一、MADDPG 算法的并行训练改造说明

文件 1：maddpg.py（核心训练逻辑）

修改前（串行训练）：

每个 agent 在 train() 中实时调用其他 agent 的 actor 网络：（原来代码 40 行左右）

```
with torch.no_grad():  # target_Q has no gradient
    # Select next actions according to the actor_target
    batch_a_next_n = [agent.actor_target(batch_obs_next) for agent, batch_ob
    Q_next = self.critic_target(batch_obs_next_n, batch_a_next_n)
    target_Q = batch_r_n[self.agent_id] + self.gamma * (1 - batch_done_n[sel
```

修改后（策略冻结，避免训练中交叉依赖）：

增加 cached_actions 参数，训练前统一缓存所有 agent 的 target 动作（代码 40 行左右）：

```
if cached_actions is None:
    #让每个 agent 都用自己当前的 actor_target 网络对 next_obs 做前向推理；
    #所有动作都提前计算好，相当于冻结当前策略输出。
    #后续每个 agent 都会用这个静态版本的动作训练自己的 critic，从而解耦依赖，支
    cached_actions = [
        agent.actor_target(batch_obs_next_n[i])
        for i, agent in enumerate(agent_n)
    ]
```

随后在 critic 和 actor 训练中，使用这些静态动作：

```
50          with torch.no_grad():
51              Q_next = self.critic_target(batch_obs_next_n, cached_actions)
52              target_Q = batch_r_n[self.agent_id] + self.gamma * (1 - batch_don
53
```

文件 2：inter_layer_make_env.py（内层环境控制）

修改前：

```
231     if self.replay_buffer.current_size > self.args.batch_size:
232         # Train each agent individually
233         for agent_id in range(self.args.N):
234             self.agent_n[agent_id].train(self.replay_buffer, self.agent_n)
235
```

修改后：

在调用每个 agent 训练前，**先统一缓存所有 agent 的 actor_target 输出**：

```
236         # 策略冻结：提前计算所有 UAV 的 actor 网络输出（动作）
237         with torch.no_grad():
238             static_actions = [
239                 agent.actor_target(batch_obs)
240                 for agent, batch_obs in zip(self.agent_n, self.replay_buffer.
241             ]
```

然后将其传入每个 agent 的 train 函数：

```
243         # 并行训练每个 UAV，使用静态动作作为其他 UAV 的输入
244         for agent_id in range(self.args.N):
245             self.agent_n[agent_id].train_parallel(
246                 self.replay_buffer,
247                 static_actions,  # 所有 UAV 的静态策略动作
248                 agent_id
249             )
```

二、无人机收益记录和角度弧度问题矫正

先遍历所有的 UAV 找到计算最快的 UAV：

```
126             # 遍历所有 UAV，找到处理当前任务耗时最短的那个
127             for idx, uav in enumerate(self.uavs):
128                 d_u_v, d_u_cloud = uav.get_uv_distance(user.position, myCon
129                 trans_rate_user, trans_rate_cloud = uav.get_uav_transmissio
130                 temp_time = uav.get_uav_transmission_computing_time(task,
131
132                 if temp_time < task_min_time:
133                     task_min_time = temp_time
134                     uav_cope_task_index = idx
```

把让每个 UAV 都去计算收益的循环移除了，只让最优的 UAV 去计算收益：

```
137     if uav_cope_task_index != -1:
138         uav = self.uavs[uav_cope_task_index]
139 
140         # 如果 UAV 命中缓存并且任务尚未完成
141         if (task[0] in uav.cached_service_type or task[0] in uav.cached_cont
142             # 扣除资源
143             uav.storage_space -= task[0]
144 
145             if task[0] in uav.cached_service_type:
146                 uav.F_cpu -= task[2]
147                 temp_profit = 1.0 * (task[3] / task_min_time) * (
148                         task[1] / uav.storage_space +
149                         task[2] / uav.F_cpu +
150                         (task[2] * uav.alpha / uav.F_cpu) / uav.remain_power
151                         task[-2] / 15
152                 )
153                 uav.F_cpu += task[2]  # 释放资源
154             else:
```

同时让角度输入前先转成弧度：

```
95                 for i in range(len(self.uavs)):
96                     temp_uav = self.uavs[i]
97                     move_direction = 180 * inter_layer_actions[i] + 180
98             #-------------------------下面是改动的地方-------------------------------
99                     move_direction = math.radians(move_direction)
100                    temp_uav.uav_move(move_direction, move_distance)
101 
```

训练后的效果如下：