第二十八届中国机器人及人工智能大赛"空地协同赛道" - 仿真操作文档

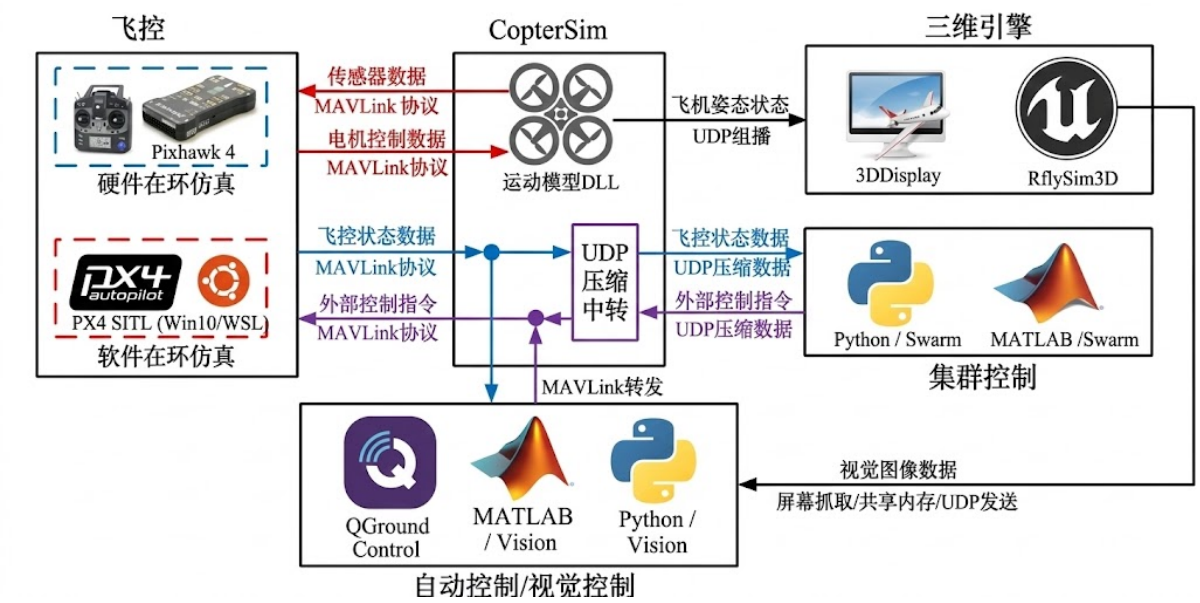
> **注**：本文档为"空地协同赛道"的官方仿真技术操作手册，专注 RflySim 仿真环境的部署、运行与接口调用。关于比赛规则与评分机制，请参阅赛事综合说明。

> **文档版本**: v2.2 | **最后更新**: 2026年3月11日 | **维护者**: 罗旭 (qq2179391312)

1. 📐 系统架构与代码库总览

1.1 仿真架构简述

RflySim是一套专为科研和教育打造的Pixhawk /PX4和MATLAB/Simulink生态系统或工具链，采用基于模型设计（Model-Based Design，MBD）的思想，可用于无人系统的控制和安全测试，RflySim工具链目前公开免费的版本已经能够满足绝大多数的无人系统开发功能，可便于选手构建稳定、可扩展的空地协同算法。



1.2 开源代码库结构

解压官方提供的 `28com_demo` 项目包后，核心目录结构及作用如下：

```
./
├── 28com_SITL/                    # 软件在环(SITL)平台启动脚本
│   ├── CARSITL.bat                # 无人车仿真平台一键启动脚本
│   ├── CARSITL.py                 # 无人车场景渲染脚本
│   ├── UAVSITL.bat                # 无人机仿真平台一键启动脚本
│   └── UAVSITL.py                 # 无人机场景渲染脚本
├── 28com_UAV/                     # 无人机比赛基础例程
│   ├── Config.json                # 传感器配置文件
│   ├── StartSLAM.sh               # 启动激光 SLAM 节点
│   ├── WinWSL.bat                 # 进入 WSL 终端快捷脚本
│   ├── WinWSLRunCV.bat            # 视觉控制例程一键运行脚本
│   ├── WinWSLRunCtrl.bat          # 速度控制例程一键运行脚本
│   ├── WinWSLRunoffboard.bat      # 位置控制例程一键运行脚本
│   ├── main.py                    # 传感器数据采集与 ROS 话题发布节点
│   ├── offboard.py                # 位置控制例程
│   ├── testRflyRosCV.py           # 视觉感知与控制融合例程
│   ├── testRflyRosCtrl.py         # 速度控制例程
│   └── tf_cfg.yaml
└── 28com_UGV/                     # 无人车比赛基础例程
    ├── Config.json
    ├── WinWSL.bat
    ├── WinWSLRunCtrl.bat          # 无人车控制例程一键运行脚本
    ├── control.py                 # 无人车运动控制例程
    ├── main.py
    └── tf_cfg.yaml
```

↑返回目录

2. ⚙️ 环境依赖与基础配置

> **注**：阅读本文档时，默认已成功安装 RflySim 仿真平台工具链。如果尚未安装，请先查阅官方提供安装包中的`HowToInstall.pdf`完成基础部署。

2.1 算法开发环境 (WinWSL / Ubuntu)

RflySim 平台高度集成了跨平台通信与软硬件在环能力。针对本次“空地协同赛道”，官方已为开发者准备了开箱即用的底层运行环境。

- **平台内置 WinWSL 环境（官方推荐）**：
  平台自带的 WinWSL (Ubuntu 22.04) 环境已经预先配置好了 **ROS1/ROS2、Mavros和opencv** 等所有比赛必须的核心依赖，可直接作为 Ubuntu 虚拟机运行平台的所有例程。
  - **快捷启动**：在运行例程时，只需双击项目目录（如 `28com_UAV` 或 `28com_UGV`）下的 `WinWSL.bat` 脚本，即可一键进入 Ubuntu 终端环境。
  - **右键启动**：也可以进入任意例程文件夹，鼠标右键点击“Open in Terminal”（在终端打开），在弹出的 Windows 终端中输入 `wsl` 或 `bash` 进入。
  - **环境验证**：成功进入 WSL 环境后，命令行头部将显示类似 `root@用户名:文件目录` 的格式。
- **自定义物理/硬件环境（进阶开发）**：
  本比赛例程除了支持使用平台内置的 WSL 环境外，同样支持选手将算法部署到真实的物理环境中进行联机运行：
  - 独立的 Ubuntu 双系统或 Ubuntu 虚拟机。
  - 第二台处于同一局域网下的 Ubuntu 电脑。
  - 真实的机载计算机（如 NUC、Nvidia Jetson NX 等板卡）。
    (注：如需在外部自定义环境中开发，请参考官方提供的依赖清单，自行配置对应的 ROS、Mavros 及 Python 算法库。)

↑返回目录

---

3. 🚀 仿真例程运行指南

本节旨在引导选手快速跑通比赛基础例程，验证平台中相机、激光雷达等感知链路的端到端数据流（采集 -> ROS 发布 -> SLAM -> 飞控融合），以及从算法侧到飞控端的运动控制闭环。

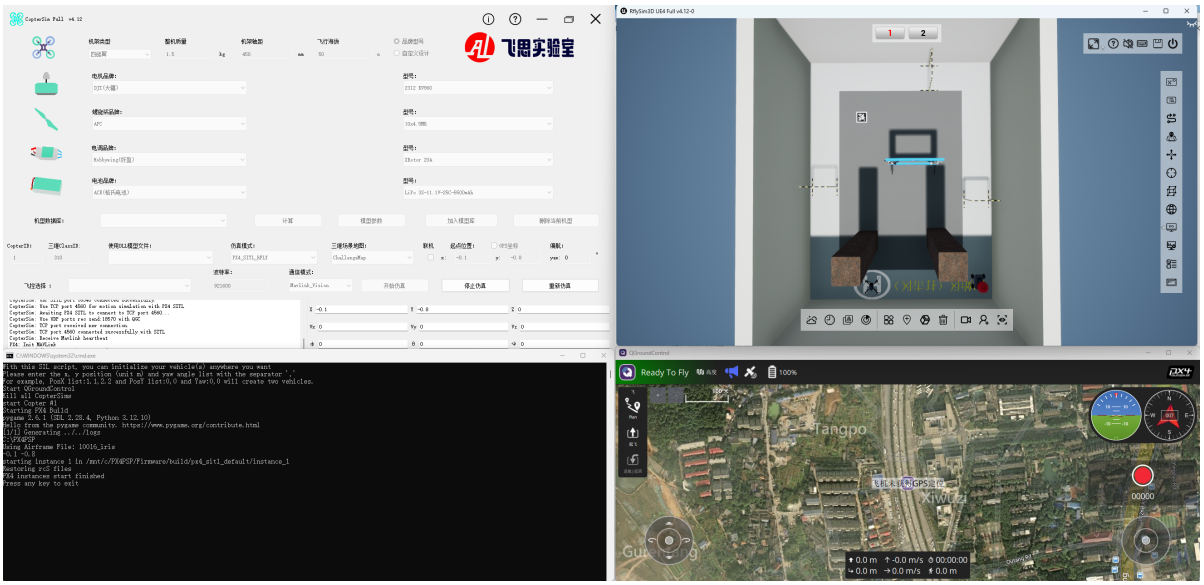
> 💡 **运行前提示**：
>
> - 请确保在运行前**关闭所有 VPN 和杀毒软件**，以免拦截本地 UDP 通信端口。
> - 若仿真运行中飞行器/车辆出现异常抖动或翻滚，通常为电脑 CPU/内存性能瓶颈，建议使用性能更佳的台式机或调低渲染画质。

3.1 🚁 无人机仿真例程

Step 1: 启动底层物理仿真环境

在 Windows 系统下，进入 `28com_SITL` 文件夹，双击运行 `UAVSITL.bat` 脚本。 启动成功后，系统将弹出以下关键窗口，如下图所示：

1. **QGroundControl (QGC) 地面站。**
2. **CopterSim 物理引擎**：请务必检查其下方日志栏，必须打印出 `GPS 3D fixed & EKF initialization finished` 字样，代表初始化完成。
3. **RflySim3D 渲染窗口**：此时应能看到一架无人机，以及一个顶着红色气球的无人车处于比赛模拟场景中。



注意：场景加载完毕后，无人车会按照比赛要求运动最后在无人机打击区停下来，这个属于环境加载代码的一部分，仅作为模拟还原比赛真实效果，并非代码控制的无人车运动。

Step 2: 运行控制代码

方法一：一键运行

在 Windows 环境下直接双击 `WinWSLRunCtrl.bat` 、`WinWSLRunCV.bat` 或 `WinWSLRunoffboard.bat` 脚本即可一键运行。系统将自动按序拉起数据采集、SLAM 定位与控制节点，可以将直接看到无人机起飞并按照代码设定进行运动。

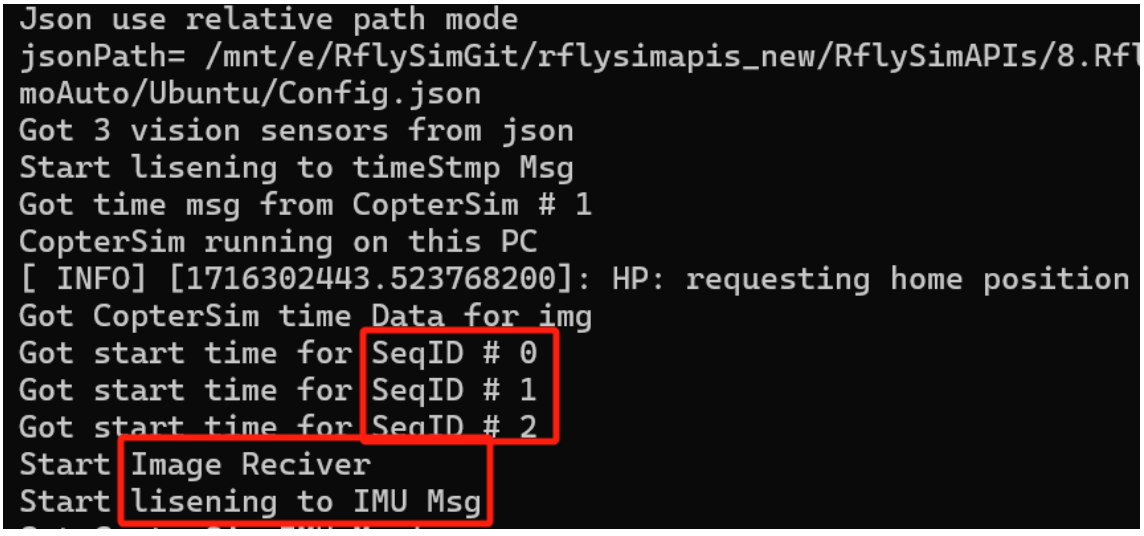
方法二：手动分步运行

对于第一次使用的参赛选手，我们建议先手动分布启动各个节点，以便清晰地查验每一个环节的数据流。具体步骤如下：

1. **启动中间件与传感器 ROS 发布节点**： 双击 `WinWSL.bat` 进入 Ubuntu 终端，执行以下命令开启 Mavros 并抓取图像/点云数据：

   ```
   python3 main.py
   ```

   现象：等待终端打印出图像和 IMU 消息发出的提示，说明传感器链路已通。

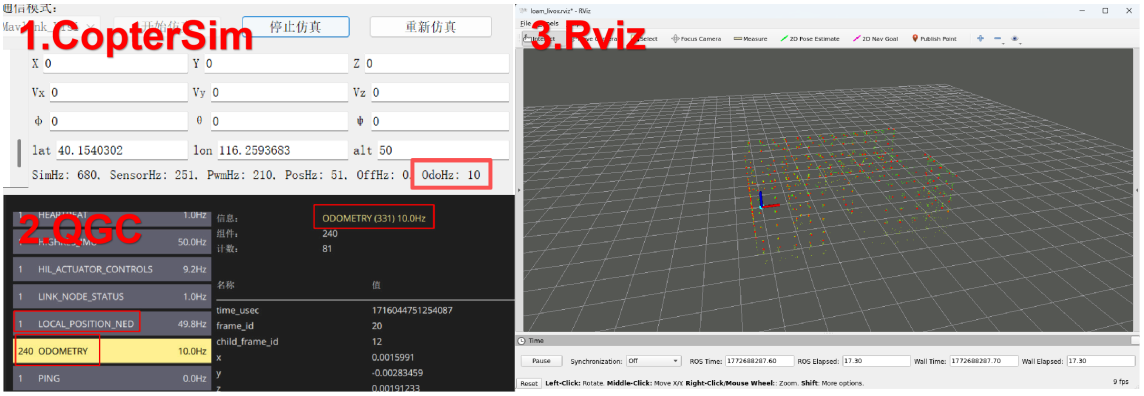
2. **启动激光 SLAM 定位节点**： 在当前目录再次双击 `WinWSL.bat` （或在例程文件夹中鼠标右键点击 `Open in Terminal`，再输入`wsl`），执行：

```
./StartSLAM.sh
```

现象：1.QGC 地面站的"MAVLink检测"页面会出现 `Odometry` 和 `Local_Position_NED` 数据；

2.CopterSim 左下角显示视觉里程计频率约为 10Hz；

3.同时会弹出 RViz 窗口显示建成的 3D 点云地图。



3. **下发控制指令**： 开启第三个 Ubuntu 终端，执行位置控制或速度控制脚本：
   - **位置控制测试**：

     ```
     python3 offboard.py
     ```

     现象：终端提示进入 Offboard 模式并解锁。无人机起飞，运动到指定点。
   - **速度控制测试**：

     ```
     python3 testRflyRosCtrl.py
     ```

     现象：终端提示进入 Offboard 模式并解锁。无人机起飞，以设定速度运动。
   - **视觉测试**：

     ```
     python3 testRflyRosCV.py
     ```

     现象：终端提示进入 Offboard 模式并解锁。无人机起飞，获取并实时显示前下视摄像头图像。

3.2 🚗 无人车 仿真例程

无人车的开发逻辑与无人机完全一致，剥离了复杂的 SLAM 定位，专注于地面控制。

Step 1: 启动底层物理仿真环境

在 Windows 系统下，进入 `28com_SITL` 文件夹，双击运行 `CARSITL.bat` 脚本，加载无人车专属的车辆动力学模型与地面测试场景。

Step 2: 运行控制基线代码

方法一：一键运行

双击 `WinWSLRunCtrl.bat` 一键启动所有脚本，可以看到无人车进入offboard模式后，按照设定轨迹运动。

方法二：手动分步运行

进入 `28com_UGV` 目录，分两步运行：

1. **启动中间件与数据转发**： 双击 `WinWSL.bat` 进入 Ubuntu 终端，执行：

   ```
   python3 main.py
   ```

   现象：终端持续打印雷达与 IMU 数据发送成功的日志。

2. **下发车辆控制指令**： 开启新的 Ubuntu 终端，执行运动控制脚本：

   ```
   python3 control.py
   ```

   现象：终端打印 `Vehicle Armed` 和 `Offboard mode enabled`。此时可在 3D 场景中观察到无人车先向前移动一段时间，随后向右转弯并最终停止。

↑ 返回目录

---

4. 🔌 接口调用与二次开发指南

在跑通基线代码后，选手需基于官方例程开发自己的空地协同算法。本节详细说明了核心脚本的作用边界和数据接口字典

4.1 核心代码与接口调用

官方提供的仿真例程遵循了严格的模块化设计，涵盖了数据采集、建图定位、运动控制与视觉感知四大链路。以下将结合具体源码，详细阐述各模块的核心接口功能及二次开发规范。

1. 传感器模块（`main.py` & `Config.json`）

本模块主要负责连接底层物理仿真引擎，获取视觉及传感器数据，并将其转化为 ROS 话题进行发布。

- **传感器配置文件（`Config.json`）：**
  该文件定义了仿真平台加载的传感器类型与参数。若需调整相机分辨率或增删传感器，建议直接修改此文件，而无需更改底层读取逻辑，以下为 `Config.json` 核心字段的说明字典：

字段名	作用说明与配置规则
`SeqID`	视觉传感器序号（0, 1, 2...）。如果填 `0`，系统将自动递增排序。
`TypeID`	传感器类型。`1`：RGB，`2`：深度，`3`：灰度，`4`：分割，`5`：测距，`7`：深度转点云，`20-23`：激光雷达。
`TargetCopter`	传感器绑定的飞行器 ID。***注：免费版仅支持绑定 1 号飞机。***
`TargetMountType`	绑定方式。`0`：固定飞机几何中心，`1`：固定飞机底部中心，`2`：固定地面上，`3`：弱固定（姿态不随动），`4`：绑定其他视觉传感器。
`DataWidth/Height`	输出图像的像素分辨率（宽 / 高）。
`DataCheckFreq`	图像检查更新频率。图像的总延迟由“UE渲染刷新率 + 此检查频率”共同决定。

字段名	作用说明与配置规则
`SendProtocol`	传输协议数组，共 8 位： • **[0] 协议类型**: `0` 共享内存(限Win), `1` UDP(JPEG压缩/点云直传), `2` UDP无损图片, `3` UDP(PNG压缩)。 • **[1-4] 目标IP**: 默认填 `0` (代表127.0.0.1本机)。 • **[5] 端口号**: 图像回传端口，默认填 `0` (自动使用 9999+SeqID)。
`CameraFOV`	视觉传感器的视场角 (FOV)，可间接影响相机焦距效果。
`EularOrQuat`	姿态表示模式。 `0`: 使用欧拉角 (`SensorAngEular`)， `1`: 使用四元数 (`SensorAngQuat`)。
`SensorPosXYZ`	传感器的安装位置 `[x, y, z]`，相对于 `TargetMountType` 中心的偏移量，单位为米。
`SensorAngEular`	传感器的安装姿态 `[roll, pitch, yaw]`，采用欧拉角表示，单位为度。

- **数据抓取与通信接口 (`main.py`):**

```
# 1. 加载 Config.json 中的传感器配置文件参数
vis.jsonLoad(1)

# 2. 开启取图循环机制。执行该接口后，算法端可通过 vis.Img[i] 实时读取图像阵列数据
vis.startImgCap()

# 3. 发送数据回传请求。从目标飞机的 CopterSim 底层引擎读取 IMU 数据
# 默认回传频率为 200Hz，执行后自动开启数据监听
vis.sendImuReqCopterSim(StartCopterID, TargetIP)
```

2. 激光 SLAM 定位模块 (`StartSLAM.sh`)

该模块负责接收点云与 IMU 数据，运行定位算法并向飞控输出里程计。

```
# 启动 faster_lio 算法节点，并同步开启 rviz 可视化工具以显示 3D 点云地图
roslaunch faster_lio rflysim.launch rviz:=true
```

开发建议：

1、在比赛基础阶段，确保该脚本正常运行且 QGC 接收到 `Odometry` 即可，一般无需对该启动器做深度修改；

2、faster_lio算法包已集成在平台中，如果想要修改，可以通过 `rospack find faster_lio` 得到算法包所在路径，对此感兴趣的同学可以自行学习如何优化。

3. 运动控制实现逻辑

官方例程提供了两种级别的运动控制开发接口，参赛选手可根据自身的技术栈选择合适的开发路径：

方式一：基于 RflySim SDK 的高层 API 控制 (`testRflyRosCtrl.py`) [推荐]

该例程引用了平台高度封装的 `PX4MavCtrlV4ROS.py` 库，大幅简化了状态读取与模式切换的复杂度，极为适合开发基于视觉反馈的智能控制算法。

```
# 1. 实例化控制对象，绑定目标飞行器 ID
```

```
mav = PX4MavCtrl.PX4MavCtrler(CopterID=1)

# 2. 一键初始化：自动处理 MAVROS 繁琐的解锁与 Offboard 模式切换请求
mav.initOffboard()

# 3. 状态读取接口：支持直接获取底层 NED 坐标系下的位置、速度、欧拉角与角速度参数
print('PosE', mav.uavPosNED)
print('VelE', mav.uavVelNED)
print('Euler', mav.uavAngEular)

# 4. 运动控制接口解析：
# 4.1 发送期望位置点，NED地球坐标系
mav.SendPosNED(x,y,z,yaw)
# 4.2 发送速度，NED地球坐标系
mav.SendVelNED(vx,vy,vz,yawrate)
# 4.3 发送速度，FRD机体坐标系，通常而言，设定vx和yawrate即可
mav.SendVelFRD(vx,vy,vz,yawrate)
# 4.4 同时控制飞机位置和速度
mav.SendPosVelNED(PosE=[0,0,0],VelE=[0,0,0],yaw,yawrate)
```

方式二：基于原生 MAVROS 的控制（`offboard.py` / `control.py`）

该方式直接调用 ROS 的 Topic 和 Service，适合熟悉 ROS 架构的参赛选手。

```
# 1. 状态订阅：获取飞行器/车辆的当前连接状态与飞行模式
state_sub = rospy.Subscriber("mavros/state", State, callback = state_cb)

# 2. 模式与解锁服务：调用对应 Service 请求进入 OFFBOARD 模式并解锁设备
set_mode_client = rospy.ServiceProxy("mavros/set_mode", SetMode)
arming_client = rospy.ServiceProxy("mavros/cmd/arming", CommandBool)

# 3. 位置控制
# 3.1 无人机位置控制：持续发布 PoseStamped 消息至位置控制话题
local_pos_pub = rospy.Publisher("mavros/setpoint_position/local", PoseStamped,
queue_size=10)

# 3.2 无人车速度控制：持续发布 Twist 消息至速度控制话题
pub = rospy.Publisher("/mavros/setpoint_velocity/cmd_vel_unstamped", Twist,
queue_size=10)
```

4. 视觉图像订阅与实时显示（`testRflyRosCV.py`）

RflySim 平台将仿真相机图像以标准 ROS 话题的形式发布，选手可通过以下接口直接订阅并获取 OpenCV 图像帧。

```
# 前置摄像头回调：将 ROS Image 消息转换为 OpenCV BGR 格式并缓存
def front_image_callback(msg):
    ...
    cv_image = bridge.imgmsg_to_cv2(msg, desired_encoding="bgr8")
    # 在此处调用前视视觉算法（目标检测等）
    with lock:
        latest_image1 = cv_image.copy()  # copy() 防止后续帧覆盖当前缓存
    ...
```

```python
# 下视摄像头回调：同上，用于俯视识别（二维码、降落点等）
def down_image_callback(msg):
    ...
    cv_image2 = bridge.imgmsg_to_cv2(msg, desired_encoding="bgr8")
    # 在此处调用下视视觉算法
    with lock:
        latest_image2 = cv_image2.copy()
    ...

# 注册话题订阅，ROS 将以回调形式异步推送图像数据
rospy.Subscriber("/rflysim/sensor1/img_rgb", Image, front_image_callback)  # 前置摄像头
rospy.Subscriber("/rflysim/sensor2/img_rgb", Image, down_image_callback)   # 下视摄像头

# 在主循环中安全读取缓存帧并渲染（10Hz，性能不足时可注释 imshow 相关行）
while not rospy.is_shutdown():
    with lock:
        if latest_image1 is not None:
            cv2.imshow("Front_image", latest_image1)
        if latest_image2 is not None:
            cv2.imshow("Down_image", latest_image2)
    cv2.waitKey(1)
    rate.sleep()

cv2.destroyAllWindows()
```

4.2 核心 ROS 话题介绍与开发指南

在空地协同赛道中，参赛选手的算法节点需要与 RflySim 仿真底层进行高频的数据交互。官方环境已将底层的物理传感器数据与飞控状态映射为标准的 ROS 话题 (Topics) 和服务 (Services)。

1. 关键 ROS 话题说明

在编写订阅节点前，请务必确认话题对应的数据结构，以便在代码中正确导入相应的模块。

📡 `/mavros/state`

- **消息类型**: `mavros_msgs/State`
- **作用说明**: 设备状态。包含连接状态 (`connected`)、解锁状态 (`armed`) 以及当前模式 (`mode`，如是否处于 `OFFBOARD`)。

📍 `/mavros/local_position/pose`

- **消息类型**: `geometry_msgs/PoseStamped`
- **作用说明**: 局部位置反馈。实时获取飞行器/车辆在 ENU (东北天) 坐标系下的当前位置与姿态。

🗺️ `/mavros/odometry/out`

- **消息类型**: `nav_msgs/Odometry`
- **作用说明**: 里程计数据。由 SLAM 节点解算并融合后的定位数据，包含三维空间位姿与速度信息。

📷 `/rflysim/sensor1/img_rgb`

- **消息类型**: `sensor_msgs/Image`

- **作用说明**: RGB图像流。Rflysim平台输出的 RGB 原始图像数据，话题名由平台配置，可通过修改Config.json改变传感器类型。

🌀 `/rflysim/sensor0/mid360_lidar`

- **消息类型**: `sensor_msgs/PointCloud2`
- **作用说明**: 激光点云流。Rflysim平台输出的激光雷达数据，话题名由平台配置，可通过修改Config.json改变传感器类型。

🎯 `/mavros/setpoint_position/local`

- **消息类型**: `geometry_msgs/PoseStamped`
- **作用说明**: 位置控制指令。算法端向飞控下发的目标位置指令。

🏎️ `/mavros/setpoint_velocity/cmd_vel_unstamped`

- **消息类型**: `geometry_msgs/Twist`
- **作用说明**: 速度控制指令。算法端向底层下发的速度控制指令，在无人车底盘闭环控制中最常用。

2. 标准数据订阅代码协议 (Python)

为了确保参赛选手能够稳定获取并解析上述数据，以下提供两套标准订阅器 (Subscriber) 范例代码。

范例 A：订阅 MAVROS 状态与位置信息

开发运动控制算法时，必须实时获取设备状态以构建状态机。

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import rospy
from mavros_msgs.msg import State
from geometry_msgs.msg import PoseStamped

class MavrosStateMonitor:
    def __init__(self):
        self.current_state = State()
        self.current_pose = PoseStamped()
        # 1. 订阅设备状态话题
        rospy.Subscriber("/mavros/state", State, self.state_cb)
        # 2. 订阅局部位置话题
        rospy.Subscriber("/mavros/local_position/pose", PoseStamped,
self.pose_cb)
    def state_cb(self, msg):
        """
        状态话题回调函数：提取解锁状态与飞行模式
        实际开发中，可在此处加入状态机逻辑，如判断 msg.mode 是否为 "OFFBOARD"
        """
        self.current_state = msg
    def pose_cb(self, msg):
        """位置话题回调函数：提取 ENU 坐标系下的位置数据"""
        self.current_pose = msg
        x = msg.pose.position.x
        y = msg.pose.position.y
        z = msg.pose.position.z
```

范例 B：订阅视觉图像流并转换为 OpenCV 格式

获取图像并进行目标检测是协同任务的关键。请遵循以下 `CvBridge` 协议处理 ROS 图像与 OpenCV 图像的格式转换。若需同时接入前置（`sensor1`）和下视（`sensor2`）两路摄像头，可以通过线程锁保护跨线程共享的图像缓存。

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import rospy
import cv2
import threading
from sensor_msgs.msg import Image
from cv_bridge import CvBridge, CvBridgeError

bridge = CvBridge()
lock = threading.Lock()
latest_image1 = latest_image2 = None

def front_image_callback(msg):
    """前置摄像头回调：图像格式转换后缓存，供主循环渲染"""
    global latest_image1
    try:
        cv_image = bridge.imgmsg_to_cv2(msg, desired_encoding="bgr8")
        # ---- 在此处加入前视视觉算法（目标检测等）----
        with lock:
            latest_image1 = cv_image.copy()  # copy() 防止后续帧覆盖当前缓存
    except CvBridgeError as e:
        print(e)

def down_image_callback(msg):
    """下视摄像头回调：用于俯视识别（二维码、降落点等）"""
    global latest_image2
    try:
        cv_image2 = bridge.imgmsg_to_cv2(msg, desired_encoding="bgr8")
        # ---- 在此处加入下视视觉算法 ----
        with lock:
            latest_image2 = cv_image2.copy()
    except CvBridgeError as e:
        print(e)

# 注册话题订阅（下视摄像头也可在需要时再动态调用注册，以节省计算开销）
rospy.Subscriber("/rflysim/sensor1/img_rgb", Image, front_image_callback)  # 前置摄像头
rospy.Subscriber("/rflysim/sensor2/img_rgb", Image, down_image_callback)   # 下视摄像头

# 主循环中线程安全地读取缓存帧渲染（性能不足时可注释 imshow 相关行）
rate = rospy.Rate(10)
while not rospy.is_shutdown():
    with lock:
        if latest_image1 is not None: cv2.imshow("Front_image", latest_image1)
        if latest_image2 is not None: cv2.imshow("Down_image",  latest_image2)
    cv2.waitKey(1)
    rate.sleep()
```

↑返回目录

5. ❓ 常见问题

下面按故障现象给出排查要点，操作中只引用本例程文件名或命令提示：

1. **点击UAVSITL.bat和CARSITL.bat一键启动仿真平台后，Rflysim3D未加载出比赛的仿真场景或缺少部分场景**
   - 一键退出当前的平台程序，重新运行bat脚本；若依然存在场景加载问题，重启电脑后再运行启动脚本；
2. **话题没有出现或图像/点云为空**
   - 检查是否已运行 `main.py`。在 Ubuntu 环境或 WSL 终端中运行：

     ```
     python3 main.py
     ```

   - 确认 `Config.json` 中传感器索引正确，且在控制台没有网络/权限错误。
3. **SLAM 无法发布里程计（QGC 中看不到 Odometry）**
   - 启动 SLAM：运行 `StartSLAM.sh`，并检查终端日志中对点云和 IMU 的订阅是否正常。
   - 用 `rostopic hz /mavros/odometry/out` 检查频率（预期约 10Hz）。
4. **Rviz 界面白屏或不显示点云**
   - 先确认 `StartSLAM.sh` 已运行或手动载入 `rflysim.rviz`。
   - 如果白屏，尝试关闭后重启 rviz；在虚拟化环境下，需确保图形加速/桌面会话正常。
5. **无人机或无人车运行后抖动或不稳定**
   - 检查主机性能与资源分配（在虚拟机场景下，建议增加内存与 CPU 分配）。
   - 在使用 `testRflyRosCtrl.py`、`testRflyRosCV.py`、`offboard.py` 或 `control.py` 前，先确认 `main.py` 与 SLAM 已稳定输出话题并且飞控已连接。
6. **坐标系与控制方向不一致（控制命令与实际方向错位）**
   - 请注意：MAVROS（脚本中多数接口）使用 ENU，而 PX4 的 local_position 使用 NED。查看 `offboard.py` 或 `control.py` 中的坐标映射实现并按需修正。

↑返回目录