如果智能体能维持合适的速度（包括大小和方向），要给相应的正奖励，这通过 z 轴线性加速度来控制。降落任务采用跟悬停任务相同的 DDPG 模型。

3.2.5 任务 4：组合（combined）

最后，设计一个完整的飞行系统，将前面三个基础动作融合进一个智能体中。我选择了

使用一个智能体完成整套动作，combined 任务设置了分段奖励函数，按照时间顺序依次执行 takeoff,hover 和 landing。由于一开始 takeoff 智能体的设置较为简单，进行 hover 和 landing 任务时对该智能体做了一定更改和优化，实验发现优化后的智能体也能很好的完成起飞任务，因此选用一个智能体进行连续任务的训练。三段奖励函数的设定基本跟单独任务的一致，去除了一些重复和冲突并做了简单调整，加入了对是否起飞成功的检查（5s 内不能起飞就结束阶段），把悬停开始的条件改为达到目标高度（以包括起飞时间少于 5s 的清空）。

3.3 改进

3.3.1 奖励函数改进

hover 任务的奖励函数设置经历了反复多次的修改和调整，起初只考虑了智能体的高度，刚开始的设定较为简单，当飞行器达到目标高度后给以奖励，如果没有达到就给惩罚，并且奖励和惩罚都是常量。这种设定没有考虑到不同高度的差异，过于简单，无法促使智能体逐渐向目标靠近；因此进行了改进，在没有达到高度的情况下，根据位置的不同设置不同的惩罚力度（距离目标越远惩罚越大），后来为了加大惩罚力度，也尝试过使用高次函数作奖励函数，但是效果一直不理想，飞行器无法维持稳定的悬停，经常上下来回波动。

在经过一段时间的试错和经验总结后，发现两个关键的问题：仅靠飞行高度的控制是不够的，实现悬停还有一个关键的条件是速度，只有当智能体维持速度为零才能一直保持悬停姿态，另外通过对速度正负的惩罚也可以控制飞行方向，促使飞行器在错误的位置上（高于或低于目标）尽快调整姿态。因此后来我在 state 中加入了一个特征量：自定义的瞬时速度，并且把智能体原本自带的 z 轴线性加速度也考虑了进来。在奖励函数中对速度和加速度施加惩罚，并当其在目标高度满足速度要求时给以奖励。第二点是需要设法增大对智能体达到目标的正奖励，让它在上下起伏的过程中逐渐“发现”越多次的达到目标奖励越大，因此我记录下了一个 episode 中 agent“击中”目标（即达到目标高度）的次数，把相应的正奖励设为击中次数的二次函数。

在对奖励函数进行改进后，agent 的表现出现了明显的提升。

3.3.2 智能体的随机探索问题

在实验过程中，注意到 agent 时常出现行为过于固定的现象，比如会连续很多次都保持同样的 force_z 值，导致它持续上升不断加速。我意识到这可能是由于随机探索性不足造成的，我使用的是 Ornstein–Uhlenbeck process，根据维基百科，分布曲线的长期固定方差为：

$\mathrm{var}=\sigma^2/2\theta$，$\sigma$越大，$\theta$越小，会使得方差越大，产生的随机数变动的幅度也就越大。因此我尝试向增大方差的方向调整，发现把方差调大后，随机探索行为明显增强了，agent 的动作幅度开始出现变化，但如果把方差调的太大就会导致动作不稳定，时常掉落到地面。经过一些试错和对比后，我找到了较为合适的参数值，能一定程度的促进 agent 的随机探索行为，同时又不至于动作幅度变化太大。

3.3.3 经验回放机制改进

原先的 agent 模型中，记忆存放机制较为简单，直接按时间顺序存放经验，并随机取样，当空间填满后就直接清空重来。这种机制的随机性过大，容易错过有价值的经验。尤其是在训练的早期，高奖励的经验出现次数较少，大部分的经验都是用处不大的错误动作，而如果直接随机存取，很容易错失这些宝贵的经验。为了让智能体更有效的利用经验学习，设置了优先级存放机制。用 deque 存放经验，根据优先级（奖励的高低）排序，如果存满，优先级低的经验会先被“挤出去”，优先级高的经验留存的时间更长。取样时，用 random 的概率机制，概率通过优先级设定，优先级高的经验取样概率更大，但同时也有一定的随机性，以保证取样的灵活。

改进了讲演回放机制后，智能体就能更有效的利用高价值经验，学习效率提升，奖励曲线收敛的更快。

3.3.4 Actor 和 Critic 模型参数调节

第一个起飞任务较为简单，Actor 和 Critic 直接使用了初始的设定就能实现。但在 hover 和 landing 任务中发现初始设定的效果不好，于是从几个方面对 Actor 和 Critic 的设置进行了改进。

首先尝试了不同的网络层数和大小，刚开始试过较大的网络结构（参考原 DDPG 论文的设定），但发现并不适用，智能体的动作过于固定，后来又逐渐把 size 减小，发现有所改进。然后为了防止 critic 出现过拟合，加入了 L2 正则化；为了优化 actor 的输出，对其的激活函数也尝试了不同的方案，发现前两层采用 relu，最后一层采用 tanh 的效果最好；为了加快收敛，对 Actor 和 Crtic 的优化器进行了调整。另外，由于 hover, landing 和 combine 任务都加入了新的特征量，而这两个量跟高度的单位不同，为了排除数量级不统一对训练的影响，我在 Actor 网络的输入层后加了一个批归一化层。最终经过不断尝试，找到了较为合适的组合方案。

3.3.5 DDPG 主模型超参数调节

对于 DDPG 主模型的超参数，包括折扣率 gamma，软更新参数，经验存放区容量和取样量，也都进行了不同的尝试和调节。gamma 值初始设定为 0.99，尝试过把该值调小，但效果欠佳，因为 gamma 越大，agent 对以往经验的重视程度就越高，而对于飞行器的四个飞行任务，因为训练前期会经常出现错误动作，奖励高的经验较少，因此对以往经验的学习更为重要，这能够帮助 agent 更快的找到合适的动作方案。软更新参数略微调大，设置为 0.05，经过实验和分析，我认为该参数的取值要维持较小，以保证 soft-update 的“缓慢”，以使得 target 模型维持固定，这样更有利于训练的稳定。最后，经验存放区的容量我设置了较大的值 10000，考虑到很多任务的训练时间较长，如果容量太小会过早的丢弃有用经验，取样量设置在 150，比初始的 64 有所增大，以适当加快学习速度。

4 结果

4.1 任务完成情况

4.1.1 起飞任务

起飞任务较为顺利，构建好 GGPD 模型后，经过几次实验简单调整，很快就实现了在规定时间内 (5s) 平稳起飞并达到目标高度 (距离地面 10 个单位) 的要求。

该任务完成的阶段奖励曲线如图 1 所示，曲线记录了在 80 个 episode 的运行过程中，每个阶段总奖励值的变化趋势。从图中可看出，奖励曲线收敛较快，并在达到峰值后一直保持稳定。刚开始曲线出现了一小段快速下降，前 10 个 episode 的总奖励值最低，约为-1500 左右，然后在第 12-13 个 episode 间急剧上升，快速达到了峰值-600，其后一直保持-600 左右的总奖励值。

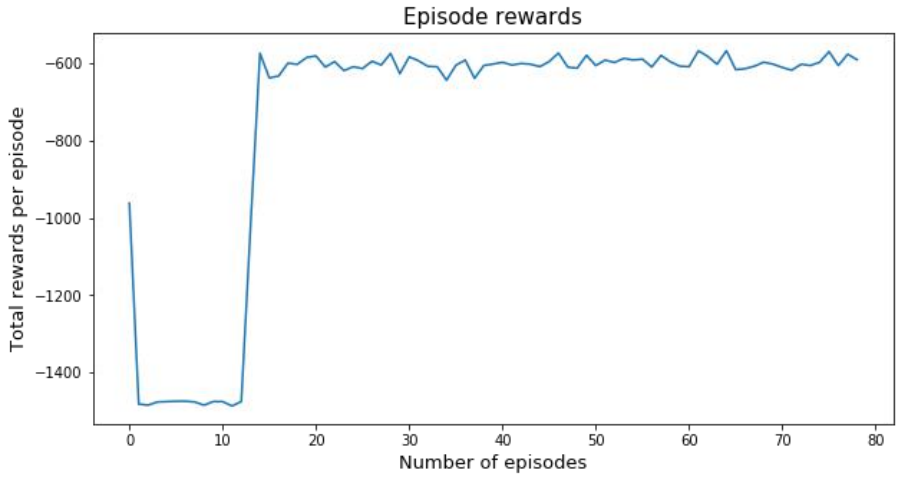

图 1：起飞任务的奖励曲线

从运行情况来看，初期飞行器不断改变 force_z 的大小和方向，经历了几个阶段的上下颠簸后，很快就找到了起飞的最佳条件，此后每个阶段都能实现平稳快速的起飞。在达到稳定后我录制了一小段视频，展示了飞行器连续多次平稳起飞并达到目标高度的情况（相关视频见附件“takeoff.mp4”）。

4.1.2 悬停任务

相比于起飞，悬停任务难度较大，不但经历了反复试验和不断改进调整，飞行器学习过程也更加困难。最后完成任务，实现了飞行器在指定高度 (距离地面 10 个单位) 维持一段时间 (5s) 的悬停，相应的阶段奖励曲线图 2 所示。

从奖励曲线来看，曲线收敛时间更长，智能体学习速度较慢。总共运行时间约为 60 个 episode，前期经历了缓慢上升和反复波动，在大约第 27 个 episode 时开始出现快速上升，在大约第 30 个 episode 后曲线达到收敛，收敛后的峰值越为-5600 左右，相比最低值增大了约 12 倍。

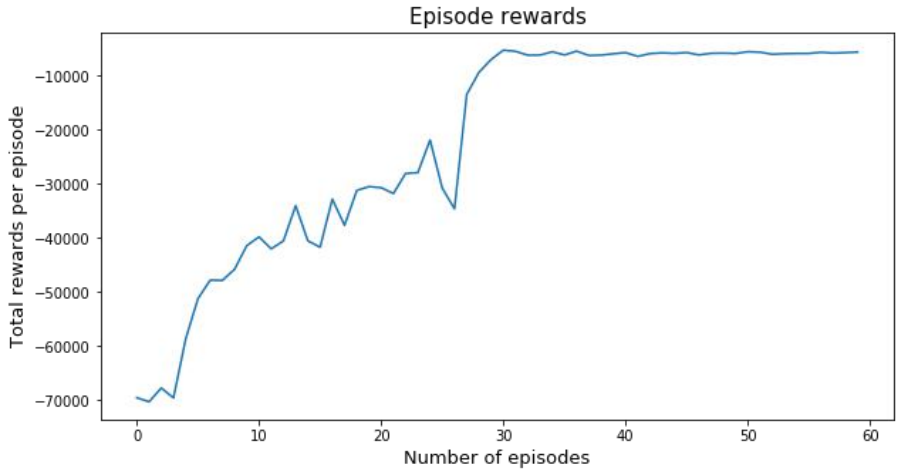

图 2：悬停任务的奖励曲线

从运行表现来看，飞行器前期经常下落或持续上升，经过一段时间的探索和经验学习后，终于找到了维持在目标高度悬停的条件，后期表现较为稳定。达到稳定后录制了一短简短的视频，展示了飞行器实现维持在目标高度悬停的行为（视频见附件“hover.mp4”）。

4.1.3 降落任务

降落任务难度更大些，飞行器经历了较长时间的尝试和经验学习后，完成从目标高度(距离地面 10 个单位) 缓慢降落，降落时间在 6-7s 之内，图 3 为相应的阶段奖励曲线。

整个运行时间约为 185 个 episode，前 20 个 episode 内，阶段总奖励值始终维持在-10000 以下，从第 25 个 episode 开始上升，后面又经历了一段时间的反复波动后，最终在第 85 个 episode 左右收敛至峰值，此后一直维持着-2000 左右的阶段总奖励值。整个过程中，奖励曲线收敛时间较长，收敛后的奖励值提升了约 5 倍。

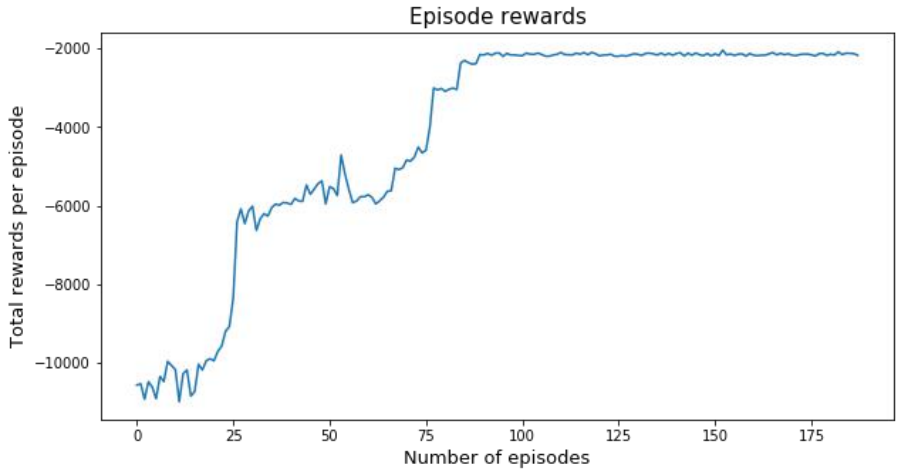

图 3：降落任务的奖励曲线

从运行表现来看，飞行器前期经历了较长时间的试错和调整，刚开始下落速度太快，然后逐渐学会减慢速度，但中间出现了几次持续上升的情况，一段时间后终于找到了合适的方法，在一定范围内不断快速变换 force_z 的大小以维持速度稳定在合适的值，实现了缓慢降落。在收敛后录制了简短视频，展示了飞行器从指定高度缓慢降落的行为（视频见附件“landing.mp4”）。

4.1.4 组合任务

最后的组合任务，训练一个智能体完成了从起飞，悬停再到降落的整套连续动作，相应的奖励曲线如图 4 所示。

整个过程运行了 300 个 episode 以上，曲线收敛较慢，从开始一直持续缓慢的上升，直到约 120 个 episode 后达到收敛，收敛后阶段总奖励值增大了约 3 倍左右。

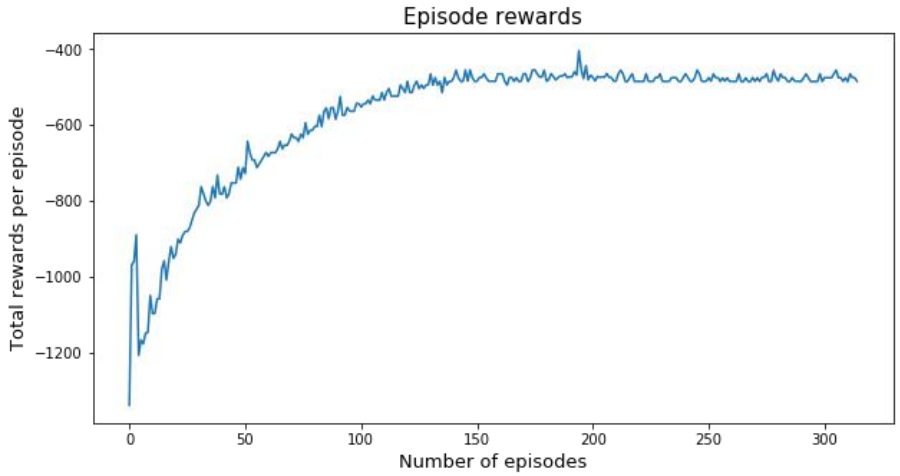

图 4：整套动作组合任务的奖励曲线

从运行表现来看，智能体学习的时间更长。由于三个任务的组合难度更大，刚开始训练时智能体很快学会了顺利起飞，但悬停任务又经过了一段时间的试错后才找到合适的条件，最后的降落任务也花了一段时间探索，才做到在起飞并悬停后缓慢稳定的降落。最终实现整套连续动作的视频见附件（“combined.mp4”）。

4.2 分析讨论

第一个起飞任务任务相对比较简单，比较容易就能实现，智能体训练时间也较短，学习曲线在开始有一段下降，然后出现急速上升，很快收敛，之后就一直维持平稳。该任务中，智能体学习速度较快，很快就能找到合适的起飞条件，奖励曲线收敛后也一直维持很稳定的性能。

从第二个任务开始，难度逐渐增大。智能体在进行第二个 hover 任务时，明显遇到了更大的困难，奖励曲线要经过较长的时间才能收敛，学习速度较慢。由于 hover 任务要求飞行器在指定高度的悬停，也就是要保持速度为零的状态，这需要飞行器的得重力与升力维持平衡。因此智能体必须不断调整线性推力 force_z 以维持飞行姿态的稳定，在训练的早期，飞行器时常出现上下起伏波动，一直在试图向着目标高度靠近，不断调整错误的飞行方向，随着时间推移，方向调整的频率越来越快，上下位置的变动幅度也越来越小，最终达到平衡。与前两个任务相比，降落的难度更大些。由于 landing 需要向下飞行，并且还要保持速度足够小以实现缓慢降落，这需要飞行器不断的快速调整 force_z，以维持飞行方向始终向下，并且速度还要稳定在足够小的范围内，因而找到合适的条件以满足要求就需要花费很长的时

间。最后的 combine 任务难度最大，需要顺利完成之前的三个独立动作，不仅要在适当的时间切换飞行姿态，调整速度，还要保持动作的连贯性，这就需要更长时间的探索和学习。

虽然任务难度较大，但通过不断的实验和改进，最终找到了合适的奖励函数和模型参数设定。在算法的控制下，智能体在早期更多的是进行随机探索，需要经过一段时间的试错后，积累了足够的经验，而奖励函数的合理设定能让智能体充分的了解到什么样的动作更好，什么样的动作更差，在后期智能体探索行为减弱，会对之前积累的经验有效学习和利用，不断优化模型参数使得输出的动作越来越接近目标。

5 结论

5.1 总结和思考

在这个项目中，我实现了用深度强化策略梯度（DDPG）算法在模拟环境中控制四轴飞行器完成四个飞行任务：起飞，悬停，降落和整套动作组合。为了简化任务，并防止不必要的倾斜和翻转，我限制了动作和状态空间，只保留了完成任务所需的 z 轴线性推力和 z 高度，并基于悬停和降落任务的需要，给状态向量加入了两个新的特征量：瞬时速度和加速度，通过奖励函数的设定，能更有效的调节飞行姿态。

我采用 DDPG 算法构建了智能体模型，该模型由 DDPG 主模型，Actor 模型，Critic 模型，经验缓存区 ReplayBuffer，OU 噪点生成器五个部分组成，首先用该模型实现了起飞任务（任务程序已经实现定义好）。然后对于悬停和降落任务，分别单独设定了奖励函数，根据任务的要求设置合适的惩罚和奖励。最后的 combine 使用了一个智能体来训练，设置分段奖励函数来实现。在进行后三个任务的过程中，我在实验中不断优化改进奖励函数。在经历了不断探索，多次反复的试错和总结后，我对原始的 DDPG 模型做了较大的改进和调整，包括调节 OU 噪点模型方差以促进智能体的探索，采用优先级存取方案来优化讲演回放机制，提高学习效率，以及对 Actor 和 Critic 网络结构和参数的调节等。最终完成了每个任务，达到任务目标，并在每个任务中获得了随时间上升并最终收敛且维持稳定的奖励曲线。

总的来说，这个项目是有一定难度的，要完成该项目的要求，顺利实现对四轴飞行器姿态的控制，最核心的地方是采用的算法模型和奖励函数的设计。对于这个典型的连续状态和动作空间的强化学习问题，我采用了目前最为合适的 DDPG 算法，算法的架构和流程参考原论文设定。在实践过程中，为了满足具体运行环境和任务的需要，还需要对具体的实现方式和模型超参数做出很多调整。而奖励函数的设定也非常重要，不合理的奖励函数无法对智能

体做出正确的指导，使得智能体的学习性能无法提升。整个项目中最困难的部分是悬停任务，由于本身难度较大，再加上前期的思考不充分走了一些弯路，比如开始没有想到悬停任务需要加上对速度条件的控制，还有开始的随机经验回放机制也存在不合理，以及 OU 噪点参数不合适导致探索行为太弱等问题。这个任务做了很长的时间，把奖励函数和模型参数配置都进行了很多次的调整，经历了多次的尝试，不断试错才最终找到合适的条件。而在后面做降落和组合任务时，由于有了前面的经验，并且 DDPG 模型已经优化到一定程度了，通过合理的奖励函数设定和调整，完成起来就没那么困难了。在这个过程中，我不仅加深了对强化学习和 DDPG 算法的理解，锻炼了编程能力，更充分了解了奖励函数的设计对智能体性能的影响，也总结了 DDPG 算法模型参数调节的一些方法和技巧，积累了宝贵的经验。

5.2 后续改进

在本项目中，我已经使用 DDPG 模型控制四轴飞行器完成了四个任务：起飞，悬停，降落和整套动作组合。但仍有需要改进的地方，组合任务虽然达到了目标，但奖励曲线收敛较慢，需要运行较长时间才能完成连续的整套动作，降落任务也存在类似的情况。这可能跟两个方面的因素有关，首先就是奖励函数的设定，为了充分考虑到各种可能的情况，我在奖励函数中设置了很多条件，使得整体有些复杂，这可能会影响智能体的学习效率。另外一个方面就是 DDPG 模型的设置和超参数，可能还需要进一步优化 Actor 和 Critic 模型，包括它们的网络结构，激活函数和优化器等。后期可以从这两个方面继续优化和改进，以提高智能体学习效率，加快奖励曲线的收敛。

另外，我为了简化任务限制了动作和状态空间，使得飞行器只能在 Z 轴方向做一维运动。后续可以考虑适当放开限制，给飞行器更多的自由度，让它能在水平 x,y 方向运动，并可以加入适当的扭矩，让它能够倾斜和翻转。这样会让飞行器更接近现实状态的情况，在高维复杂的动作和状态空间中测试智能体的性能，进一步改进和优化模型，可以增强模型的鲁棒性，让其适应更真实的环境。

参考文献

[1] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg & Demis Hassabis. (2015.) Human-level control through deep reinforcement learning

[2] Richard S. Sutton, David McAllester, Satinder Singh, Yishay Mansour,AT&T Labs - Research, 180 Park Avenue, Florham Park, NJ 07932 . Policy gradient methods for reinforcement learning with function approximation.

[3] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, Daan Wierstra. (2015). Continuous control with deep reinforcement learning