1.实验名称及目的

1.1 实验名称

使用 MAVROS C++ 进行 OFFBOARD 控制实验

1.2 实验目的

运行MAVROS的C++程序将飞行器切换到offboard模式并解锁，然后将其位置设定为固定的坐标点。

1.3 关键知识点

先运行一个python程序，打开mavros主节点，将mavlink消息转化成ros消息,然后，再运行一个mavros 的C++程序，进行mavros控制。

首先创建一个ROS包，创建 ROS 包和 scripts 文件夹后。在 scripts 文件夹中，创建名为offb_node.cpp的c++文件并授予其可执行权限。在 offboard_py 包中，在 ~/catkin_ws/src/offboard_py/src 目录中创建一个名为 launch的文件夹。在launch文件夹里创建一个名为start_offb.launch的启动文件。

本实验主要是实现通过Python接口RflyRosStart.py（见RflySimAPIs\RflySimSDK\ctrl目录），展示用Python mavros进行飞机控制。关键代码解析如下：

本例子和其他分布式例子的区别，主要在于使用mavros进行飞机控制，其中控制代码是c++编写的，采用c++语言进行控制。不需要启动控制文件，编译完成后，利用roslaunch的方式启动控制飞机程序。

关键知识点1：通过ReqCopterSim可以自动从局域网获取到仿真电脑的IP地址，从而自动建立连接，不再需要手动指定IP地址。不过，此种连接方式，可能在局域网中产生干扰（多台电脑同时打开多个CopterSim会产生误识别），不适合多个实验同时进行的场景。

1） ReqCopterSim接口使用（自动获取ip接口）

```
req = ReqCopterSim.ReqCopterSim() # 获取局域网内所有CopterSim程序的电脑IP列表

TargetIP = req.getSimIpID(StartCopterID) #自动获取CopterSim的StartCopterID号程序所在电脑的

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

req.sendReSimUdpMode(TargetID,2) \# 强制切换MAVLINK_FULL
```

2）ros/ros.h使用

```
ros::init(argc, argv, "offb_node"); \# 创建节点

ros::NodeHandle nh; \# 创建句柄

ros::Subscriber state_sub = nh.subscribe\<mavros_msgs::State\>("mavros/state",10, sta

ros::Publisher local_pos_pub = nh.advertise<geometry_msgs::PoseStamped>("mavros/setpo
10); #发布mavros/setpoint_position/local话题消息，消息类型是PoseStamped，限制排队的消息数量为

ros::ServiceClient arming_client = nh.serviceClient<mavros_msgs::CommandBool>("mavros

ros::Rate rate(20.0); \# 以 20 赫兹（20 次每秒）的频率运行代码

local_pos_pub.publish(pose); \#发布pose消息，这个消息属于前面说的mavros/setpoint_position/

ROS_INFO("Offboard enabled"); \# 终端打印OFFBOARD enabled信息

set_mode_client.call(offb_set_mode) &&offb_set_mode.response.mode_sent \# 向mavros/se
```

3）其余代码说明

```
ros = RflyRosStart.RflyRosStart(TargetID,TargetIP) \#创建ros实例，也就是发起roscore

mavros_msgs::SetMode offb_set_mode \# offb_set_mode设置为SetMode 消息类型
```

2.实验效果

3.文件目录

文件夹/文件名称	说明
offb_node	C++版本ROS控制测试文件夹
C++RosLearn.bat	软件在环一键启动脚本
Python38Run.bat	Windows下Python程序运行脚本
WinWSL.bat	WSL1/Ubuntu 20.04环境程序运行脚本
WslGUI.bat	WSL1/Ubuntu 20.04可视化界面脚本

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/doc/zh/HowToInstall.pdf

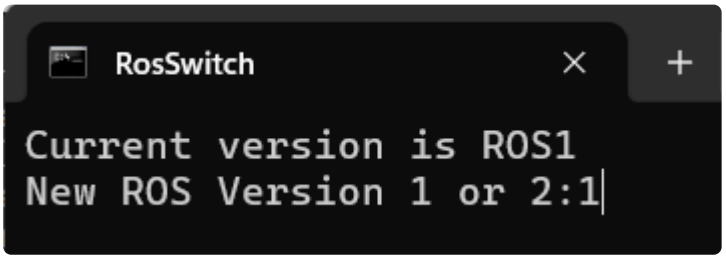
5.实验步骤

5.1 必做实验：WinsWSL控制

Step 1：WSL里编译工作空间

1.在ubuntu20.04里构建工作空间。

双击打开"*\桌面\RflyTools\RosSwitch.lnk"，确认切换到ROS 1环境下。



双击打开"*\桌面\RflyTools\WslGUI.lnk”，启动WSL可视化界面。

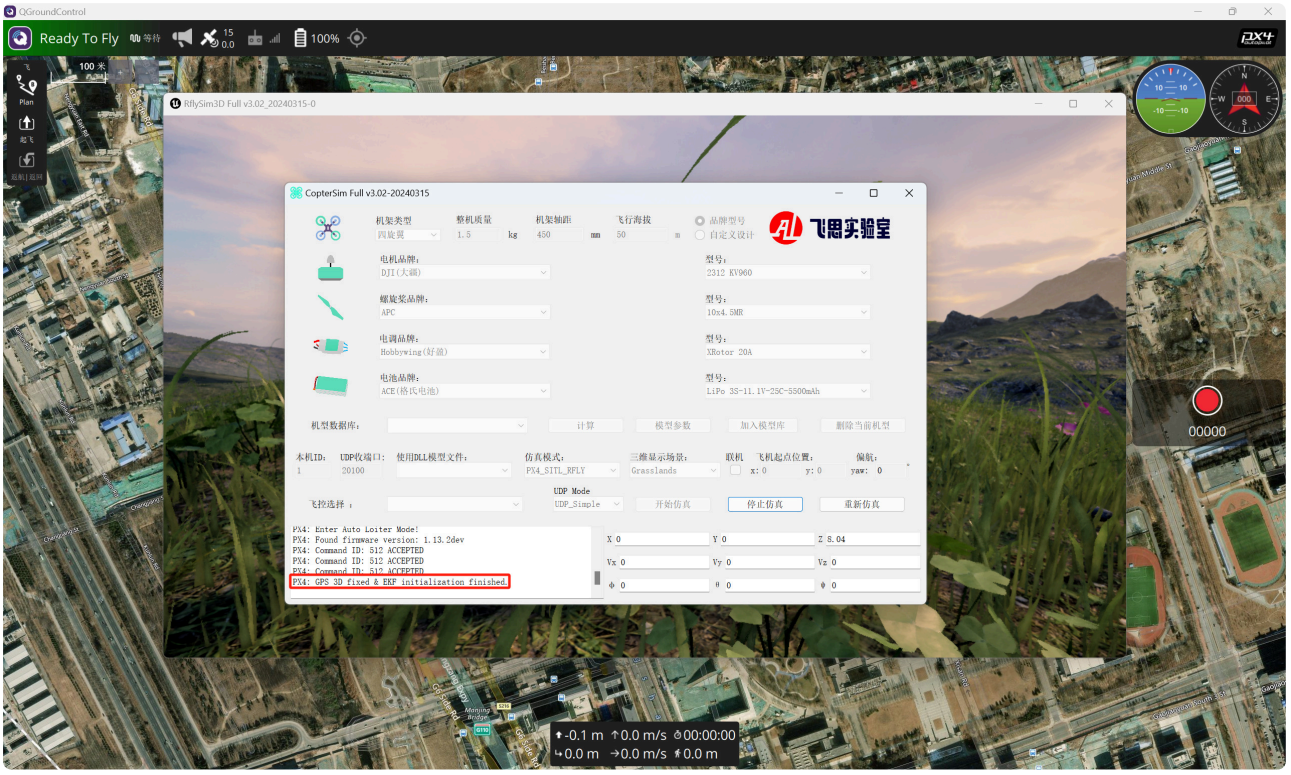
打开终端运行命令：`mkdir -p catkin_ws/src`

2.把文件offb_node拷贝到 catkin_ws/src里面

3.编译：编译之前请确保WSL中的ROS环境为ROS Noetic，可运行"*\桌面\RflyTools\RosSwitch.lnk"进行切换。打开终端进入到catkin_ws目录下，运行命令 catkin_make 进行编译;

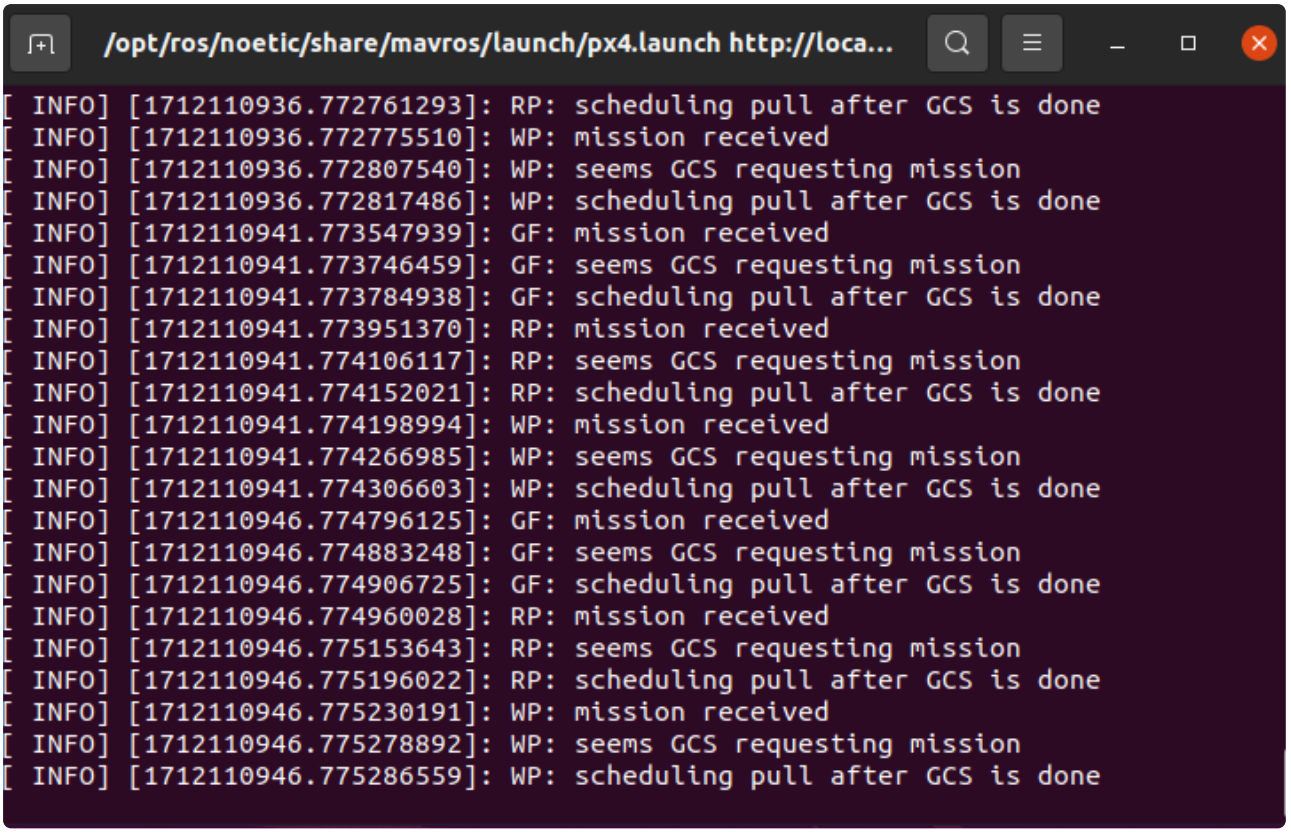
Step 2：开启仿真

在 windows端双击运行C++RosLearn.bat脚本，等待CopterSim日志打印初始化完成。

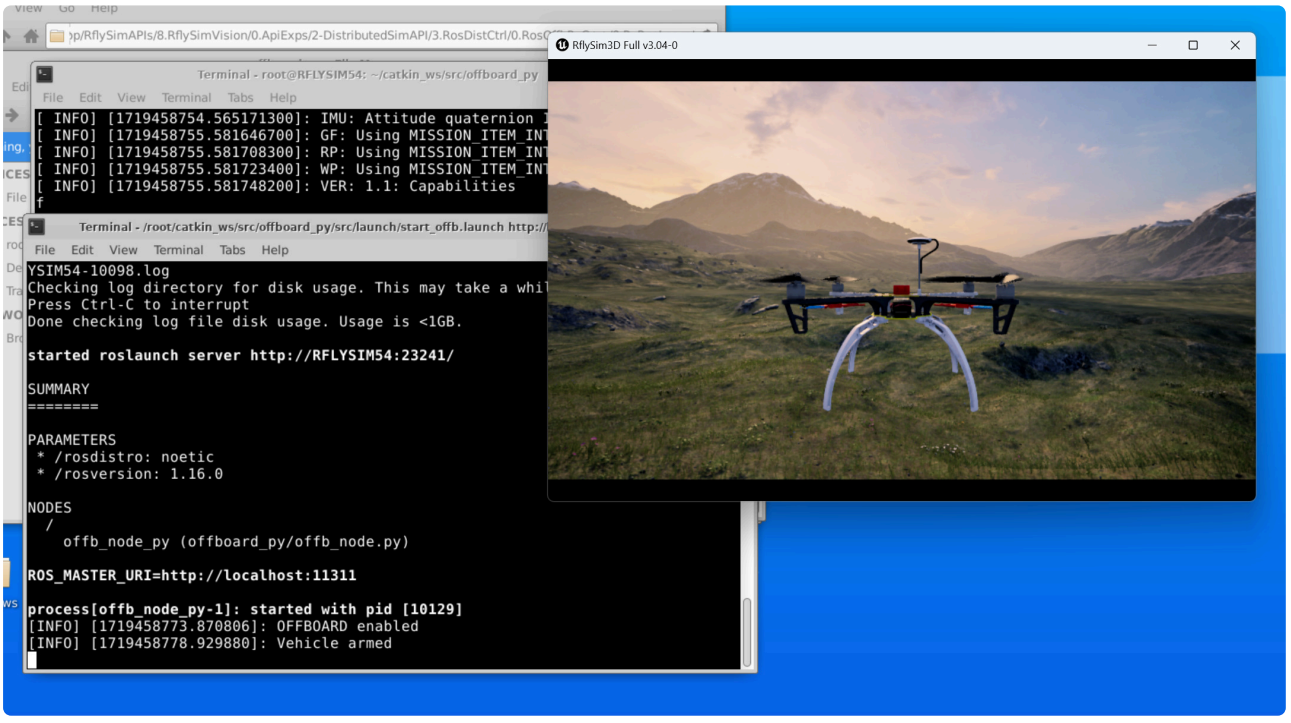


Step 3：运行控制程序

1.打开终端进入到catkin_ws/src/offboard_py目录下，运行`python3 mavros.py`程序启动mavros。



2.打开终端，切换到catkin_ws目录下，运行`source devel/setup.bash`，再运行`roslaunch offb_node offb_node.launch`，可以看到进入offboard，飞机起飞。



5.2. 选作实验

准备工作：

虚拟机或NX的配置方法是相同的。

1）Ubuntu虚拟机环境下，进行分布式联机实验。先参考
[安装目录]\RflySimAPIs\8.RflySimVision\0.ApiExps\0.Preparation\1.VMwareUbuntu\Readme.pdf
，完成虚拟机的下载与配置。

2）用第二台Ubuntu电脑或NX板卡，实现联机实验。其他Ubuntu电脑的配置，先看

[安装目录]\RflySimAPIs\8.RflySimVision\0.ApiExps\0.Preparation\2.GenenralUbuntuConfig\Readme.pdf
；NX板卡的配置方法，先看
[安装目录]\RflySimAPIs\8.RflySimVision\0.ApiExps\0.Preparation\3.NXwithPX4Config\Readme.pdf
。

扩展实验：

5.2.1在虚拟机/视觉板卡/另一台Ubuntu上接收图像实验

Step 1：虚拟机里编译工作空间

1.在ubuntu20.04里构建工作空间。

打开虚拟机，进入虚拟机界面。

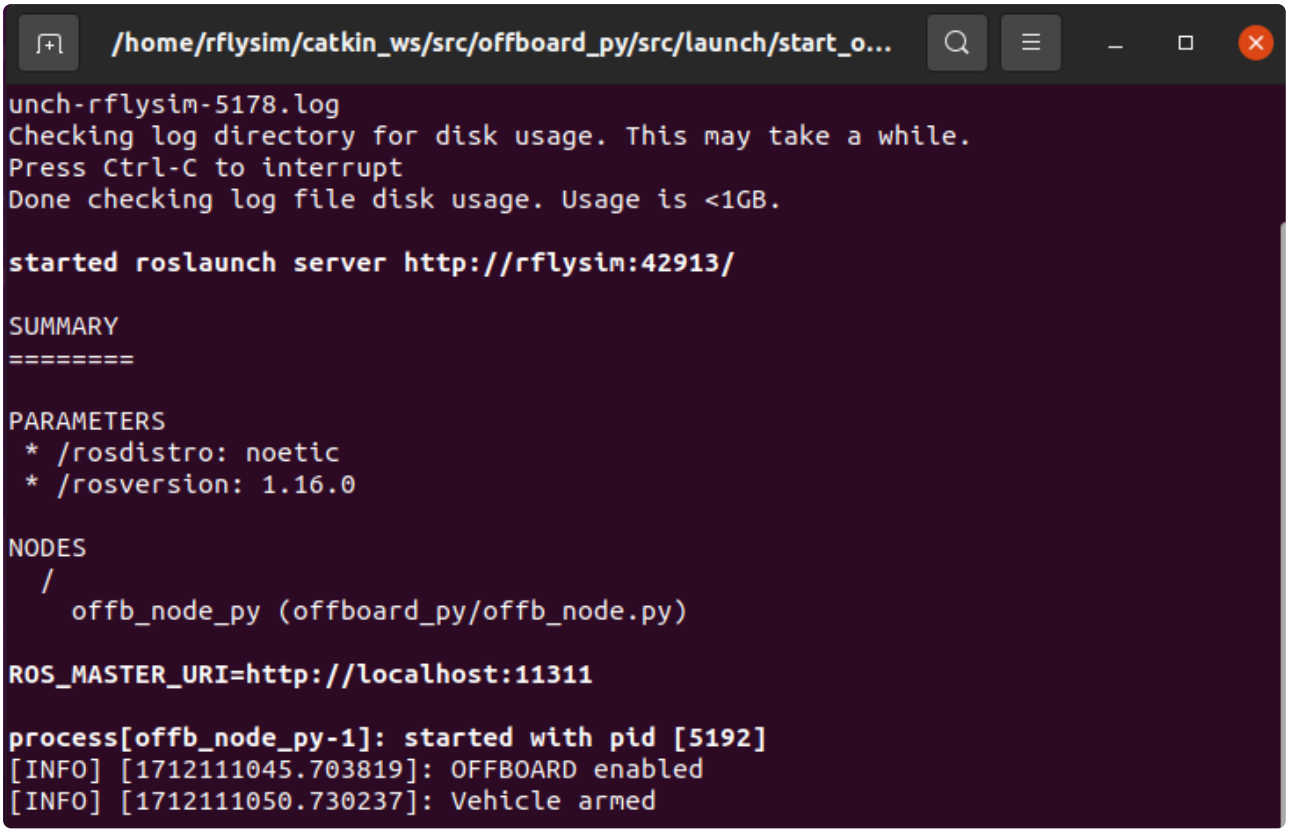
打开终端运行命令: `mkdir -p catkin_ws/src`

2.把文件offboard_py拷贝到 catkin_ws/src里面

3.编译：打开终端进入到catkin_ws 目录下，运行命令 `catkin_make` 进行编译;

Step 2：运行控制程序

步骤2同上面的Step3步骤。



6.参考资料

7. C++代码解释：

mavros_msgs 功能包中包含了MAVROS包服务和主题所需的全部自定义消息文件。
所有服务和主题及其相应的消息类型都可以在mavros wiki (opens new window)中找到。

```
#include <ros/ros.h>

#include <geometry_msgs/PoseStamped.h>

#include <mavros_msgs/CommandBool.h>

#include <mavros_msgs/SetMode.h>

#include <mavros_msgs/State.h>
```

我们创建了一个简单的回调函数来储存飞控当前的状态。这样我们就可以检查连接状态，是否解锁以及_Offboard_ 标志位。

```
mavros_msgs::State current_state;

void state_cb(const mavros_msgs::State::ConstPtr& msg){

current_state = *msg;

}

```

我们构建了一个发布者来发布本地位置指令并请求客户端进行加解锁状态及控制模式的切换。请注意，对于您自己的系统，"mavros"前缀可能不同，取决于节点启动文件中指定的名称。

```
ros::Subscriber state_sub = nh.subscribe<mavros_msgs::State>("mavros/state",10, state

ros::Publisher local_pos_pub = nh.advertise<geometry_msgs::PoseStamped>("mavros/setpo
10);

ros::ServiceClient arming_client = nh.serviceClient<mavros_msgs::CommandBool>("mavros

ros::ServiceClient set_mode_client = nh.serviceClient<mavros_msgs::SetMode>("mavros/s
```

PX4 在两个 *Offboard* 命令之间设置了500毫秒的超时检查。如果超时，飞控会立即切换回进入 *Offboard* 模式之前的飞行模式。这也是为什么发布频率 **必须** 大于2Hz的原因。 这样是为什么我们推荐从 *位置*模式进入 *offboard* 模式，因为如果飞控掉出 *Offboard*模式，无人机会悬停于当前位置。

```
//the setpoint publishing rate MUST be faster than 2Hz
ros::Rate rate(20.0);在发布任何消息之前，我们需要等待飞控和MAVROS建立连接。在收到心跳包之后，代码
geometry_msgs::PoseStamped pose;
pose.pose.position.x = 0;
pose.pose.position.y = 0;
pose.pose.position.z = 2;
```

尽管PX4在航空航天常用的NED坐标系下操控飞机，但MAVROS将自动将该坐标系切换至常规的ENU坐标系下，反之亦然。这也就是为什么我们设置z为+2。

```
//send a few setpoints before starting

for(int i = 100; ros::ok() && i \> 0; --i){

local_pos_pub.publish(pose);

ros::spinOnce();

rate.sleep();

}
```

在进入 Offboard 模式之前，您必须已经开始流式传输设定点。否则，模式切换将被拒绝。这里的100 可以被设置为任意数。

```
mavros_msgs::SetMode offb_set_mode;
offb_set_mode.request.custom_mode = "OFFBOARD";
```

该代码的其余部分完全是自解释性的。我们尝试切换到离机模式，然后我们武装四边形以使其飞行。我们每隔五秒去调用一次该服务，避免飞控被大量的请求阻塞。在同一个循环中，我们按照指定的频率持续发送期望点设定值信息给飞控。

```
mavros_msgs::CommandBool arm_cmd;

arm_cmd.request.value = true;

ros::Time last_request = ros::Time::now();

while(ros::ok()){

if( current_state.mode != "OFFBOARD" &&

(ros::Time::now() - last_request \> ros::Duration(5.0))){

if( set_mode_client.call(offb_set_mode) &&

offb_set_mode.response.mode_sent){

ROS_INFO("Offboard enabled");

}

last_request = ros::Time::now();

} else {

if( !current_state.armed &&

(ros::Time::now() - last_request \> ros::Duration(5.0))){

if( arming_client.call(arm_cmd) &&

arm_cmd.response.success){

ROS_INFO("Vehicle armed");

}

last_request = ros::Time::now();

}

}

local_pos_pub.publish(pose);

ros::spinOnce();

rate.sleep();

}
```

7.常见问题

Q1：无

A1：无