PX4 SITL与质点模型混合集群实验

1. 实验目的

本实验旨在通过 RflySim 仿真平台，掌握混合异构无人机蜂群的飞行控制方法。具体目标如下：

1. **理解两种仿真模式的差异**：掌握 PX4 SITL（软件在环）精确模式与点质量模型（NoPX4）轻量化模式在初始化方式、控制接口和适用场景上的本质区别。
2. **掌握 MAVLink Offboard 控制流程**：学习通过 `PX4MavCtrlV4` 库对 PX4 飞控发送位置/速度指令，理解 Arm 解锁、Offboard 模式切换等关键操作步骤。
3. **实现编队跟随控制**：设计并实现以单架 PX4 SITL 无人机为 Leader、4 架点质量模型无人机为 Followers 的菱形编队飞行，Followers 实时读取 Leader 的 NED 位置并保持固定偏移量跟随。
4. **掌握 NED 坐标系约定**：理解 RflySim 中北-东-下（NED）坐标系的定义，以及 UE4 地图坐标系与 NED 坐标系的对应关系，能够正确设置飞行高度与平面运动目标点。
5. **学习定频控制循环设计**：理解 30 Hz 精确定时控制循环的实现原理，掌握基于时间补偿的主控循环结构，避免因计算延迟导致的控制频率漂移。

2. 实验要求

- 软件要求：Windows 10及以上版本；RflySim工具链[1]。
- 硬件要求：笔记本/台式电脑1台[2]。

3.实验地址

例程目录：[安装目录]\RflySimAPIs\10.RflySimSwarm\2.AdvExps\e4_PX4+MPSwarm

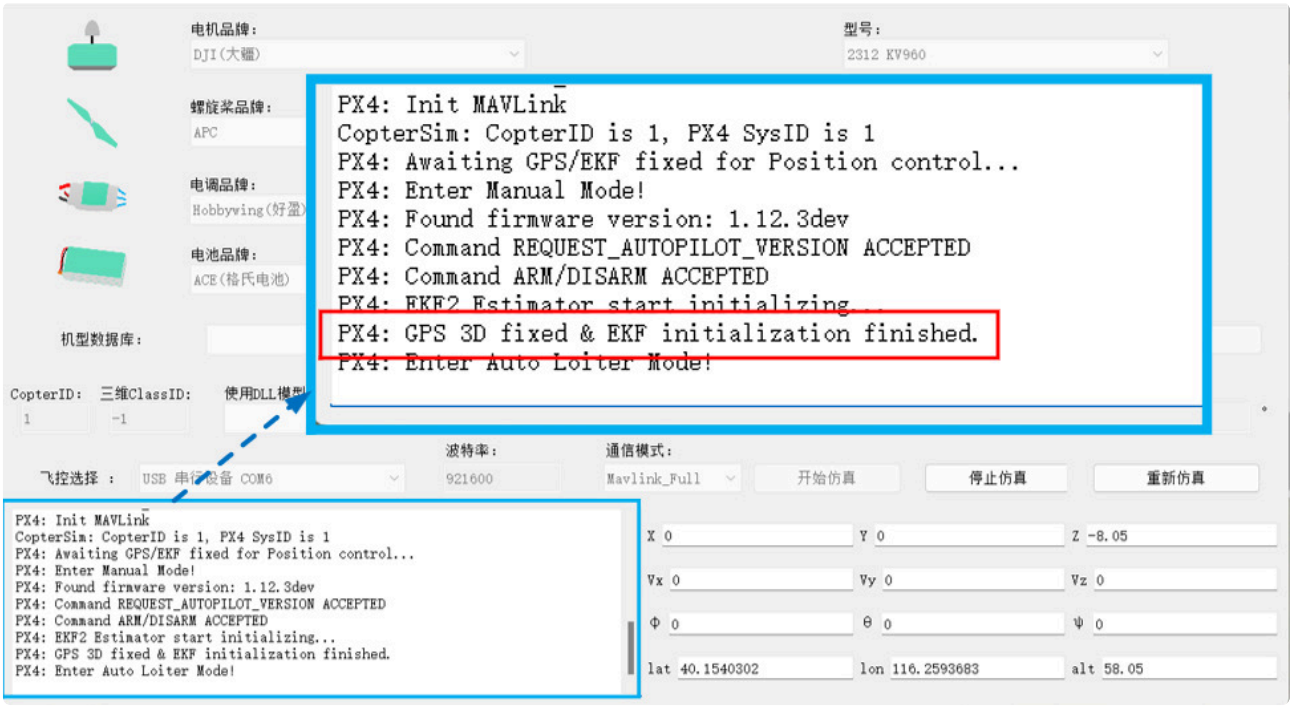
文件目录结构说明：

```
e4_PX4+MPSwarm\
├── HybridSwarmCtrl.py       # 【主控程序】1架PX4 Leader + 4架点质量模型Followers混合蜂群控制
├── UAVCtrlSILRun.bat        # 一键启动PX4 SITL完整仿真环境脚本
├── Python38Run.bat          # RflySim Python 3.8运行环境启动脚本
```

4. 实验内容或步骤

4.1 仿真初始化

1、双击运行`UAVCtrlSILRun.bat`文件，等待仿真环境初始化完成。脚本将会启动 1 个 QGC 地面站， 1 个 CopterSim 、1 个 RflySim3D 软件， 等待CopterSim软件下侧日志栏必须打印出`GPS 3D fixed & EKF initialization finished`字样代表初始化完成。如下图所示：



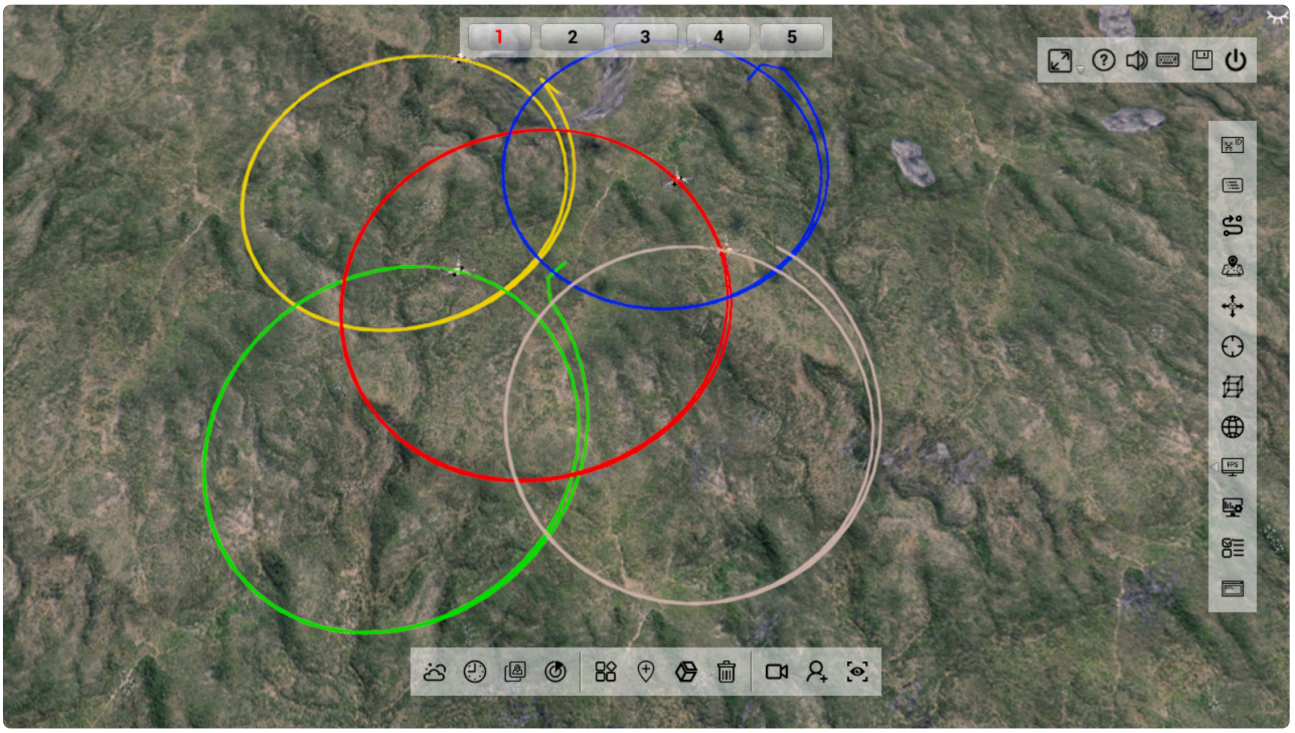
4.2 启动1+4架无人机控制程序

双击`Python38Run.bat`脚本，在弹出的CMD对话框中输入：`python HybridSwarmCtrl.py`，等待控制程序启动完成。

```
D:\RflySim\RflySimAPIs\10.RflySimSwarm\2.AdvExps\e4_PX4+MPSwarm>python HybridSwarmCtrl.py
============================================================
  混合蜂群控制程序启动
  Leader(PX4):  ID = 1
  Followers(点质量模型): ID = 2 ~ 5
============================================================
[Init] 创建 4 架 Follower 实例 (ID 2~5)
[Init] 初始化 Followers 点质量模型...
  Follower 1 (ID=2) 初始化于 X=2.0, Y=0.0, Yaw=0°
  Follower 2 (ID=3) 初始化于 X=-2.0, Y=0.0, Yaw=0°
  Follower 3 (ID=4) 初始化于 X=0.0, Y=-2.0, Yaw=0°
  Follower 4 (ID=5) 初始化于 X=0.0, Y=2.0, Yaw=0°
```

4.3 查看无人机仿真效果

在RflySim3D窗口下按下`T`键，即可打开所有无人机的飞行轨迹显示，具体飞行效果如下图所示。



5. 关键知识点

5.1 整体实验框架与思路

本实验的核心思路是**混合异构编队**：将高精度 PX4 SITL 飞控（适合验证控制算法）与轻量化点质量模型（适合大规模编队）有机结合，在同一仿真环境中实现协同飞行。整体框架分为五个阶段：

```
┌──────────────────────────────────────────────────────────┐
│  阶段1：实例化            阶段2：初始化          阶段3：起飞  │
│  创建Leader/Follower  ──→  点质量模型初始化  ──→  编队起飞  │
│  MAVLink通信实例          MAVLink Offboard      悬停就位    │
├──────────────────────────────────────────────────────────┤
│  阶段4：主控制循环（30 Hz）                                 │
│  Leader圆形轨迹 ──→ 实时读取Leader NED位置 ──→ Followers跟随 │
├──────────────────────────────────────────────────────────┤
│  阶段5：安全退出                                           │
│  降落 ──→ 退出Offboard ──→ 上锁 ──→ 停止点质量模型实例       │
└──────────────────────────────────────────────────────────┘
```

Leader 与 Follower 的核心差异对比：

对比项	Leader（PX4 SITL, ID=1）	Followers（点质量模型, ID=2~5）
仿真模式	CopterSim PX4_SITL_RFLY	NoPX4 点质量模型
初始化接口	`InitMavLoop()`	`initPointMassModel(alt, stat`
解锁方式	`SendMavArm(True)`	无需解锁，直接响应位置指令
位置控制	`SendPosNED(x, y, z, yaw)`	`SendPosNED(x, y, z, yaw)`
状态读取	`leader.uavPosNED`	不需要（被动跟随）

5.2 NED 坐标系与 UE4 坐标系

本实验涉及两套坐标系，理解其差异是避免初始化位置错误的关键：

- **NED 坐标系**（北-东-下）：`SendPosNED` 所使用的坐标系。`x` 朝北为正，`y` 朝东为正，`z` 朝下为正。因此飞机在地面上方时 `z < 0`，高度 8 m 对应 `z = -8.0`。
- **UE4 地图坐标系**：`initPointMassModel` 的第一个参数 `intAlt` 和第二个参数 `intState=[X, Y, Yaw]` 使用 UE4 地图坐标，`X` 近似对应北方向，`Y` 近似对应东方向，高度从地面（0）计量。

```
# NED 坐标系示例：飞到北方 5m、东方 0m、高度 8m
leader.SendPosNED(5.0, 0.0, -8.0, 0)   # z=-8.0 ← 负值表示地面以上

# UE4 坐标系示例：在地图 X=2.0m, Y=0.0m 的位置初始化，高度使用-8(m)
mav.initPointMassModel(-8, [2.0, 0.0, 0])
#   参数1: -8 → 初始化在距地面 8m 高的位置
#   参数2: [X=2.0, Y=0.0, Yaw=0°] → 地图平面坐标与初始偏航
```

5.3 Leader 初始化流程（PX4 MAVLink Offboard）

PX4 SITL 飞机的 Offboard 控制必须按照严格顺序执行，否则将导致 Arm 失败或 Offboard 模式无法激活：

```python
import PX4MavCtrlV4 as PX4MavCtrl

# 第1步：创建通信实例，ID=1 对应 CopterSim #1
leader = PX4MavCtrl.PX4MavCtrler(1)

# 第2步：启动 MAVLink UDP 数据接收循环（默认 UDPMode=2，即 MAVLink_Full）
#   - 开始监听来自 CopterSim 的飞机状态数据
#   - 之后可通过 leader.uavPosNED 读取实时位置
leader.InitMavLoop()
time.sleep(0.5)

# 第3步：进入 Offboard 模式（必须在 Arm 之前调用）
#   - 该函数内部会先发送若干条 SetPoint 消息，再切换到 Offboard 模式
leader.initOffboard()

# 第4步：发送初始目标位置（Arm前必须有有效目标点，否则Arm会被拒绝）
leader.SendPosNED(0, 0, -8.0, 0)   # 目标：原点上方 8m
time.sleep(0.5)

# 第5步：发送解锁（Arm）命令
leader.SendMavArm(True)
```

注意：`initOffboard()` 与 `SendMavArm()` 之间必须先发送至少一条 `SendPosNED`，PX4 会拒绝在没有有效 Setpoint 的情况下切换到 Offboard 模式并解锁。

5.4 Followers 点质量模型初始化

4 架 Followers 使用点质量模型，无需 PX4 固件，通过 `initPointMassModel` 直接在仿真环境中生成并控制：

```python
# 菱形队形各 Follower 的初始地图位置（UE4 坐标，单位 m）
FOLLOWER_INIT_POSITIONS = [
    [ 2.0,  0.0, 0],   # Follower 1 (ID=2): Leader 正前方 2m
    [-2.0,  0.0, 0],   # Follower 2 (ID=3): Leader 正后方 2m
    [ 0.0, -2.0, 0],   # Follower 3 (ID=4): Leader 正左方 2m
    [ 0.0,  2.0, 0],   # Follower 4 (ID=5): Leader 正右方 2m
]

for i, mav in enumerate(followers):
    x, y, yaw = FOLLOWER_INIT_POSITIONS[i]
    # intAlt=-8：直接以悬停高度 8m 初始化，省去起飞等待过程
    # intState=[x, y, yaw]：UE4 地图平面位置与初始偏航角
    mav.initPointMassModel(-8, [x, y, yaw])
```

5.5 菱形编队跟随算法

编队控制的核心逻辑是：**每个 Follower 的目标位置 = Leader 当前实时 NED 位置 + 固定偏移量**。这种方法无需复杂的轨迹预测，实现简单且实时性好：

```
# 菱形队形偏移量（NED 坐标系，单位：米）
FORMATION_OFFSETS = [
    [ 3.0,  0.0,  0.0],   # Follower 1: 前方 3m（北方向+3）
    [-3.0,  0.0,  0.0],   # Follower 2: 后方 3m（北方向-3）
    [ 0.0, -3.0,  0.0],   # Follower 3: 左方 3m（东方向-3）
    [ 0.0,  3.0,  0.0],   # Follower 4: 右方 3m（东方向+3）
]

# 在主控制循环中，每个控制周期（1/30s）执行一次：
leader_pos = leader.uavPosNED   # 实时读取 Leader 的 NED 位置 [x, y, z]

for i, mav in enumerate(followers):
    offset = FORMATION_OFFSETS[i]
    follower_x = leader_pos[0] + offset[0]   # 北方向目标
    follower_y = leader_pos[1] + offset[1]   # 东方向目标
    follower_z = leader_pos[2] + offset[2]   # 高度与 Leader 完全同步
    mav.SendPosNED(follower_x, follower_y, follower_z, 0)
```

5.6 圆形轨迹生成（Leader）

Leader 飞行的圆形轨迹通过参数化方程实时计算，前 5 秒悬停等待 Followers 稳定，之后进入圆周运动：

```
TRAJ_RADIUS = 5.0    # 圆形轨迹半径（m）
TRAJ_PERIOD = 20.0   # 完成一圈所需时间（s）

# 在主控制循环中实时计算目标点：
t = time.time() - startTime   # 已飞行秒数

if t < 5.0:
    # 前 5s：原点悬停，等待编队稳定就位
    leader_target_x = 0.0
    leader_target_y = 0.0
else:
    # 5s 后：圆形轨迹
    # angle 从 0 增大到 2π，即完成一圈
    angle = 2 * math.pi * (t - 5.0) / TRAJ_PERIOD
    # cos(0)=1，减去 TRAJ_RADIUS 使轨迹从原点(0,0)出发
    leader_target_x = TRAJ_RADIUS * math.cos(angle) - TRAJ_RADIUS
    leader_target_y = TRAJ_RADIUS * math.sin(angle)

leader.SendPosNED(leader_target_x, leader_target_y, LEADER_HOVER_ALT, 0)
```

5.7 精确定时控制循环（30 Hz）

主控制循环使用**时间补偿机制**确保控制频率精确稳定，避免因代码执行时间不均匀导致控制周期漂移：

```python
lastTime     = time.time()
timeInterval = 1.0 / CTRL_FREQ   # 控制周期 = 1/30 ≈ 0.0333s

while True:
    # 累加下一次期望触发时刻
    lastTime = lastTime + timeInterval
    # 计算还需等待的时间
    sleepTime = lastTime - time.time()
    if sleepTime > 0:
        time.sleep(sleepTime)   # 正常情况：等待到下一个时刻
    else:
        lastTime = time.time()  # 超时（计算量过大）：重置基准时刻，防止积累偏差

    # 以下为 30Hz 控制代码
    # ...
```

> 这种定时方式比 `time.sleep(1/30)` 更精确，因为每次循环都会自动补偿上一次执行超时的误差。

5.8 安全退出流程

PX4 飞机需要按严格顺序退出，避免飞机失控：

```python
# 1. 发送近地目标位置，使飞机缓慢降落
leader.SendPosNED(0, 0, -0.1, 0)
time.sleep(1)

# 2. Followers 同步降落
for mav in followers:
    mav.SendPosNED(0, 0, -0.1, 0)
time.sleep(2)

# 3. 退出 Offboard 模式（切换回位置控制模式）
leader.endOffboard()

# 4. 上锁（停止电机）
leader.SendMavArm(False)

# 5. 停止 MAVLink 数据接收循环
leader.stopRun()

# 6. 停止各 Follower 的点质量模型仿真实例
for mav in followers:
    mav.EndPointMassModel()
```

6.参考资料

1. RflySim官方文档：https://rflysim.com/doc/zh/
2. RflySim工具链下载与安装指南：https://rflysim.com/doc/zh/HowToInstall.pdf
3. MAVLink通用消息协议定义（POSITION_TARGET_LOCAL_NED）：https://mavlink.io/en/messages/common.html#SET_POSITION_TARGET_LOCAL_NED
4. PX4 Offboard控制官方文档：https://docs.px4.io/main/en/flight_modes/offboard.html
5. PX4 飞行模式枚举定义（MAV_CMD）：https://mavlink.io/en/messages/common.html#MAV_CMD

7.常见问题

Q1：运行 `HybridSwarmCtrl.py` 后，Leader 无法解锁（Arm），程序卡住或 CopterSim 日志显示 Arm 被拒绝。

A1：该问题通常由以下原因导致，请按顺序排查：

1. **EKF 未初始化完成**：CopterSim 日志栏必须出现 `GPS 3D fixed & EKF initialization finished` 字样后才能运行控制程序。若未等待该

提示就运行脚本，PX4 内部状态机尚未就绪，会拒绝 Arm 请求。

2. **Offboard 模式激活失败**：PX4 要求在切换到 Offboard 模式之前，必须已经以足够高的频率收到 Setpoint 消息（通常 > 2 Hz）。程序中 `initOffboard()` 内部已处理此逻辑，但若 UDP 通信异常（如端口冲突），可能导致 Offboard 切换失败。请检查防火墙设置，确保 UDP 端口 `20100`、`20200` 未被占用。
3. **已有旧 CopterSim 进程残留**：`UAVCtrlSILRun.bat` 启动时会自动 `taskkill` 旧进程，但若手动关闭后重开，需等待 5~10 秒让端口彻底释放再运行控制脚本。

Q2：Followers（点质量模型飞机）出现在错误位置，或初始化后立即飘移而非悬停。

A2：原因在于 `initPointMassModel` 的坐标系理解有误。该函数第一个参数 `intAlt` 是 **UE4 地图中的绝对高度**（单位：米，地面为 0），而不是 NED 坐标系中的 z 值。若要让 Followers 从空中 8m 高度初始化，应传入 `-8`（约定负值表示地面以上）。第二个参数 `intState=[X, Y, Yaw]` 对应 UE4 地图平面坐标，单位为米，与 NED 坐标的 x/y 方向近似对应。请对照代码中的注释核对：

```
# 正确写法：从 8m 高度初始化，X=2m 前方，Y=0m
mav.initPointMassModel(-8, [2.0, 0.0, 0])

# 错误写法：intAlt=0 会使飞机从地面生成，需要额外等待起飞
# mav.initPointMassModel(0, [2.0, 0.0, 0])  ← 不推荐
```

Q3：运行过程中 Followers 的跟随存在明显滞后，与 Leader 的实际距离明显大于设定偏移量。

A3：跟随滞后是点质量模型的固有特性。点质量模型使用简化的动力学方程，其位置跟踪响应速度受内部参数限制，在 Leader 高速运动时（如圆形轨迹半径大、周期短时）滞后现象会加剧。解决方法：

1. **降低 Leader 运动速度**：增大 `TRAJ_PERIOD`（延长圆形轨迹周期），或减小 `TRAJ_RADIUS`（缩小轨迹半径），使 Leader 的运动速度不超过点质量模型的跟踪能力（通常建议最大飞行速度 < 3 m/s）。
2. **采用预测跟随策略**：将 Follower 的目标点改为 Leader 的**目标位置（规划值）**而非当前实际位置（`leader.uavPosNED`），可以减少动态滞后。本实验默认使用实际位置，如需优化可修改如下：

```
# 当前实现（跟随 Leader 实际位置，存在滞后）
leader_pos = leader.uavPosNED

# 优化实现（跟随 Leader 目标位置，可降低动态滞后）
leader_pos = [leader_target_x, leader_target_y, leader_target_z]
```

---

1. https://rflysim.com/ ↩
2. 推荐配置请见： https://rflysim.com/doc/zh/HowToInstall.pdf ↩