

## 1. 实验名称及目的

### i. 实验名称

飞机组网实验

## 2. 实验目的

飞机在飞行过程中获取其他飞机的飞行状态信息。

## 3. 关键知识点

以1号飞机为例，直接指定了发往2 3 4号飞机的端口 60002 60003 60004，同时2 3 4号飞机也都指定了会发往 60001号端口（被1号飞机监听）。因此，每个飞机都能收到其他三个飞机的数据。

## 2. 实验效果

可以观察到飞机依次飞出。增加监听指令后，可以看到飞机收到其他飞机的信息。

## 3. 文件目录

例程目录：

[[安装目录](#)]\RflySimAPIs\9.RflySimComm\3.CustExps\e2.NS3\_UAVCommExps\3.NS3-PyViz\uav

文件夹/文件名称	说明
Python38Run	仿真配置文件
SITLRun4MavlinkFull	软件在环仿真配置文件
UAV1Ctrl	飞机控制程序

## 4. 运行环境

序号	软件要求	硬件要求	
		名称	数量
1	Windows 10及以上版本	笔记本/台式电脑 <sup>①</sup>	1
2	RflySim工具链		

## 5. 实验步骤

## Step 1:

运行SITLRun4UDPSimple.bat会创建四个飞机，双击Python38Run.bat四次，会创建四个命令提示框，在四个Python命令提示框中，分别输入下面四条指令的1条（现阶段只输入，不要回车）

```
python UAV1Ctrl.py
```

```
python UAV2Ctrl.py
```

```
python UAV3Ctrl.py
```

```
python UAV4Ctrl.py
```



```
C:\Windows\system32\cmd.exe
Python3.8 environment has been set with openCV+pymavlink+numpy+pyulog etc.
You can use pip or pip3 command to install other libraries
Put your python scripts 'XXX.py' into the folder 'C:\Users\dream\Desktop\PythonNetSimAPI'
Use the command: 'python XXX.py' to run the script with Python
C:\Users\dream\Desktop\PythonNetSimAPI>python UAV1Ctrl.py

C:\Windows\system32\cmd.exe
Python3.8 environment has been set with openCV+pymavlink+numpy+pyulog etc.
You can use pip or pip3 command to install other libraries
Put your python scripts 'XXX.py' into the folder 'C:\Users\dream\Desktop\PythonNetSimAPI'
Use the command: 'python XXX.py' to run the script with Python
C:\Users\dream\Desktop\PythonNetSimAPI>python UAV3Ctrl.py

C:\Windows\system32\cmd.exe
Python3.8 environment has been set with openCV+pymavlink+numpy+pyulog etc.
You can use pip or pip3 command to install other libraries
Put your python scripts 'XXX.py' into the folder 'C:\Users\dream\Desktop\PythonNetSimAPI'
Use the command: 'python XXX.py' to run the script with Python
C:\Users\dream\Desktop\PythonNetSimAPI>python UAV2Ctrl.py

C:\Windows\system32\cmd.exe
Python3.8 environment has been set with openCV+pymavlink+numpy+pyulog etc.
You can use pip or pip3 command to install other libraries
Put your python scripts 'XXX.py' into the folder 'C:\Users\dream\Desktop\PythonNetSimAPI'
Use the command: 'python XXX.py' to run the script with Python
C:\Users\dream\Desktop\PythonNetSimAPI>python UAV4Ctrl.py
```

注意：输入"python u"然后按Tab键，可以快速补全

## Step 2:

等待RflySim3D显示所有飞机已fixed，或4号CopterSIM上看到初始化完毕

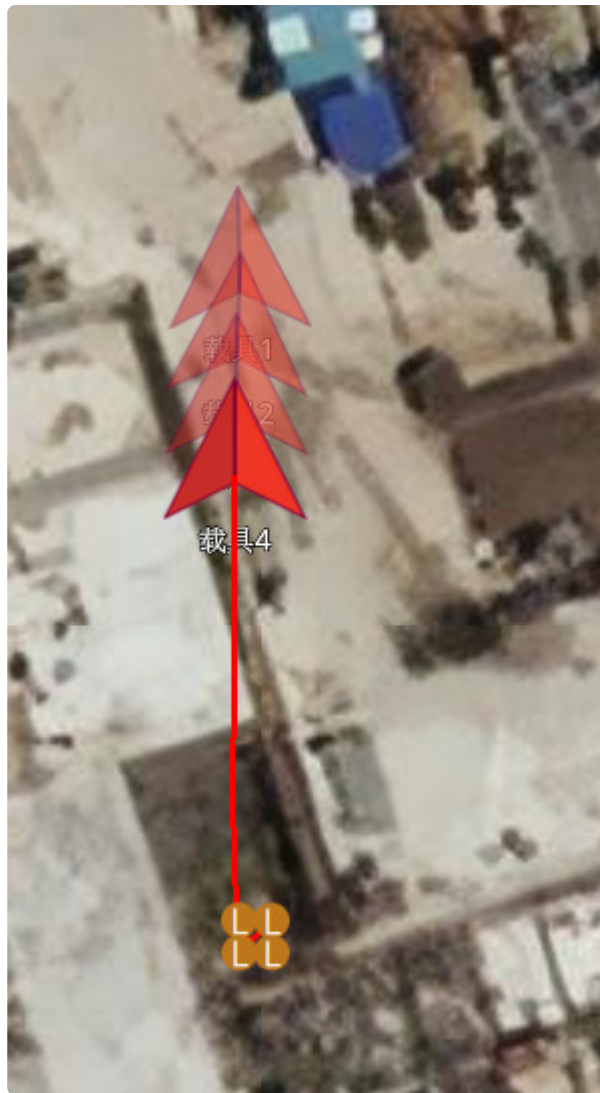
本机ID: 4    UDP收端口: 20106    使用DLL模型文件: [v]    仿真模式: PX4\_SITL\_RFL

飞控选择: [v]    UDP Me: [v]    UDP\_S: [v]

```
PX4: Found firmware version: 1.12.3dev
PX4: Command ID: 512 ACCEPTED
PX4: Command ID: 512 ACCEPTED
PX4: Command ID: 512 ACCEPTED
PX4: GPS 3D fixed & EKF initialization finished.
PX4: Enter Auto Loiter Mode!
```

依次在四个Python命令框中按下回车，执行四个Python程序，可以看到四个飞机一个跟一个地飞出去





在30秒左右，1号飞机会停一下，其他三个飞机依次停止，各自相距1米，停顿10秒后，四个飞机开始依次跟踪向东飞

## 6. 参考资料

主要是修改了PX4MavCtrlV4.py增加了几个函数机制

1) 增加一个事件，使得收到CopterSim的包后，能够通知上层的NetSimAPIV4.py去做数据转发

```
self.netEvent = threading.Event()
```

2) 增加了一个函数，使能事件

```
def netForwardData(self):
```

```
self.netEvent.set()
```

3) 在def getMavMsg(self)函数中，等数据接收并更新完毕后，调用netForwardData函数，通知上层去将数据发出。

其次是增加了一个NetSimAPIV4.py接口类函数

1. 使能方法：要先建立mav实例，再以此建立net实例

```
import PX4MavCtrlV4 as PX4MavCtrl
```

```
import NetSimAPIV4
```

```
#Create a new MAVLink communication instance, UDP sending port (CopterSim's  
receiving port) is 20100
```

```
# 创建#1号飞机的通信实例，和CopterSim#1号相连
```

```
mav = PX4MavCtrl.PX4MavCtrl(20100)
```

```
net = NetSimAPIV4.NetSimAPI(mav)
```

2. 增加了转发使能函数

```
def enNetForward(self,PortList=[60002],targetIP='224.0.0.10'):
```

通过这个接口，可以让本飞机，将自己收到的数据，发送到指定targetIP和端口PortList  
(端口可以指定多个)

注意，我们这里默认使用60000系列端口来作为网路通信接口

相关代码包括

```
# 启用网络转发，将本机收到的消息，发给CopterIDList飞机列表，使用默认60000系列端  
口
```

```
def enNet2UavID(self,CopterIDList=[2]): # 默认只发给2号飞机
```

```
if not isinstance(CopterIDList,list): # 如果是一维标量，就转为数组
```

```
CopterIDList = [CopterIDList]
```

```
PortList=[]
```

```
for i in range(len(CopterIDList)):
```

```
PortList = PortList + [CopterIDList[i]+60000] # 自动填充默认端口规则
```

```
self.enNetForward(PortList)
```

```
# 启用网络转发，将本机收到的消息，转发给targetIP上的PortList系列端口
```

```
# 默认发给2号飞机端口60002，以及组播IP 224.0.0.10

def enNetForward(self,PortList=[60002],targetIP='224.0.0.10'):

self.ForwardIP=targetIP

if not isinstance(PortList,list):# 如果是一维标量，就转为数组

PortList=[PortList]

self.ForwardPort=PortList

self.enForward=True
```

### 3. 转发的数据机制

在mav的getMavMsg()函数中，等数据收到之后，调用mav.netForwardData()函数，会发送事件，被net中的getMavEvent死循环监听到，调netForwardBuf，用会自动将数据转发到上面指定的端口和IP

注意：netForwardBuf函数在转发时，会在数据包最前面加上校验位和CopterID和接受消息的飞机信息

包头协议：

```
#checksum=12345678 # int32型，校验位
```

```
#CopterID, int32型，当前飞机的ID号
```

```
#sendMode, int32型，发送模式，有组播，也有单播
```

```
#StartIdx, int32型，发送列表的起始飞机序号
```

```
#SendMask, uint64型，从序号开始的64个飞机是否发送
```

```
# 编码规则 iiiiQ
```

# 注意：sendMode、StartIdx、和SendMask是组网仿真程序需要使用的，接收端用不着可忽略

转发的数据结构为

```
# double uavTimeStmp 时间戳
```

```
# float uavAngEular[3] 欧拉角 弧度
```

```
# float uavVelNED[3] 速度 米
```

```
# double uavPosGPSHome[3] GPS 维度 (度)、经度 (度)、高度 (米)
```

```
# double uavPosNED[3] 本地位置 米 (相对起飞点)
```

```
# double uavGlobalPos[3] 全局位置 (相对与所有飞机的地图中心)
```

```
# d6f9d = 长度104
```

相关代码:

```
def getMavEvent(self):
```

```
while True:
```

```
if not self.enForward:
```

```
break
```

```
# 如果mav收到了消息
```

```
self.mav.netEvent.wait()
```

```
# double uavTimeStmp 时间戳
```

```
# float uavAngEular[3] 欧拉角 弧度
```

```
# float uavVelNED[3] 速度 米
```

```
# double uavPosGPSHome[3] GPS 维度 (度)、经度 (度)、高度 (米)
```

```
# double uavPosNED[3] 本地位置 米 (相对起飞点)
```

```
# double uavGlobalPos[3] 全局位置 (相对与所有飞机的地图中心)
```

```
# d6f9d = 长度104
```

```
if self.enForward: #如果开启网络转发
```

```
# 提取mav的必要数据, 向外转发
```

```
buf=struct.pack('d6f9d',self.mav.uavTimeStmp,*self.mav.uavAngEular,*self.mav.uavVelNED,*self.mav.uavPosGPSHome,*self.mav.uavPosNED,*self.mav.uavGlobalPos)
```

```
self.netForwardBuf(buf)
```

```
def netForwardBuf(self,buf):
```

```
# 将数据发送到ForwardIP和ForwardPort (端口或列表)
```

```

if self.enForward: #如果开启网络转发

checkSum=12345678 # int32型，校验位

CopterID = self.CopterID # int32型，当前飞机的ID号

sendMode = self.netSendMode # int32型，发送模式，有组播，也有单播

for j in range (len(self.netSendStarList)):

StartIdx = self.netSendStarList[j] # int32型，发送列表的起始飞机序号

SendMask = self.netSendMaskList[j] # uint64型，从序号开始的64个飞机是否发送

#checkSum=12345678 # int32型，校验位

#CopterID, int32型，当前飞机的ID号

#sendMode, int32型，发送模式，有组播，也有单播

#StartIdx, int32型，发送列表的起始飞机序号

#SendMask, uint64型，从序号开始的64个飞机是否发送

# 编码规则 iiiiQ

# 注意：sendMode、StartIdx、和SendMask是组网仿真程序需要使用的，接收端用不着
可忽略

bufID = struct.pack('iiiiQ',checkSum,CopterID,sendMode,StartIdx,SendMask) # 封装校
验位和飞机ID

myBuf = bufID+buf #包头封装一个飞机ID号

for i in range(len(self.ForwardPort)): # 向端口列表转发数据

self.udp_socket.sendto(myBuf,(self.ForwardIP,self.ForwardPort[i])) # 将数据转发给网络
仿真器

#print('Send')

```

4. 增加监听函数和接口，使得mav可以监听到发往自己的数据

```
StartNetRec(self,MultiPort=60001,MultiIP='224.0.0.10')
```

监听到的数据会存在mav.UavData这个结构体列表中

UavData的定义如下:

```
# UAVSendData

# double uavTimeStmp 时间戳

# float uavAngEular[3] 欧拉角 弧度

# float uavVelNED[3] 速度 米

# double uavPosGPSHome[3] GPS 纬度 (度)、经度 (度)、高度 (米)

# double uavPosNED[3] 本地位置 米 (相对起飞点)

# double uavGlobalPos[3] 全局位置 (相对与所有飞机的地图中心)

# d6f9d = 长度104

class UAVSendData:

    def __init__(self):

        self.hasUpdate=False

        self.CopterID=0

        self.uavTimeStmp=0

        self.uavAngEular=[0,0,0]

        self.uavVelNED=[0,0,0]

        self.uavPosGPSHome=[0,0,0]

        self.uavPosNED=[0,0,0]

        self.uavGlobalPos=[0,0,0]
```

注意: hasUpdate为是否有数据更新标志位, 每次读数后, 应该置0; 收到数据后会自动置于1, 通过此机制, 可以判断数据是否阻塞。

注意: UavData是一个列表, 里面会存储所有收到的飞机的数据, 以CopterID作为区分。

相关代码如下:

```
# 开始监听发往自己飞机的数据, 端口根据CopterID自动设置

def StartNetRecOwn(self):
```

```

self.StartNetRec(60000+self.CopterID)

# 开启接收发往指定飞机的数据

# 默认接收1号飞机端口60001，和组播IP 224.0.0.10

def StartNetRec(self,MultiPort=60001,MultiIP='224.0.0.10'):

# 设置组播监听IP和地址，并开启监听

ANY = '0.0.0.0'

self.udp_socketNet = socket.socket(socket.AF_INET, socket.SOCK_DGRAM) # Create
socket

self.udp_socketNet.setsockopt(socket.SOL_SOCKET, socket.SO_BROADCAST, 1)

self.udp_socketNet.bind((ANY, MultiPort))

status = self.udp_socketNet.setsockopt(socket.IPPROTO_IP,
socket.IP_ADD_MEMBERSHIP,
socket.inet_aton(MultiIP) + socket.inet_aton(ANY))

self.stopFlagNet=False

self.t1Net = threading.Thread(target=self.getMavMsgNet, args=())

self.t1Net.start()

def endNetLoop(self):

""" The same as stopRun(), stop message listenning from 20100 or serial port
"""

self.stopFlagNet=True

self.t1Net.join()

self.udp_socketNet.close()

# 在这里接收来自组播端口的飞机数据，目前只支持UDP_FULLL和UDP_Simple

def getMavMsgNet(self):

""" Start loop to listen mavlink data from 20100 series port or COM port

```

```

"""

while True:

if self.stopFlagNet:

break

try:

buf,addr = self.udp_socketNet.recvfrom(65500)

#print('Data Rec')

# 如果数据包太小, 说明不正确, 直接跳过

if len(buf)<=24:

continue

#checkSum=12345678 # int32型, 校验位

#CopterID, int32型, 当前飞机的ID号

#sendMode, int32型, 发送模式, 有组播, 也有单拨

#StartIdx, int32型, 发送列表的起始飞机序号

#SendMask, uint64型, 从序号开始的64个飞机是否发送

# 编码规则 iiiiQ

# 注意: sendMode、StartIdx、和SendMask是组网仿真程序需要使用的, 接收端用不着
可忽略

checkSum,CopterID,sendMode,StartIdx,SendMask=struct.unpack('iiiiQ',buf[0:24]) # 获
取校验位和通知

if checkSum==12345678: # 如果校验通过,符合预设的密码 (这里是为了排除无干的消息)

buf = buf[24:]

# UAVSendData

# double uavTimeStmp 时间戳

# float uavAngEular[3] 欧拉角 弧度

```

```
# float uavVelNED[3] 速度 米
# double uavPosGPSHome[3] GPS 纬度 (度)、经度 (度)、高度 (米)
# double uavPosNED[3] 本地位置 米 (相对起飞点)
# double uavGlobalPos[3] 全局位置 (相对与所有飞机的地图中心)
# d6f9d = 长度104
if len(buf)==104:
    UIV=struct.unpack('d6f9d',buf)
    uavTimeStmp=UIV[0]
    uavAngEular=UIV[1:4]
    uavVelNED=UIV[4:7]
    uavPosGPSHome=UIV[7:10]
    uavPosNED=UIV[10:13]
    uavGlobalPos=UIV[13:16]
    isCopterExist=False
    for i in range(len(self.UavData)): #遍历数据列表, 飞机ID有没有出现过
        if self.UavData[i].CopterID == CopterID: #如果出现过, 就直接更新数据
            idx=i
            isCopterExist=True
            self.UavData[idx].hasUpdate=True
            self.UavData[idx].uavTimeStmp=uavTimeStmp
            self.UavData[idx].uavAngEular=uavAngEular
            self.UavData[idx].uavVelNED = uavVelNED
            self.UavData[idx].uavPosGPSHome = uavPosGPSHome
            self.UavData[idx].uavPosNED = uavPosNED
```

```

self.UavData[idx].uavGlobalPos = uavGlobalPos

break

if not isCopterExist:#如果没有出现过，就创建一个结构体

vsr=UAVSendData()

vsr.hasUpdate=True

vsr.CopterID = CopterID

vsr.uavTimeStmp=uavTimeStmp

vsr.uavAngEular=uavAngEular

vsr.uavVelNED = uavVelNED

vsr.uavPosGPSHome = uavPosGPSHome

vsr.uavPosNED = uavPosNED

vsr.uavGlobalPos = uavGlobalPos

self.UavData = self.UavData + [copy.deepcopy(vsr)] #扩充列表，增加一个元素

except:

self.stopFlagNet=True

break

```

### 5. 增加数据包封装目标飞机ID列表功能

netAddUavSendList可以输入一个待发送飞机的列表，函数对列表按每64位间隔内飞机划分为一组的原则，将飞机列表分为若干组，然后将飞机总列表存储在self.netSendIDList里面，将包头的起始飞机ID列表存储在self.netSendStarList中，将发送使能列表存储在self.netSendMaskList中。

设定好飞机列表后，包头中就包含了这个消息包的目标飞机，在组网仿真器中，会根据目标飞机ID，将这个包转发出去，给到特定的飞机。

相关代码：

```

def netResetSendList(self,sendMode=0): # 清空待发送列表

self.netSendIDList=[0]

```

```
self.netSendStarList=[0]

self.netSendMaskList=[0]

self.netSendMode=sendMode

def netAddUavSendList(self,uavList=[]):

for i in range(len(uavList)): # 遍历待发送飞机列表

    CurID = uavList[i]

    isIdExist=False

    for j in range(len(self.netSendIDList)): # 判断是否已经在列表中

        if CurID== self.netSendIDList[j]:

            isIdExist=True

            break

        if not isIdExist: # 不在列表中就加入列表

            self.netSendIDList = self.netSendIDList + [CurID]

            # 开始更新目标发送列表

            if len(self.netSendIDList)==0: # 列表为空的话，就还原一下

                self.netSendIDList=[0]

                self.netSendStarList=[0]

                self.netSendMaskList=[0]

            return

        self.netSendIDList.sort() # 从小到大重新排序

    netGrp=[]

    self.netSendStarList=[]

    self.netSendMaskList=[]

    for i in range(len(self.netSendIDList)):
```

```
curlID = self.netSendIDList[i]

if len(netGrp)==0:

netGrp=[curlID]

else:

if curlID-netGrp[0]<64: # 如果还没有越界，就累加

netGrp=netGrp+[curlID]

# 到达

if curlID-netGrp[0]>=64 or i==len(self.netSendIDList)-1:

# 开始将上一组数据，写到列表中

startIdx = netGrp[0]

mask = 0x0000000000000000 # 64位的mask

for j in range(len(netGrp)):

dist = netGrp[j]-startIdx

mask = mask | 1<<dist

self.netSendStarList = self.netSendStarList + [startIdx]

self.netSendMaskList = self.netSendMaskList + [mask]

# 如果超过64位了，则直接从新开启一组

if curlID-netGrp[0]>=64 and i<len(self.netSendIDList)-1:

netGrp=[curlID]

# 如果超过64位且刚好达到末尾，则直接计算矩阵

if curlID-netGrp[0]>=64 and i==len(self.netSendIDList)-1:

startIdx = curlID

mask = 1

self.netSendStarList = self.netSendStarList + [startIdx]
```

```
self.netSendMaskList = self.netSendMaskList + [mask]
```

## 6. 增加了请求飞机返回数据的接口

和发送一样，减了一些列表

```
# 网络仿真时，请求消息的飞机ID
```

```
self.netReqIDList=[0]
```

```
self.netReqStarList=[0]
```

```
self.netReqMaskList=[0]
```

netResetReqList函数可以重置列表

netAddUavReqList函数可以增加列表

StartReqUavData会开启一个线程，以每秒一次的频率，发送数据请求

EndReqUavData可以结束线程

sendReqUavLoop是发送线程死循环

发送的消息只有包头，但是sendMode设置成了12345

```
sendMode = 12345 # int32型，发送模式，有组播，也有单拨 # 这里12345表示是一个请求消息包
```

getMavMsgNet在飞机的监听线程中，收到请求消息后，看一下自己的飞机是不是在列表中，如果在列表中，就将发送消息的飞机ID，记录在本机的发送列表netAddUavSendList中，这样，后面数据就会往这个飞机发送一份。

## 7. 常见问题

Q1: \*\*\*

A1: \*\*\*