基于综合模型ROSTrans控制实验

1. 实验目的

在常规的 RflySim 外部控制实验中，飞行动力学由 PX4 固件（SITL）解算。本实验则演示另一条技术路线：在**不依赖 PX4 固件**的情况下，CopterSim 直接加载综合动力学模型（DLL Model，即 `MulticopterNOpx4.dll`）完成飞行解算，再通过 RosTrans 中间件将该模型的遥测与控制数据桥接为标准的 MAVROS 风格 ROS 主题和服务。用户因此可以用与真实 MAVROS 完全一致的接口（同样的主题名、消息类型与服务名）来订阅遥测、下发控制指令，从而把已有的 ROS 控制代码无缝迁移到“无 PX4”的综合模型仿真上。

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\3.BasicRosTopicCtrlNoPx4

本实验文件夹下的完整文件目录结构如下：

```
3.BasicRosTopicCtrlNoPx4/
├── CopterSimNOpx4.bat                  # SIL 启动脚本：复制 DLL 模型并启动 CopterSim + 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（启用 DLL 模型与 UE4 通道）
├── dllModel_flight.py                  # 8 字飞行例程：演示全部 RX/TX 主题、服务与 UE4 视口控制
└── 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

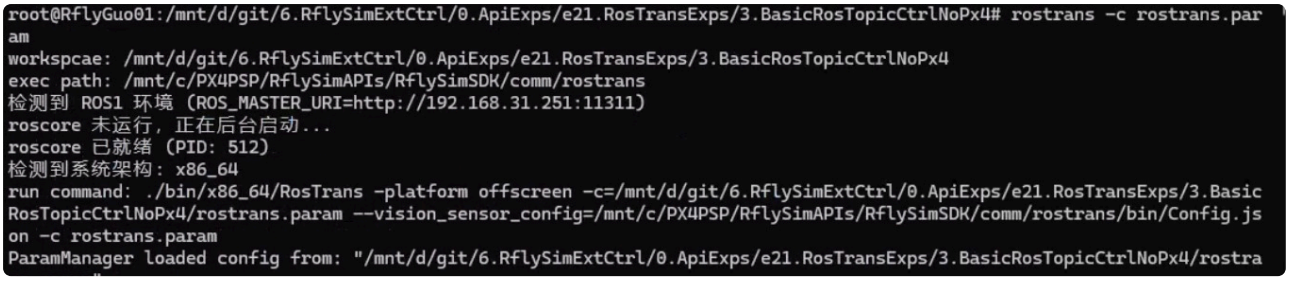
4.1 启动仿真环境

双击运行 `CopterSimNOpx4.bat` 文件，等待仿真环境初始化完成。脚本将会启动 1 个 CopterSim、1 个 RflySim3D 软件，查看CopterSim软件，将显示使用DLL模型文件为 MulticopterNOpx4，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。如下图所示：

4.2 启动 RosTrans 中间件

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
```



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

- 连接 CopterSim（DLL Model 模式），接收遥测数据并发布到 ROS 主题
- 创建 MAVROS 兼容的服务端点（set_mode, arming, takeoff, land 等）
- 如已启用 `enable_ue3d=true`，还将桥接 UE4 显示控制通道

`rostrans.param` 关键配置说明：

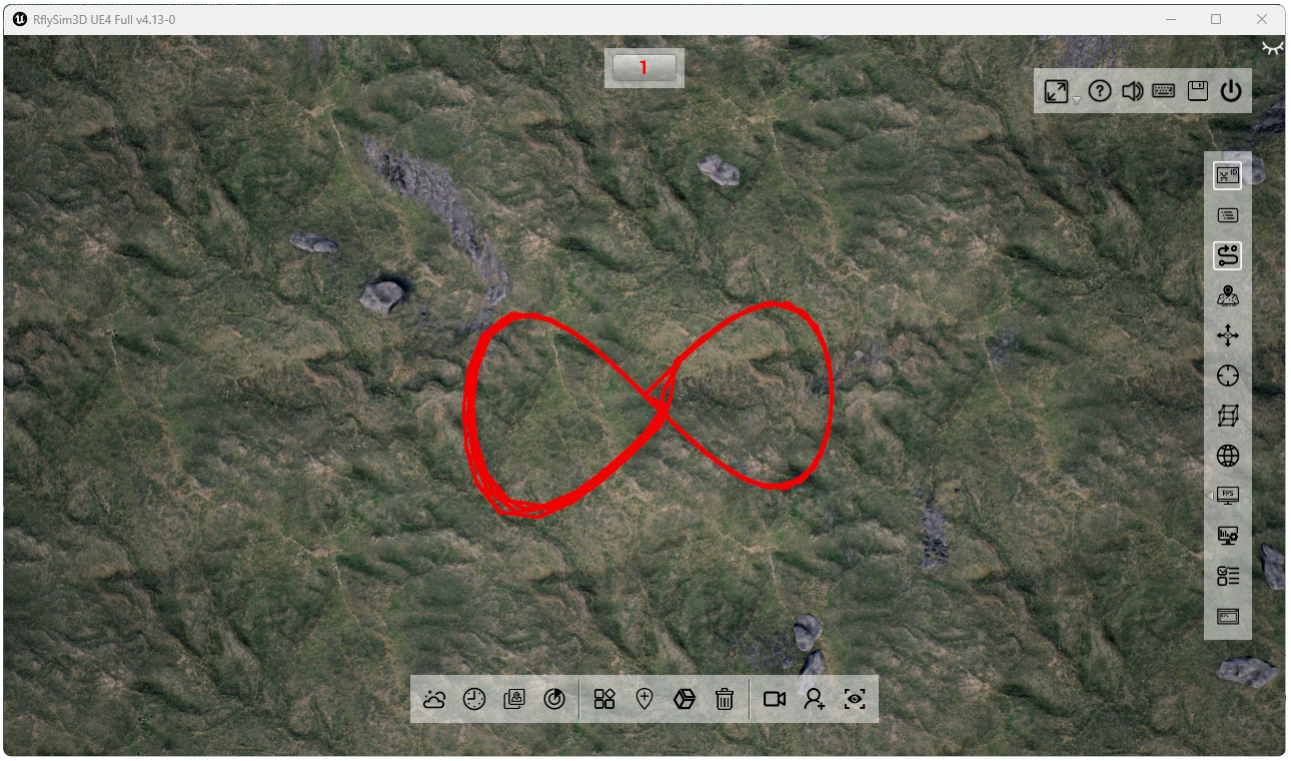
```
enable_dllModel=true          # 启用 DLL 综合模型（无 PX4）
coptersim_count=1             # 仿真飞机数量
coptersim_targetip=127.0.0.1  # CopterSim IP 地址
enable_ue3d=true              # 启用 UE4 3D 显示控制
```

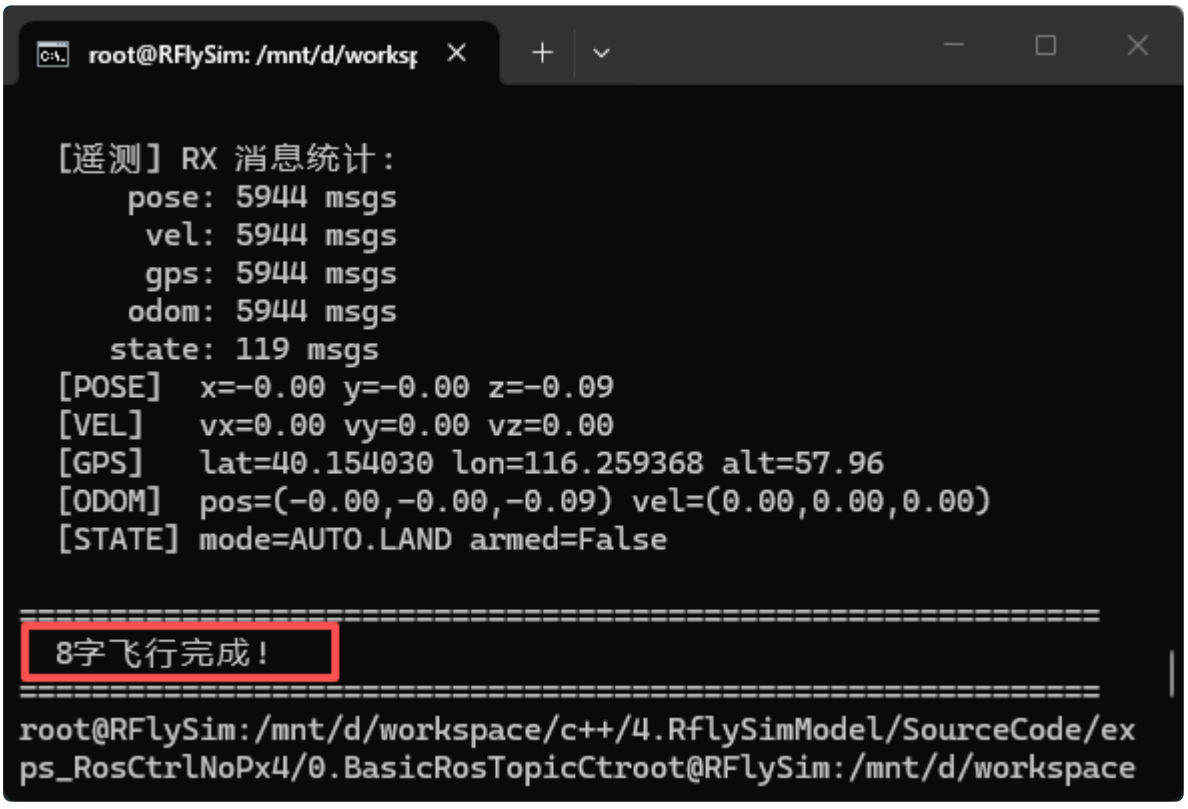
4.3 运行 8 字飞行例程

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（单机，高度10m，半径5m，2圈）
python3 dllModel_flight.py
```

可在RflySim3D中观察到无人机开始绕“8”字飞行。





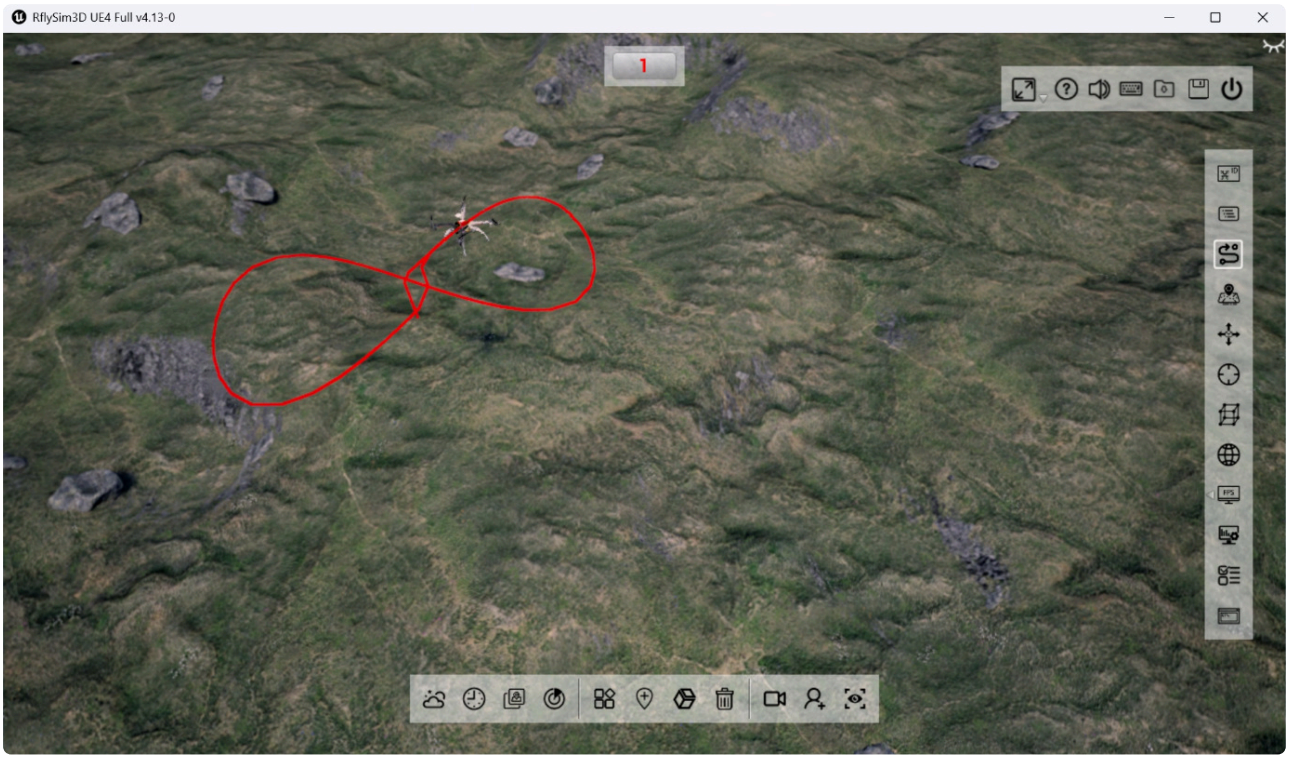
运行结束，该程序也支持传入参数的形式来，自定义设置，再次双击'WinWSL.bat'脚本，在弹出的对话框中运行如下代码：

```
# 自定义参数
python3 dllModel_flight.py --alt 8 --radius 6 --laps 3
```

则表示执行 3 圈 8 字飞行，高度为 10m，半径为 6m。

也可再次双击'WinWSL.bat'脚本，在弹出的对话框中运行如下代码：

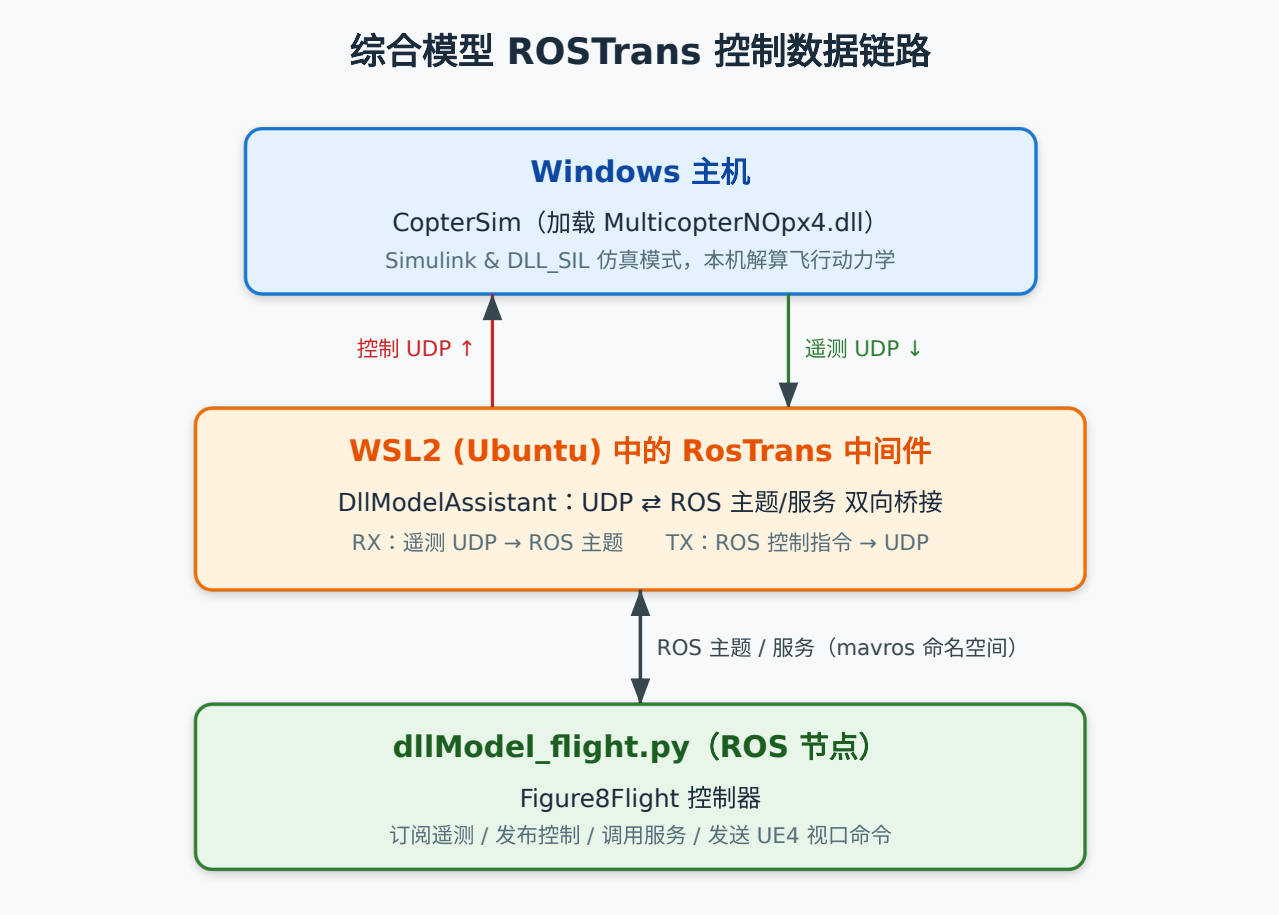
```
# 指定飞机 ID 和 UE4 窗口
python3 dllModel_flight.py --id 1 --win 0
```



5. 关键知识点

5.0 实验整体思路与框架

本实验的核心是把“CopterSim 综合模型”这一无 PX4 的仿真对象，包装成与真实 MAVROS 一致的 ROS 接口。整条数据链路如下：



整个例程 `dllModel_flight.py` 围绕一个 `Figure8Flight` 控制器类组织，其设计思路可归纳为四点：

1. **接口全覆盖**：一次飞行任务中遍历 DllModelAssistant 提供的全部 5 个 RX 主题、6 个 TX 主题与 5 个服务，便于学习者逐一对照接口含义；
2. **多控制模式混合**：把一个完整的“8 字”轨迹切成 4 段，每段使用一种不同的 TX 控制接口（位置 / 位姿 / 带时间戳速度 / 无时间戳速度），最后再单独演示两种姿态通道；
3. **后台周期发布**：用独立线程以约 20 Hz 的频率持续发布当前 setpoint，满足 OFFBOARD 模式“必须持续收到控制流”的要求；
4. **ROS1/ROS2 双兼容**：同一份代码通过 `ROS_VERSION` 环境变量自动选择 rospy 或 rclpy 的 API。

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

5.1 ROS 通信接口总览

RX 主题（CopterSim → ROS，订阅）：

```
主题                                          消息类型         说明
--------------------------------------------- ------------- ------------------------
--
/mavros/local_position/pose                   PoseStamped     ENU 本地位姿（位置 + 四元
数）
/mavros/local_position/velocity_local         TwistStamped    ENU 线速度 + 角速度
/mavros/global_position/global                NavSatFix       GPS 经纬度高度
/mavros/local_position/odom                   Odometry        完整里程计（位姿 + 速度）
/mavros/state                                 State           飞控状态（armed / mode）
```

TX 主题（ROS → CopterSim，发布）：

```
主题                                          消息类型         说明
--------------------------------------------- ------------- ------------------------
--
/mavros/setpoint_raw/local                    PositionTarget 位置/速度/加速度多模式控制
/mavros/setpoint_position/local               PoseStamped     位置 + 四元数控制
/mavros/setpoint_velocity/cmd_vel             TwistStamped    带时间戳速度控制
/mavros/setpoint_velocity/cmd_vel_unstamped   Twist           无时间戳速度控制
/mavros/setpoint_attitude/cmd_vel             TwistStamped    体轴角速度控制
/mavros/setpoint_raw/attitude                 AttitudeTarget 姿态目标（四元数 + 推力）
```

服务：

```
服务                        类型            说明
--------------------------- -------------- ----------------------------------
/mavros/param/set           ParamSet        设置飞控参数
/mavros/set_mode            SetMode         切换飞行模式（OFFBOARD / AUTO.LAND）
/mavros/cmd/arming          CommandBool     解锁 / 上锁
/mavros/cmd/takeoff         CommandTOL      起飞到指定高度
/mavros/cmd/land            CommandTOL      降落
```

UE4 视口控制（通过 `rflysim_msgs/UE4Cmd`）：

```
主题                            说明
------------------------------ --------------------------------
/rflysim/win0/ue4_cmd           向 UE4 渲染窗口发送显示控制命令
```

5.2 多机命名空间规则

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

5.3 标准飞行操作序列

```
start_setpoint()          # 启动 20Hz setpoint 发布线程
  → param_set(NAV_RCL_ACT, 0)  # 禁用遥控器失联保护
  → set_mode(OFFBOARD)         # 切换到 OFFBOARD 模式
  → arming(True)               # 解锁
  → takeoff(alt)               # 起飞
  → [航点飞行 / 控制测试]
  → set_target(0, 0, 0)        # 逐渐下降
stop_setpoint()
  → set_mode(AUTO.LAND)        # 切换降落模式
  → land()                     # 降落服务
  → arming(False)              # 上锁
```

5.4 程序整体框架与初始化

例程的全部逻辑封装在 `Figure8Flight` 类中。构造函数根据飞机 ID 推导命名空间，缓存遥测数据并初始化控制状态，最后调用 `_init_ros()` 完成 ROS 环境搭建：

```python
class Figure8Flight:
    def __init__(self, copter_id=1, altitude=10.0, radius=5.0, laps=1, win_id=0):
        self.cid = copter_id
        self.ns = "mavros" if copter_id == 1 else f"mavros{copter_id}"   # 多机命名空间
        self.alt = altitude
        self.radius = radius
        self.laps = laps
        self.win_id = win_id
        # RX 数据缓存 + 计数
        self.rx_counts = {"pose": 0, "vel": 0, "gps": 0, "odom": 0, "state": 0}
        # 控制状态：当前模式 / 目标值 / 偏航 / 推力
        self._running = False
        self._mode = "position"
        self._tgt = [0.0, 0.0, 0.0]
        self._init_ros()
```

要点解析：

- `self.ns` 决定了所有主题/服务的前缀，飞机 1 为 `mavros`，飞机 N 为 `mavrosN`，与 5.2 的命名空间规则一致；
- `self._mode` 与 `self._tgt` 是“控制意图”的中心状态变量，后台发布线程会根据它们决定发布哪种消息、填入什么数值；
- `rx_counts` 用于统计每个遥测主题收到的消息数，是排查“收不到数据”问题（见第 7 节 Q1）的关键依据。

5.5 ROS1 / ROS2 双版本兼容设计

程序在文件顶部根据环境变量 `ROS_VERSION` 判断运行在 ROS1 还是 ROS2 上，并据此导入不同的库：

```
ros_ver = os.getenv("ROS_VERSION", "1")
is_ros1 = (ros_ver == "1")

if is_ros1:
    import rospy
    from mavros_msgs.msg import PositionTarget, State, AttitudeTarget, ParamValue
    from mavros_msgs.srv import CommandBool, CommandTOL, SetMode, ParamSet
else:
    import rclpy
    from rclpy.qos import QoSProfile, ReliabilityPolicy, DurabilityPolicy,
HistoryPolicy
    ...
```

`_init_ros()` 中两套初始化路径的关键差异在于 **ROS2 需要显式配置 QoS 并等待 DDS 发现**：

```
def _sub_ros2(self):
    qos = QoSProfile(reliability=ReliabilityPolicy.BEST_EFFORT,   # 与发布端匹配
                     durability=DurabilityPolicy.VOLATILE,
                     history=HistoryPolicy.KEEP_LAST, depth=10)
    n = self._node
    n.create_subscription(PoseStamped, f"/{self.ns}/local_position/pose",
self._cb_pose, qos)
    ...
```

```
# ROS2 初始化结尾
self._exec = rclpy.executors.MultiThreadedExecutor()
self._exec.add_node(self._node)
threading.Thread(target=self._exec.spin, daemon=True).start()
...
time.sleep(5)   # 等待 DDS discovery 完成，否则首次服务调用会失败
```

这里的两个细节非常重要：遥测主题在 ROS2 下必须使用 `BEST_EFFORT` 可靠性策略才能与 RosTrans 的发布端匹配（否则订阅不到数据，对应 Q1）；而 DDS 节点发现需要数秒，因此初始化末尾固定 `sleep(5)`，避免起飞阶段服务调用超时（对应 Q2）。

5.6 Setpoint 周期发布线程

OFFBOARD 模式要求地面端持续不断地发送控制流。程序用一个守护线程以 20 Hz（`sleep(0.05)`）的频率循环发布，发布哪种消息完全由当前 `self._mode` 决定：

```
def _sp_loop(self):
    while self._running and not _interrupted:
        m = self._mode
        if   m == "position":      self._pub_position_target()
        elif m == "pose":          self._pub_pose_stamped()
        elif m == "vel_stamped":   self._pub_vel_stamped()
        elif m == "vel_unstamped": self._pub_vel_unstamped()
        elif m == "att_rate":      self._pub_att_rate()
        elif m == "att_target":    self._pub_att_target()
        time.sleep(0.05)           # 20 Hz

def start_sp(self):
    self._running = True
    self._sp_thread = threading.Thread(target=self._sp_loop, daemon=True)
    self._sp_thread.start()
```

这种“**状态机 + 后台线程**”的设计把“发什么”与“怎么发”解耦：主流程只需调用 `set_pos / set_vel_s / set_att_rate` 等便捷方法修改 `self._mode` 和 `self._tgt`，发布线程会立刻切换到对应接口持续发送，无需主流程关心发布频率。

5.7 六种 TX 控制接口的实现

发布线程对应的六个 `_pub_*` 方法，正是对 5.1 中 6 个 TX 主题的逐一封装。以最常用的 `setpoint_raw/local`（位置模式）为例：

```
def _pub_position_target(self):
    """TX: setpoint_raw/local (PositionTarget) — 位置模式"""
    msg = PositionTarget()
    msg.header.stamp = self._now(); msg.header.frame_id = "map"
    msg.coordinate_frame = PositionTarget.FRAME_LOCAL_NED
    msg.type_mask = (PositionTarget.IGNORE_VX | PositionTarget.IGNORE_VY |
                     PositionTarget.IGNORE_VZ | PositionTarget.IGNORE_AFX |
                     PositionTarget.IGNORE_AFY | PositionTarget.IGNORE_AFZ |
                     PositionTarget.IGNORE_YAW_RATE)
    msg.position.x, msg.position.y, msg.position.z = (float(v) for v in self._tgt)
    msg.yaw = self._yaw
    self.pub_raw.publish(msg)
```

`type_mask` 是 `PositionTarget` 的精髓：通过 `IGNORE_*` 位掩码忽略速度/加速度/偏航角速率分量，只保留位置与偏航，飞控即按“纯位置模式”跟踪。若改为忽略位置而保留速

度，则切换为速度模式。

姿态目标接口则用四元数表达期望姿态，并显式给出推力：

```
def _pub_att_target(self):
    """TX: setpoint_raw/attitude (AttitudeTarget)"""
    msg = AttitudeTarget()
    msg.header.stamp = self._now()
    msg.type_mask = self._att_mask              # mask=128 表示忽略姿态、仅用
body_rate
    msg.body_rate.x, msg.body_rate.y, msg.body_rate.z = (float(v) for v in
self._tgt)
    msg.thrust = self._thrust
    msg.orientation.z = math.sin(self._yaw / 2.0)   # yaw → 四元数（绕 Z）
    msg.orientation.w = math.cos(self._yaw / 2.0)
    self.pub_att_t.publish(msg)
```

注意偏航角到四元数的转换：仅绕 Z 轴旋转时，`qz = sin(yaw/2)`、`qw = cos(yaw/2)`，其余分量为 0。这一构造在 `setpoint_position/local` 的 `_pub_pose_stamped` 中同样出现。

5.8 “8 字”轨迹的数学生成

8 字轨迹采用 Lissajous 型参数方程，位置、速度与切向偏航分别由三个方法生成：

```
def _fig8_pos(self, t):
    """8字轨迹点: x = R*sin(t), y = R*sin(t)*cos(t)"""
    R = self.radius
    x = R * math.sin(t)
    y = R * math.sin(t) * math.cos(t)
    return x, y

def _fig8_vel(self, t, omega=1.0):
    """8字轨迹速度: dx/dt, dy/dt"""
    R = self.radius
    vx = R * omega * math.cos(t)
    vy = R * omega * (math.cos(2 * t))   # d/dt[sin(t)cos(t)] = cos(2t)
    return vx, vy

def _fig8_yaw(self, t, omega=1.0):
    """沿轨迹切线方向的 yaw 角"""
    vx, vy = self._fig8_vel(t, omega)
    return math.atan2(vy, vx)
```

关键点：`y = R·sin(t)·cos(t)` 即半幅度的 `sin(2t)`，与 `x = R·sin(t)` 组合后形成横置的“8”字（双纽线近似）；速度由位置对时间求导得到，其中

`d/dt[sin·cos] = cos(2t)`；偏航角用 `atan2(vy, vx)` 取轨迹切线方向，使机头始终朝向运动方向。位置模式段直接用 `_fig8_pos`，速度模式段则用 `_fig8_vel`，体现了同一轨迹可由不同控制接口实现。

5.9 服务调用的封装

5 个 MAVROS 服务被封装为 `srv_*` 方法，每个方法内部同样区分 ROS1/ROS2。以 `set_mode` 为例：

```
def srv_set_mode(self, mode):
    print(f"  [SRV] set_mode → {mode}")
    if is_ros1:
        rospy.wait_for_service(f"/{self.ns}/set_mode", 3)
        return rospy.ServiceProxy(f"/{self.ns}/set_mode", SetMode)(0,
mode).mode_sent
    req = SetMode.Request(); req.custom_mode = mode
    return self._call_srv_ros2(self.cli_mode, req)
```

ROS2 的异步服务调用统一通过 `_call_srv_ros2` 处理，它先等待服务可用，再轮询 future 直到完成或超时：

```
def _call_srv_ros2(self, cli, req, timeout=5.0):
    if not cli.wait_for_service(timeout_sec=timeout): return False
    f = cli.call_async(req)
    end = time.time() + timeout
    while not f.done() and time.time() < end: time.sleep(0.05)
    return f.result() is not None
```

这些服务在飞行流程中的调用顺序即 5.3 给出的标准操作序列：`param_set(NAV_RCL_ACT,0)`（关闭遥控失联保护）→ `set_mode(OFFBOARD)` → `arming(True)` → `takeoff` → … → `set_mode(AUTO.LAND)` → `land` → `arming(False)`。

5.10 UE4 视口控制

通过 `rflysim_msgs/UE4Cmd` 主题，可向 RflySim3D 渲染窗口发送字符串命令。`sendUE4Cmd` 负责封装并发布命令，`setup_ue4_viewport` 则组合多条命令完成俯视相机 + 编号 + 轨迹的配置：

```python
def sendUE4Cmd(self, cmd_str):
    if not HAS_UE4_MSG or not self.pub_ue4_cmd:
        return
    msg = UE4Cmd()
    msg.checksum = 1234567890
    msg.copter_id = self.cid
    msg.cmd = cmd_str
    self.pub_ue4_cmd.publish(msg)

def setup_ue4_viewport(self):
    self.sendUE4Cmd('RflyChangeViewKeyCmd N 6')          # N 模式(地球静止)第6视角—俯视
    self.sendUE4Cmd('RflyCameraFovDegrees 120')          # 视场角 120°
    self.sendUE4Cmd('RflyCameraPosAng %.2f %.2f %.2f %.2f %.2f %.2f'
                    % (0, 0, -23, 0, -90, 0))            # 相机位置(北东地)+角度(俯
视-90°)
    self.sendUE4Cmd('RflyChangeViewKeyCmd S')            # 显示无人机编号
    self.sendUE4Cmd('RflyChangeViewKeyCmd T 8')          # 显示飞行轨迹(模式8)
```

`UE4Cmd` 消息的三个字段含义：`checksum` 为固定校验值，`copter_id` 指定目标飞机，`cmd` 为 RflySim3D 支持的控制台命令字符串。相机位置使用北东地（NED）坐标、单位米，俯仰角 `-90°` 即垂直向下俯视。注意若 `rflysim_msgs` 包未安装，`HAS_UE4_MSG` 为 `False`，相关调用会被安全跳过（对应 Q3）。

5.11 主飞行流程 run() 的五个阶段

`run()` 把上述构件串成一次完整任务，分五个阶段执行：

```python
def run(self):
    # Phase 1: 等待遥测 + 配置 UE4 视口
    sleep_ok(3); self.print_telemetry(); self.setup_ue4_viewport()

    # Phase 2: 服务配置 + 解锁起飞
    self.set_pos(0, 0, 0); self.start_sp(); sleep_ok(1)
    self.srv_param_set("NAV_RCL_ACT", 0.0)
    self.srv_set_mode("OFFBOARD"); self.srv_arming(True);
self.srv_takeoff(self.alt)
    self.set_pos(0, 0, self.alt); sleep_ok(5)

    # Phase 3: 8字飞行(每圈4段, 各用一种TX接口)
    for lap in range(self.laps):
        # 段1 set_pos / 段2 set_pose / 段3 set_vel_s / 段4 set_vel_u
        ...

    # Phase 4: 姿态通道演示(att_rate 与 att_target 各旋转2s)
    # Phase 5: 逐步下降 → AUTO.LAND → land → disarm
```

第 3 阶段是核心：每一圈被等分为四个 1/4 弧段，依次切换到位置目标、位姿、带时间戳速度、无时间戳速度四种接口，从而在一条连续轨迹上演示不同控制模式的等效效果。第 5 阶段先以位置模式逐步降高，再切 `AUTO.LAND` 调用降落服务并上锁，最后补发一段 setpoint 以确保编码器输出 `CmdArmed=0` 的 UDP 包，使 CopterSim 侧正确进入上锁状态。

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 Python 脚本后没有收到任何遥测数据（所有 RX 计数为 0）

A1：请检查以下几点：

- 确认 CopterSim 已启动并检查DLL模型文件与仿真模式是否符合预期
- 确认 RosTrans 已启动并加载了正确的 `rostrans.param`
- 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：服务调用超时（set_mode/arming 返回 False）

A2：服务端由 RosTrans 的 DllModelAssistant 提供。请确认：

- RosTrans 配置中 `enable_dllModel=true`

- RosTrans 正在运行且未报错
- ROS2 环境下 DDS discovery 需要约 3-5 秒，脚本已内置等待

Q3：UE4 视口命令无效（无人机编号/轨迹未显示）

A3：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）。若未安装，程序中 `HAS_UE4_MSG` 为 `False`，会打印“rflysim_msgs 未安装, 跳过视口配置”并继续飞行，不影响控制本身

---

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