

---

# 1. 实验名称及目的

## 1.1 实验名称

基于Python的跨ROS版本传感器数据订阅与处理实验

## 1.2 实验目的

本实验旨在构建一个完整的Python传感器数据订阅处理系统，主要目标包括：

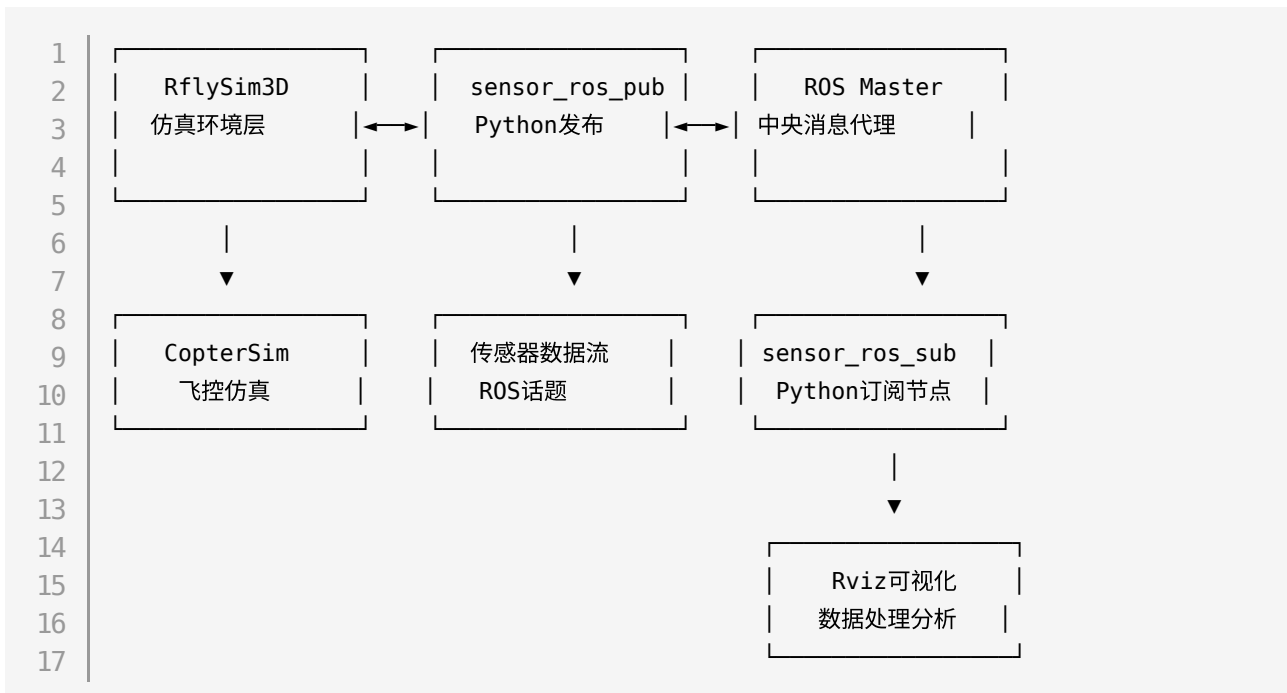
1. **分布式仿真架构**：通过Python接口从RflySim3D获取多种传感器数据并发布到ROS话题
2. **Python ROS订阅**：使用Python实现跨ROS版本的传感器数据订阅和处理
3. **多模态数据处理**：处理RGB图像、深度图像、激光雷达点云等多种传感器数据
4. **跨ROS版本兼容**：Python程序同时支持ROS1和ROS2的统一接口实现
5. **实时数据可视化**：使用Rviz和Python工具实现传感器数据的实时显示和分析
6. **数据格式转换**：掌握传感器数据在ROS消息格式和OpenCV格式间的转换技术

## 1.3 关键知识点

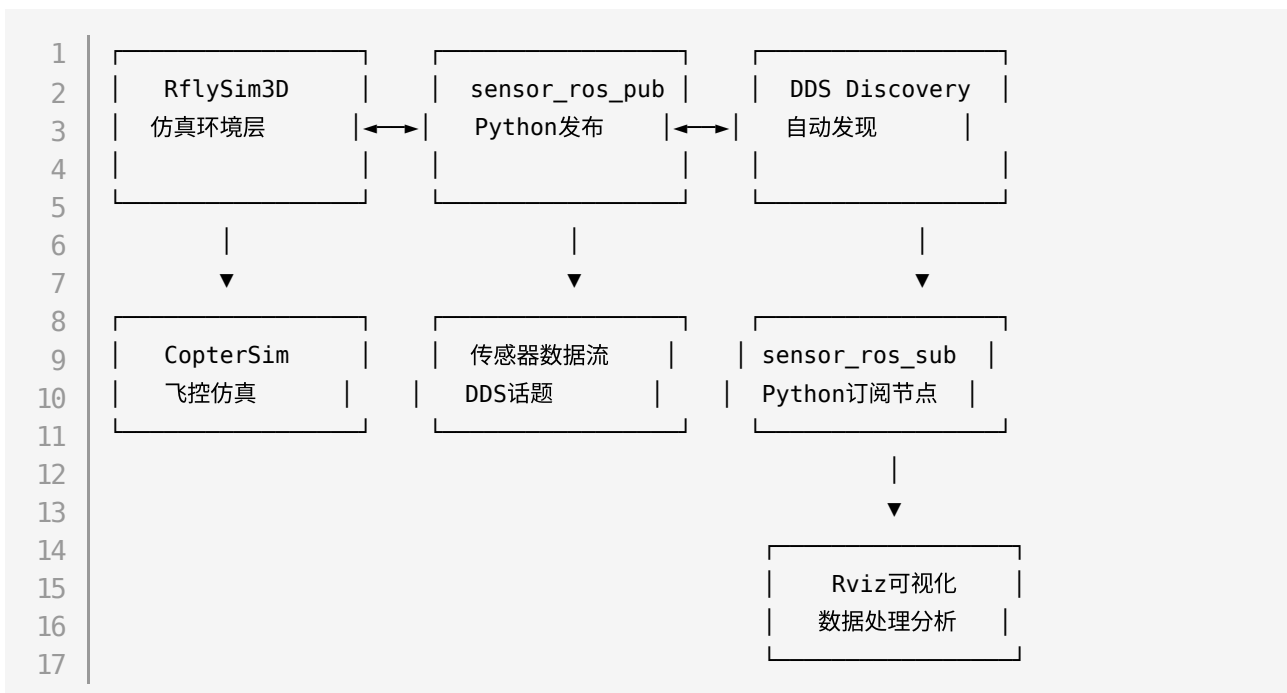
### 系统架构概述

本实验构建了一个多层次的分布式传感器数据处理架构，支持ROS1和ROS2两种不同的通信模式：

**ROS1架构（中央式）：**



## ROS2架构（分布式DDS）：



## 核心技术知识点

### 知识点0: Python传感器数据订阅架构

Python程序核心组件：

- **sensor\_ros\_sub.py**：主程序文件，实现跨ROS版本的传感器数据订阅
- **SensorRosSub类**：核心数据处理类，实现传感器数据的接收和处理
- **智能类型识别**：根据传感器TypeID自动判断数据类型和处理方式

## 核心类设计架构:

```
1 class SensorRosSub:
2     def __init__(self, vis: VisionCaptureApi.VisionCaptureApi):
3         # 传感器类型映射表
4         self.sensor_type = {
5             1: "img_rgb", 2: "img_gray", 3: "img_depth",
6             40: "img_Infrared_Gray", 41: "img_Infrared",
7             20: "vehicle_lidar", 21: "global_lidar",
8             22: "livox_lidar", 23: "mid360_lidar"
9         }
10
11         # 根据ROS版本初始化节点
12         if ros_version == "1":
13             rospy.init_node("rflsim_subscriber")
14         else:
15             rclpy.init()
16             self.node = rclpy.create_node("rflsim_subscriber")
17
18         def generic_callback(self, data, seq_id):
19             # 通用回调函数处理不同类型传感器数据
20             pass
21
22         def run(self):
23             # 运行订阅循环
24             pass
```

## 跨ROS版本兼容策略:

```
1 # ROS版本检测和适配
2 ros_version = os.getenv("ROS_VERSION")
3 if ros_version == "1":
4     import rospy
5     # ROS1订阅方式
6     rospy.Subscriber(topic_name, msg_type, callback)
7 else:
8     import rclpy
9     # ROS2订阅方式
10    node.create_subscription(msg_type, topic_name, callback, qos)
```

## 知识点1: 分布式网络通信架构

### 自动IP发现机制:

- 使用 `ReqCopterSim.py` 接口实现局域网内仿真节点的自动发现
- 支持本地 (127.0.0.1) 和远程 (192.x.x.x) 两种运行模式
- 通过UDP广播实现零配置网络连接

## 网络拓扑优势:

```
1 | # 自动获取仿真电脑IP, 实现网络透明
2 | req = ReqCopterSim.ReqCopterSim()
3 | TargetIP = req.getSimIpID(StartCopterID) # 自动发现目标IP
4 | req.sendReSimIP(CopterID) # 请求数据回传
```

## 注意事项:

- 多实验并行时可能产生IP冲突, 建议使用手动IP配置
- 需要确保防火墙允许相关端口通信

## 知识点2: 多协议传感器数据传输

### SendProtocol传输协议详解:

- `SendProtocol[0] = 0`: 共享内存模式 (仅Windows, 最高性能)
- `SendProtocol[0] = 1`: UDP + PNG压缩 (跨平台, 平衡性能)
- `SendProtocol[0] = 2`: UDP无压缩 (最低延迟, 高带宽需求)
- `SendProtocol[0] = 3`: UDP + JPEG压缩 (最低带宽, 适度质量损失)

## 传输性能对比:

协议模式	延迟	带宽占用	图像质量	跨平台性
共享内存	极低	无网络	无损	仅Windows
UDP+PNG	低	中等	无损	全平台
UDP原始	极低	极高	无损	全平台
UDP+JPEG	中等	极低	有损	全平台

## 激光雷达数据特殊处理:

- 仅支持共享内存 (0) 和UDP直传 (1) 两种模式
- 点云数据无法压缩, 建议使用千兆网络

## 知识点3: 跨ROS版本兼容性设计与架构差异

### ROS1 vs ROS2 核心架构对比:

特性	ROS1 (Noetic)	ROS2 (Humble)
通信架构	中央式 (ROS Master)	分布式 (DDS)

特性	ROS1 (Noetic)	ROS2 (Humble)
服务发现	依赖Master节点	自动P2P发现
消息传输	TCP/UDP	DDS (RTPS)
实时性	有限支持	原生实时支持
容错性	单点故障风险	高可用性
安全性	基础认证	内置安全机制

### 架构启动差异:

- **ROS1:** 必须先启动 `roscore` 作为消息代理和服务注册中心
- **ROS2:** 节点直接通过DDS自动发现和通信, 无需中央节点

### 编译时版本选择:

```

1 | #ifndef USE_ROS1
2 |     // ROS1 Noetic代码路径 - 需要NodeHandle
3 |     #include <ros/ros.h>
4 |     ros::NodeHandle nh; // 连接到ROS Master
5 | #else
6 |     // ROS2 Humble代码路径 - 直接创建节点
7 |     #include "rclcpp/rclcpp.hpp"
8 |     auto node = rclcpp::Node::make_shared("sensor_node"); // DDS自动发现
9 | #endif

```

### 运行时环境适配:

```

1 | ros_version = os.getenv("ROS_VERSION")
2 | if ros_version == "1":
3 |     import rospy
4 |     rospy.init_node("rflsim_subscriber") # 注册到ROS Master
5 | else:
6 |     import rclpy
7 |     rclpy.init() # 初始化DDS环境
8 |     node = rclpy.create_node("rflsim_subscriber") # 创建DDS节点

```

### 通信机制差异:

```

1 | # ROS1: 通过Master节点中转
2 | rospy.Publisher('/topic', String, queue_size=10)
3 |
4 | # ROS2: 直接DDS点对点通信
5 | node.create_publisher(String, '/topic', 10)

```

## 知识点4：传感器数据类型与话题映射

完整传感器支持矩阵：

```
1 | sensor_type_mapping = {
2 |     # 图像传感器类别
3 |     1: "img_rgb",          # RGB彩色相机
4 |     2: "img_depth",       # 单色灰度相机
5 |     3: "img_gray",        # 深度相机
6 |     4: "img_Segmentation", # 语义分割相机
7 |     8: "fisheye",         # 鱼眼相机
8 |     9: "img_cine",        # 吊舱相机
9 |     40: "img_Infrared_Gray", # 红外灰度相机
10 |    41: "img_Infrared",    # 红外彩色相机
11 |
12 |    # 点云传感器类别
13 |    20: "vehicle_lidar",   # 车载激光雷达
14 |    21: "global_lidar",   # 全局激光雷达
15 |    22: "livox_lidar",    # Livox激光雷达
16 |    23: "mid360_lidar",   # Mid360激光雷达
17 |    7: "Depth_Cloud",     # 深度点云
18 | }
```

ROS话题命名规范：

```
1 | /rflysim/sensor{SeqID}/{sensor_type}
2 | 例如：
3 | /rflysim/sensor1/img_rgb          # 1号RGB相机
4 | /rflysim/sensor2/mid360_lidar    # 2号Mid360激光雷达
5 | /rflysim/sensor1/camera_info     # 相机内参信息
```

## 知识点5：实时性能优化策略

数据流水线优化：

1. **异步数据采集**：传感器数据采集与ROS发布分离
2. **缓存机制**：实现帧缓存避免数据丢失
3. **线程隔离**：图像处理与网络传输使用独立线程
4. **内存池**：预分配内存减少动态分配开销

性能调优参数：

```
1 | {
2 |     "DataCheckFreq": 30,    // 采集频率：平衡实时性与性能
3 |     "DataWidth": 640,      // 分辨率：根据计算能力调整
4 |     "DataHeight": 480,     // 分辨率：根据计算能力调整
5 |     "SendProtocol": [1,0,0,0,0,0,0,0] // 传输协议选择
6 | }
```

## 核心接口使用说明

### 1) Python传感器数据获取接口

VisionCaptureApi核心功能：

```
1 | # 传感器初始化和配置
2 | vis.jsonLoad() # 加载Config.json中的传感器配置
3 | isSuss = vis.sendReqToUE4(0, TargetIP) # 向RflySim3D发送取图请求
4 | vis.startImgCap() # 开启传感器数据采集循环
5 |
6 | # 数据状态检查和获取
7 | vis.hasData[i] # 检查第i号传感器数据是否更新
8 | vis.Img[i]     # 获取第i号传感器的像素数据矩阵
9 |
10 | # 直接显示（可选，ROS发布模式下一般不使用）
11 | cv2.imshow('Img'+str(i), vis.Img[i]) # 使用OpenCV显示图像
```

### 2) Python传感器数据订阅接口

SensorRosSub核心功能：

```

1  # 自动传感器话题订阅和初始化
2  class SensorRosSub:
3      def __init__(self, vis):
4          # 根据传感器配置创建订阅者
5          for s in vis.VisSensor:
6              seq_id = str(s.SeqID)
7              type_id = s.TypeID
8              topic_name = f"/rflysim/sensor{seq_id}/{sensor_type[type_id]}"
9
10             if type_id in [1, 2, 3, 4, 8, 9, 40, 41]: # 图像类型
11                 self.msg_type = sensor_msgs.msg.Image
12             elif type_id in [20, 21, 22, 23, 7]: # 点云类型
13                 self.msg_type = sensor_msgs.msg.PointCloud2
14
15             # 创建订阅者
16             self.subscriber.append(
17                 self.create_subscriber(topic_name, self.msg_type, callback)
18             )
19
20 # 图像数据处理回调
21 def generic_callback(self, data, seq_id):
22     type_id = self.sensor[int(seq_id) - 1].TypeID
23     if type_id in [1, 2, 3, 4, 8, 9, 40, 41]: # 图像类型
24         # 根据类型选择编码格式
25         encoding = "bgr8" if type_id == 1 else "mono8"
26         img_msg = self.bridge.imgmsg_to_cv2(data, encoding)
27         print(f"Received image: {img_msg.shape}")
28     elif type_id in [20, 21, 22, 23, 7]: # 点云类型
29         print(f"Received point cloud: {sensor_type[type_id]}")

```

## 2) ReqCopterSim接口使用（自动获取ip接口）

```

1  req = ReqCopterSim.ReqCopterSim() \# 获取局域网内所有CopterSim程序的电脑IP列表
2
3  TargetIP = req.getSimIpID(StartCopterID) \#自动获取CopterSim的StartCopterID号程序所在
4  电脑的IP，作为目标IP。这里获取CopterSim所在仿真电脑的IP。
5
6  vis = VisionCaptureApi.VisionCaptureApi(TargetIP) \#创建一个视觉传感器实例，这个实例对应
7  的ip号为TargetIP

```

req.sendReSimIP(CopterID) \# 请求mavlink数据到本电脑

## 3) 相机数量和参数配置

其中，视觉传感器的初始状态由本文件夹下的Config.json决定，主要包含以下配置项：

```

1 | "SeqID":0 : 使用自动更新ID的方式, 创建了SeqID为0的视觉传感器
2 |
3 | "TypeID":20 : 传感器类型为激光雷达
4 |
5 | "TargetCopter":1 : 相机绑定在1号飞机上
6 |
7 | "SendProtocol":[1,0,0,0,0,0,0,0] : 传输模式为1: UDP网络传输模式 (图片使用jpeg压缩, 点云直
8 | 传)。
9 |
   | "SensorPosXYZ":[0,0,-0.3] : 相机分布位置。

```

## 4) 飞机控制指令

```

1 | MavList = MavList+[PX4MavCtrl.PX4MavCtrl(CopterID,TargetIP)] \#初始化并建立i号飞机的
2 | MAVLink通信连接, 连接上远端的电脑飞机, 创建一个飞机控制实例列表mavlist
3 |
4 | MavList[i].InitMavLoop() \# 初始化Mavlink监听程序, 读取第i个飞机数据
5 |
6 | MavList[i].initOffboard() \# 第i个飞机进入Offboard模式
7 |
8 | Error2UE4Map = Error2UE4Map+[-np.array([mav.uavGlobalPos[0]-
9 | mav.uavPosNED[0],mav.uavGlobalPos[1]-mav.uavPosNED[1],mav.uavGlobalPos[2]-
10 | mav.uavPosNED[2]])] \# 用于获取车辆的全局位置, 同时一个 Python 列表转换为一个 NumPy
11 | 数组, 便于进行计算处理
12 |
13 | MavList[i].SendPosNED(0, 0, -8, 0) \# 对第i个飞机发送8米高的位置控制指令
14 |
15 | mav=MavList[j] \# 从列表中取第i个实例作为当下处理的飞机实例
16 |
   | MavList[i].endOffboard() \# 对第i个飞机进入endoffboard模式
   |
   | MavList[i].stopRun() \# 终止第i个飞机

```

## 2. 实验效果

本实验实现了完整的Python传感器数据订阅处理流水线:

- **Python跨ROS版本订阅:** 实现了同时支持ROS1和ROS2的传感器数据订阅程序
- **智能类型识别:** Python程序能够根据传感器TypeID自动判断数据类型和处理方式
- **多模态数据支持:** 同时处理RGB、深度、灰度、红外图像和激光雷达点云数据
- **数据格式转换:** 使用cv\_bridge实现ROS消息与OpenCV格式的自动转换
- **实时数据监控:** 通过Rviz和终端输出实时显示传感器数据的接收情况

# 3. 文件目录

例程目录：

[安装目录]\RflySimAPIs\2.RflySimUsage\0.ApiExps\e13\_VisAPIPyCpp\2.rosPythonExps

文件夹/文件名称	类型	说明
<b>核心Python模块</b>		
<a href="#">sensor_ros_pub.py</a>	发布	Python传感器数据获取及ROS转发程序
<a href="#">sensor_ros_sub.py</a>	订阅	<b>Python传感器数据订阅处理程序（本实验核心）</b>
<b>可视化配置文件</b>		
<a href="#">vision_ros1.rviz</a>	配置	ROS1版本的Rviz可视化配置文件
<a href="#">vision_ros2.rviz</a>	配置	ROS2版本的Rviz可视化配置文件
<a href="#">vision.rviz</a>	配置	通用Rviz可视化配置文件
<b>自动化脚本</b>		
<a href="#">RunRflysim3DSITL.bat</a>	启动	一键启动RflySim3D仿真环境
<a href="#">RunWslRosCore.bat</a>	启动	WSL环境下启动ROS核心服务（ROS1专用）
<a href="#">RunWslRosTrans.bat</a>	启动	WSL环境下运行Python传感器发布程序
<a href="#">RunWslRosReceive.bat</a>	启动	<b>WSL环境下运行Python传感器订阅程序（本实验核心）</b>
<a href="#">RunWslRviz.bat</a>	启动	WSL环境下启动Rviz可视化工具
<a href="#">WinWSL.bat</a>	环境	启动WSL环境和X11图形转发服务

文件夹/文件名称	类型	说明
配置文件		
Config.json	配置	传感器参数配置（激光雷达、分辨率、频率等）

## 4. 运行环境

### 4.1 软件要求

Windows 10及以上版本；RflySim工具链；Visual Studio Code；Linux（Ubuntu 20.04）；Linux（Ubuntu 20.04）。

①：若使用Pixhawk 6X飞控，平台安装时的编译命令为：px4\_fmu-v6x\_default，推荐PX4固件版本为：1.12.3。其他配套飞控及编译命令请见：

<https://rflysim.com/doc/zh/1/Hardware.html>

### 4.2 硬件要求

笔记本/台式电脑① 1台；WinWSL 1台；虚拟机/视觉盒子/其他板卡 可选台。

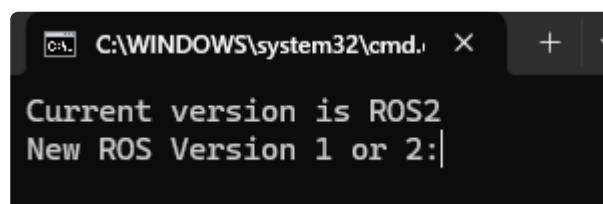
①：推荐配置请见：<https://rflysim.com/>

## 5. 实验步骤

### 5.1 前置准备步骤

#### Step 1: 配置ROS环境

在 `\Desktop\RflyTools` 中找到双击运行“RosSwitch.bat”，切换需要的ROS版本



```
C:\WINDOWS\system32\cmd. x + v
Current version is ROS2
New ROS Version 1 or 2:|
```

## ROS版本选择说明:

- **选择ROS1:** 需要双击运行 `RunWslRosCore.bat` 脚本启动ROS Master节点
- **选择ROS2:** 无需启动核心服务, DDS会自动处理节点发现和通信

## ROS1模式 - 启动核心服务:

### 功能说明:

在WSL环境中启动ROS Master作为中央消息代理和服务注册中心

### 备选方法:

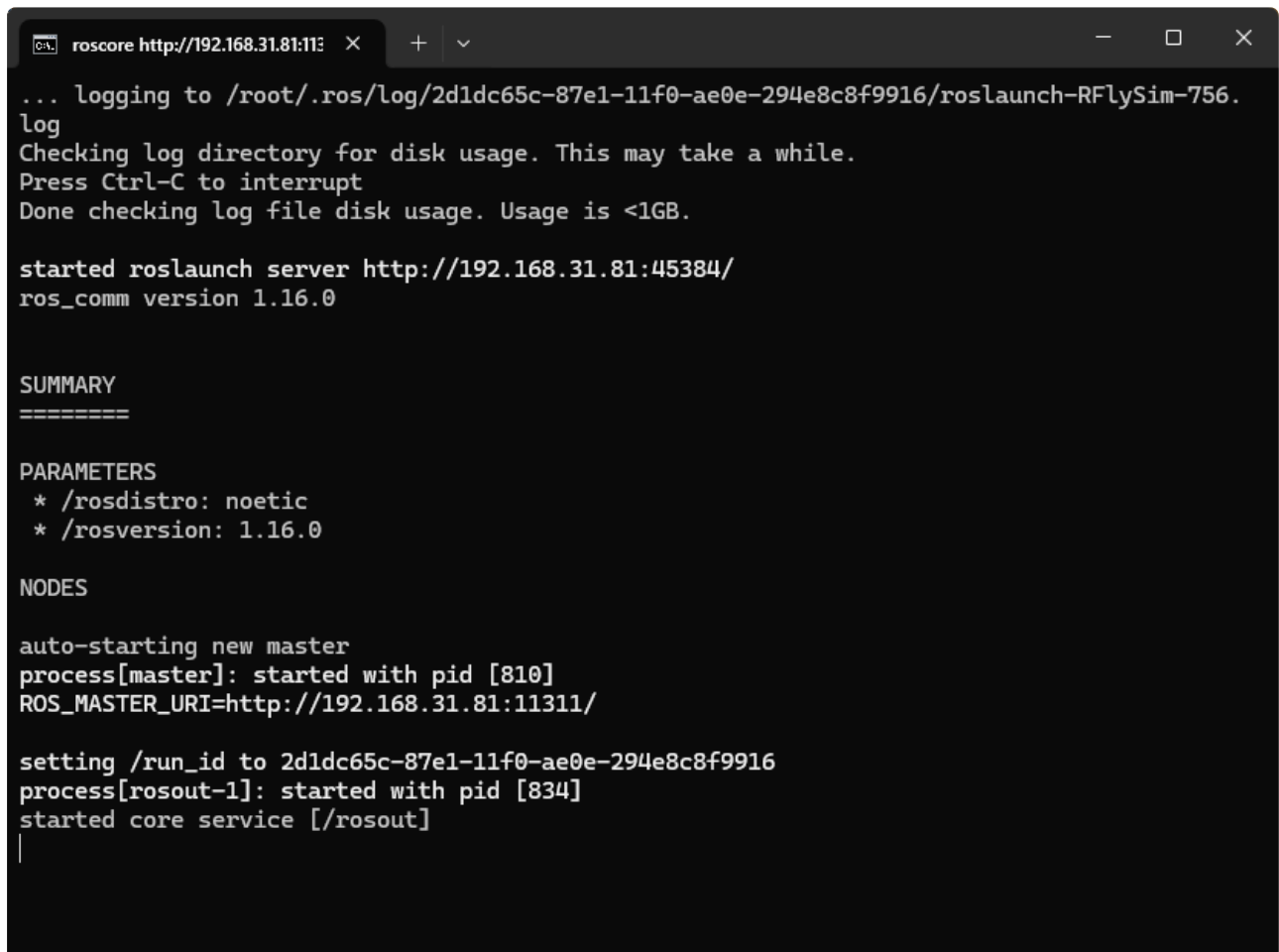
也可双击 `WinWSL.bat` 进入WSL环境, 然后手动输入 `roscore` 命令

### 预期结果:

ROS Master成功启动, 终端显示节点注册信息和参数服务器状态

## ROS2模式说明:

ROS2使用DDS (Data Distribution Service) 实现分布式通信, 节点间可直接发现和通信, 无需中央Master节点。系统启动后, 各节点会自动通过DDS协议进行服务发现和建立通信链路。



```
roscore http://192.168.31.81:11311 x + v
... logging to /root/.ros/log/2d1dc65c-87e1-11f0-ae0e-294e8c8f9916/roslaunch-RFlySim-756.
log
Checking log directory for disk usage. This may take a while.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://192.168.31.81:45384/
ros_comm version 1.16.0

SUMMARY
=====

PARAMETERS
* /roscore: noetic
* /rosversion: 1.16.0

NODES

auto-starting new master
process[master]: started with pid [810]
ROS_MASTER_URI=http://192.168.31.81:11311/

setting /run_id to 2d1dc65c-87e1-11f0-ae0e-294e8c8f9916
process[rosout-1]: started with pid [834]
started core service [/rosout]
|
```

## 5.2 ROS图像转发步骤

### Step 2: 启动RflySim3D仿真环境

#### 操作方法:

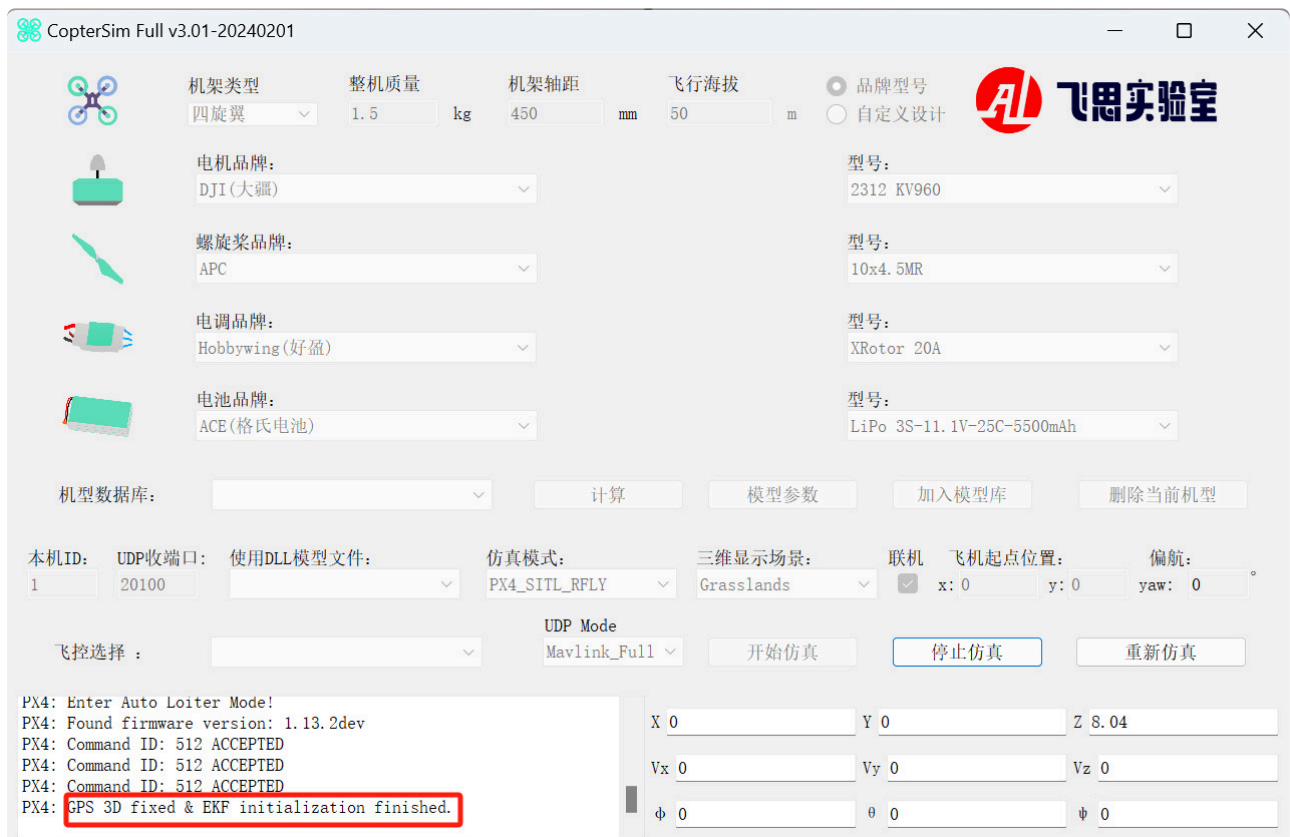
双击运行 `RunRflysim3DSITL.bat` 脚本

#### 预期结果:

- 启动1个QGC地面站
- 启动1个CopterSim仿真软件
- 启动1个RflySim3D三维仿真环境
- CopterSim软件下侧日志栏显示 `GPS 3D fixed & EKF initialization finished`
- RflySim3D软件内显示1架无人机模型

#### 状态检查:

确保所有仿真组件正常运行，无人机初始化完成



### Step 3: 启动传感器数据发布节点

#### 操作方法:

双击运行 `RunWslRosTrans.bat` 脚本

## 功能说明:

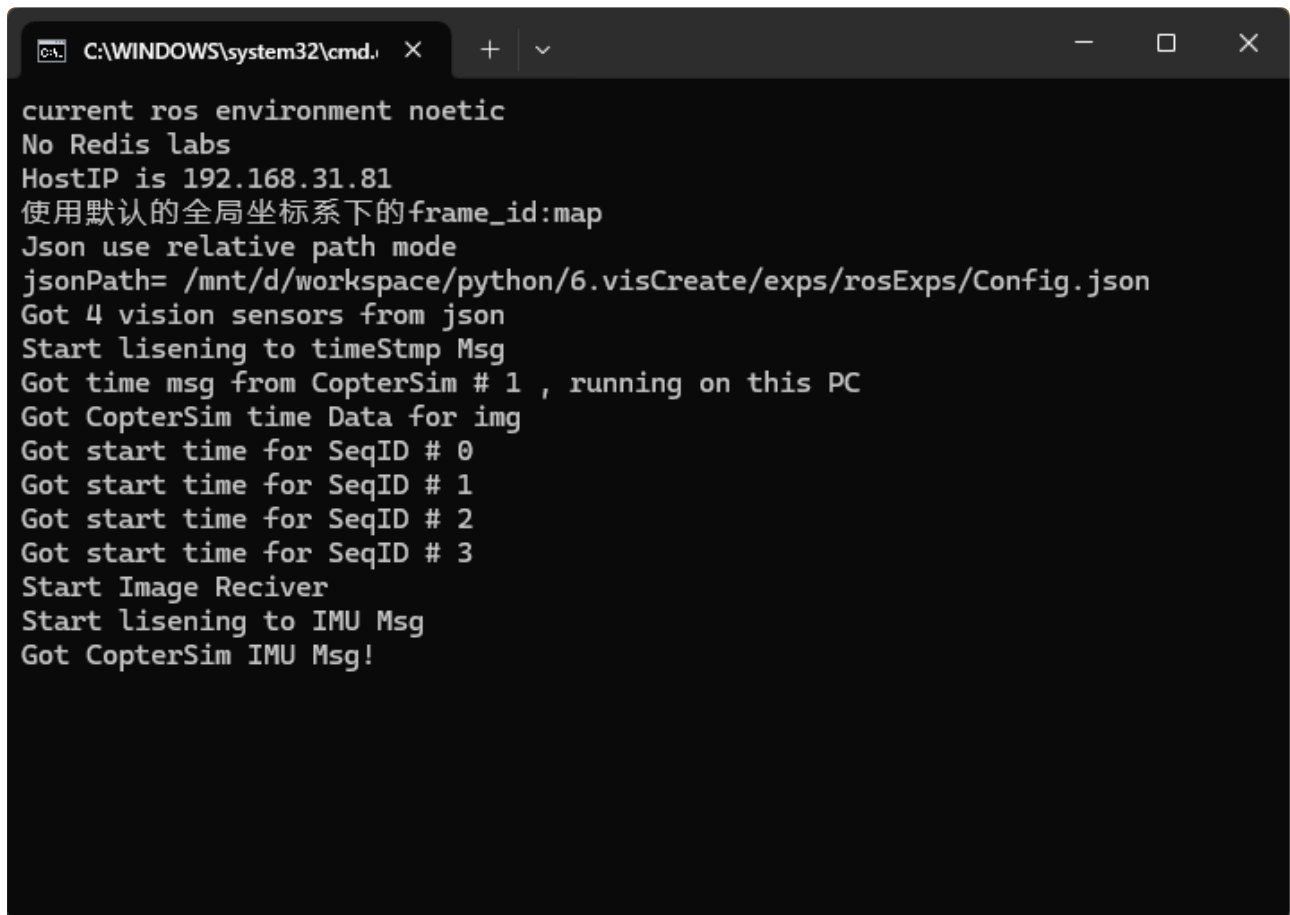
- 运行 `sensor_ros_pub.py` Python程序
- 从RflySim3D获取传感器数据（RGB图像、激光雷达点云等）
- 控制无人机起飞到指定高度
- 将传感器数据发布到ROS话题

## 核心流程:

1. 自动获取仿真电脑IP地址
2. 建立与RflySim3D的通信连接
3. 加载传感器配置（Config.json）
4. 启动传感器数据采集循环
5. 控制无人机执行起飞任务
6. 持续发布传感器数据到ROS话题

## 备选方法:

双击 `WinWSL.bat` 进入WSL环境，手动执行 `python3 sensor_ros_pub.py`



```
C:\WINDOWS\system32\cmd. x + v - □ ×
current ros environment noetic
No Redis labs
HostIP is 192.168.31.81
使用默认的全局坐标系下的frame_id:map
Json use relative path mode
jsonPath= /mnt/d/workspace/python/6.visCreate/exps/rosExps/Config.json
Got 4 vision sensors from json
Start lisening to timeStmp Msg
Got time msg from CopterSim # 1 , running on this PC
Got CopterSim time Data for img
Got start time for SeqID # 0
Got start time for SeqID # 1
Got start time for SeqID # 2
Got start time for SeqID # 3
Start Image Reciver
Start lisening to IMU Msg
Got CopterSim IMU Msg!
```

## 5.3 Python传感器数据订阅步骤

### Step 4: 启动Rviz可视化监测程序

操作方法:

双击运行 `RunWslRviz.bat` 脚本

功能说明:

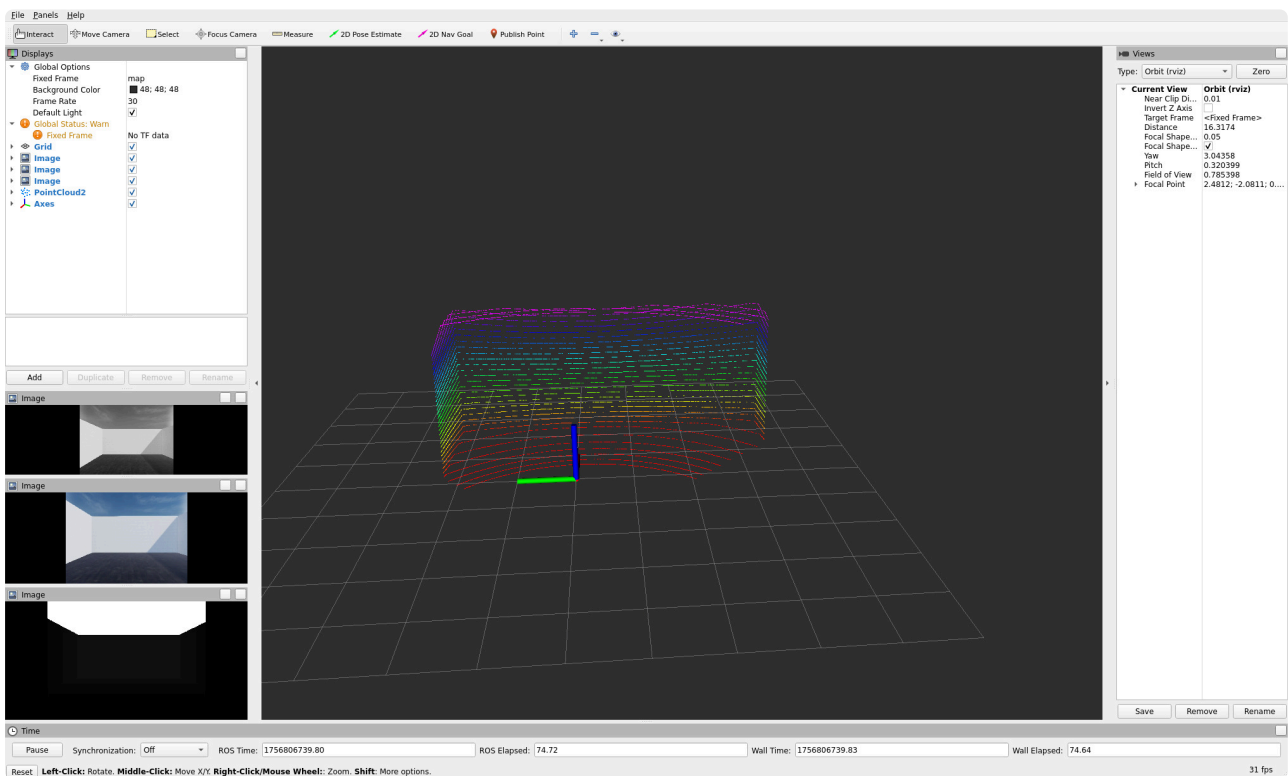
- 启动Rviz可视化工具
- 自动加载传感器数据显示配置
- 实时显示激光雷达点云数据
- 可视化传感器数据发布状态

预期效果:

- Rviz界面成功打开
- 能够看到传感器话题列表
- 点云数据在Rviz中实时更新显示

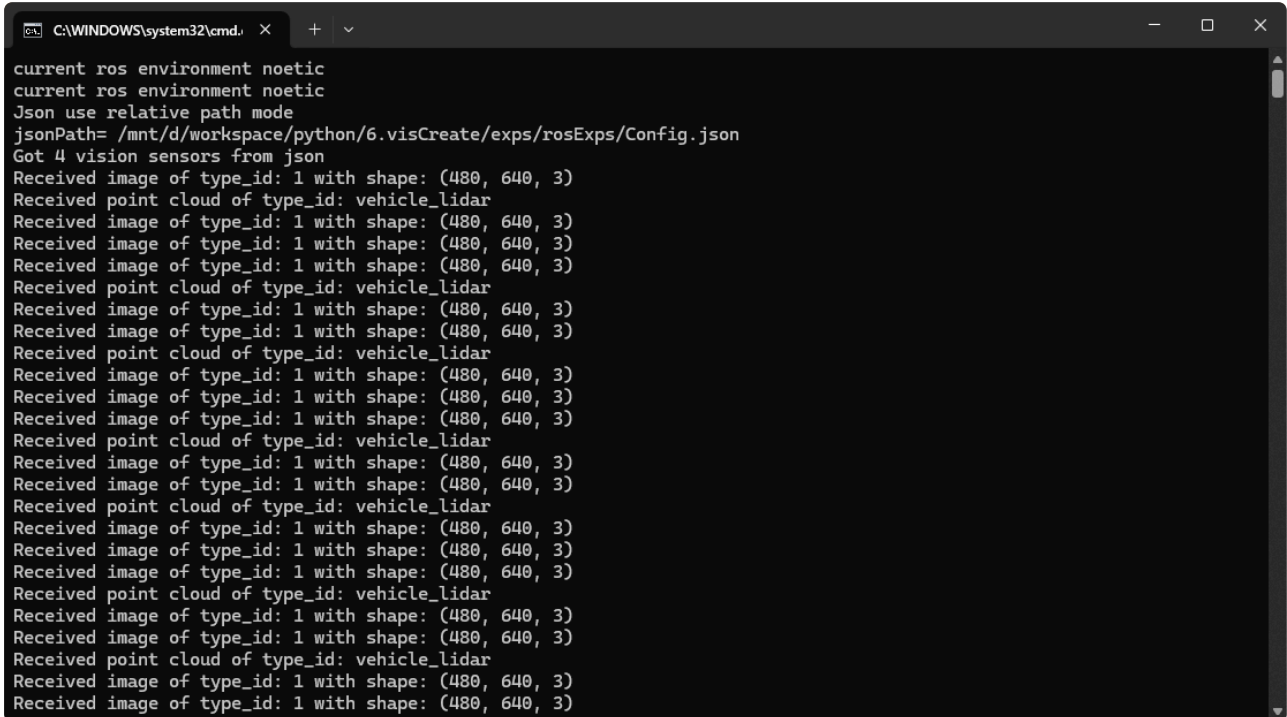
备选方法:

双击 `WinWSL.bat` 进入WSL环境, 手动输入 `rviz` 或 `rviz2` 命令



## Step 5: 开启Python/ROS接收程序

双击 [RunWslRosReceive.bat](#)，会自动在WSL下运行Python取图/ROS转发程序脚本（注：也可双击 [WinWSL.bat](#)，再输入Python3 [sensor\\_ros\\_sub.py](#) 指令），如下会从ROS空间接收到大量图片/点云数据。

A terminal window showing the execution of a Python script in a ROS environment. The output displays a sequence of received sensor data, including images and point clouds. The terminal text is as follows:

```
current ros environment noetic
current ros environment noetic
Json use relative path mode
jsonPath= /mnt/d/workspace/python/6.visCreate/exps/rosExps/Config.json
Got 4 vision sensors from json
Received image of type_id: 1 with shape: (480, 640, 3)
Received point cloud of type_id: vehicle_lidar
Received image of type_id: 1 with shape: (480, 640, 3)
Received image of type_id: 1 with shape: (480, 640, 3)
Received image of type_id: 1 with shape: (480, 640, 3)
Received point cloud of type_id: vehicle_lidar
Received image of type_id: 1 with shape: (480, 640, 3)
Received image of type_id: 1 with shape: (480, 640, 3)
Received point cloud of type_id: vehicle_lidar
Received image of type_id: 1 with shape: (480, 640, 3)
Received image of type_id: 1 with shape: (480, 640, 3)
Received image of type_id: 1 with shape: (480, 640, 3)
Received point cloud of type_id: vehicle_lidar
Received image of type_id: 1 with shape: (480, 640, 3)
Received image of type_id: 1 with shape: (480, 640, 3)
Received point cloud of type_id: vehicle_lidar
Received image of type_id: 1 with shape: (480, 640, 3)
Received image of type_id: 1 with shape: (480, 640, 3)
Received image of type_id: 1 with shape: (480, 640, 3)
Received point cloud of type_id: vehicle_lidar
Received image of type_id: 1 with shape: (480, 640, 3)
Received image of type_id: 1 with shape: (480, 640, 3)
Received point cloud of type_id: vehicle_lidar
Received image of type_id: 1 with shape: (480, 640, 3)
Received image of type_id: 1 with shape: (480, 640, 3)
```

## 实验技巧与注意事项

### 启动顺序建议

建议按照以下顺序启动各个组件，以确保系统稳定运行：

#### ROS1启动序列：

1. 首先启动ROS Master：运行 `RunWslRosCore.bat`
2. 启动RflySim3D仿真环境
3. 等待无人机完全初始化后，启动传感器数据发布节点
4. 最后启动可视化工具和接收节点

#### ROS2启动序列：

1. 直接启动RflySim3D仿真环境（无需ROS Master）
2. 等待无人机完全初始化后，启动传感器数据发布节点
3. 启动可视化工具和接收节点（DDS自动处理节点发现）

**注意：** ROS2的分布式架构使得节点启动顺序更加灵活，但建议仍按上述顺序以确保数据流的连续性。

## 网络连接提示

- 如果使用多台电脑进行分布式仿真，请确保所有设备在同一局域网内
- 检查防火墙设置，确保相关端口未被阻塞
- 如果IP自动获取失败，可以手动配置目标IP地址

## Python程序性能优化建议

### Python环境优化：

- 使用Python虚拟环境隔离项目依赖
- 安装优化版本的NumPy和OpenCV: `pip install numpy[mkl] opencv-python`
- 配置OpenCV使用多核处理: `cv2.setNumThreads(4)`

### 数据处理优化：

- **内存管理：** 使用NumPy数组存储图像数据，避免频繁的内存拷贝
- **异步处理：** 使用Python多线程或异步处理图像数据
- **数据压缩：** 对于大尺寸图像，可考虑在处理前进行适度压缩

### ROS通信优化：

- 调整ROS订阅者的队列大小，避免数据丢失
- 对于ROS2，优化QoS设置以平衡实时性和可靠性
- 使用合适的消息缓存策略，避免内存溢出

### 系统调优：

- 在Config.json中调整传感器参数（分辨率、频率）以适应系统性能
- 使用合适的传输协议平衡质量与性能
- 监控Python进程的CPU和内存使用情况

## 6. 参考资料

1. RflySim官方文档: <https://rflysim.com>
2. ROS官方教程: <http://wiki.ros.org>
3. OpenCV-Python教程: <https://opencv-python-tutroals.readthedocs.io>

4. PX4飞控固件: <https://px4.io>

## I 7. 常见问题

### Q1: 启动仿真后, 无人机无法正常起飞怎么办?

A1: 检查以下几点:

- 确保CopterSim日志显示"GPS 3D fixed & EKF initialization finished"
- 检查QGC地面站中的传感器状态是否正常
- 确认MAVLink连接建立成功
- 检查无人机是否已解锁 (armed)

### Q2: 终端中看不到传感器数据显示怎么办?

A2: 请按ROS版本分别排查:

#### ROS1排查步骤:

- 确认ROS Master正在运行: 检查 `roscore` 进程
- 确认传感器数据发布节点正常运行
- 检查ROS话题是否正确发布: `rostopic list`
- 验证话题数据: `rostopic echo /rflsim/sensor1/img_rgb`
- 检查节点连接状态: `rostopic list` 和 `rostopic info /sensor_node`

#### ROS2排查步骤:

- 确认DDS环境正常: `ros2 daemon status`
- 检查可用话题: `ros2 topic list`
- 验证话题数据: `ros2 topic echo /rflsim/sensor1/img_rgb`
- 检查节点状态: `ros2 node list` 和 `ros2 node info /sensor_node`
- 验证QoS兼容性: `ros2 topic info /rflsim/sensor1/img_rgb --verbose`

### Q3: Python传感器订阅程序运行出错怎么办?

A3: 请按以下步骤检查和解决:

#### Python依赖检查:

- 检查Python版本: `python3 --version` (需要Python 3.6+)
- 安装基础依赖: `pip3 install opencv-python numpy yaml`

- 安装ROS Python依

赖: `sudo apt install python3-rospy python3-sensor-msgs python3-cv-bridge`  
(ROS1)

- 安装ROS2 Python依

赖: `sudo apt install python3-rclpy python3-sensor-msgs-py` (ROS2)

### 环境配置检查:

- 检查ROS环境变量: `echo $ROS_VERSION` 和 `echo $ROS_DISTRO`
- 确保正确加载ROS环境: `source /opt/ros/$ROS_DISTRO/setup.bash`
- 检查Python路径: `which python3`

### 常见错误解决:

- 如果出现模块导入错误, 请检查PYTHONPATH环境变量
- cv\_bridge错误: `sudo apt install ros-$ROS_DISTRO-cv-bridge`
- 如果出现权限问题, 请检查Python脚本的执行权限

### Q4: WSL环境中无法显示图形界面怎么办?

A4: 需要配置X11转发:

- 确保VcXsrv X Server正在运行
- 在WSL中设置DISPLAY环境变量: `export DISPLAY=:0`
- 检查防火墙是否允许VcXsrv通信
- 重启WSL后重新尝试

### Q5: Python订阅程序能正常运行但接收不到数据怎么办?

A5: 请按以下步骤排查:

#### 检查传感器发布节点:

- 确保 `sensor_ros_pub.py` 程序正在运行
- 检查仿真环境是否正常启动和连接
- Verify RflySim3D和CopterSim的连接状态

#### 检查ROS话题状态:

- ROS1: `rostopic list | grep rflysim` 查看传感器话题
- ROS2: `ros2 topic list | grep rflysim` 查看传感器话题
- 检查话题数据发布频率: `rostopic hz /rflysim/sensor1/global_lidar`

### 检查订阅节点配置:

- 检查 `sensor_ros_sub.py` 中的传感器类型映射是否正确
- 验证 `VisionCaptureApi` 配置与 `Config.json` 中的传感器参数一致
- 检查 ROS 消息类型和话题名称的匹配性

### Q6: 多台电脑联机仿真时连接失败怎么办?

A6: 请检查网络配置:

- 确保所有电脑在同一局域网内
- 检查 IP 地址获取是否正确
- 关闭 Windows 防火墙或添加例外
- 使用 ping 命令测试网络连通性
- 手动指定 IP 地址而不使用自动获取