1.实验名称及目的

1.1 实验名称

使用 MAVROS Python 进行 OFFBOARD 控制实验

1.2 实验目的

运行MAVROS的Python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

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.py的python 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，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）rospy接口使用

```
rospy.init_node("offb_node_py") # 创建节点

state_sub = rospy.Subscriber("mavros/state", State, callback=state_cb) #订阅mavros/sta

local_pos_pub = rospy.Publisher("mavros/setpoint_position/local", PoseStamped,queue_s

rospy.wait_for_service("/mavros/cmd/arming") # 等待/mavros/cmd/arming服务

arming_client = rospy.ServiceProxy("mavros/cmd/arming", CommandBool) #在mavros/cmd/ar

rate = rospy.Rate(20) # 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/lo

rospy.loginfo("OFFBOARD enabled") # 终端打印OFFBOARD enabled信息

set_mode_client.call(offb_set_mode).mode_sent # 向 /mavros/set_mode服务发送请求，尝试将无
```

3）其余代码说明

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

offb_set_mode = SetModeRequest() # offb_set_mode设置为SetModeRequest()服务类型
```

2.实验效果

3.文件目录

文件夹/文件名称	说明
Offboard_py	Python版本ROS控制测试文件夹
PyRosLearn.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WslGUI.bat，启动WSL可视化界面。

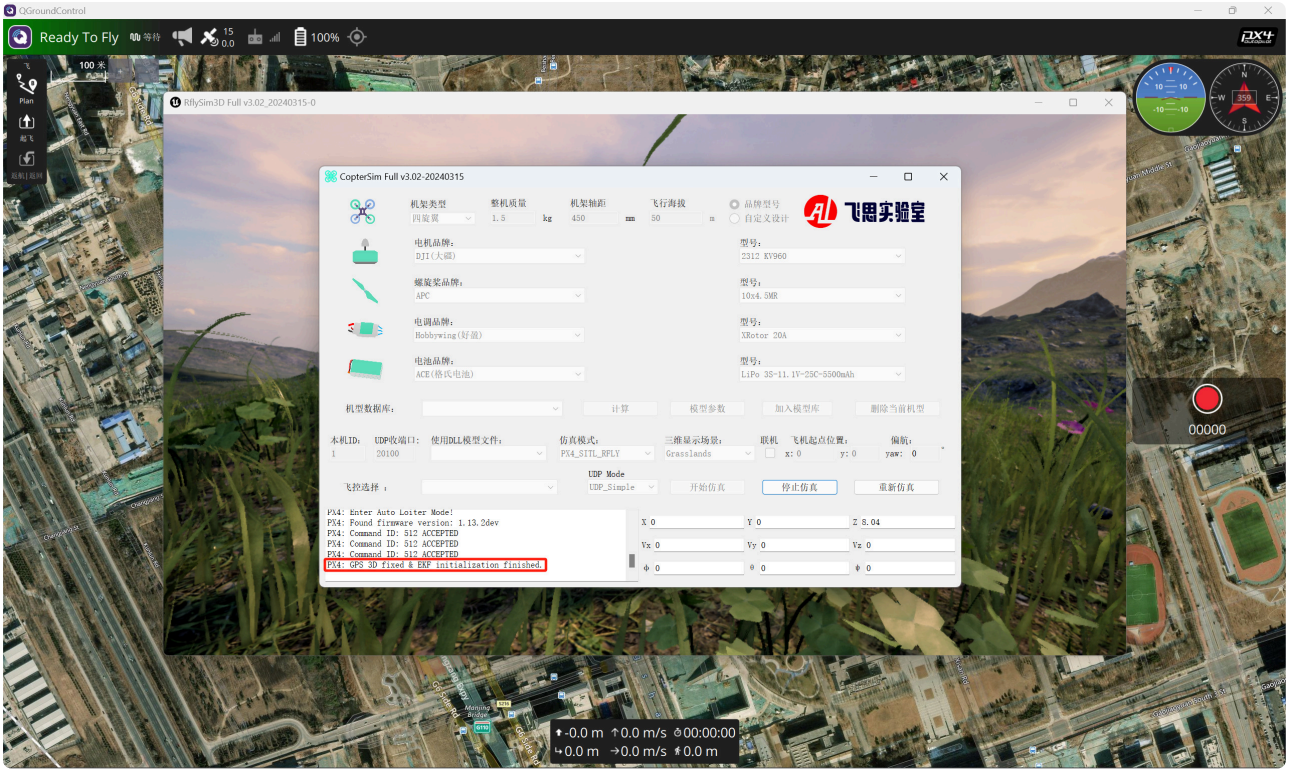
打开终端运行命令: `mkdir -p catkin_ws/src`

2.把文件offboard_py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;

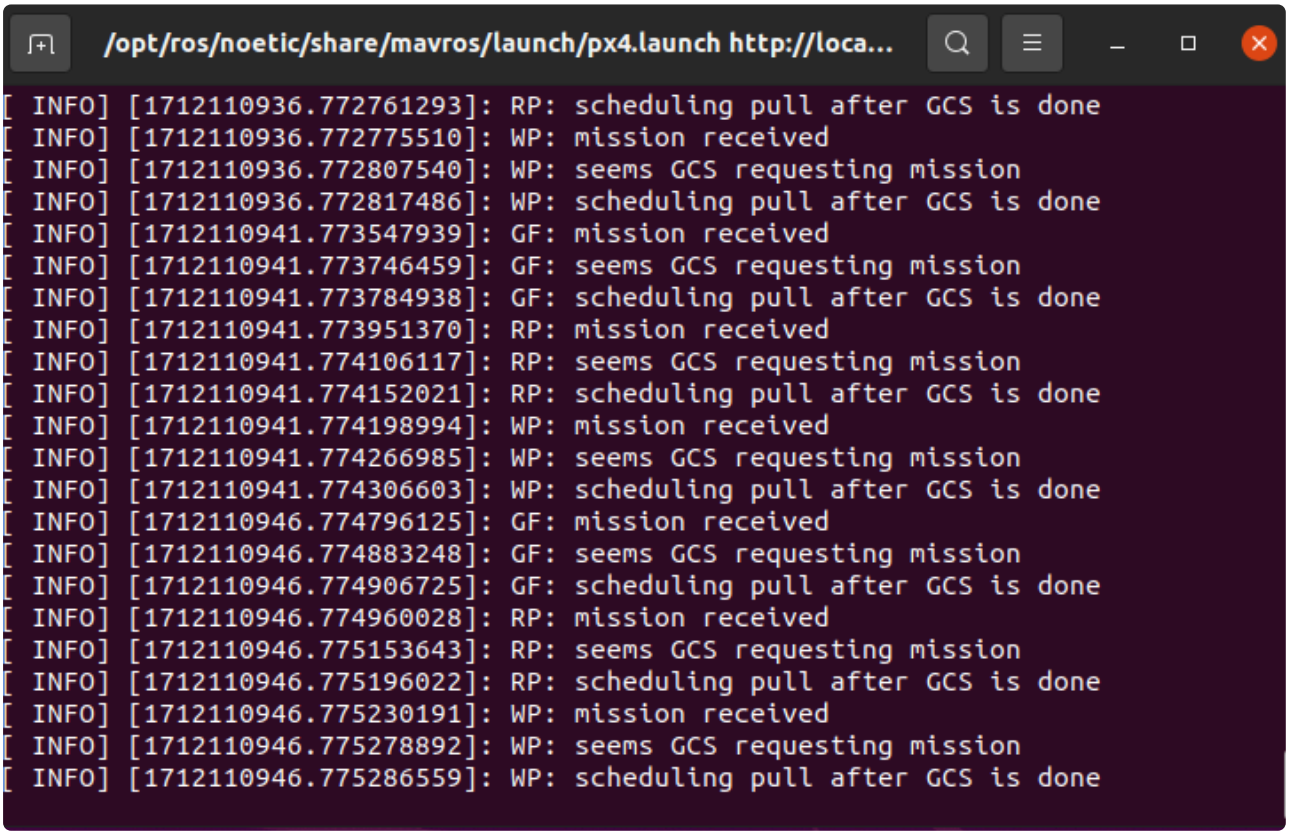
Step 2：开启仿真

1.在 windows端双击运行PyRosLearn.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。

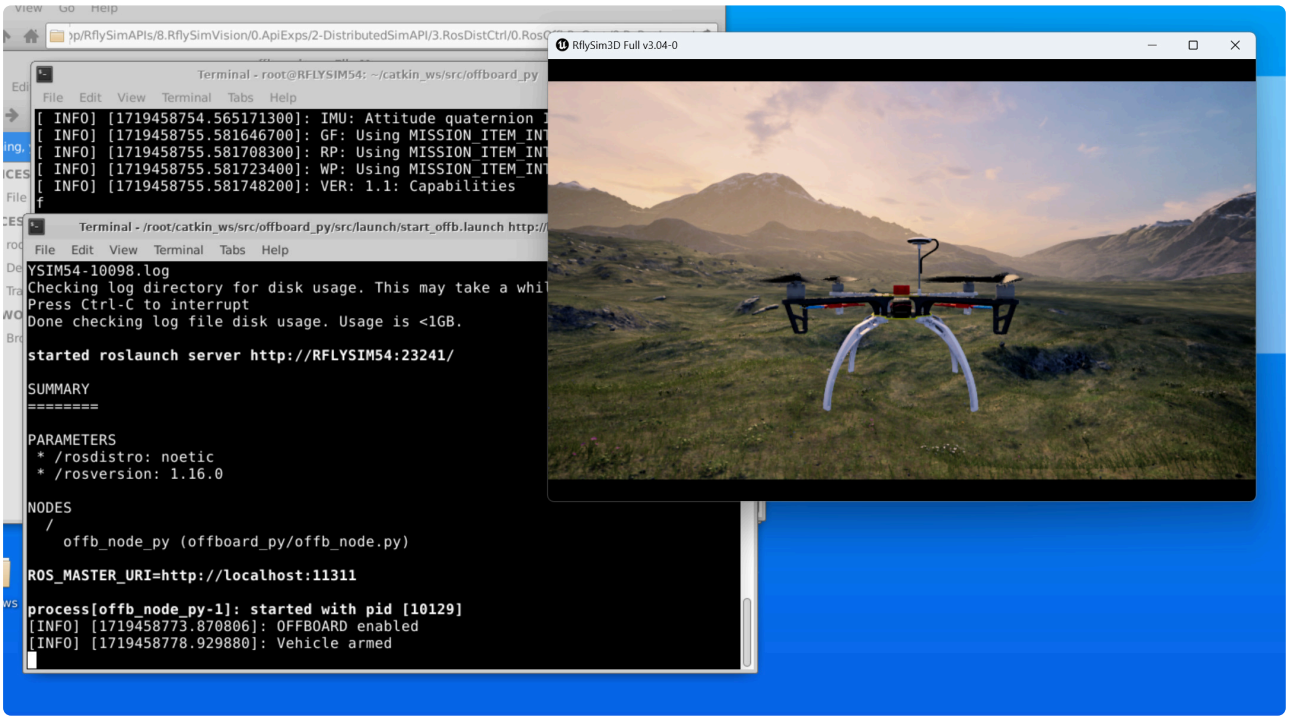


2. 安装 dos2unix 工具（如果尚未安装），打开终端，运行`sudo apt-get install dos2unix`。

3. 然后转换文件格式，切换到catkin_ws目录下，打开终端，运行`dos2unix src/offboard_py/scripts/offb_node.py`。

```
root@RFLYSIM54:~/catkin_ws# sudo apt-get install dos2unix
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages were automatically installed and are no longer required:
  libdc1394-25 usb-modeswitch usb-modeswitch-data
Use 'sudo apt autoremove' to remove them.
The following NEW packages will be installed:
  dos2unix
0 upgraded, 1 newly installed, 0 to remove and 36 not upgraded.
Need to get 374 kB of archives.
After this operation, 1342 kB of additional disk space will be used.
Get:1 https://mirrors.tuna.tsinghua.edu.cn/ubuntu focal/universe amd64 dos2unix
amd64 7.4.0-2 [374 kB]
Fetched 374 kB in 1s (549 kB/s)
Selecting previously unselected package dos2unix.
(Reading database ... 215319 files and directories currently installed.)
Preparing to unpack .../dos2unix_7.4.0-2_amd64.deb ...
Unpacking dos2unix (7.4.0-2) ...
Setting up dos2unix (7.4.0-2) ...
Processing triggers for man-db (2.9.1-1) ...
root@RFLYSIM54:~/catkin_ws#
```

4. 再运行`source devel/setup.bash`，再运行`roslaunch offboard_py start_offb.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。

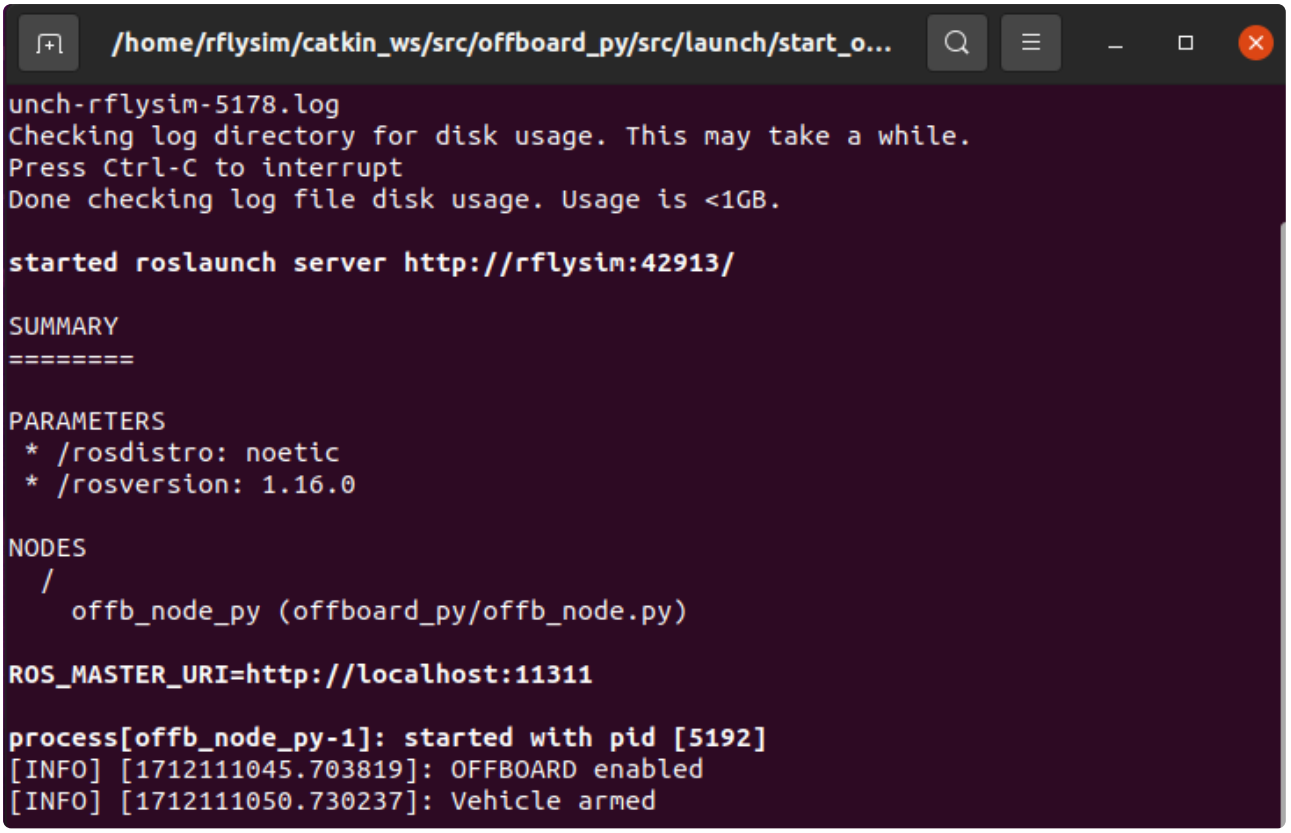
打开终端运行命令: `mkdir -p catkin_ws/src`

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

python代码解释：

mavros_msgs包包含操作 MAVROS
包提供的服务和主题所需的所有自定义消息。所有服务和主题及其相应的消息类型都记录在

mavros wiki (opens new window) 中。

```
import rospy

from geometry_msgs.msg import PoseStamped

from mavros_msgs.msg import State

from mavros_msgs.srv import CommandBool, CommandBoolRequest, SetMode,SetModeRequest
```

我们创建一个简单的回调，它将保存自动驾驶仪的当前状态。这将允许我们检查连接、布防和OFFBOARD 标志。

```
current_state = State()

def state_cb(msg):

global current_state

current_state = msg
```

我们实例化发布者以发布命令的本地位置，并实例化相应的客户端以请求布防和模式更改。请注意，对于您自己的系统，“mavros”前缀可能会有所不同，因为它取决于其启动文件中为节点指定的名称。

```
state_sub = rospy.Subscriber("mavros/state", State, callback = state_cb)

local_pos_pub = rospy.Publisher("mavros/setpoint_position/local", PoseStamped,queue_s

rospy.wait_for_service("/mavros/cmd/arming")

arming_client = rospy.ServiceProxy("mavros/cmd/arming", CommandBool)

rospy.wait_for_service("/mavros/set_mode")

set_mode_client = rospy.ServiceProxy("mavros/set_mode", SetMode)
```

PX4 在两个 OFFBOARD 命令之间的超时时间为 500毫秒。如果超过此超时，指挥官将回退到车辆进入 OFFBOARD模式之前的最后一个模式。这就是为什么发布速率必须快于 2 Hz才能考虑可能的延迟。这也是为什么建议从位置模式进入 OFFBOARD模式的原因，这样如果车辆退出 OFFBOARD 模式，它将停止在轨道上并悬停。

在这里，我们适当地设置发布速率：

```
\# Setpoint publishing MUST be faster than 2Hz
rate = rospy.Rate(20)
```

在发布任何内容之前，我们等待 MAVROS和自动驾驶仪之间建立连接。收到检测信号消息后，此循环应立即退出。

```
\# Wait for Flight Controller connection
while(not rospy.is_shutdown() and not current_state.connected):
    rate.sleep()
```

尽管 PX4 在航空航天 NED 坐标系中运行，但 MAVROS 将这些坐标转换为标准 ENU坐标系，反之亦然。这就是为什么我们将 z 设置为正 2：

```
pose = PoseStamped()pose.pose.position.x = 0
pose.pose.position.y =0pose.pose.position.z = 2
```

在进入 OFFBOARD模式之前，您必须已经开始流式传输设定点。否则，模式开关将被拒绝。下面，选择 100作为任意数量。

```
\# Send a few setpoints before startingfor i in range(100):
if(rospy.is_shutdown()):
    break
local_pos_pub.publish(pose)
rate.sleep()
```

我们准备用于将自定义模式设置为 OFFBOARD 的消息请求。支持的模式列表可供参考。

```
offb_set_mode = SetModeRequest()
offb_set_mode.custom_mode = 'OFFBOARD'
```

代码的其余部分在很大程度上是不言自明的。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。我们将服务调用间隔5秒，以免请求淹没自动驾驶仪。在同一循环中，我们继续以先前定义的速率发送请求的姿势。

```
arm_cmd = CommandBoolRequest()

arm_cmd.value = True

last_req = rospy.Time.now()

while(not rospy.is_shutdown()):

    if(current_state.mode != "OFFBOARD" and (rospy.Time.now() - last_req) rospy.Durat

        if(set_mode_client.call(offb_set_mode).mode_sent == True):

            rospy.loginfo("OFFBOARD enabled")

            last_req = rospy.Time.now()

    else:

        if(not current_state.armed and (rospy.Time.now() - last_req) rospy.Duration(5

            if(arming_client.call(arm_cmd).success == True):

                rospy.loginfo("Vehicle armed")

                last_req = rospy.Time.now()

                local_pos_pub.publish(pose)

                rate.sleep()
```

7.常见问题

Q1：无

A1：无