多无人机广播缓冲区API实验

1. 实验目的

本实验旨在深入理解和掌握多无人机系统中基于缓冲区的数据广播机制。通过搭建SITL（Software In The Loop）仿真环境，学习如何使用NetBufAPI实现多架无人机之间的数据传输与共享。实验将使学习者熟悉以下关键概念和技能：

- 理解无人机集群中数据广播的原理和应用场景
- 掌握NetBufAPI缓冲区数据结构的设计与实现方法
- 学习使用struct模块进行数据打包与解包操作
- 掌握多线程编程在无人机通信中的应用
- 实现跨平台（Windows/Linux）的无人机协同通信机制

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\9.RflySimComm\0.ApiExps\e1.RflyNetAPIExps\5.BroadBufAPI

- NetBufAPI.py：网络缓冲区API接口文件
- Python38Run.bat：RflySim Python运行脚本
- SITLRun3MavlinkFull.bat：启动3架无人机SITL仿真环境脚本
- UAV1Ctrl.py：1号无人机控制程序（发送端）
- UAV2Ctrl.py：2号无人机控制程序（发送端）
- UAV3Ctrl.py：3号无人机控制程序（接收端）
- UavPythonRunALL.bat：批量启动所有无人机控制程序脚本

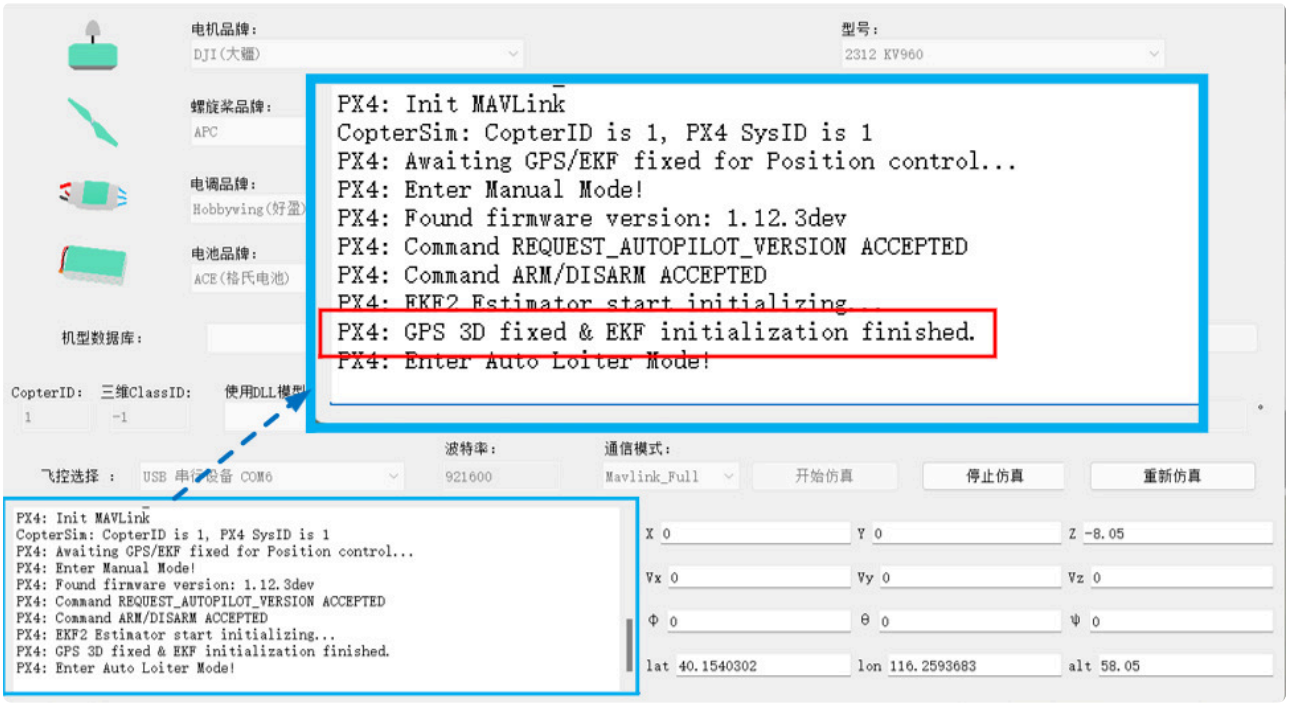
4. 实验内容或步骤

本实验分为两个小实验。

4.1. 本机实验

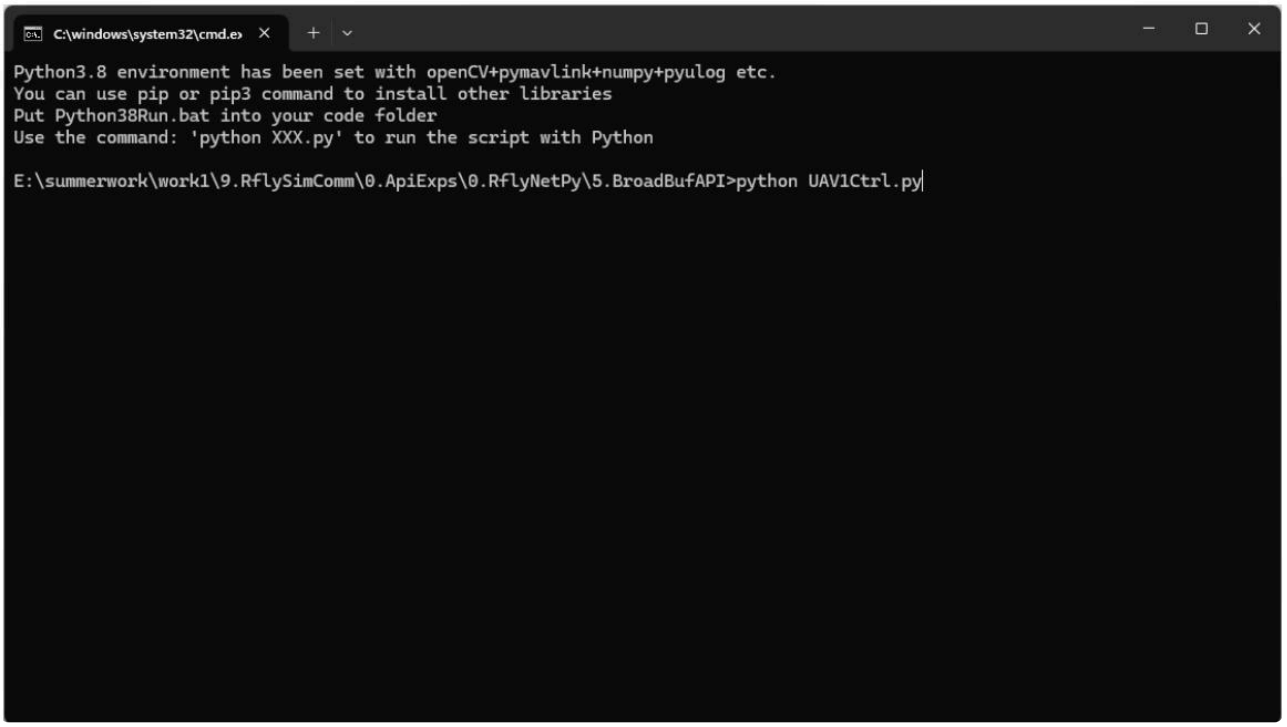
4.1.1. 启动仿真环境

双击运行`SITLRun3MavlinkFull.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， 3 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：



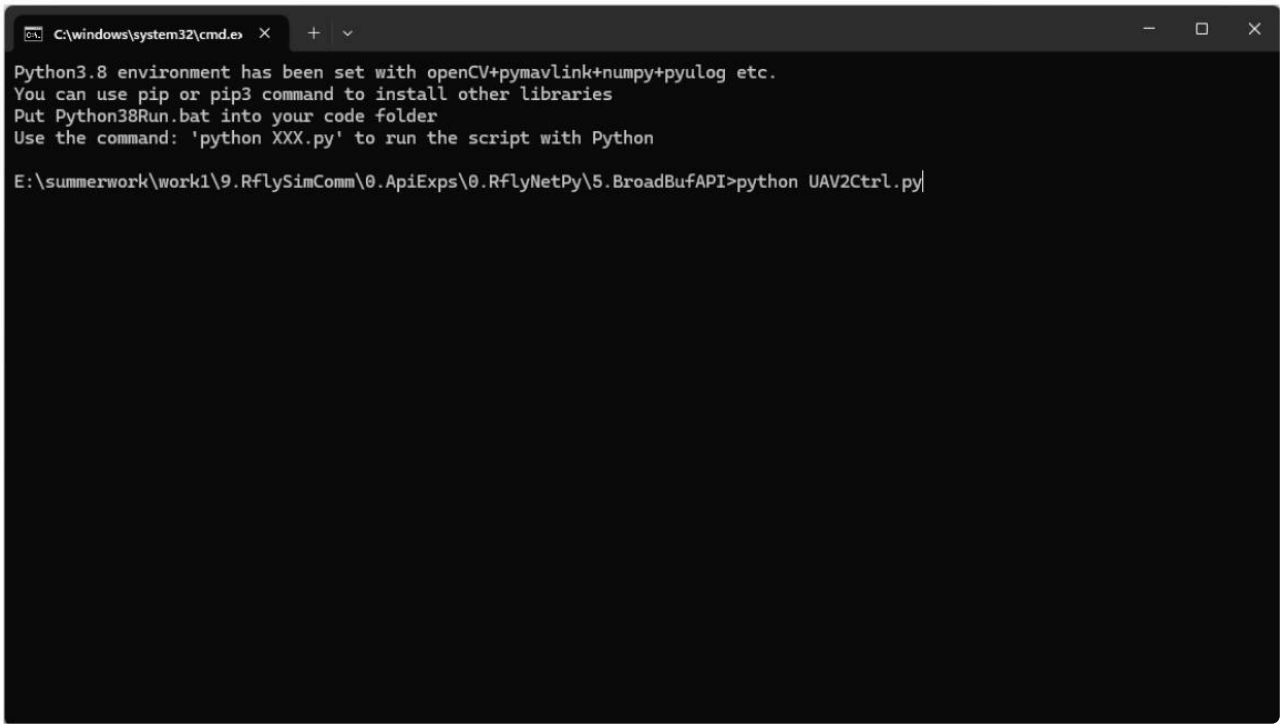
4.1.2. 启动1号无人机广播发送程序

双击运行"Python38Run.bat"，在其中输入"python UAV1Ctrl.py"，启动 1 号飞机广播发送程序。



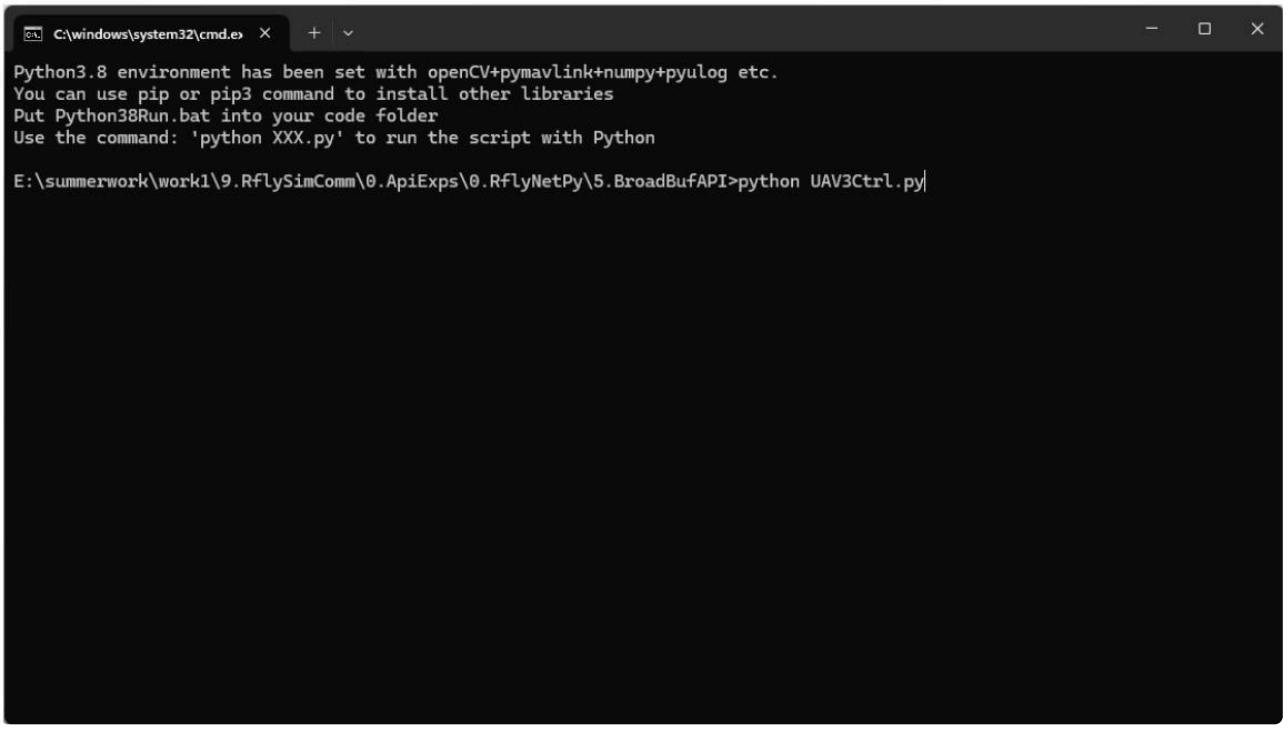
4.1.3. 启动2号无人机广播发送程序

双击运行"Python38Run.bat"，在其中输入"python UAV2Ctrl.py"，启动 2 号飞机广播发送程序。



4.1.4. 启动3号无人机广播接收程序

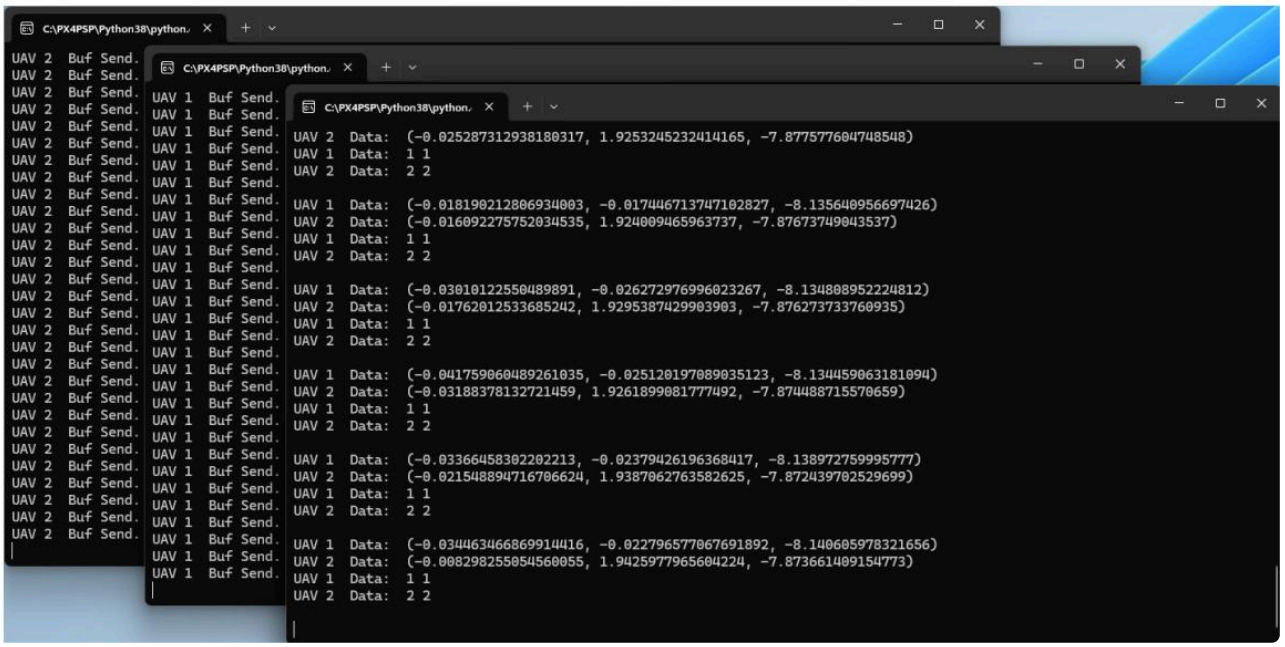
双击运行"Python38Run.bat"，在其中输入"python UAV3Ctrl.py"，启动 3 号飞机广播接收程序。



4.1.5. 观察实验结果

观察每个窗口，每个程序都打印自己收到的数据，并输出。1-2号飞机，将结构体转为buf，并发出，在命令提示符窗口打印。3号飞机，收到包后，解析并打印。

> 注：上述步骤可以用直接双击运行"UavPythonRunALL.bat"来替代，会自动完成上述操作。



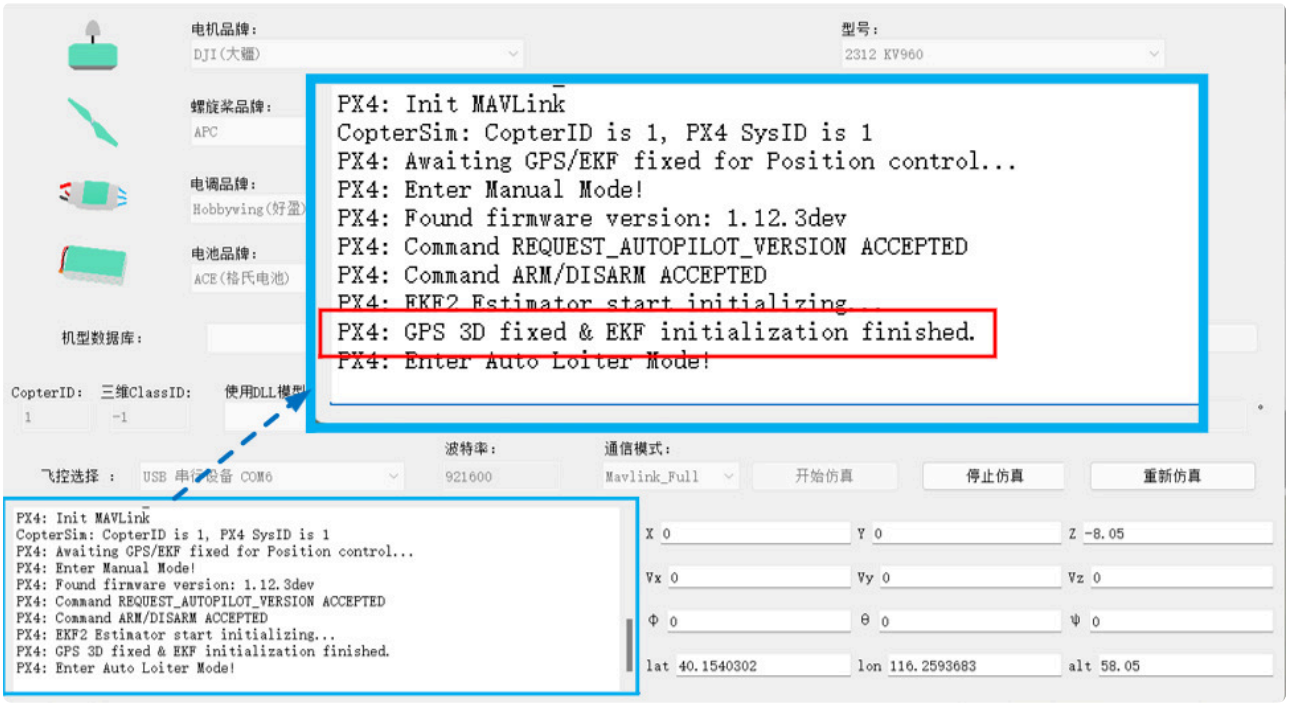
4.2. 联机实验

4.2.1. 准备工作

1）准备3台任意系统主机（可以是NX主机、Ubuntu虚拟机、Windows电脑）。确保两台机器是在同一个局域网内。
2）按照[RflySim按安装理解]\RflySimAPIs\1.RflySimIntro\2.AdvExps\e3.PythonConfig实验将[RflySim安装路径]\RflySimAPIs\RflySimSDK\RflySimSDK库复制到每台主机上，并确保RflySimSDK已经导入当前主机的Python环境中。

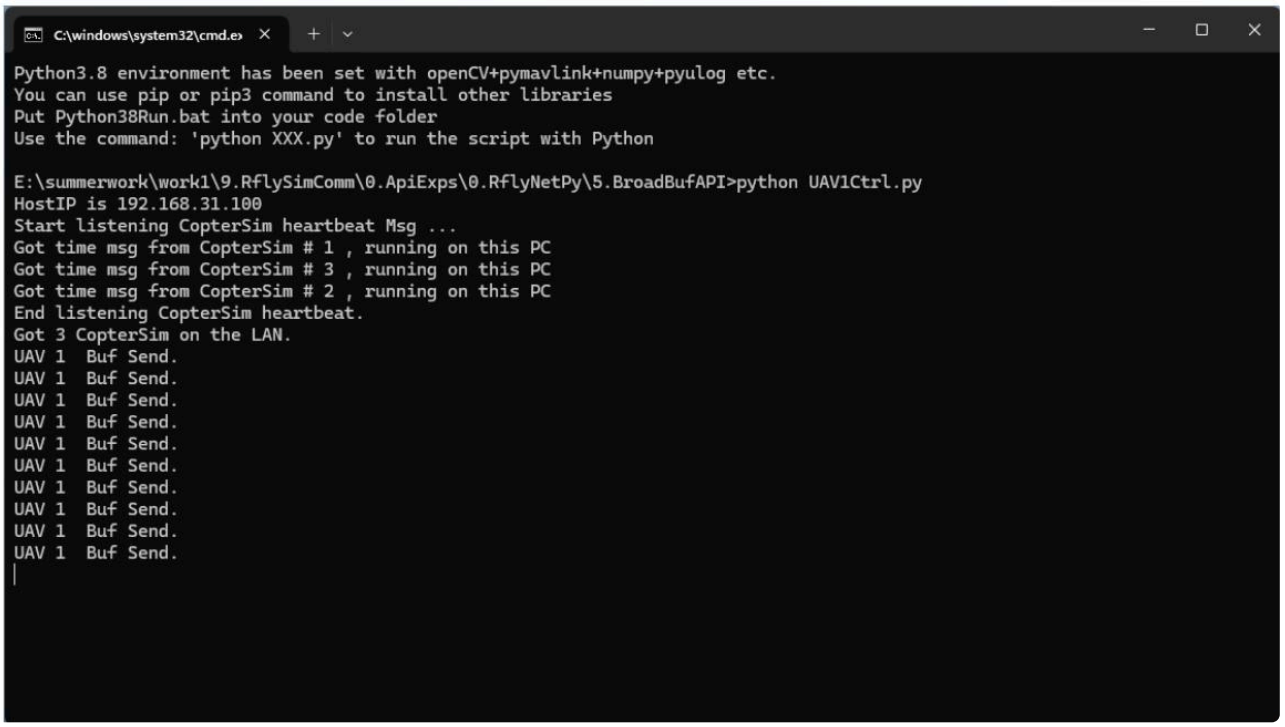
4.2.2. 启动仿真环境

双击运行`SITLRun3MavlinkFull.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， 3 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：



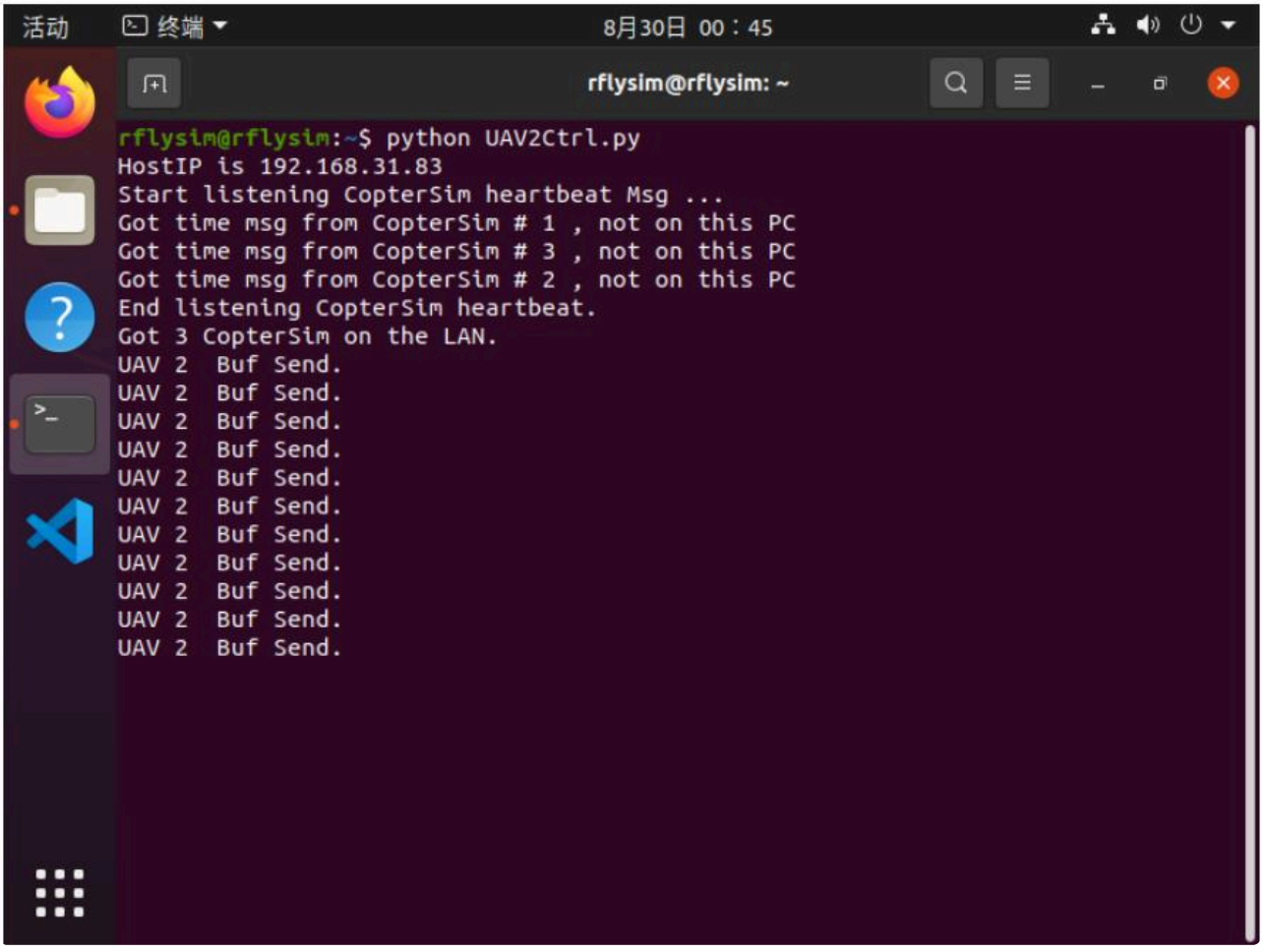
4.2.3. 在Windows主机中启动1号无人机程序

在 Windows 主机中，运行 UAV1Ctrl.py，启动 1 号飞机广播发送程序。



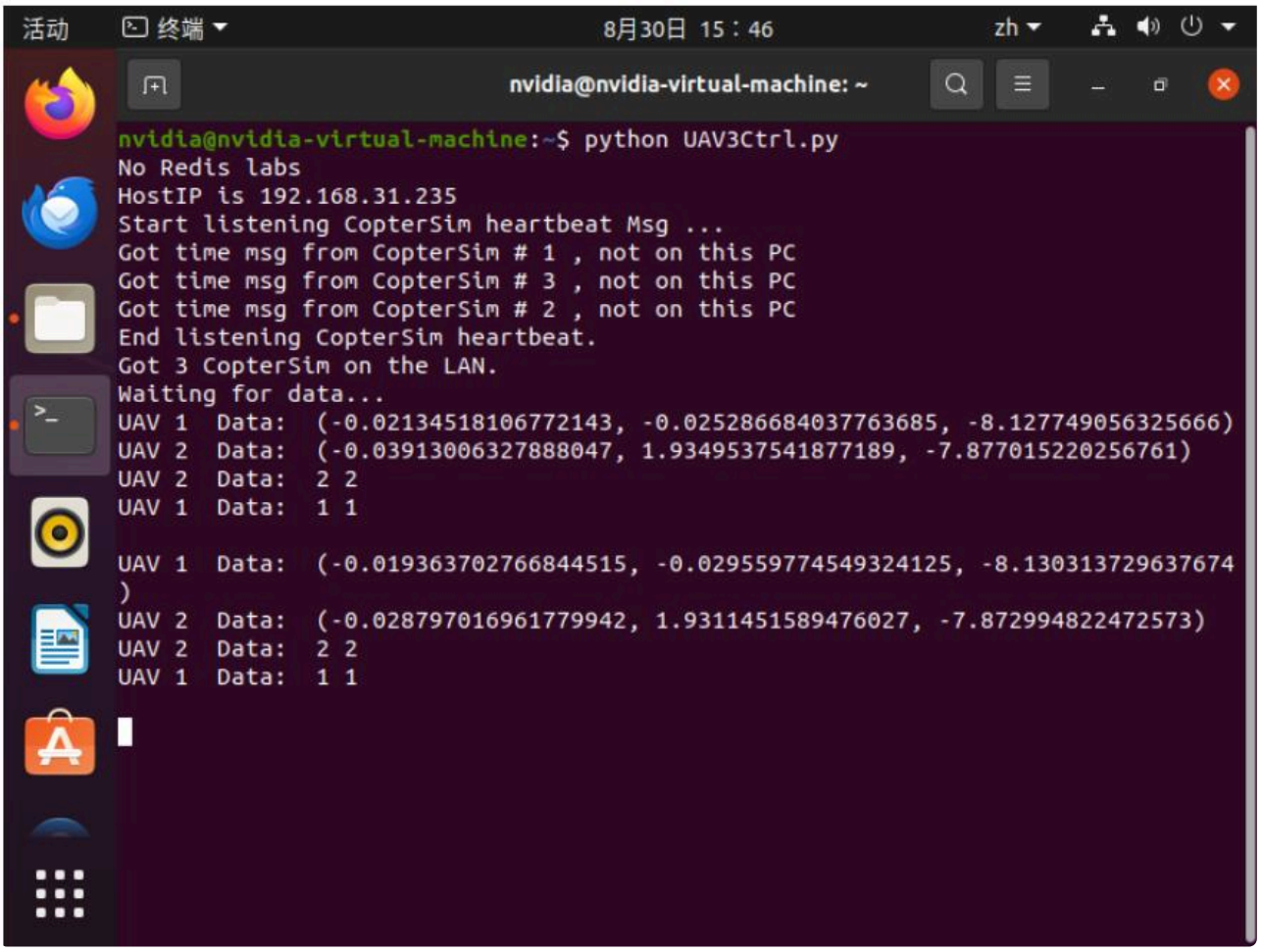
4.2.4. 在Ubuntu虚拟机中启动2号无人机程序

在一台Ubuntu虚拟机中，运行UAV2Ctrl.py，启动 2号飞机广播发送程序。控制程序中需要导入"NetBufAPI.py"文件，所以需要将该文件一同复制到虚拟机中。



4.2.5. 在Ubuntu虚拟机中启动3号无人机程序

在另一台 Ubuntu 虚拟机中，运行 UAV3Ctrl.py，启动 3 号飞机广播接收程序。控制程序中需要导入"NetBufAPI.py"文件，所以需要将该文件一同复制到虚拟机中。



4.2.6. 验证实验效果

能看到和本地测试一样的效果，说明测试正确。

5. 关键知识点

本实验的整体思路是构建一个多无人机系统间的通信框架，通过网络缓冲区API实现无人机间的数据交换。整个系统采用发布-订阅模式，发送方将数据封装成特定格式的缓冲区，接收方通过解析缓冲区数据获取所需信息。

NetBufAPI.py 解析：

该文件是本实验的核心，实现了网络缓冲区API，主要包括以下几个关键组件：

1. **UAVData 类**：用于存储单架无人机的数据

```python
class UAVData:
    def __init__(self):
        self.hasUpdate=False
        self.CopterID=False
        self.Data1=False
        self.Data2=False

    def update(self,Data):
        self.hasUpdate=True
        self.CopterID=Data[1]
        self.Data1=Data[2]
        self.Data2=Data[2]
```

该类用于存储从网络接收到的无人机数据，包括更新标志、无人机ID和两个自定义数据字段。

2. **NetBufAPI 类**：主要API接口类

```python
class NetBufAPI:
    def __init__(self, net):
        self.BufDataDict={}
        self.BufData=[]
        self.net = net

    def getMavEvent(self):
        print('Waiting for data...')
        while True:
            self.net.bufEvent.wait() # 会一直阻塞在这里，直到有新的数据包收到
            bufData = copy.deepcopy(self.net.bufData) # 立刻将包拷贝出来
            bufHead = copy.deepcopy(self.net.bufHead) # 立刻将包头拷贝出来
            self.net.bufEvent.clear() # 清除set位，使得下层能够继续更新数据

            if len(self.net.bufData) == 16:
                Data = struct.unpack('iiii',self.net.bufData) # 获取所有数据向量
                checksum = Data[0]
                if checksum==123456789: # 校验通过，说明是对的包
                    CopterID = Data[1]

                    if self.BufDataDict.__contains__(str(CopterID)): # 如果飞机数据已经在字典中
                        self.BufDataDict[str(CopterID)].update(Data) # 直接更新数据
                    else:
                        uData = UAVData() # 创建一个新的空结构体
                        uData.update(Data) # 更新数据
                        self.BufDataDict[str(CopterID)] = copy.deepcopy(uData) # 拷贝并加入字典
                        self.BufData=self.BufData+[self.BufDataDict[str(CopterID)]]

                    continue # 成功解析数据包，就跳过本循环，后续代码将不会执行
```

这个方法实现了数据包的接收和解析，使用多线程技术确保主线程不被阻塞。通过校验码验证数据包的合法性，并根据无人机ID将数据存储到对应的对象中。

3. **SendMyBuf 方法**：发送数据的方法

```
def SendMyBuf(self,Data1,Data2):
    buf = struct.pack('iiii',123456789,self.net.CopterID,Data1,Data2) # 封装一个数据包，备注1号飞机
    self.net.netForwardBuf(buf) # 发出buf包
```

该方法将数据打包成特定格式并通过网络发送出去。

4. **StartBufRec 方法**：启动接收线程

```
def StartBufRec(self):
    # 开启一个线程去接受buf数据并解析
    self.t1 = threading.Thread(target=self.getMavEvent, args=())
    self.t1.start()
```

通过开启独立线程处理数据接收，避免阻塞主程序流程。

UAV控制程序解析：

1. **UAV1Ctrl.py 和 UAV2Ctrl.py**：这两个程序都是发送端程序，结构基本相同，主要功能包括：
   - 初始化通信实例，建立与对应无人机的连接
   - 创建NetBufAPI实例，用于数据发送
   - 循环发送包含无人机ID和自定义数据的数据包
   - 打印发送状态信息
2. **UAV3Ctrl.py**：接收端程序，主要功能包括：
   - 初始化通信实例
   - 启动数据接收线程
   - 循环读取接收到的数据并打印出来
   - 显示来自不同无人机的数据

这些程序共同构成一个完整的多无人机通信系统，实现了数据的发送、接收和解析功能。

6.参考资料

1. RflySim官方文档：https://rflysim.com/doc/zh/

2. PX4 官方文档： https://docs.px4.io/
3. MAVLink 协议规范： https://mavlink.io/en/

7.常见问题

Q1：运行程序时出现"无法找到NetBufAPI模块"错误?

A1：确保将NetBufAPI.py文件放在与UAV控制程序相同的目录下，或将其放置在Python解释器能够找到的路径中。如果是在分布式环境中运行，需要确保目标机器上也存在该文件。

Q2：UAV3无法接收到UAV1和UAV2发送的数据?

A2：首先确认SITL仿真环境已正确启动，各CopterSim实例正常运行。检查网络连接是否正常，确保广播端口和接收端口一致。另外，检查发送的数据包格式是否符合预设的结构（长度为16字节，校验码为123456789）。

Q3：在Linux/Ubuntu环境下运行出现问题?

A3：确保Python环境配置正确，相关依赖库已安装。在Linux环境下可能需要调整某些路径设置，同时确保NetBufAPI.py文件和其他必要的API文件都已复制到Linux系统中。检查权限设置，确保脚本具有执行权限。

---

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