综合模型 SITL 的 6 机纵一字跟随转圈实验

1. 实验目的

本实验在前序单机 ROSTrans 控制实验的基础上，进一步演示**多机**场景：在不依赖 PX4 固件的情况下，CopterSim 加载综合动力学模型（DLL Model，即 `MulticopterNOpx4.dll`）完成飞行解算，RosTrans 中间件的 DllModelAssistant 插件为 6 架无人机分别建立独立的 ROS 通信通道（`/mavros`、`/mavros2`…`/mavros6`），由一段 Python 节点统一调度，实现基于 ROS 的领航-跟随（Leader-Follower）纵一字编队转圈飞行。其中 UAV1 作为 Leader 沿圆形轨迹飞行，UAV2~UAV6 作为 Follower 沿 Leader 航向后方依次排成纵一字队形跟随。

通过本实验，学习者应能够：

- 理解“6 个 CopterSim 综合模型 ↔ RosTrans 多机桥接 ↔ 单一 ROS 节点”的多机数据链路与按命名空间区分的寻址方式；
- 掌握 RosTrans 多机控制的 ROS 通信接口（每机独立的 RX 主题、TX 主题与服务）与 `/mavrosN` 命名空间规则；
- 理解纵一字编队的几何计算——Follower 偏移量如何随 Leader 航向实时旋转，使队形始终沿飞行方向排列；
- 学会 Leader-Follower 编队控制策略：Leader 做圆形轨迹位置规划，Follower 基于目标点的比例（P）控制跟随并限速；
- 掌握多机同步的标准飞行操作序列（统一 Pre-arm → OFFBOARD → 解锁 → 起飞 → 编队 → 降落 → 上锁）；
- 了解通过 `rflysim_msgs/UE4Cmd` 配置 UE4 俯视视口（相机、无人机编号、飞行轨迹显示）；
- 熟悉同一套 Python 代码兼容 ROS1（rospy）与 ROS2（rclpy）的多机工程化写法。

2. 实验要求

- 软件要求：Windows 10 及以上版本；RflySim 工具链[1]；WSL2 Ubuntu 环境（用于运行 ROS/ROS2 节点）。
- 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\6.RflySimExtCtrl\0.ApiExps\e21.RosTransExps\5.SixCopterSwarmNoPx4

本实验文件夹下的完整文件目录结构如下：

```
5.SixCopterSwarmNoPx4/
├── MulticopterNOpx4.bat                # SIL 启动脚本：复制 DLL 模型并启动 6 个
CopterSim + 1 个 RflySim3D
├── MulticopterNOpx4.dll                # CopterSim 综合动力学模型（不含 PX4 固件）
├── rostrans.param                      # RosTrans 中间件配置文件（6 机 DllModel 模式）
├── SixFollowFlightCircle.py            # 6 机纵一字跟随编队转圈飞行脚本
└── WinWSL.bat                          # Windows—WSL2 网络/图形桥接脚本（启动 VcXsrv
并进入 WSL）
```

4. 实验内容或步骤

4.1 启动仿真环境

双击运行 `MulticopterNOpx4.bat` 文件，等待仿真环境初始化完成。脚本将会启动 1 个 RflySim3D 软件，并依次启动 6 个 CopterSim 实例（编号 1~6），6 架飞机按网格排列（间距 2m），地图为 Grasslands。

等待所有 CopterSim 窗口的仿真模式，均显示`Simulink&DLL SIL`模式，即表示初始化完成。

4.2 启动 RosTrans 中间件

2、在 WSL 中启动 RosTrans 并加载参数。双击 'WinWSL.bat'，在弹出的对话框中输入：

```
rostrans
```

该指令将加载本目录下的 `rostrans.param` 配置文件。RosTrans 将自动：

- 连接 6 个 CopterSim（DLL Model 模式），为每架飞机创建独立的 ROS 通信通道
- 在 `/mavros`、`/mavros2`、…、`/mavros6` 命名空间下发布遥测主题和服务
- 桥接 UE4 显示控制通道（`enable_ue3d=true`）

`rostrans.param` 关键配置说明：

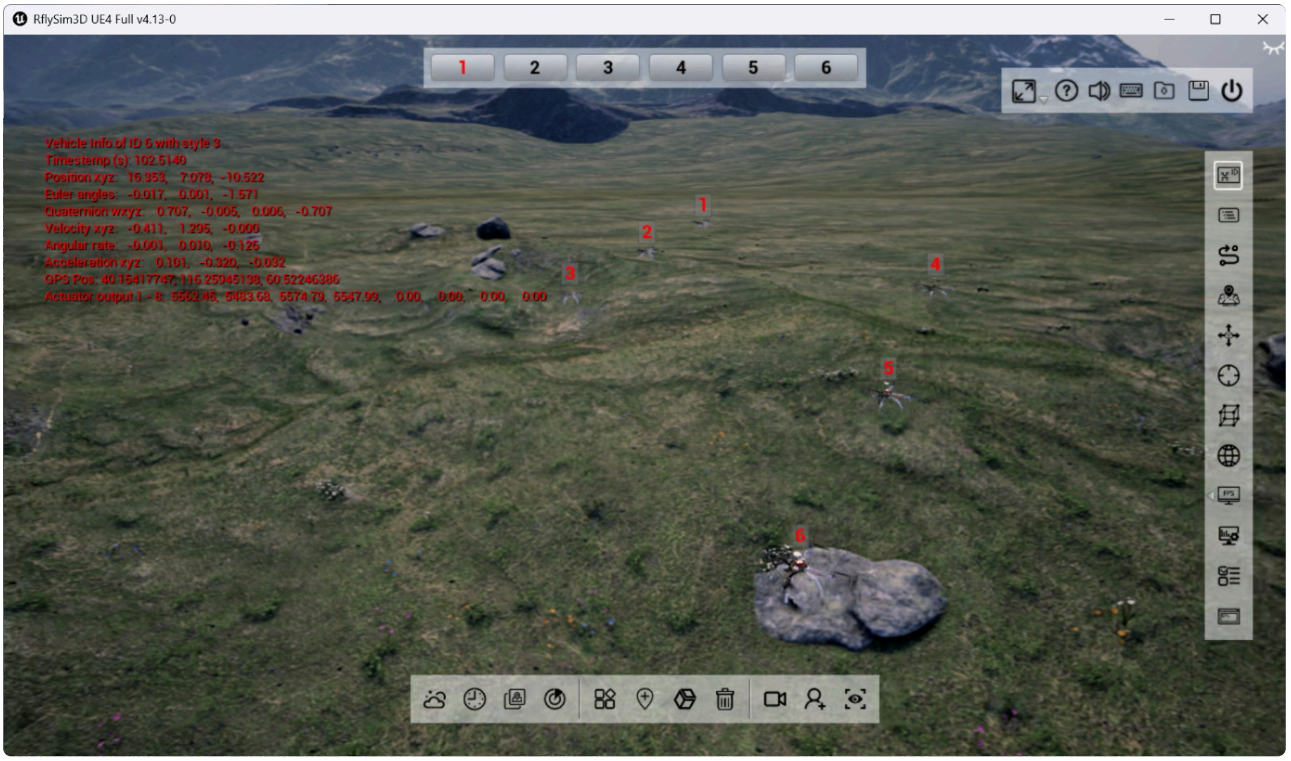
```
enable_dllModel=true          # 启用 DLL 综合模型（无 PX4）
coptersim_startid=1           # 飞机起始编号
coptersim_count=6             # 仿真飞机数量
coptersim_targetip=127.0.0.1  # CopterSim IP 地址
enable_ue3d=true              # 启用 UE4 3D 显示控制
enable_vision=false           # 本实验不使用视觉传感器
enable_mavros=false           # 不启用独立 MAVROS 节点
```

4.3 运行 6 机编队转圈飞行

3、再次双击 'WinWSL.bat' 脚本，在弹出的对话框中运行如下代码：

```
# 基本用法（6机，高度8m，绕圈半径4m，3圈）
python3 SixFollowFlightCircle.py
```

可在 RflySim3D 中观察到 6 架无人机依次解锁起飞，并以纵一字队形绕“圆”飞行。



运行结束后，该程序也支持传入参数自定义设置。再次双击 'WinWSL.bat' 脚本，在弹出的对话框中运行如下代码：

```
# 自定义参数（高度 8m，绕圈半径 4m，切向速度 0.5m/s，3 圈）
python3 SixFollowFlightCircle.py --alt 8 --radius 4 --speed 0.5 --laps 3
```

也可再次双击 'WinWSL.bat' 脚本，在弹出的对话框中运行如下代码，以调整纵一字编队间距：

```
# 调整编队间距为 3m
python3 SixFollowFlightCircle.py --spacing 3.0
```

全部命令行参数说明如下：

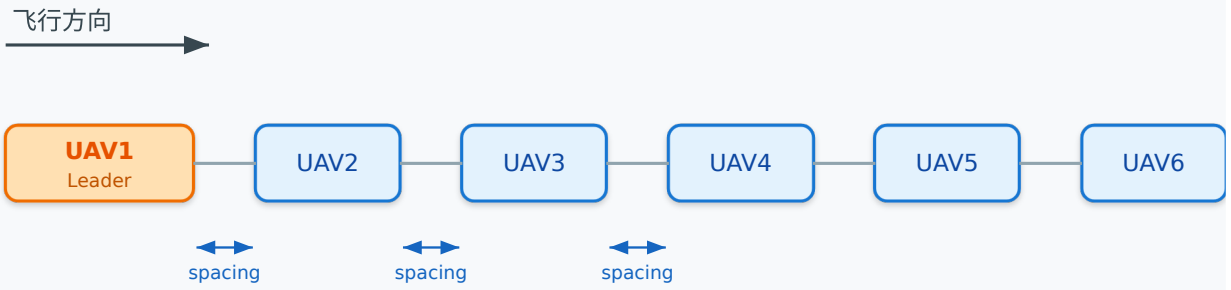
```
参数          默认值    说明
----------  ------  ------------------------------
--alt         8.0     飞行高度 (m)
--radius      4.0     Leader 绕圈半径 (m)
--speed       0.5     Leader 切向速度 (m/s)
--laps        3       飞行圈数
--spacing     2.0     纵一字编队间距 (m)
--win         0       UE4 窗口 ID
```

4.4 飞行流程说明

飞行流程分为 4 个阶段：

```
阶段        内容                          说明
--------  ------------------------  ---------------------------------
Phase 0   等待遥测 + 配置 UE4 视口      俯视相机 + 编号 + 轨迹显示
Phase 1   Pre-arm + OFFBOARD + 解锁    6 架飞机同时切模式、解锁、起飞
Phase 2   起飞至目标高度                速度模式爬升，到达后悬停稳定
Phase 3   编队绕圈飞行                  Leader 飞圆形轨迹，Followers 纵一字跟随
Phase 4   各回起始位 + 降落             各 UAV 回各自出生位置后逐步降落
```

编队控制原理如下：



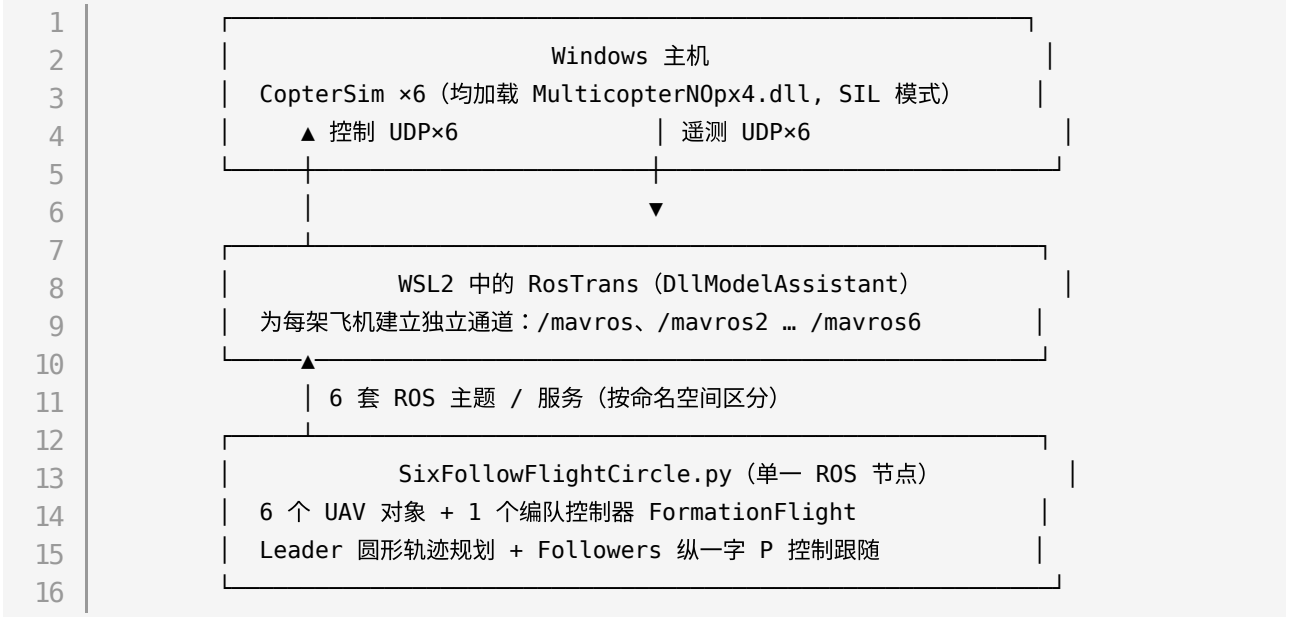
各 Follower 沿 Leader 飞行方向后方排列，偏移量随航向实时旋转，始终保持纵一字队形

- **Leader (UAV1)**：按圆形轨迹生成位置 setpoint，使用 `setpoint_raw/local` 位置模式
- **Followers (UAV2~6)**：读取 Leader 当前绕圈角度，计算沿飞行方向后方的偏移量，生成各自的位置 setpoint
- 编队偏移随 Leader 航向实时旋转，保持纵一字队形始终沿飞行方向排列

5. 关键知识点

5.0 实验整体思路与框架

本实验是单机 ROSTrans 控制的多机扩展。其核心是：让 6 个独立的 CopterSim 综合模型实例各自跑飞行解算，由 RosTrans 统一桥接成 6 套 MAVROS 风格的 ROS 接口，再用**一个** Python 节点同时驱动 6 架飞机协同飞行。整条数据链路如下：



程序 `SixFollowFlightCircle.py` 的设计可归纳为三层：

1. **单机封装层** `UAV`：把“一架飞机”的订阅、发布、服务调用封装成对象，构造时根据 ID 自动绑定 `/mavrosN` 命名空间；6 架飞机就是 6 个 `UAV` 实例；
2. **编队控制层** `FormationFlight`：持有 6 个 `UAV`，区分 1 个 Leader 与 5 个 Follower，负责轨迹规划、编队几何计算与整体飞行流程（Phase 0~4）；
3. **可视化层** `UE4Viewport`：通过 `rflysim_msgs/UE4Cmd` 配置 RflySim3D 俯视视口、编号与轨迹显示。

下文先给出多机接口与操作序列总览（5.1–5.3），再逐段解析 `SixFollowFlightCircle.py` 的关键代码（5.4 起）。

5.1 多机 ROS 通信接口

多机命名空间规则（飞机 1 用 `mavros`，其余用 `mavrosN`）：

```
飞机 ID    命名空间
--------  -----------------
  1        /mavros/...
  2        /mavros2/...
  N        /mavrosN/...
```

RX 主题（CopterSim → ROS，每架飞机独立）：

```
主题                                消息类型          说明
--------------------------------  -----------    ------------------------
/mavrosN/local_position/pose       PoseStamped     ENU 本地位姿
/mavrosN/state                     State           飞控状态（armed / mode）
```

TX 主题（ROS → CopterSim）：

```
主题                                  消息类型           说明
----------------------------------  --------------   --------------------
/mavrosN/setpoint_raw/local           PositionTarget    位置/速度多模式控制
/mavrosN/setpoint_position/local      PoseStamped       位置 + 四元数控制
```

服务（每架飞机独立）：

```
服务                          类型            说明
-------------------------  ------------    --------------
/mavrosN/param/set           ParamSet        设置飞控参数
/mavrosN/set_mode            SetMode         切换飞行模式
/mavrosN/cmd/arming          CommandBool     解锁 / 上锁
/mavrosN/cmd/takeoff         CommandTOL      起飞
/mavrosN/cmd/land            CommandTOL      降落
```

5.2 纵一字编队偏移计算

```python
# 第 idx 个 Follower (0~4) 在 Leader 后方 (idx+1)*spacing 处
dist = (idx + 1) * spacing
# 飞行反方向 = Leader 绕圈角度 + π/2 (切线) + π (反向)
back_angle = leader_heading + π/2 + π
offset_east  = dist * sin(back_angle)
offset_north = dist * cos(back_angle)
```

偏移量随 Leader 绕圈角度实时旋转，确保队列始终沿飞行方向排列。

5.3 标准多机飞行操作序列

```
[所有 UAV] pub_hover()              # Pre-arm setpoint 流
[所有 UAV] param_set(NAV_RCL_ACT, 0)
[所有 UAV] set_mode(OFFBOARD)
[所有 UAV] arming(True)
[所有 UAV] takeoff(alt)
[Phase 2]  爬升至目标高度
[Phase 3]  Leader 绕圈 + Followers 跟随
[Phase 4]  各 UAV 回 (0,0) 原点降落
[所有 UAV] set_mode(AUTO.LAND)
[所有 UAV] land()
[所有 UAV] arming(False)
```

5.4 单机控制封装：UAV 类

每架飞机都被封装为一个 `UAV` 对象。构造时根据飞机 ID 推导命名空间，并完成订阅与发布的初始化；6 架飞机即 6 个独立对象，互不干扰：

```python
class UAV:
    def __init__(self, uav_id, node=None):
        self.id = uav_id
        self.ns = "mavros" if uav_id == 1 else f"mavros{uav_id}"   # 命名空间随 ID
        self._node = node
        self.last_pose = None
        self.last_state = None
        self.pos = [0.0, 0.0, 0.0]   # ENU: [east, north, up]
        self._init_comms()
```

`_init_comms()` 同样区分 ROS1/ROS2：ROS2 下订阅端必须使用 `BEST_EFFORT` 的 QoS 才能与 RosTrans 发布端匹配（否则收不到遥测，对应 Q1）。位姿回调直接把 ENU 坐标缓存到 `self.pos`，供编队跟随时实时读取：

```python
def _cb_pose(self, msg):
    self.last_pose = msg
    p = msg.pose.position
    self.pos = [p.x, p.y, p.z]   # ENU, East/North/Up
```

控制指令统一通过 `setpoint_raw/local`（`PositionTarget`）下发，并用 `type_mask` 切换“位置模式”或“速度模式”——这是同一主题实现多种控制的关键：

```python
def pub_position(self, x, y, z, yaw=0.0):
    msg = PositionTarget()
    msg.header.stamp = self._now()
    msg.coordinate_frame = PositionTarget.FRAME_LOCAL_NED
    msg.type_mask = POS_ONLY_MASK     # 仅保留位置 + yaw，忽略速度/加速度
    msg.position.x, msg.position.y, msg.position.z = float(x), float(y), float(z)
    msg.yaw = float(yaw)
    self.pub_raw.publish(msg)

def pub_velocity(self, vx, vy, vz):
    msg = PositionTarget()
    msg.header.stamp = self._now()
    msg.coordinate_frame = PositionTarget.FRAME_LOCAL_NED
    msg.type_mask = VEL_ONLY_MASK     # 仅保留速度，忽略位置/加速度/yaw
    msg.velocity.x, msg.velocity.y, msg.velocity.z = float(vx), float(vy),
float(vz)
    self.pub_raw.publish(msg)
```

其中两个掩码常量分别屏蔽不同分量：

```
POS_ONLY_MASK =
(IGNORE_VX|IGNORE_VY|IGNORE_VZ|IGNORE_AFX|IGNORE_AFY|IGNORE_AFZ|IGNORE_YAW_RATE)
VEL_ONLY_MASK =
(IGNORE_PX|IGNORE_PY|IGNORE_PZ|IGNORE_AFX|IGNORE_AFY|IGNORE_AFZ|IGNORE_YAW|IGNORE_Y
AW_RATE)
```

`UAV` 还提供 `at_altitude`、`is_armed`、`is_offboard` 等属性，便于飞行流程判断各机状态。

5.5 Leader 圆形轨迹与纵一字编队偏移

Leader（UAV1）按参数化圆生成位置 setpoint，偏航取切线方向，使机头始终朝向飞行方向：

```python
def _leader_circle_pos(self, theta):
    """Leader 在圆上的 ENU 位置, theta 顺时针增加"""
    cn, ce = CIRCLE_CENTER
    north = cn + self.radius * math.cos(theta)
    east  = ce + self.radius * math.sin(theta)
    yaw   = theta + math.pi / 2    # 切线方向
    return east, north, yaw
```

Follower 的位置由 Leader 当前角度推出。第 `idx` 个 Follower（0~4 对应 UAV2~UAV6）位于 Leader 飞行方向**后方** `(idx+1)*spacing` 处，偏移角 = 飞行方向 + π（反向），因此队形随 Leader 转弯实时旋转，始终保持纵一字：

```python
def _formation_offset(self, idx, leader_heading):
    dist = (idx + 1) * self.form_spacing      # 距 Leader 的距离
    back_angle = leader_heading + math.pi / 2 + math.pi   # 飞行反方向(后方)
    dx_e = dist * math.sin(back_angle)        # East 偏移
    dy_n = dist * math.cos(back_angle)        # North 偏移
    return dx_e, dy_n
```

这里 `leader_heading + π/2` 是切线（飞行）方向，再 `+ π` 转到正后方；`spacing` 越大队列越长，但过小可能碰撞、过大可能滞后（见第 7 节 Q3）。

5.6 Follower 比例（P）控制跟随

Phase 3 中，Leader 直接发位置 setpoint，Follower 则先算出各自的目标点（Leader 位置 + 编队偏移），再用比例控制计算速度并限速，使其平滑收敛到目标点：

```python
for i, fol in enumerate(self.followers):
    off_e, off_n = self._formation_offset(i, leader_heading)
    tx = lx + off_e            # 目标 East
    ty = ly + off_n            # 目标 North
    ex = tx - fol.pos[0]       # 位置误差
    ey = ty - fol.pos[1]
    vx = FOLLOW_KP * ex        # P 控制 (FOLLOW_KP=1.2)
    vy = FOLLOW_KP * ey
    spd = math.sqrt(vx*vx + vy*vy)
    if spd > FOLLOW_VEL_MAX:   # 限速 (FOLLOW_VEL_MAX=1.5 m/s)
        scale = FOLLOW_VEL_MAX / spd
        vx *= scale; vy *= scale
    fol.pub_position(tx, ty, self.alt, lyaw)
```

`FOLLOW_KP` 决定跟随响应速度（增益越大跟得越紧但易超调），`FOLLOW_VEL_MAX` 防止误差过大时速度飙升。代码在计算速度后实际仍以位置模式 `pub_position` 下发目标点、由飞控内环跟踪，P 控制的速度量在此用于评估收敛与限幅逻辑。

5.7 主飞行流程 run() 的五个阶段

`FormationFlight.run()` 按统一节拍（`RATE_HZ=20`）驱动全部 6 架飞机，分五个阶段执行：

```python
def run(self):
    # Phase 0: 等待遥测 + 配置 UE4 俯视视口
    sleep_ok(2); self.ue4.setup(); sleep_ok(1)

    # Phase 1: Pre-arm setpoint 流 → 统一切 OFFBOARD → 解锁 → 起飞
    end_t = time.time() + PRE_ARM_SEC
    while time.time() < end_t and not _interrupted:
        for uav in self.uavs: uav.pub_hover()      # 6 机同时刷 setpoint
        self._rate_sleep(rate)
    for uav in self.uavs: uav.srv_param_set("NAV_RCL_ACT", 0.0)
    for uav in self.uavs: uav.srv_set_mode("OFFBOARD")
    for uav in self.uavs: uav.srv_arm(True)
    for uav in self.uavs: uav.srv_takeoff(self.alt)

    # Phase 2: 速度模式爬升至目标高度，到达后悬停稳定 2s
    # Phase 3: Leader 绕圈 + Followers 纵一字 P 控制跟随（laps 圈）
    # Phase 4: 各机回各自起始位 → 逐步下降 → AUTO.LAND → 上锁
```

要点：

- **Pre-arm setpoint 流**：OFFBOARD 模式要求切模式前已有持续的 setpoint，故先用 `pub_hover()` 给 6 架飞机刷 `PRE_ARM_SEC`（默认 3s）的悬停流，否则部分飞机可能切不进 OFFBOARD（对应 Q2）；
- **统一调度**：每个服务调用都用 `for uav in self.uavs` 对 6 架逐一下发，实现“多机同步”；
- **Phase 2 起飞**：未达高度的用速度模式 `pub_takeoff_vel` 爬升，已达高度的切位置模式锁定，直到全部到位；
- **Phase 4 收尾**：各机先回各自本地原点 `(0,0)`（即出生点）再逐步降高，最后切 `AUTO.LAND`、上锁，并补发一段着地 setpoint。

5.8 UE4 俯视视口配置

`UE4Viewport` 通过 `rflysim_msgs/UE4Cmd` 主题向 RflySim3D 发送控制台命令，组合出“俯视相机 + 编号 + 轨迹”的观察视角：

```python
def setup(self):
    if not self.pub:
        print("  [UE4] rflysim_msgs 未安装, 跳过视口配置"); return
    self.send('RflyChangeViewKeyCmd N 6')                          # 地球静止俯视
    self.send('RflyCameraFovDegrees 120')                          # 视场角 120°
    self.send('RflyCameraPosAng 0.00 0.00 -30.00 0.00 -90.00 0.00')  # 正上方俯视
    self.send('RflyChangeViewKeyCmd S')                            # 显示无人机编号
    self.send('RflyChangeViewKeyCmd T 8')                          # 显示飞行轨迹
```

相机位置使用北东地（NED）坐标、单位米，`-30` 表示位于场景上方 30m，俯仰角 `-90°` 即垂直向下俯视，便于俯瞰整个编队队形。若 `rflysim_msgs` 未安装，`HAS_UE4_MSG` 为 `False`，视口配置会被安全跳过，不影响飞行（对应 Q4）。

6. 参考资料

1. RflySim官方文档：https://rflysim.com/doc/zh/
2. MAVROS 功能包 Wiki（Offboard 控制与主题/服务说明）：http://wiki.ros.org/mavros
3. ROS2 QoS（服务质量）配置指南：https://docs.ros.org/en/humble/Concepts/About-Quality-of-Service-Settings.html
4. mavros_msgs 消息与服务定义参考：http://docs.ros.org/en/api/mavros_msgs/html/index.html

7. 常见问题

Q1：运行脚本后没有收到遥测数据

A1：请检查以下几点：

- 确认 6 个 CopterSim 已全部启动并显示 `DLL Model` 模式
- 确认 RosTrans 已启动并加载了正确的 `rostrans.param`（`coptersim_count=6`）
- 确认 WSL2 与 Windows 网络互通（可运行 `WinWSL.bat` 配置网络桥接）
- ROS2 环境下需确保 QoS 匹配（订阅端使用 `BEST_EFFORT`）

Q2：部分飞机未解锁或未起飞

A2：DllModelAssistant 服务初始化需要时间。请确认：

- RosTrans 配置中 `enable_dllModel=true` 且 `coptersim_count=6`
- 脚本的 `PRE_ARM_SEC`（默认 3 秒）足够 RosTrans 完成所有飞机的服务初始化
- 如果部分飞机持续失败，可增大 `PRE_ARM_SEC` 或手动重试

Q3：编队队形不正确（Followers 位置偏离）

A3：请检查：

- `--spacing` 参数是否合理（默认 2m，过小可能碰撞，过大可能超出通信范围）
- Leader 绕圈半径是否足够大（默认 4m），过小时 Followers 会因惯性滞后
- 可调整 `FOLLOW_KP`（跟随增益）提高跟随响应速度

Q4：UE4 视口命令无效（编号/轨迹未显示）

A4：请确认：

- `rostrans.param` 中 `enable_ue3d=true`
- RflySim3D 已启动
- `rflysim_msgs` 包已正确安装（ROS1 需 source 对应的 workspace）

---

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