基于ROS2集群控制实验

1. 实验目的

本实验旨在通过实际操作掌握基于ROS2的无人机集群控制系统的设计与实现方法。通过本实验，学习者将能够：

1. 理解ROS2在无人机集群控制中的应用机制；
2. 掌握使用ROS2与PX4飞控系统进行通信的方法；
3. 学习如何通过Python脚本实现多无人机协同控制；
4. 熟悉Offboard模式下无人机的位置控制原理；
5. 了解无人机集群编队飞行的基本实现方式；
6. 掌握WSL2环境下ROS2开发环境的配置与使用。

通过完成本实验，学习者将具备独立开发基于ROS2的无人机集群控制系统的能力，为进一步研究无人机集群智能决策、路径规划等高级功能打下坚实基础。

2. 实验要求

- 软件要求：Windows 10及以上版本；RflySim工具链[1]。
  > ①：工具链安装时，PX4固件版本必须为：1.14.4及以上版本。
- 硬件要求：笔记本/台式电脑1台[2]。
  > ①：推荐配置请见：https://rflysim.com/

3.实验地址

例程目录：[安装目录]\RflySimAPIs\10.RflySimSwarm\0.ApiExps\e10.ROS2Swarm

- Ros2ComMode.bat：ROS2通信模式切换脚本，用于切换WSL中系统DDS模式配置
- RosSwitch.bat：ROS版本切换脚本，用于在ROS1和ROS2之间切换
- SITL4Run.bat：SITL仿真启动脚本，用于启动4个无人机的仿真环境

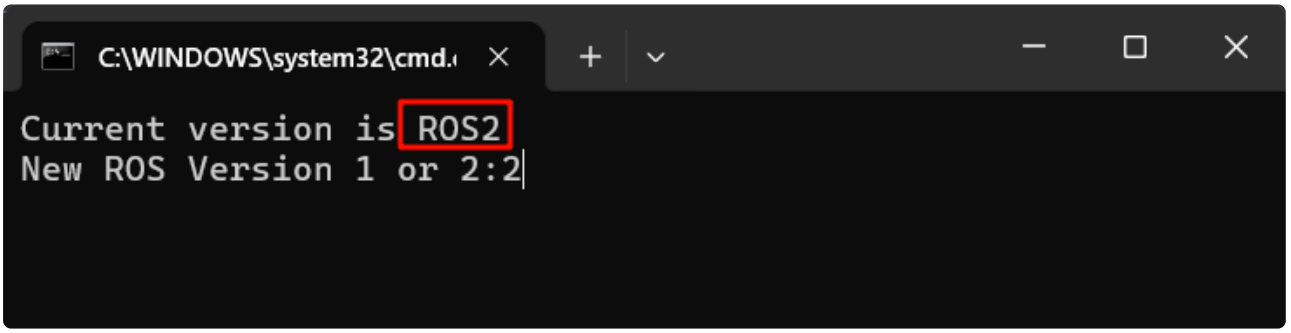
- WinWSL.bat：WSL启动脚本，用于打开WSL终端环境
- WinWSL_StartuXRCE.bat：MicroXRCE代理启动脚本，用于启动uXRCE-DDS代理服务
- extract_and_copy.bat：文件提取复制脚本，用于将实验文件复制到ROS工作空间
- px4_swarm_control.py：集群控制核心脚本，实现多无人机协同控制功能
- setup_px4_ros.sh：PX4 ROS依赖安装脚本，用于安装必要的ROS2功能包

4. 实验内容或步骤

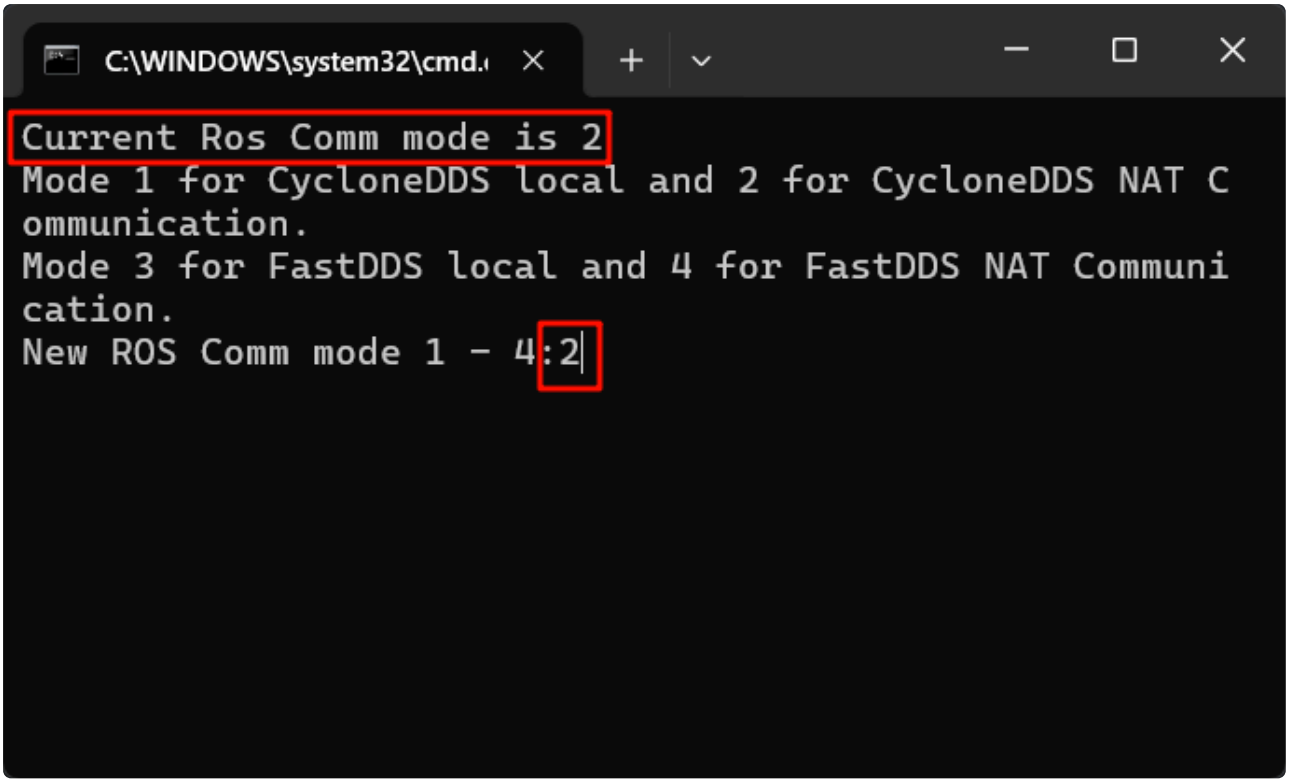
4.1 步骤1：环境确认

本实验前请确认WSL得系统为 WSL 2，可在CMD窗口输入 `wsl -l -v` 即可查看版本。

若不是WSL 2，请自行重新安装RflySim配置为WSL 2。本实验基于ROS2开发，因此实验前需要确认RflySim中的ROS环境为ROS 2，可双击运行RosSwitch.bat脚本进行切换。



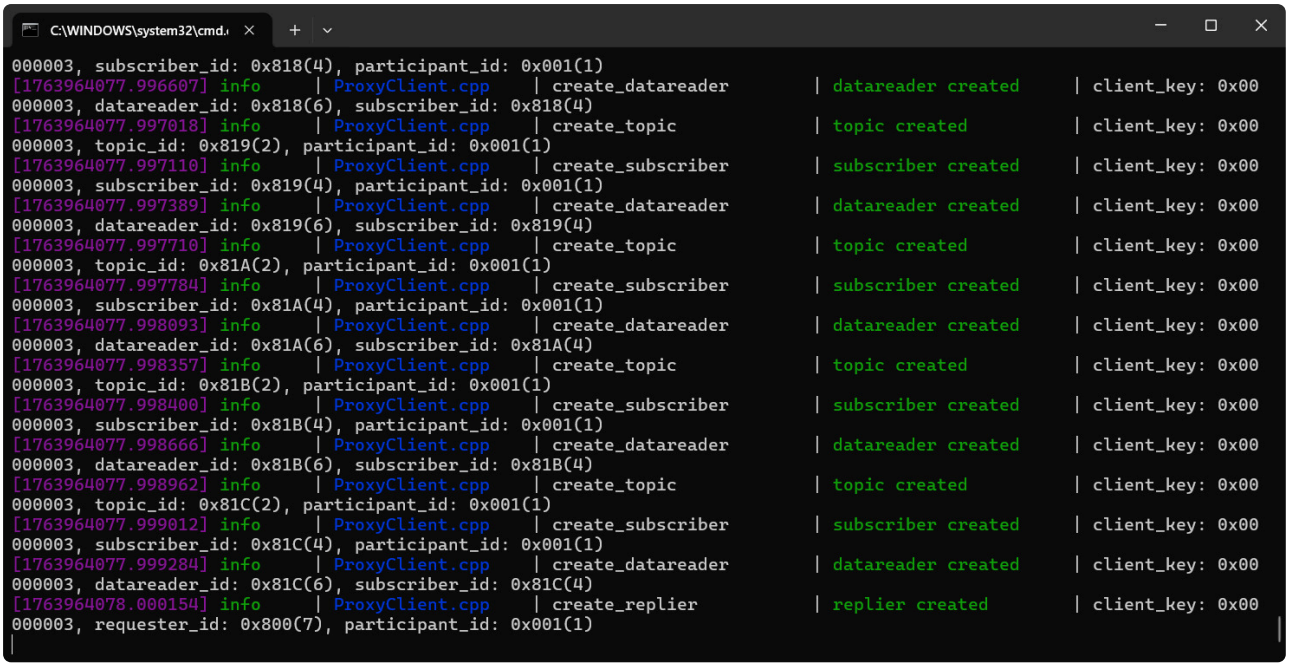
双击运行Ros2ComMode.bat脚本，输入2回车，将WSL中系统切换到CycloneDDS+广播模式。



4.2 步骤2：初始化仿真环境

双击运行SITL4Run.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， 4 个 CopterSim 、1 个 RflySim3D 软件， 等待4个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：

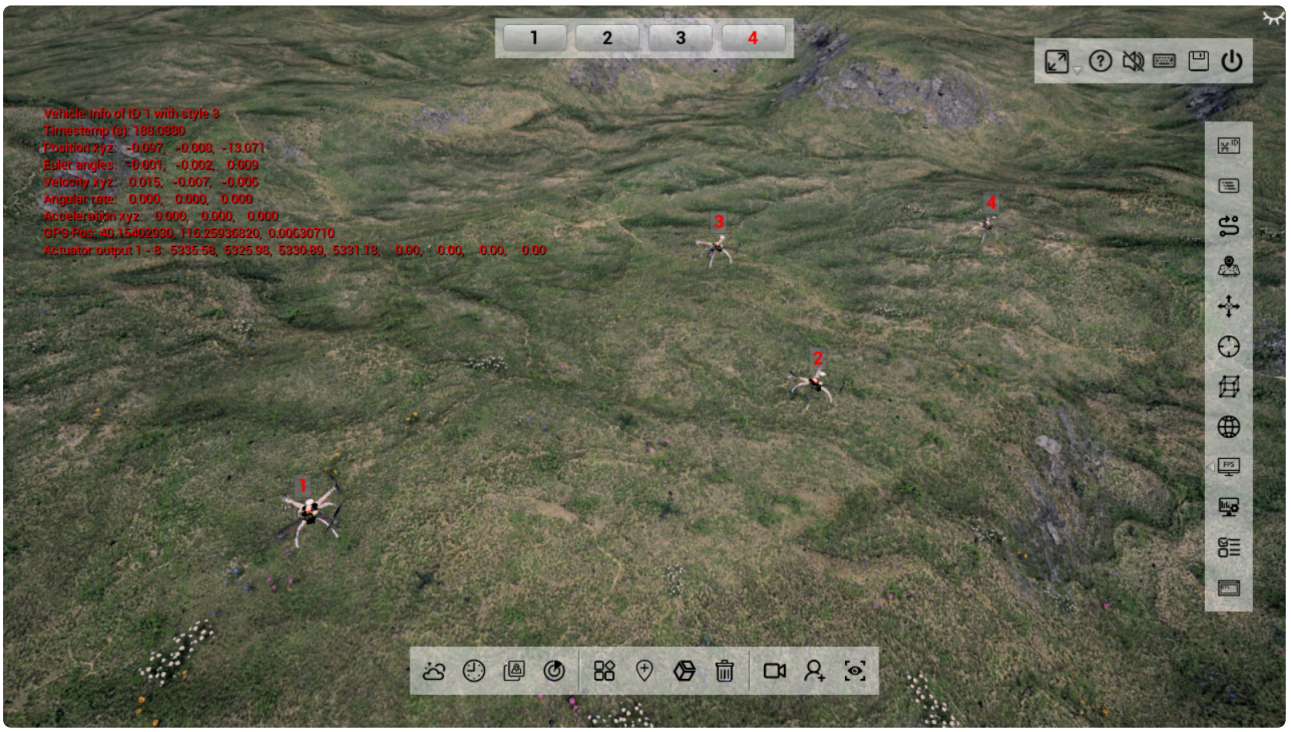
双击运行WinWSL_StartuXRCE.bat脚本，启动MicroXRCEAgent。



4.3 步骤3：构建ROS 2工作空间并运行

双击打开WinWSL.bat脚本，在打开的WSL系统中输入：`mkdir -p ~/ros2swarm/src/`来创建ROS 2工程目录。

然后，双击运行extract_and_copy.bat脚本，将本实验所需文件复制到ROS 2工程目录下并开始编译ROS 2工程。等待编译完成后，将自动开始运行。



4.4 步骤4：运行环境还原

实验完成后，务必需要运行Ros2ComMode.bat脚本将DDS的模式切换到1，避免干扰其他实验，ROS版本和WinWSL版本可选择性切换。

5. 关键知识点

关键知识点1：ROS2与PX4通信机制

ROS2与PX4之间的通信是通过uXRCE-DDS代理实现的。PX4作为服务端，ROS2节点作为客户端，两者通过DDS中间件进行数据交换。在本实验中，我们通过订阅和发布PX4定义的各类消息来实现对无人机的控制。

核心的消息类型包括：

- OffboardControlMode：用于声明offboard控制模式下激活的控制量
- TrajectorySetpoint：用于发送轨迹设定点（位置、速度、加速度等）
- VehicleCommand：用于发送车辆控制命令，如解锁、更改模式等
- VehicleStatus：用于接收车辆当前状态信息

- VehicleOdometry：用于接收车辆里程计信息

关键知识点2：PX4 Offboard控制模式

Offboard模式是PX4的一种特殊模式，允许外部控制器完全接管飞行器的控制。当处于Offboard模式时，飞行器会持续监听来自外部控制器的设定点指令。为了安全起见，PX4要求必须以高于2Hz的频率发送Offboard心跳信号，否则会自动退出该模式。

关键知识点3：无人机集群控制架构

本实验采用主从式控制架构，由一个主节点(SwarmCommander)负责协调管理多个无人机实例。每个无人机实例都有自己的通信接口(SingleDroneInterface)，封装了与特定无人机通信所需的发布者和订阅者。

关键知识点4：px4_swarm_control.py程序详解

px4_swarm_control.py是本实验的核心控制程序，实现了四旋翼无人机集群的控制功能。程序主要包含两个类：SingleDroneInterface和SwarmCommander。

SingleDroneInterface类

SingleDroneInterface类是单个无人机的控制接口，负责与特定无人机进行通信：

```
class SingleDroneInterface:
    """
    单机控制接口类：负责与某一特定命名空间（如 /px4_1）的飞机通信
    """
    def __init__(self, node: Node, drone_id: int):
        self.node = node
        self.drone_id = drone_id
        # 构造命名空间前缀，例如 "px4_1"
        # 注意：PX4 官方多机脚本通常生成的 topic 格式为 /px4_{id}/fmu/...
        # self.ns_prefix = f"/px4_{drone_id}"
        if drone_id == 0:
            # Instance 0 (第一架飞机) 不使用命名空间前缀
            self.ns_prefix = ""
        else:
            # 其他飞机使用 /px4_{id} 前缀
            # 注意：这里我们使用 /px4_{drone_id}，但前面不带斜杠，稍后构造话题时再加
            self.ns_prefix = f"px4_{drone_id}"
```

该类根据无人机ID确定对应的命名空间，其中ID为0的无人机不使用命名空间前缀，而其他无人机则使用"px4_{id}"格式的命名空间。

在初始化过程中，该类创建了必要的发布者和订阅者：

```
# 心跳包：告诉 PX4 "我还活着，别切回手动模式"
self.pub_offboard_mode = node.create_publisher(
    OffboardControlMode,
    f'{self.ns_prefix}/fmu/in/offboard_control_mode',
    qos_profile)

# 轨迹设定点：发送位置/速度指令
self.pub_trajectory = node.create_publisher(
    TrajectorySetpoint,
    f'{self.ns_prefix}/fmu/in/trajectory_setpoint',
    qos_profile)

# 车辆命令：用于解锁、更改模式
self.pub_vehicle_command = node.create_publisher(
    VehicleCommand,
    f'{self.ns_prefix}/fmu/in/vehicle_command',
    qos_profile)
```

同时，也创建了用于接收无人机状态和位置信息的订阅者：

```
# 获取当前状态（是否解锁，是否在 Offboard）
self.sub_status = node.create_subscription(
    VehicleStatus,
    f'{self.ns_prefix}/fmu/out/vehicle_status',
    self.status_callback,
    qos_profile)

# 获取当前位置（用于闭环控制或编队计算）
self.sub_odometry = node.create_subscription(
    VehicleOdometry,
    f'{self.ns_prefix}/fmu/out/vehicle_odometry',
    self.odometry_callback,
    qos_profile)
```

该类还提供了几个重要的控制方法：

```python
def publish_offboard_heartbeat(self):
    """发送 Offboard 心跳，必须 > 2Hz"""
    msg = OffboardControlMode()
    msg.position = True
    msg.velocity = False
    msg.acceleration = False
    msg.attitude = False
    msg.body_rate = False
    msg.timestamp = int(self.node.get_clock().now().nanoseconds / 1000)
    self.pub_offboard_mode.publish(msg)

def send_position_setpoint(self, x, y, z, yaw=0.0):
    """发送 NED 坐标系下的位置指令"""
    msg = TrajectorySetpoint()
    msg.position = [float(x), float(y), float(z)]
    msg.yaw = float(yaw)
    msg.timestamp = int(self.node.get_clock().now().nanoseconds / 1000)
    self.pub_trajectory.publish(msg)
```

SwarmCommander类

SwarmCommander类是集群控制的主节点，负责协调管理多个无人机：

```python
class SwarmCommander(Node):
    """
    集群指挥官：负责调度多架飞机
    """
    def __init__(self):
        super().__init__('swarm_commander')

        # 定义集群包含的飞机 ID 列表
        self.drone_ids = [0, 1, 2, 3]
        # self.drone_ids = [1, 2, 3, 4, 5, 6, 7, 8]
        self.drones = []

        # 初始化每架飞机的接口
        for did in self.drone_ids:
            self.drones.append(SingleDroneInterface(self, did))

        # 创建主控制循环 (20Hz)
        self.timer = self.create_timer(0.05, self.control_loop)
        self.tick_count = 0

        print("集群节点已启动，等待 Offboard 信号流建立...")
```

主控制逻辑在control_loop方法中实现：

```python
def control_loop(self):
    self.tick_count += 1

    # 1. 必须先给所有飞机发心跳，否则无法切入 Offboard
    for drone in self.drones:
        drone.publish_offboard_heartbeat()

    # 2. 逻辑状态机 (示例)
    if self.tick_count == 20: # 1秒后
        print(">>> 尝试切换到 Offboard 模式并解锁")
        for drone in self.drones:
            drone.set_offboard_mode()
            drone.arm()

    elif self.tick_count > 100: # 5秒后开始执行任务
        # 示例：让飞机排成一排起飞
        # NED坐标系：Z轴向下为正，所以 -5.0 是向上飞 5 米

        for i, drone in enumerate(self.drones):
            # 目标：高度 -5m，Y轴每架飞机间隔 2m
            target_x = 0.0
            target_y = i * 2.0
            target_z = -5.0

            drone.send_position_setpoint(target_x, target_y, target_z)
```

控制循环以20Hz频率运行，首先向所有无人机发送Offboard心跳信号，然后在适当时机发送模式切换和解锁命令，最后控制无人机按预定位置飞行。

6.参考资料

此处编写参考资料，编写样式如下：

1. 《PX4 Developer Guide》
2. 《ROS 2 Documentation》
3. 《PX4-Autopilot Source Code》
4. 《RflySim》

7.常见问题

Q1：无人机无法起飞或者飞行不稳定

A1：可能原因有以下几个方面：

1. 未正确切换到Offboard模式，确保已经运行WinWSL_StartuXRCE.bat脚本并启动MicroXRCEAgent；
2. Offboard心跳信号发送频率过低，程序中设定为20Hz，如果频率低于2Hz会导致PX4退出Offboard模式；
3. GPS信号不佳，等待CopterSim日志中出现"GPS 3D fixed & EKF initialization finished"后再尝试起飞；
4. 无人机未成功解锁，检查是否执行了arm()命令。

Q2：ROS2节点无法与PX4通信

A2：这类问题通常由以下原因导致：

1. DDS配置不正确，确保运行Ros2ComMode.bat脚本并将模式设置为"CycloneDDS+广播模式"；
2. MicroXRCEAgent未启动或端口配置错误，确认WinWSL_StartuXRCE.bat脚本正确执行；
3. 命名空间配置错误，检查px4_swarm_control.py中关于无人机ID和命名空间的配置是否与仿真环境一致；
4. WSL版本问题，确认系统使用的是WSL2而不是WSL1。

Q3：编译过程中出现依赖库缺失错误

A3：这通常是由于缺少必要的ROS2功能包导致的。解决方法包括：

1. 确保已正确安装PX4_ROS功能包，运行setup_px4_ros.sh脚本进行安装；
2. 检查extract_and_copy.bat脚本是否完整执行，确认所有必需文件都已复制到正确位置；
3. 在WSL中手动安装缺失的功能包，使用sudo apt install ros-humble-命令；
4. 清理之前的编译缓存，使用colcon build --cmake-clean-cache重新编译。

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