RflySim Platform Visual Interface Document

directory

1. Introduction to the RflySim Visual Architecture

The RflySim vision architecture is divided into two parts: software-in-the-loop (SITLR un communication framework) and hardware-in-the-loop (HITL+ data communication frame work).

During the SITL software-in-the-loop simulation, the PX4 flight controller runs comple tely in the Win10WSL virtual machine and uses px4_sitl firmware. The PX4 flight control ler runs in a Win10WSL virtual machine using px4_sitl firmware. PX4 communicates dire ctly with CopterSim via the MAVLink protocol, and then CopterSim sends and receives e xternal Python messages through port 20100/20101. The Python program communicates dir ectly with the RflySim3D program through UDP, and obtains visual images through shared memory/UDP sending.

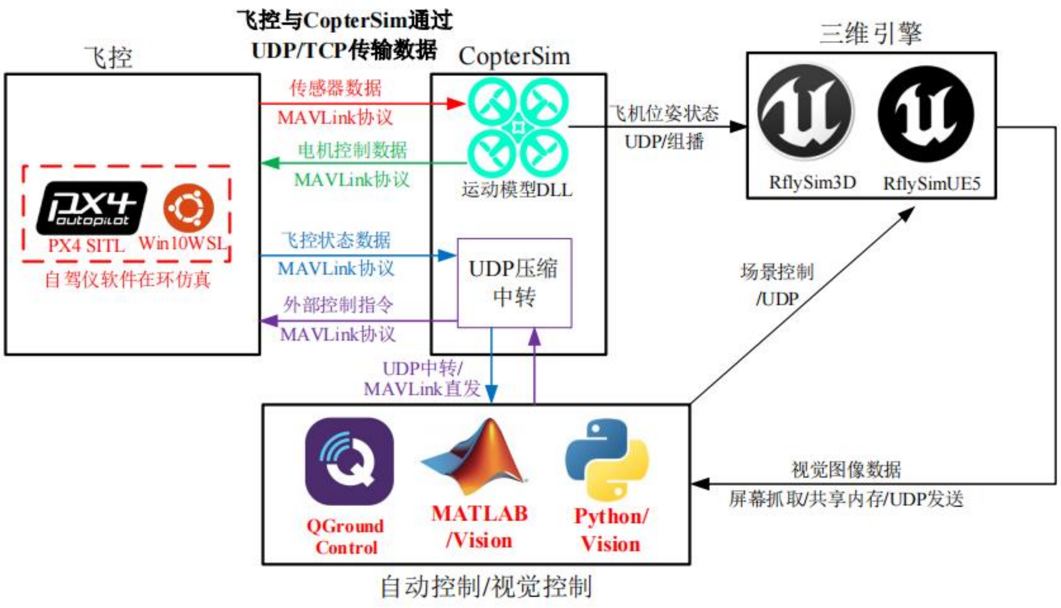


Figure 1 SITLRun software-in-the-loop simulation communication architecture

In the HITL hardware-in-the-loop simulation, the PX4 flight controller algorithm runs with the Pixhawk hardware, using the px4_fmu-v5 firmware. PX4 communicates directly w ith CopterSim through the MAVLink protocol of the USB serial port, and then CopterSim implements the message transmission and reception with external Python programs through the data transmission serial port. The Python program communicates directly with the Rfly Sim3D program through UDP, and obtains visual images through shared memory/UDP sen ding.

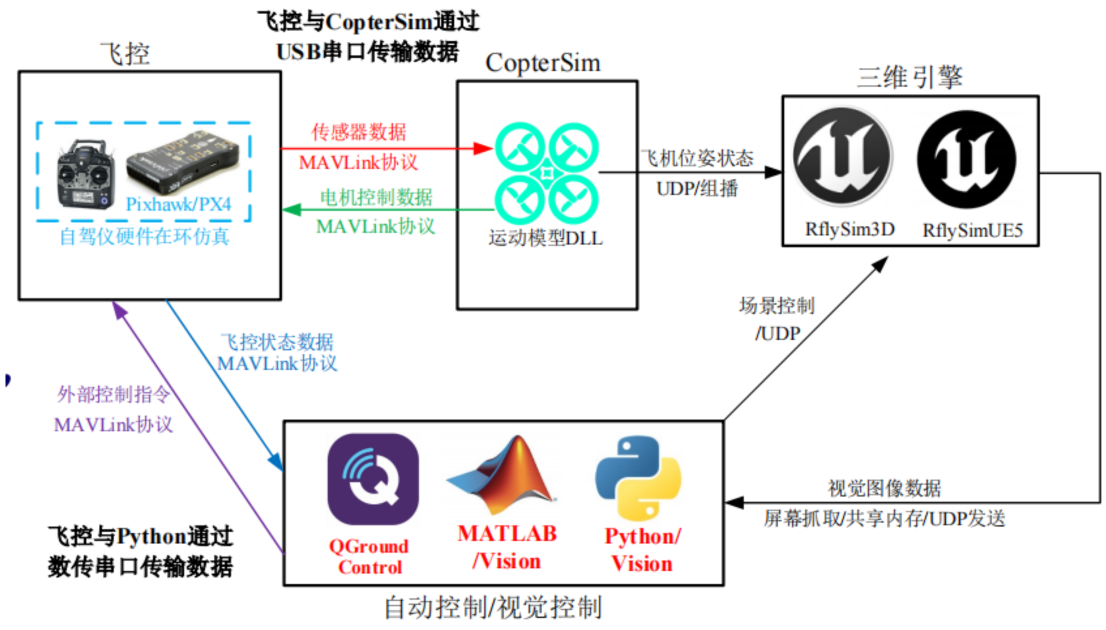


Figure 2 HITLRun hardware-in-the-loop simulation communication architecture

2. Configuration of visual function development environment

The vision function development environment of the RflySim platform is divided into two parts: the Windows development environment and the Ubuntu virtual machine develop ment environment. The Windows development environment is used to use the various inter faces of the RflySim platform to control the motion of the drone, the scene control, and obtain visual image data, while the Ubuntu virtual machine development environment adap ts to the RflySim platform in order to adapt to the development needs of ros in the Linu x system environment.

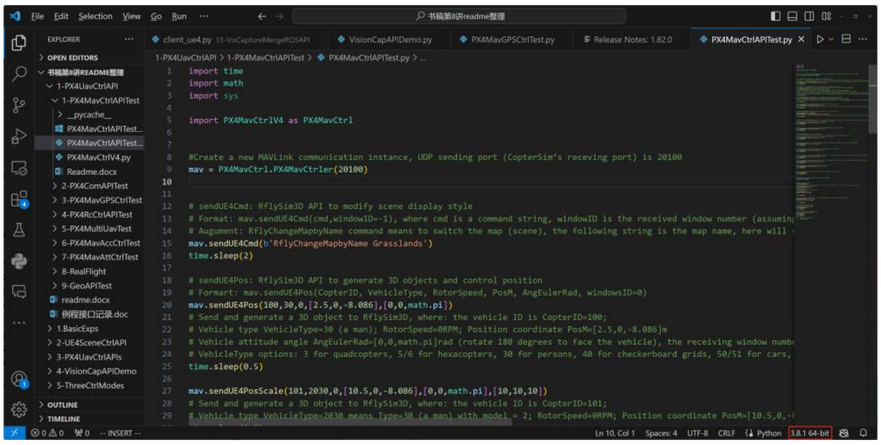
2.1. Windows Development Environment

2.1.1. The platform comes with a VS Code development environment

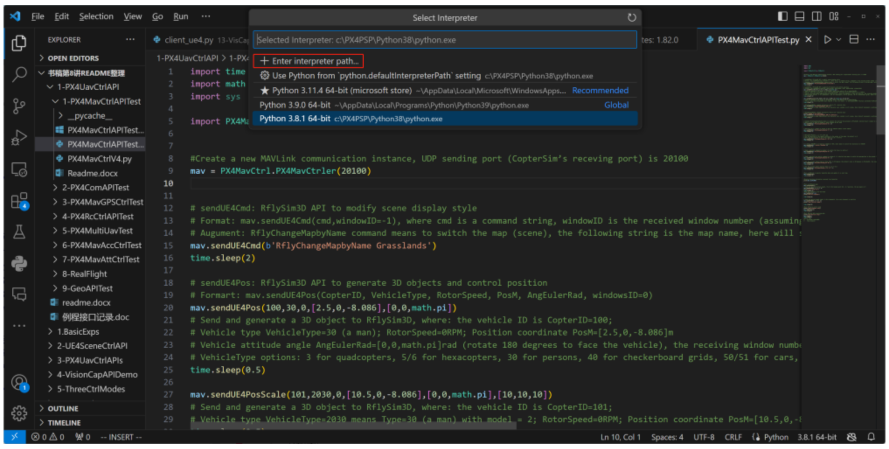
After running the one-click platform installation script, the python environment on which the RflySim platform interface depends will be installed, and the default path is C:\PX4PSP\Python38\python.exe 。 Once the VS Code software is installed, run a software-in-the-loop example, configure the environment in VS Code, and then use the interface to the

RflySim platform. The configuration is as follows:

1) Click on the environment path key, then click on the Enter interpreter path, then click on Find:



```
import time
import math
import sys

import PX4MavCtrlV4 as PX4MavCtrl


#Create a new MAVLink communication instance, UDP sending port (CopterSim's receving port) is 20100
mav = PX4MavCtrl.PX4MavCtrler(20100)


# sendUE4Cmd: RflySim3D API to modify scene display style
# Format: mav.sendUE4Cmd(cmd,windowID=-1), where cmd is a command string, windowID is the received window number (assuming
# Augument: RflyChangeMapbyName command means to switch the map (scene), the following string is the map name, here will
mav.sendUE4Cmd(b'RflyChangeMapbyName Grasslands')
time.sleep(2)

# sendUE4Pos: RflySim3D API to generate 3D objects and control position
# Formart: mav.sendUE4Pos(CopterID, VehicleType, RotorSpeed, PosM, AngEulerRad, windowsID=0)
mav.sendUE4Pos(100,30,0,[2.5,0,-8.086],[0,0,math.pi])
# Send and generate a 3D object to RflySim3D, where: the vehicle ID is CopterID=100;
# Vehicle type VehicleType=30 (a man); RotorSpeed=0RPM; Position coordinate PosM=[2.5,0,-8.086]m
# Vehicle attitude angle AngEulerRad=[0,0,math.pi]rad (rotate 180 degrees to face the vehicle), the receiving window numb
# VehicleType options: 3 for quadcopters, 5/6 for hexacopters, 30 for persons, 40 for checkerboard grids, 50/51 for cars,
time.sleep(0.5)

mav.sendUE4PosScale(101,2030,0,[10.5,0,-8.086],[0,0,math.pi],[10,10,10])
# Send and generate a 3D object to RflySim3D, where: the vehicle ID is CopterID=101;
# Vehicle type VehicleType=2030 means Type=30 (a man) with model = 2; RotorSpeed=0RPM; Position coordinate PosM=[10.5,0,-
```



Select Interpreter

Selected Interpreter: c:\PX4PSP\Python38\python.exe

+ Enter interpreter path...

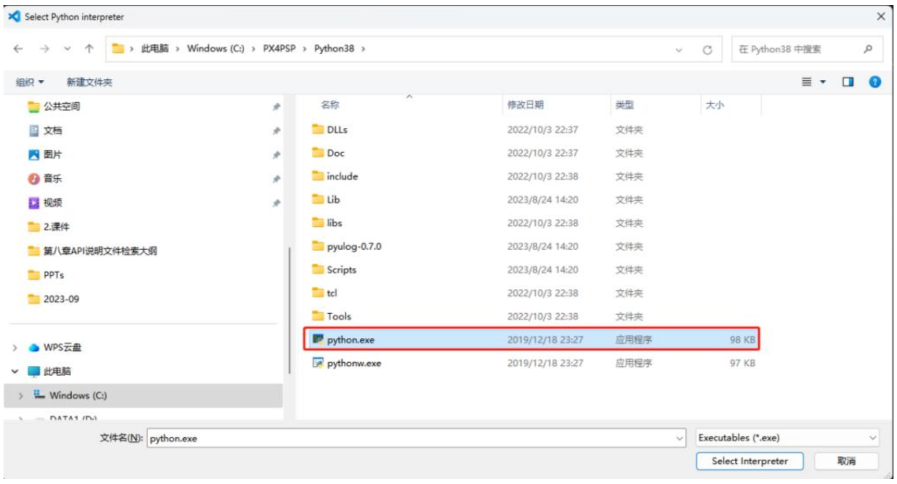
Use Python from `python.defaultInterpreterPath` setting c:\PX4PSP\Python38\python.exe

★ Python 3.11.4 64-bit (microsoft store) ~\AppData\Local\Microsoft\WindowsApps... Recommended

Python 3.9.0 64-bit ~\AppData\Local\Programs\Python\Python39\python.exe Global

Python 3.8.1 64-bit c:\PX4PSP\Python38\python.exe

2）Select the corresponding Python environment：



Select Python interpreter

此电脑 › Windows (C:) › PX4PSP › Python38 ›

名称	修改日期	类型	大小
DLLs	2022/10/3 22:37	文件夹	
Doc	2022/10/3 22:37	文件夹	
include	2022/10/3 22:38	文件夹	
Lib	2023/8/24 14:20	文件夹	
libs	2022/10/3 22:38	文件夹	
pyulog-0.7.0	2023/8/24 14:20	文件夹	
Scripts	2023/8/24 14:20	文件夹	
tcl	2022/10/3 22:38	文件夹	
Tools	2022/10/3 22:38	文件夹	
python.exe	2019/12/18 23:27	应用程序	98 KB
pythonw.exe	2019/12/18 23:27	应用程序	97 KB

文件名(N): python.exe　Executables (*.exe)

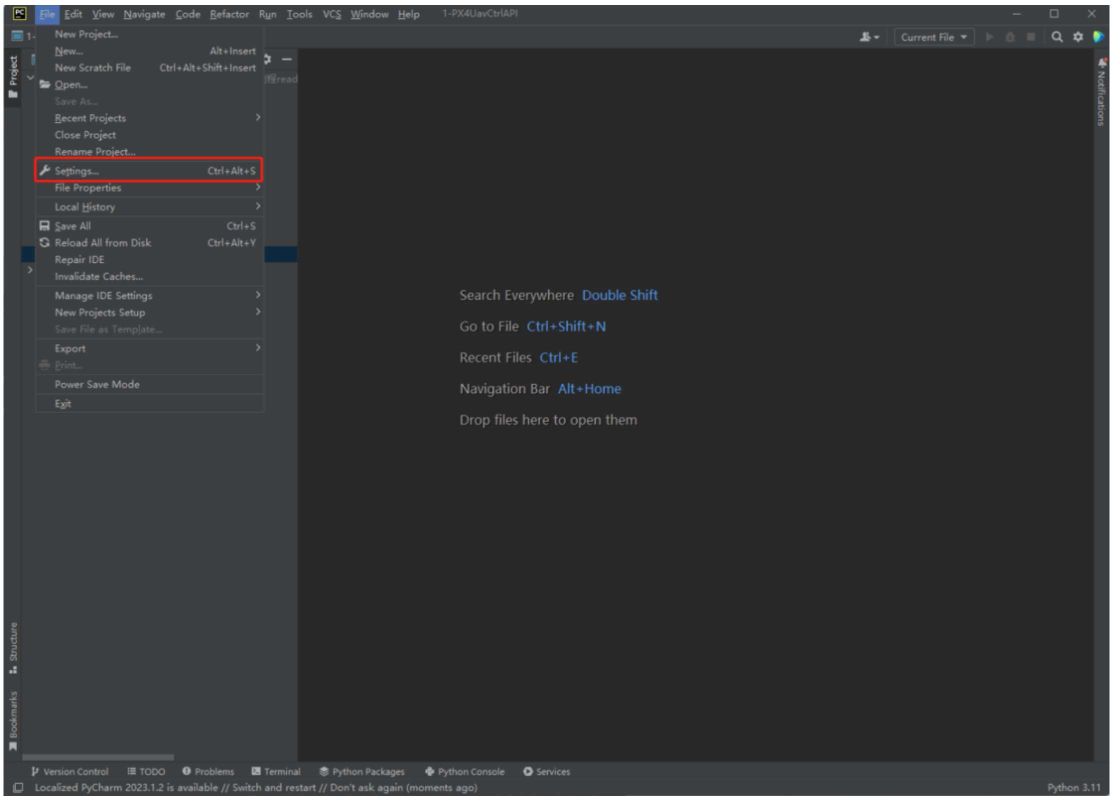
Select Interpreter　取消

2.1.2. Pycharm Development Environment

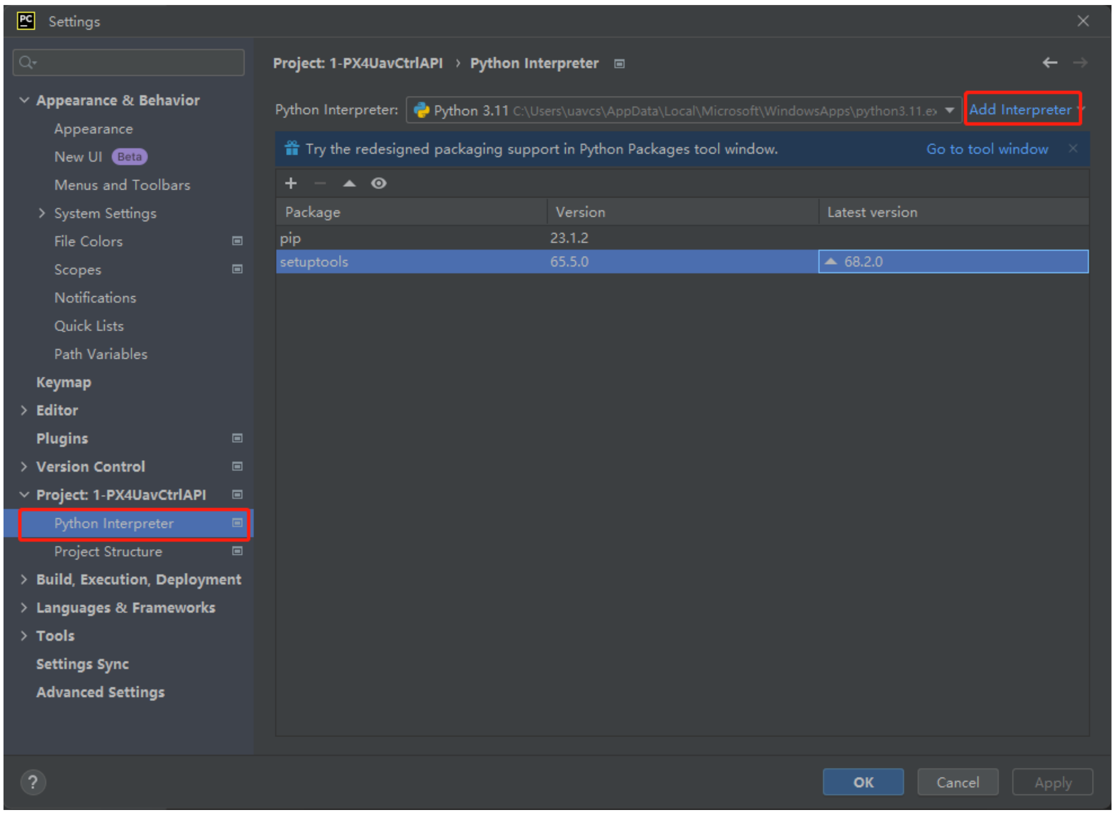
To use the RflySim platform under Pycharm, you need to configure the python environment. The configuration environment is as follows:

1) Open pycharm and click file--settings, then click on Python Interpeter under profect, and

click Add Interpreter.



Localized PyCharm 2023.1.2 is available // Switch and restart // Don't ask again (moments ago)

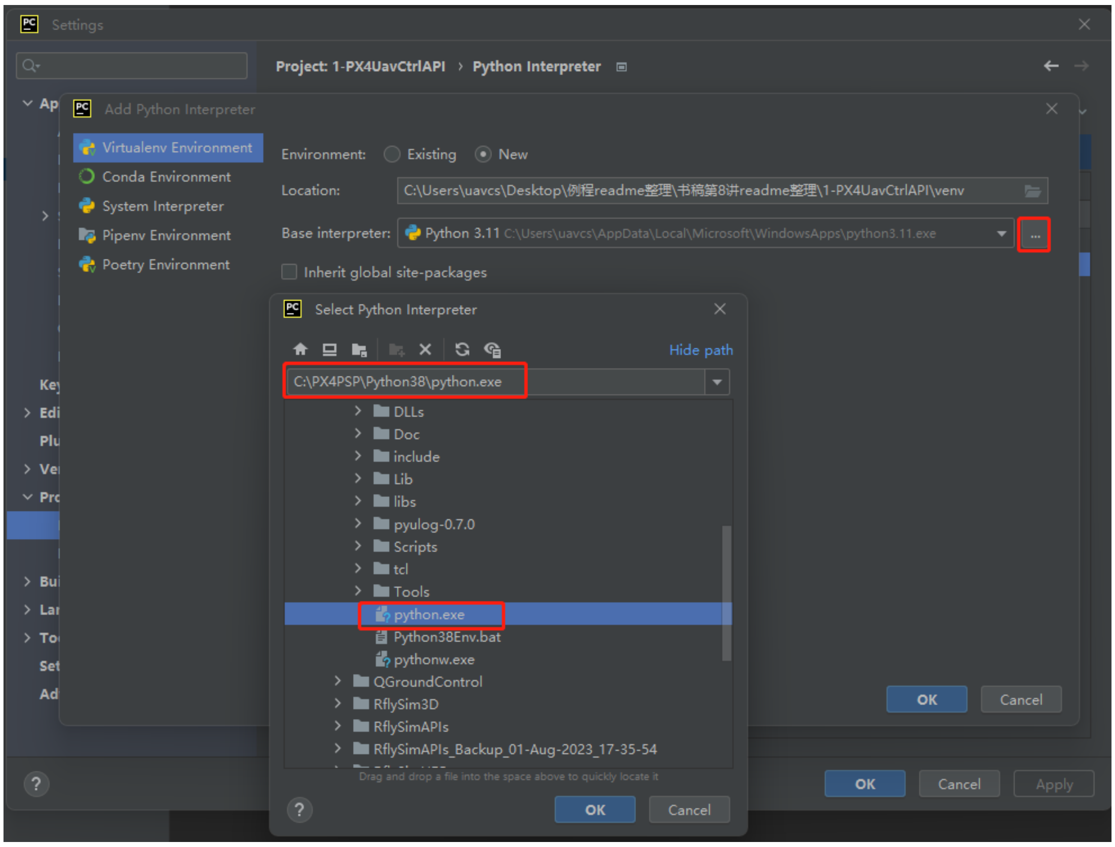


Python Interpreter: Python 3.11 C:\Users\uavcs\AppData\Local\Microsoft\WindowsApps\python3.11.ex

Try the redesigned packaging support in Python Packages tool window.

Package	Version	Latest version
pip	23.1.2	
setuptools	65.5.0	▲ 68.2.0

2)Select the python path when the platform is installed.



Environment: Existing [✓ New]

Location: C:\Users\uavcs\Desktop\例程readme整理\书稿第8讲readme整理\1-PX4UavCtrlAPI\venv

Base interpreter: Python 3.11 C:\Users\uavcs\AppData\Local\Microsoft\WindowsApps\python3.11.exe

Inherit global site-packages

C:\PX4PSP\Python38\python.exe

2.2. Ubuntu Virtual Machine Development Environment

Before using an Ubuntu VM, you need to download the VMware software and run the Ubuntu VM image on the VMware software. It is recommended to use NAT mode for network communication between the virtual machine and the host, which requires the host to connect to an external router, so that a physical IP address can be assigned to the virtual machine, so that the virtual machine can also access the external network. When there is no external route, the host uses a private network connection to connect to the virtual machine, it should be noted that some computers use private network communication, which will block some CopterSim communication with the virtual machine, of course, private network communication, in rare cases, such a problem will occur.

2.3. Ubuntu Development Environment

The RflySim platform vision algorithm is developed on the Ubuntu development environment in order to have a good interaction between the RflySim platform and the Linux system environment. Conducted joint development of RflySim platform for Windows and Linux. It is also designed to be closely related to the ROS and ROS2 environments under Linux and mavros.
If you need to use a GPU, you need to build a cuda library and tensorRT acceleration.

3. Introduction to development tools that rely on the development of visual algorithms

The RflySim vision algorithm development dependency tool is divided into two parts: the simulation part and the algorithm dependency library, the simulation part is mainly related to the application of the RflySim platform (software and hardware in the loop) and the communication part (pymavlink, mavros, ros, ros2); The algorithm relies on public libraries (OpenCV, Eigen, etc.), as well as some libraries based on AI target training and detection (cuda, torch, trensorRT, etc.).

The RflySim vision architecture is divided into two parts: software-in-the-loop (SITLRun communication framework) and hardware-in-the-loop (HITL+ data communication framework). Software-in-the-loop development only requires the RflySim platform interface and the following dependent development tools, such as MAVLink, mavros, pymavlink, ROS, and ROS2. However, hardware-in-the-loop development requires Pixhawk flight controller hardware in addition to the software mentioned above, or a remote controller if required.

For AI, you need the cuda library of the GPU and the corresponding tensorRT library.

3.1. MAVLink/mavros/pymavlink

MAVLink is a lightweight, efficient drone communication protocol for communication between drones, ground stations, and other drone-related systems. It provides a highly reliable and flexible approach to real-time data transmission and control. MAVLink supports a variety of communication media, including SERIAL, UDP, TCP, and CAN. It defines a variety of message types, commands, and status information for acquiring sensor data, sending control commands, and more. For more learning materials, please refer to: https://mavlink.io/en/.

mavros is an open-source ROS (Robot Operating System) software package that implements the bridging between MAVLink and ROS. It provides ROS nodes and services, enabling developers to communicate and control UAVs through ROS. The mavros acts as an intermediate layer between the UAS and the ROS, and the ROS function package can be used for condition monitoring, task assignment and control of the robot, among other things. It provides many convenient functions such as flight mode switching, waypoint navigation, attitude control, etc., as well as the reception and release of various sensor data. For more learning materials, please refer to: http://wiki.ros.org/mavros.

pymavlink is a MAVLink library for Python development. It provides parsing and generation capabilities for the MAVLink protocol, enabling developers to communicate with drones in the Python programming language. pymavlink can be used to write Python scripts that receive and send MAVLink messages to process sensor data, send control commands, etc. It provides an advanced encapsulation of the MAVLink protocol, making it easier for developers to operate and interact with UAS. For more learning materials, please refer to: https://mavlink.io/zh/mavgen_python/.

3.2. ROS/ROS2

ROS (Robot Operating System) and ROS2 (Robot Operating System 2) are very common and powerful tools. They are open-source robotics software development platforms that provide a modular, reusable, and distributed way for robotic systems to build software.

1. ROS (Robot Operating System): ROS is a flexible and extensible software development framework designed to support the development and operation of robot systems. It provides a range of tools, libraries, and conventions for handling aspects such as hardware abstraction, device drivers, library integration, shared messaging, package management, and visualization for bots. ROS uses a publish-subscribe model for communication between nodes, allowing modular software components to interact via messaging. This architecture enables developers to divide system functions into independent nodes and build robotics applications in a composable and reusable manner. For more learning materials, please refer to: http://wiki.ros.org/cn.

2. ROS2 (Robot Operating System 2): ROS2 is the next-generation version of ROS, which improves some of the limitations and shortcomings in ROS. It is a multi-language, cross-platform, real-time, and scalable robotics software development framework. ROS2 introduces Data Distribution Service (DDS) as the underlying communication protocol to support higher-level real-time communication and security. It also offers features such as better real-time behavior, distributed system support, multi-sensor fusion, and better encapsulation. ROS2 has been developed with a focus on improving performance, reliability, and scalability, making it suitable for a wider range of robotics applications and scales. For more learning materials, please refer to: Https://www.ros.org.

3.3. OpenCV

OpenCV (Open Source Computer Vision Library) is an open-source computer vision and machine learning library, which is widely used in vision algorithm development and image processing tasks. It provides a rich set of functions and tools that can be used to process image and video data and perform a variety of computer vision tasks, such as feature extraction, object detection, image segmentation, and image and video processing and analysis.

Wide range of features: OpenCV provides more than 2,500 optimized algorithm functions, including image processing, feature detection, computer vision, machine learning, deep learning, and image transformation. It supports a variety of programming languages, such as C++, Python, Java, and MATLAB, allowing developers to choose the right language to develop according to their preferences and needs.

Cross-platform support: OpenCV is cross-platform and runs on multiple operating systems, including Windows, Linux, macOS, Android, and iOS. This allows developers to use the same interface and code in different environments, facilitating the development and deployment of cross-platform applications.

Efficient image processing and computation: OpenCV uses optimized algorithms and underlying implementations to efficiently process images and computer vision operations. It utilizes hardware acceleration and parallel computing to improve the performance and efficiency of the algorithm. In addition, OpenCV provides powerful matrix and vector manipulation that simplifies the programming of image processing and computations.

Hundreds of image processing functions and algorithms: OpenCV provides a wealth of image processing functions and algorithms, including filtering, edge detection, morphology manipulation, histogram processing, color space conversion, image inpainting and enhancement, and more. These functions can be used to process and manipulate different aspects of images, making it easy for developers to implement a variety of image processing tasks.

Machine learning and deep learning support: OpenCV also provides modules and functions for machine learning and deep learning. It supports common machine learning algorithms and techniques, such as Support Vector Machine (SVM), k-nearest neighbor (KNN), Random Forest, etc. In addition, OpenCV integrates deep learning frameworks, such as TensorFlow and PyTorch, to enable the construction and inference of deep learning models. For more learning materials, please refer to: https://opencv.org/.

4. RflySim platform visual configuration steps

4.1. bat Script Configuration Method (Configuration for Vision)

4.1.1. bat Script Template Selection (Select an existing script template based on your needs)

4.1.1.1. SITLRun.bat software-in-the-loop simulation base script

4.1.1.2. SITLPos.bat Initialize the position control software in the loop script

4.1.1.3. Initialize the position control software in the loop script SITLPosStr.bat the form of a string

4.1.1.4. HITLRun.bat Hardware-in-the-loop simulation base script

4.1.1.5. HITLPos.bat Initialize Position Control Hardware-in-the-Loop Script

4.1.1.6. Initialize the Position Control Hardware-in-the-Loop Script HITLPosStr.bat Strings

4.1.1.7. HITLPosSysID.bat configure the initial position sequence hardware-in-the-loop script

4.1.1.8. The value of CopterID is determined by SysID HITLPosSysIDStr.bat, and the initial position sequence can be configured

4.1.2. START_INDEX(Coptic starting number in Copticsim)

The starting aircraft serial number, the CopterID of the aircraft generated by this script, is initially valued at this START_INDEX, and is incremented by 1 in turn. This option is useful for simulations with multiple computers.

4.1.3. UDP_START_PORT (Port number for MavLink UDP communication in CopterSim)

Set the starting UDP port of SIMULINK/OFFBOARD API, this option should not be modified to cluster simulation, the UDP communication interface that receives external control data, corresponding to the CopterID, will automatically add 2 to 20100+CopterID*2, which usually does not need to be modified here, and is only modified when the computer port is occupied.

4.1.4. SimMode (Hardware/Software Emulation Mode Selection)

Set the CopterSim simulation mode, which can be identified by number or name, set to 0 or PX4_HITL for hardware-in-the-loop simulation, and set to 2 or PX4_SITL_RFLY for software-in-the-loop simulation.

4.1.5. UE4_MAP (Specifying the scene map in RflySim3D)

Set up the map, use the index or name of the map on CopterSim, and control the RflySim 3D scene by specifying the scene map name. Use the LowGPU scenario to ensure that low-spec computers can run the platform.

4.1.6. IS_BROADCAST (Select CopterSim MavLink message communication method)

This variable is used to set whether to broadcast to other computers, and if IS_BROADCAST=0, it will only be transmitted locally. IS_BROADCAST=1 indicates intra-LAN broadcast transmission. Or use an IP address to boost your speed. Note: When filling in the IP address mode, IS_BROADCAST=0 is equivalent to IS_BROADCAST=127.0.0.1, and IS_BROADCAST=1 is equivalent to IS_BROADCAST=255.255.255.255. You can also use the IP address sequence in English","or"; "Separate the specified send. For example: 127.0.0.1, 192.168.1.4, 192.168.1.5, note that this only affects the CopterSim communication, and does not affect the RflySim3D communication, in other words, no matter what the value of the IS_BROADCAST is, it does not affect the acquisition of sensor data from RflySim3D.

4.1.7. UDPSIMMODE （ MavLink Messaging Communication UDP Emulation Mode Settings ）

UDPSIMMODE value selection	illustrate
0	Indicates the use of a custom UDP_FULL communication mode (full mode, with the most complete data, but a large amount of data transferred)
1	Indicates the use of a custom UDP_Simple communication mode (the I/O is streamlined, and the structure of UDP transmission is also streamlined, so there is no additional information in the UDP_FULL mode, and in addition, MAVLink communication is optimized for multi-computer vision hardware-in-the-loop simulation
2	Mavlink_Full mode (directly use the MAVLink data stream to communicate between Python and PX4 (through CopterSim relay), which has a large amount of data and occupies a high bandwidth, but it is closer to the real machine and has perfect functions, and it is easy to block the bandwidth in large-scale clusters)
3	Mavlink_Simple mode (Python-PX4 communication directly using MAVLink data stream (via CopterSim relay), but the amount of data has been reduced)
4	Corresponding to the Mavlink_NoSend mode, CopterSim will not send MAVLink data to the outside, this mode needs to cooperate with hardware-in-the-loop simulation + data transmission serial port communication, and transmit MAVLink through wired mode, this mode has the smallest amount of data in the local area network, suitable for distributed vision hardware-in-the-loop simulation, and the number of drones is not limited.
5	mavlink_full mode without GPS positioning

4.1.8. TOTOAL_COPTER （ The total number of aircraft ）

The total number of drones, automatically arranging positions.

4.1.9. VEHICLE_INTERVAL （ Initialize the aircraft position interval ）

Set the distance between the two drones in meters.

4.1.10. Flight Controller Initialization Pose Settings (Pos, Yaw, and Distributed Multi-Machine Settings)

Set the origin of the drone in RflySim3D to be in meters and the declination in degrees. It is also possible to give the coordinate position and declination angle in the form of a list, separated in English, and set the posture of multiple UAVs separately.

4.1.11. IsSysID （Hardware-in-the-loop unique parameters determine whether CopterID uses the sys_id of the flight controller）

If a hardware-in-the-loop experiment is performed, the CopterSim will use the ID number of the flight controller as the CopterID for simulation.

4.1.12. More BAT Script Template Descriptions (e.g. Independently Set Pose for Each Aircraft, etc.)

You can set the pose of multiple drones in the script, such as setting the list of PosXStr, PosYStr, and YawStr with ",". For example, SET PosXStr=1.1,2.2,SET PosYStr=1,2,SET YawStr=0,0 can set the pose of two UAVs.

4.2. PX4 software-in-the-loop simulation steps

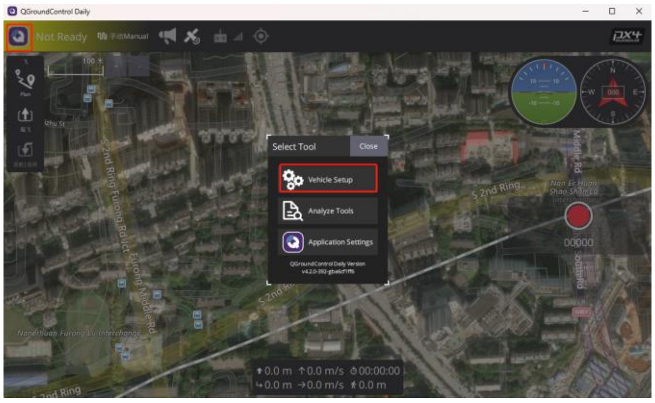
4.2.1. Bat Script Settings

Please modify it on the basis of the SITLRun.bat software-in-the-loop simulation script, and the software-in-the-loop simulation first sets the simulation mode SimMode=2 or SimMode=PX4_SITL_RFLY. Then set the simulated rack to the corresponding rack, for example, set PX4SitlFrame=iris. And set it to SET IS_BROADCAST=1.

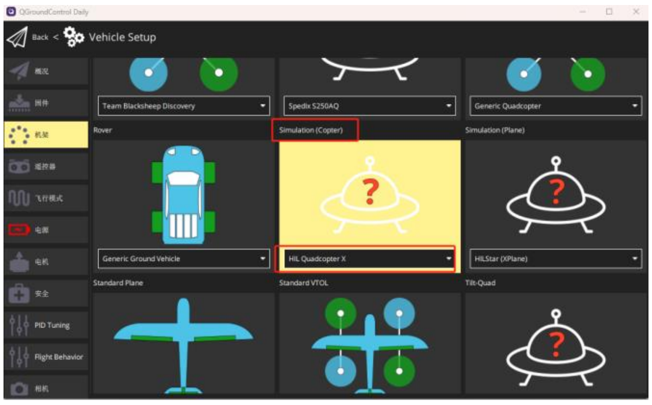
4.3. PX4 Hardware-in-the-Loop Simulation Steps (Flight Controller Needs to Restore Official Firmware)

Please modify it based on the hardware-in-the-loop simulation script HITLRun.bat, and set the

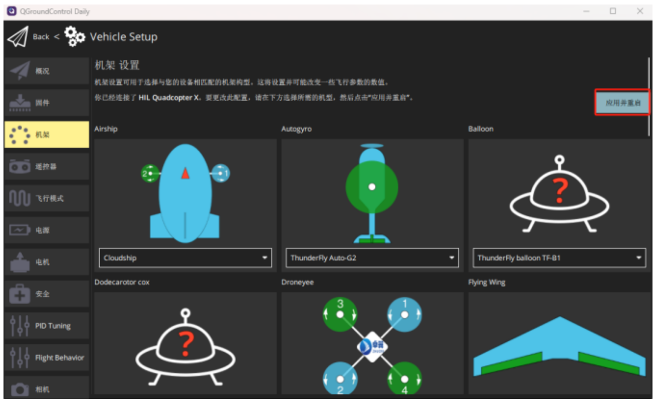
simulation mode SimMode=0 or SimMode=PX4_HITL for hardware-in-the-loop simulation. Set SET /a IsSysID=1 to enable the function of automatically calculating CopterID from SysID.
Then connect the flight controller to the computer, and turn on the QGC ground station to set up the rack, as shown in the following figure:
1. Select the vehicle settings:



Select Rack Settings, and then select HIL



After the selection is completed, click Apply and restart to set the rack successfully.



机架 设置

机架设置可用于选择与您的设备相匹配的机架构型。这将设置并可能改变一些飞行参数的数值。

你已经连接了 HIL Quadcopter X。要更改此配置，请在下方选择所需的机型，然后点击"应用并重启"。

4.4. PX4 Software-in-the-Loop + Virtual Machine in the Loop

4.4.1. Bat script configuration

Please modify it on the basis of the SITLRun.bat software-in-the-loop simulation script, and the software-in-the-loop simulation first sets the simulation mode SimMode=2 or SimMode=PX4_SITL_RFLY. Set the simulated rack to the corresponding rack, for example, set PX4SitlFrame=iris. And set it to SET IS_BROADCAST=1 VM configuration.
When creating one or more drones, specify the port number and IP address in the python program, PX4MavCtrler(port,ip) for a single drone for PX4MavCtrler (20100,'127.0.0.1') for a single drone and PX4MavCtrler (20100+ID*2,'127.0.0.1' for multiple drones). IP address 127.0.0.1 is local communication, IP address 255.255.255.255 is local LAN communication. You can also specify a fixed IP address.

4.5. PX4 Software-in-the-Loop + NX Hardware-in-the-Loop

4.5.1. Bat script configuration

Please modify it on the basis of the SITLRun.bat software-in-the-loop simulation script, and the software-in-the-loop simulation first sets the simulation mode SimMode=2 or SimMode=PX4_SITL_RFLY. Then set the simulated rack to the corresponding rack, for example, set PX4SitlFrame=iris. And set it to SET IS_BROADCAST=1.

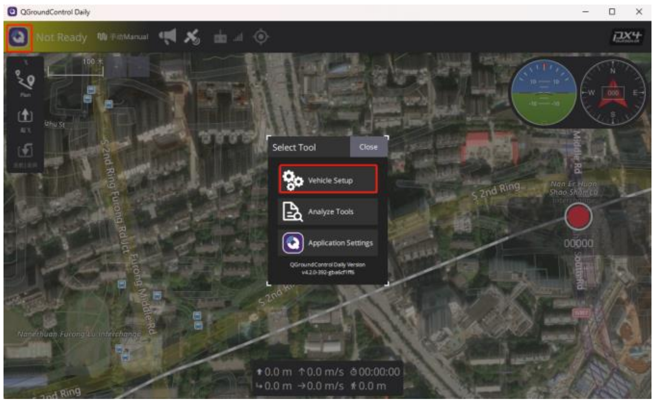
4.5.2. NX side configuration

When creating one or more drones, specify the port number and IP address in the python program, PX4MavCtrler(port,ip) for a single drone for PX4MavCtrler (20100,'127.0.0.1') for a single drone and PX4MavCtrler (20100+ID*2,'127.0.0.1' for multiple drones). IP address 127.0.0.1 is local communication, IP address 255.255.255.255 is local LAN communication. You can also specify a fixed IP address.

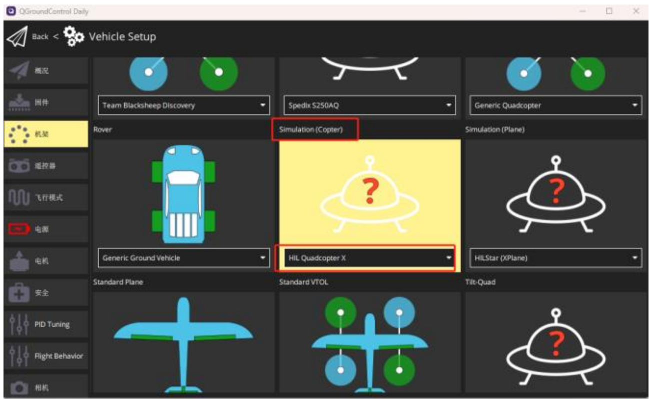
4.6. PX4 Hardware-in-the-Loop + Virtual Machine in the Loop

First, connect the flight controller to the computer, and open the QGC ground station to set the rack, as shown in the following figure:

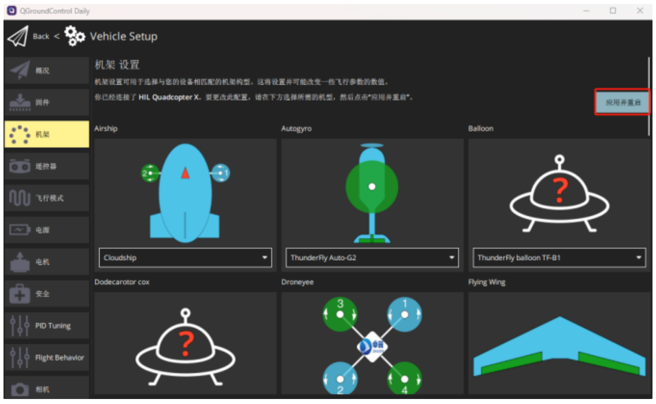
1. Select the vehicle settings:



2. Select Rack Settings, and then select HIL



3. Once you've made your selection, click Apply and restart for the rack setup to be successful.



4.6.1. Bat script configuration

Please modify it based on the hardware-in-the-loop simulation script HITLRun.bat, and set the simulation mode SimMode=0 or SimMode=PX4_HITL for hardware-in-the-loop simulation. Set SET /a IsSysID=1 to enable the function of automatically calculating CopterID from SysID.

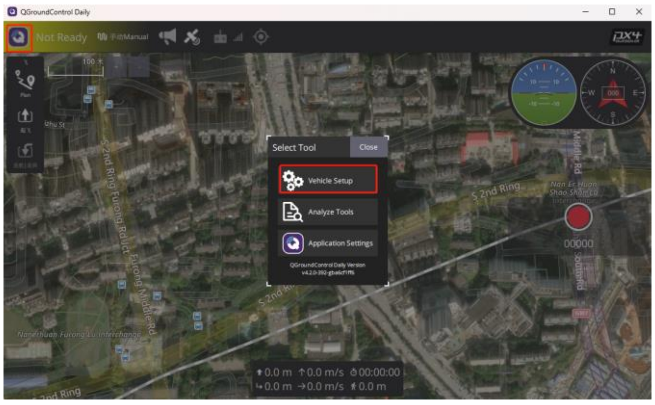
4.6.2. Virtual Machine Configuration

Identify the port number of the hardware flight controller in the computer, and specify the port number or port number plus baud rate in the python program, such as PX4MavCtrler('COM3') or PX4MavCtrler('COM3:57600').

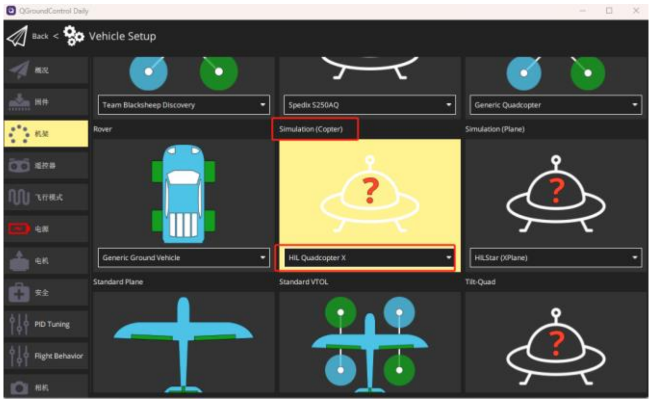
4.7. PX4 Hardware-in-the-Loop + NX Hardware-in-the-Loop

First, connect the flight controller to the computer, and open the QGC ground station to set the rack, as shown in the following figure:

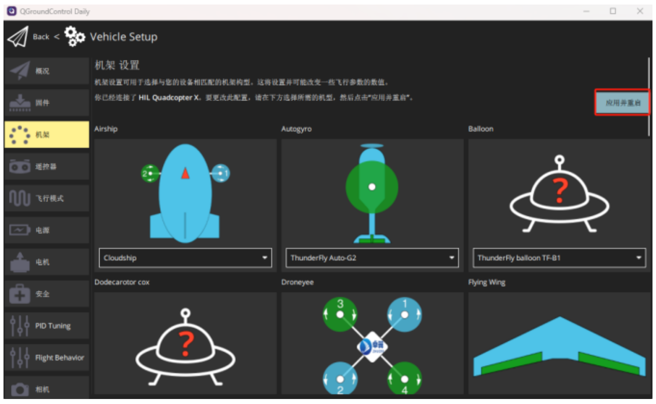
1. Select the vehicle settings:



2、Select Rack Settings, and then select HIL



3、Once you've made your selection, click Apply and restart for the rack setup to be successful.



4.7.1. Bat Script Configuration

Please modify it based on the hardware-in-the-loop simulation script HITLRun.bat, and set the simulation mode SimMode=0 or SimMode=PX4_HITL for hardware-in-the-loop simulation. Set SET /a IsSysID=1 to enable the function of automatically calculating CopterID from SysID.

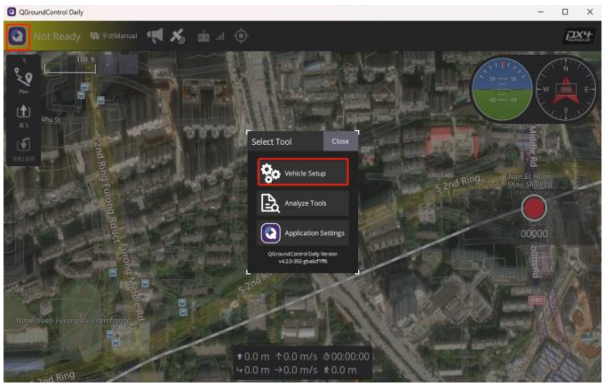
4.7.2. NX Configuration

Identify the port number of the hardware flight controller in the Ubuntu system in NX, such as /dev/ttyACM0, and then enter the command "chmod +x /dev/ttyACM0" in the terminal to authorize the port to execute, and other ports are also operated accordingly. Then specify the port number or port number plus baud rate in the python program, e.g. PX4MavCtrler('/dev/ttyACM0') or e.g. PX4MavCtrler('/dev/ttyACM0:57600').

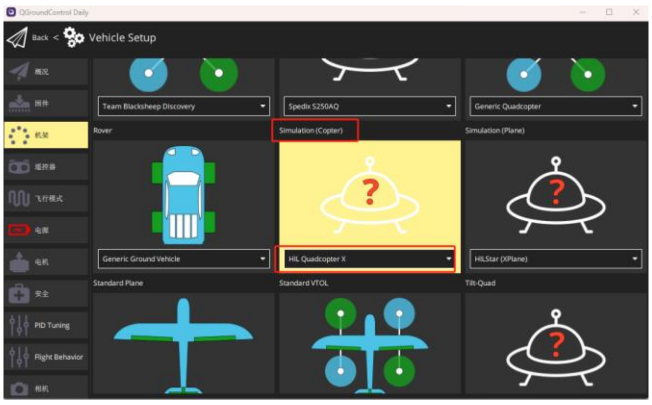
4.8. Multi-PX4+ Multi-NX Hardware-in-the-Loop

First, connect the flight controller to the computer, and open the QGC ground station to set the rack, as shown in the following figure:

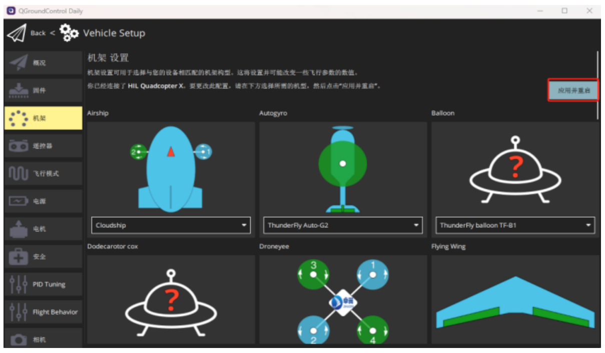
1. Select the vehicle settings:



2、Select Rack Settings, and then select HIL



3、Once you've made your selection, click Apply and restart for the rack setup to be successful.

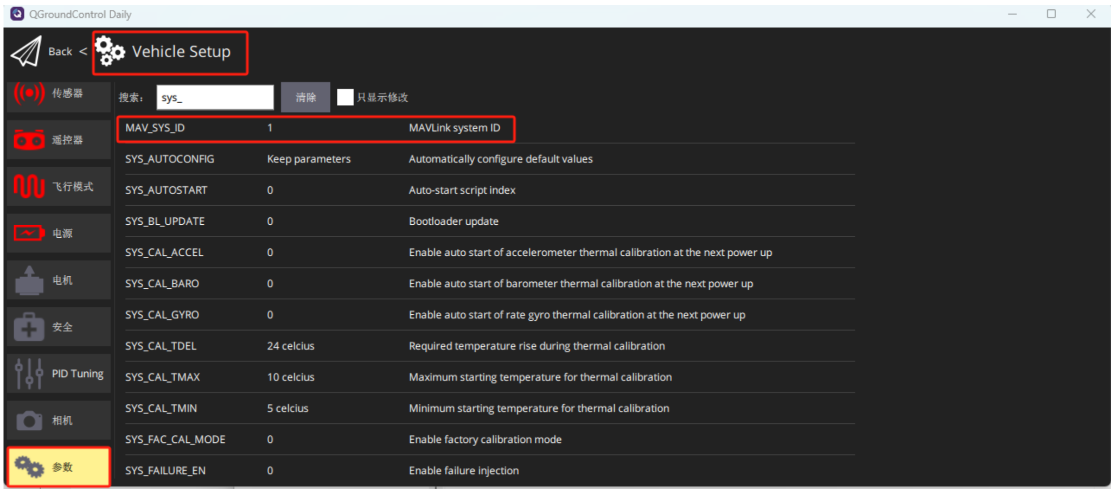


4.8.1. Bat Script Configuration

Please modify it based on the hardware-in-the-loop simulation script HITLRun.bat, and set the simulation mode SimMode=0 or SimMode=PX4_HITL for hardware-in-the-loop simulation. Set SET /a IsSysID=1 to enable the function of automatically calculating CopterID from SysID. It is possible to set the initial pose of each drone.

4.8.2. Flight Control Parameter Settings

Connect each hardware flight controller to NX via USB cable, open QGroundControl software, and set the ID number. This is so that the CopterSimID is consistent with the flight controller ID. The settings are as follows:



sys_

MAV_SYS_ID	1	MAVLink system ID
SYS_AUTOCONFIG	Keep parameters	Automatically configure default values
SYS_AUTOSTART	0	Auto-start script index
SYS_BL_UPDATE	0	Bootloader update
SYS_CAL_ACCEL	0	Enable auto start of accelerometer thermal calibration at the next power up
SYS_CAL_BARO	0	Enable auto start of barometer thermal calibration at the next power up
SYS_CAL_GYRO	0	Enable auto start of rate gyro thermal calibration at the next power up
SYS_CAL_TDEL	24 celcius	Required temperature rise during thermal calibration
SYS_CAL_TMAX	10 celcius	Maximum starting temperature for thermal calibration
SYS_CAL_TMIN	5 celcius	Minimum starting temperature for thermal calibration
SYS_FAC_CAL_MODE	0	Enable factory calibration mode
SYS_FAILURE_EN	0	Enable failure injection

4.8.3. NX Configuration

Identify the port number of the hardware flight controller in the Ubuntu system in NX, such as /dev/ttyACM0, and then enter the command "chmod 777 /dev/ttyACM0" in the terminal to authorize the port with read and write rights, and other ports are also operated accordingly. Then specify the port number or port number plus baud rate in the python program, e.g. PX4MavCtrler('/dev/ttyACM0') or e.g. PX4MavCtrler('/dev/ttyACM0:57600').

5. Introduction to vision-related communication ports

5.1. Description of UDP mutual communication port between the visual interface and CopterSim and RflySim

5.1.1. Port 20010 (default single RflySim3D window, the port where the vision interface communicates with RflySim3D for requests, such as collision detection, target properties, camera properties, etc.)

Quite the same as the specified port to communicate with different RflySim3D, 20010~20029 these 20 ports are one-to-one correspondence, for example, RflySim3D-0 refers to it is listening on 20010, and RflySim3D-3 refers to it listening on 20013

5.1.2. Port 20100 (CopterSim accepts emulation port to restart an aircraft)

CopterSim provides the mavlink communication port to the external of the current aircraft, of course this value can be something else, the key is the configuration in the bat that starts the RflySim platform. For more information about BAT script configuration, see BAT Script Configuration Method (Configuration for Vision)

5.1.3. Port 30100 (CopterSim Single Aircraft Accepts External Requests for IMU Data by Default)

The IMU data is requested for each aircraft through this port.

5.1.4. Port 31000 (Vision Interface Configuration: The current system accepts IMU data by default)

For example, you can call the sendImuReqClient(self, copterID=1, IP="", port=31000, freq=200) API to send a SensorReqCopterSim instance to request Imu data. Or call the sendImuReqServe(self, copterID=1, port=31000) API to create a thread, which receives the Imu data sent back by CopterSim in the thread and does not block the main thread. You can also call the sendImuReqCopterSim(self, copterID=1, IP="", port=31000, freq=200) API to request Imu data by sending a SensorReqCopterSim instance, and then start a thread to listen to and process the Imu data sent back by CopterSim.

5.1.5. Port 20005 (Vision Interface Requests Timestamp of Aircraft Takeoff from CopterSim)

This port mainly stores the timestamp of the terminal computer and the simulation, and detects whether the connection between the terminal computer and the simulation program is normal through the heartbeat. It is used to call the StartTimeStmplisten(self, cpID=1) interface, bind port 20005 of the machine, receive the heartbeat and timestamp data sent back by CopterSim in the subthread, ensure that the connection between the terminal and CopterSim is normal, and does not block the main thread. where cpID represents the ID of CopterSim.

5.1.6. Port 20006 (Listening for Data from RflySim3D)

This port is the struct that will be sent when RflySim3D turns on the data return mode (using the

"RflyReqVehicleData 1" command). It contains mainly collision-related data, which will be sent to the multicast IP "224.0.0.10:20006". It is also used to initialize self.udp_socketUE4 on the initUE4MsgRec(self) interface and start a thread t4(self. UE4MsgRecLoop) starts listening on 224.0.0.10:20006, as well as its own 20006 port. It can also be used to listen for 224.0.0.10:20006 on the UE4MsgRecLoop(self) interface, as well as its own handler on port 20006 to process messages returned by RflySim3D or CopterSim.

5.1.7. Description of data structures related to communication

5.1.7.1. RflyTimeStmp (Time information received from CopterSim)

5.1.7.2. VisionSensorReq (Reading the structure from the Config.json file to request sensor configuration from RflySim3D)

5.1.7.3. imuDataCopter (accepts IMU data structures from CopterSim)

5.1.7.4. SensorReqCopterSim (request relevant data from CopterSim, e.g. IMU)

5.1.7.5. reqVeCrashData (Listen to RflySim3D to object property data when it is hit by an airplane)

5.1.7.6. CoptReqData (request to RflySim3D to obtain the attributes of the specified aircraft)

5.1.7.7. ObjReqData (request to RflySim3D to get the properties of the specified object)

CameraData (request RflySim3D to get the properties of the specified camera)
The vision-related communication of the RflySim platform is divided into Python programs communicating with CopterSim, Python programs communicating with RflySim3D, and Python programs communicating with other ROS nodes in Ubuntu.

fifo: Abstract the first-in-first-out queue with python lists, and implement write and read methods to achieve the effect of similar operations to files.
1.def write(self, data):
I. Parameter Explanation:

1) data: the item to be added to the team

II. Function Explanation:

This method inserts data into the queue head.

2.def read(self):

I. Parameter Explanation:

II. Function Explanation:

This method deletes and returns the element at the end of the queue, and throws an IndexError exception if the queue is empty.

PX4SILIntFloat: This class encapsulates SILInts and SILFloats data output to the CopterSim DLL model.

I. _ _ init _ _.

```
def __init__(self):
self.checksum = 0
self.CopterID = 0
self.inSILInts = [0, 0, 0, 0, 0, 0, 0, 0]
self.inSILFLoats = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

def __init__(self, iv):
self.checksum = iv[0]
self.CopterID = iv[1]
self.inSILInts = iv[2:10]
self.inSILFLoats = iv[10:30]
```

This class has two constructors, where:

1. checksum： the check digit
2. CopterID： the Copter ID of the message
3. inSILInts： SILInts data exported to the CopterSim DLL model
4. inSILFLoats： SILFoats data exported to the CopterSim DLL model

reqVeCrashData ： This class is a struct sent by RflySim3D, which is sent when RflySim3D turns on the data postback mode (using the "RflyReqVehicleData 1" command). It contains mainly collision-related data, which will be sent to the multicast IP "224.0.0.10:20006".

I. _ _ init _ _.

```
def __init__(self):
self.checksum = 1234567897
self.copterID = 0
self.vehicleType = 0
self.CrashType = 0
self.runnedTime = 0
self.VelE = [0, 0, 0]
self.PosE = [0, 0, 0]
self.CrashPos = [0, 0, 0]
self.targetPos = [0, 0, 0]
self.AngEuler = [0, 0, 0]
self.MotorRPMS = [0, 0, 0, 0, 0, 0, 0, 0]
self.ray = [0, 0, 0, 0, 0, 0]
self.CrashedName = ""
```

1. copterID: data indicating which copter the structure belongs to.
2. vehicleType: indicates the style of the Copter

3. CrashType: indicates the type of collision object, -2 indicates the ground, -1 indicates the scene static object, 0 indicates no collision, and 1 or more indicates the ID number of the aircraft being hit
4. runnedTime: The timestamp of the current aircraft
5. VelE: Current aircraft velocity (m/s)
6. PosE: Current Aircraft Position (North East, in meters)
7. CrashPos: Coordinates of the collision point (north-east, in meters)
8. targetPos: the center coordinate of the object to be touched (north-east, unit: meters)
9. AngEuler: Roll, Pitch, Yaw, Arc of the current aircraft
10. MotorRPMS: The motor speed of the current aircraft
11. Ray: The plane's front, back, left, right, and up and down scanning lines
12. CrashedName: The name of the object being touched

RflyTimeStmp：This class encapsulates the message sent to the specified remote computer port 20005, mainly stores the timestamp of the terminal computer and the simulation, and detects whether the connection between the terminal computer and the emulator program is normal through the heartbeat.

__init__

```
def __init__(self):
self.checksum = 1234567897
self.copterID = 0
self.SysStartTime = 0
self.SysCurrentTime = 0
self.HeartCount = 0
```

This is the constructor of the class, where:

1. checksum：is the check digit of the data, which is used to check whether an exception has occurred during data transmission.
2. copterID：indicates the copter ID of the message.
3. SysStartTime：is the timestamp of the start of the simulation in milliseconds, using the Greenwich Mean starting point.
4. SysCurrentTime：the current timestamp in Windows, in milliseconds, using the Greenwich Mean starting point.
5. HeartCount：The counter of the heartbeat packet.

PX4ExtMsg：This class encapsulates the pixhawk state data returned by CopterSim.

I. _ _ init _ _.

```
def __init__(self):
self.checksum=0
self.CopterID=0
self.runnedTime=0
self.controls=[0,0,0,0,0,0,0,0]
```

This is the constructor of the class, where:

1. checksum：data validation.
2. CopterID：the Copter ID of the message.
3. runnedTime：timestamp.
4. controls：Status data in Pixhawk.

CoptReqData: This class encapsulates the aircraft data returned by RflySim3D, and after the PX4MavCtrler.reqCamCoptObj() function calls RflySim3D's "RflyReqObjData" command, RflySim3D sends information about a Copter according to the parameters of the command (the Copter ID is attached when sending the command to RflySim3D).

I. _ _ init _ _.

```
    def __init__(self):
          self.checksum = 0 # 1234567891 as a check
   Self. CopterID = 0 # Aircraft ID
          self.PosUE = [0,0,0]
          self.angEuler = [0,0,0]
          self.boxOrigin = [0,0,0]
          self.BoxExtent = [0,0,0]
          self.timestmp = 0
          self.hasUpdate=True
```

This is the constructor of the class, where:

checksum：is the check digit of the data, which is used to check whether an exception has occurred during data transmission.

CopterID：indicates the copter ID of the message.

PosUE：Indicates the location of the Copter (m, North East)

angEuler：Represents the angle of the Copter (radians, Roll, Pitch, Yaw)

boxOrigin：Surround the geometric center of the box (m, north-east)

BoxExtent：Half the length of the side of the box (m, X-axis (forward), Y-axis (right), Z-axis (up))

Timestmp：timestamp

hasUpdate：This value is not sent by RflySim3D

ObjReqData：This class encapsulates the data returned by RflySim3D, and when the PX4MavCtrler.reqCamCoptObj() function calls RflySim3D's "RflyReqObjData" command, RflySim3D sends information about an object according to the parameters of the command (the name of the object is attached when sending the command to RflySim3D).

__init__

```
   def __init__(self):
          self.checksum = 0 # 1234567891 as a check
          self.seqID = 0
          self.PosUE = [0,0,0]
          self.angEuler = [0,0,0]
          self.boxOrigin = [0,0,0]
          self.BoxExtent = [0,0,0]
          self.timestmp = 0
          self.ObjName=''
          self.hasUpdate=True
```

This is the constructor of the class, where:

1. checksum: is the check digit of the data, which is used to check whether an exception has occurred during data transmission.
2. seqID: always 0. (Obj is not Copter, it has no ID to return)
3. PosUE: Indicates the position of the object (m, north-east)
4. angEuler: Angle (radian, Roll, Pitch, Yaw)
5. boxOrigin: Surround the geometric center of the box (m, north-east)
6. BoxExtent: Half the length of the side of the box (m, X-axis (forward), Y-axis

(right), Z-axis (up))

7. Timestmp: The time that the current scene has existed (seconds, the time when the scene exists (because it is not a Copter, there is no timestamp))
8. ObjName: The name of the object
9. hasUpdate: This value is not sent by RflySim3D

CameraData: This class encapsulates the data returned by RflySim3D, after the PX4MavCtrler.reqCamCoptObj() function calls the "RflyReqObjData" command of RflySim3D, RflySim3D sends information about a certain camera (vision sensor) according to the parameters of the command (when sending a command to RflySim3D, The SeqID of the camera (vision sensor) is included).

One __init__

```
def __init__(self):
        self.checksum = 0 #1234567891 as a check
        self.SeqID = 0
        self.TypeID = 0
        self.DataHeight = 0
        self.DataWidth = 0
        self.CameraFOV = 0
        self.PosUE = [0,0,0]
        self.angEuler = [0,0,0]
        self.timestmp = 0
        self.hasUpdate=True
```

This is the constructor of the class, where:

1. checksum: is the check digit of the data, which is used to check whether an exception has occurred during data transmission.
2. seqID: The ID of the vision sensor
3. TypeID: Type of vision sensor (RPG, depth map, etc.)
4. DataHeight: The height of the data (pixels)
5. DataWidth: The width of the data (pixels)
6. CameraFOV: Camera's horizontal field of view (in degrees)
7. PosUE: Indicates the position of the object (m, north-east)
8. angEuler: indicates the angle of the camera (radian, Roll, Pitch, Yaw)
9. Timestmp: timestamp
10. hasUpdate: This value is not sent by RflySim3D

5.2. Python program communication interface with CopterSim

5.2.1. PX4SILIntFloat （ SILInts and SILFloats data exported to CopterSim DLL models ）

I. _ _ init _ _.

```
def __init__(self):
self.checksum = 0
self.CopterID = 0
self.inSILInts = [0, 0, 0, 0, 0, 0, 0, 0]
self.inSILFLoats = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

def __init__(self, iv):
self.checksum = iv[0]
self.CopterID = iv[1]
self.inSILInts = iv[2:10]
self.inSILFLoats = iv[10:30]
```

This class has two constructors, where:

1. checksum: the check digit
2. CopterID: the Copter ID of the message
3. inSILInts: SILInts data exported to the CopterSim DLL model

5.2.2. InitTrueDataLoop （Start two threads to receive real data from CopterSim and PX4 data respectively）

```
def InitTrueDataLoop(self):
```

I. Parameter Explanation:

II. Function Explanation:

Start two threads to receive real data from CopterSim and PX4 data respectively.

5.2.3. InitMavLoop （ Enable MAVLink to listen to CopterSim data and update it in real time ）

```
def InitMavLoop(self,UDPMode=2)
```

一、Parameter Explanation:s

1) UDPMode：UDPmode

二、Parameter Explanation:

Enable MAVLink to listen to CopterSim data and update it in real time. where the

UDPModel value is:

0: Corresponding to the UDP_Full mode, Python transmits complete UDP data to CopterSim, and the amount of data transmitted is small. After CopterSim receives the data, it converts it to Mavlink and transmits it to the PX4 flight controller. It is suitable for simulation of small and medium-sized clusters (less than 10 in number).

1: Corresponding to the UDP_Simple mode, the packet size and sending frequency are smaller than those of the UDP_Full mode; It is suitable for large-scale swarm simulation, and the number of drones is less than 100.

2: Corresponding to Mavlink_Full mode (default mode), Python directly sends MAVLink messages to CopterSim, and then forwards them to PX4, which is suitable for stand-alone control with large data volume; It is suitable for single aircraft or a small number of aircraft simulation, and the number of UAVs is less than 4;

3: Corresponding to the Mavlink_Simple mode, it will block some MAVLink message packets, reduce the data frequency, and send much smaller data than MAVLink_Full, which is suitable for multi-machine cluster control. It is suitable for small-scale swarm simulation, and the number of drones is less than 8.

4: Corresponding to the Mavlink_NoSend mode, CopterSim will not send MAVLink data to the outside, this mode needs to cooperate with hardware-in-the-loop simulation + data transmission serial port communication, and transmit MAVLink through wired mode, this mode has the smallest amount of data in the LAN, suitable for distributed vision hardware-in-the-loop simulation, and the number of drones is not limited.

5.2.4. TimeStmploop（Listen to the timestamp data of a specified CopterSim in a loop）

```
def TimeStmploop(self)
```

I. Parameter Explanation:

II. Function Explanation:

Listen to the timestamp data of a specified CopterSim in a loop.

5.2.5. StartTimeStmplisten（The thread loop listens to the timestamp data of the specified CopterSim and does not block the main thread）

```
def StartTimeStmplisten(self,cpID=0)
```

一、Parameter Explanatio：

1) cpID：copter ID

二、Function Explanation：

Create a thread loop to listen to the timestamp data of the specified CopterSim and not block the main thread.

5.2.6. endMavLoop （ Stop receiving CopterSim data, which is the same as stopRun ）

```
def endMavLoop(self)
```

一、Parameter Explanatio:

二、Function Explanation:

Stop receiving CopterSim data, which is the same as stopRun.

5.3. Python program to communicate with RflySim3D Interface

5.3.1. ReqVeCrashData (structure sent by RflySim3D, mainly data related to collision)

I. _ _ init _ _.

```
def __init__(self):
self.checksum = 1234567897
self.copterID = 0
self.vehicleType = 0
self.CrashType = 0
self.runnedTime = 0
self.VelE = [0, 0, 0]
self.PosE = [0, 0, 0]
self.CrashPos = [0, 0, 0]
self.targetPos = [0, 0, 0]
self.AngEuler = [0, 0, 0]
self.MotorRPMS = [0, 0, 0, 0, 0, 0, 0, 0]
self.ray = [0, 0, 0, 0, 0, 0]
self.CrashedName = ""
```

This class is the struct sent by RflySim3D. It is the struct that will be sent when RflySim3D turns on the data return mode (using the "RflyReqVehicleData 1" command). It mainly contains data related to collision, which will be sent to multicast IP "224.0.0.10: 20006".

1. CopterID: indicates the data of which copter the struct is.
2. VehicleType: Indicates the style of this Copter
3. CrashType: indicates the collision object type, -2 indicates the ground, -1 indicates the scene static object, 0 indicates no collision, and above 1 indicates the ID number of the collided aircraft
4. RunnedTime: Time stamp of the current aircraft
5. VelE: Current aircraft speed (m/s)

6. PosE: Current aircraft position (NE, in meters)
7. CrashPos: Coordinates of the impact point (NE, in meters)
8. TargetPos: coordinates of the center of the impacted object (northeast, unit: m)
9. AngEuler: Euler angle of the current aircraft (Roll, Pitch, Yaw, radians)
10. MotorRPMS: Current aircraft motor speed
11. Ray: Front, back, left, right, up and down scan lines of the aircraft
12. CrashedName: The name of the touched object

5.3.2. CoptReqData (aircraft data returned by RflySim3D, information of a Copter sent)

I. _ _ init _ _.

```
 def __init__(self):
Self. Checksum = 0 # 1234567891 as check
Self. CopterID = 0 # Aircraft ID
       self.PosUE = [0,0,0]
       self.angEuler = [0,0,0]
       self.boxOrigin = [0,0,0]
       self.BoxExtent = [0,0,0]
       self.timestmp = 0
       self.hasUpdate=True
```

For the aircraft data returned by RflySim3D, after the PX4MavCtrler.reqCamCoptObj () function calls the "RflyReqObjData" command of RflySim3D, RflySim3d, according to the parameters of the command, The information of a Copter sent (the ID of the Copter is attached when sending the command to RflySim3D).

This is the constructor for this class, where:

1. Check sum: is the check bit of the data, which is used to check whether an exception occurs during data transmission.
2. CopterID: indicates the copter ID corresponding to the message.
3. PosUE: indicates the location of the Copter (m, NE)
4. AngEuler: represents the angle of this Copter (radians, Roll, Pitch, Yaw)
5. BoxOrigin: Geometric center of bounding box (m, NE)
6. Box Extent: Half of the side length of the bounding box (meters, X axis (forward), Y axis (right), Z axis (up))
7. Times TMP: timestamp
8. HasUpdate: Set to True if the data can be obtained normally, call it outside, set to True to indicate that the data can be read, and then set to False

5.3.3. ObjReqData (data returned by RflySim3D, information about an object sent)

一. __init__

```
 def __init__(self):
    Self. Checksum = 0 # 1234567891 as check
           self.seqID = 0
           self.PosUE = [0,0,0]
           self.angEuler = [0,0,0]
           self.boxOrigin = [0,0,0]
           self.BoxExtent = [0,0,0]
           self.timestmp = 0
           self.ObjName=''
           self.hasUpdate=True
```

For the data returned by RflySim3D, after the PX4MavCtrler.reqCamCoptObj () function calls the "RflyReqObjData" command of RflySim3D, Rflymim3d, according to the parameters of the command, The information of an object sent (the Name of the object is attached when sending the command to RflySim3D).

This is the constructor for this class, where:

1. Check sum: is the check bit of the data, which is used to check whether an exception occurs during data transmission.
2. SeqID: always 0. (Obj is not a Copter, it has no ID to return)
3. PosUE: indicates the position of the object (m, NE)
4. AngEuler: represents the angle of this Obj (radians, Roll, Pitch, Yaw)
5. BoxOrigin: Geometric center of bounding box (m, NE)
6. Box Extent: Half of the side length of the bounding box (meters, X axis (forward), Y axis (right), Z axis (up))
7. Timestmp: Time the current scene has been in existence (seconds, the time the scene has been in existence (because it is not a Copter, there is no timestamp))
8. ObjName: The name of the object
9. HasUpdate: Set to True if the data can be obtained normally, call it outside, set to True to indicate that the data can be read, and then set to False

5.3.4. CameraData (data returned by RflySim3D, information of a certain camera (vision sensor) sent)

One __init__

```
    def __init__(self):
    Self. Checksum = 0 # 1234567891 as check
           self.SeqID = 0
           self.TypeID = 0
           self.DataHeight = 0
           self.DataWidth = 0
           self.CameraFOV = 0
           self.PosUE = [0,0,0]
           self.angEuler = [0,0,0]
           self.timestmp = 0
           self.hasUpdate=True
```

For the data returned by RflySim3D, after the PX4MavCtrler.reqCamCoptObj () function calls the "RflyReqObjData" command of RflySim3D, Rflymim3d, according to the parameters of the command, The information of a camera (vision sensor) sent (the SeqID of the camera (vision sensor) is attached when the command is sent to RflySim3D).

This is the constructor for this class, where:

1. Check sum: is the check bit of the data, which is used to check whether an exception occurs during data transmission.
2. SeqID: ID of the vision sensor
3. TypeID: The type of visual sensor (RPG, depth map, etc.)
4. DataHeight: Data height (pixels)
5. DataWidth: Data width (pixels)
6. CameraFOV: Camera field of view (in degrees)
7. PosUE: indicates the position of the object (m, NE)
8. AngEuler: indicates the angle of the Camera (radian, Roll, Pitch, Yaw)
9. Times TMP: timestamp
10. HasUpdate: Set to True if the data can be obtained normally, call it outside, set to True to indicate that the data can be read, and then set to False

5.3.5. UE4MsgRecLoop (used to process the messages returned by RflySim3D or CopterSim, there are 6 kinds of messages in total)

```
def UE4MsgRecLoop(self)
```

一、Explanation of parameters:

二、Function interpretation:

It monitors 224.0.0.10: 20006 and its own 20006 port. It is used to process the messages returned by RflySim3D or CopterSim. There are 6 kinds of messages in total:

1) CopterSimCrash with length of 12 bytes:

```
struct CopterSimCrash {
int checksum;
int CopterID;
int TargetID;
}
```

For the collision data returned by RflySim3D, RflySim3D will start the collision detection mode when the P key is pressed in RflySim3D. If a collision occurs, the data will be returned. The P + number can select the sending mode (0 local sending, 1 LAN sending, 2 LAN sending only in case of collision). The default is local sending.

2) PX4SILIntFloat of length 120 bytes:

```
struct PX4SILIntFloat{
int checksum;//1234567897
int CopterID;
int inSILInts[8];
float inSILFLoats[20];
};
```

3) ReqVeCrashData of length 160 bytes:

```
struct reqVeCrashData {
int checksum; //Packet check code 1234567897.
int copterID; //ID number of the current aircraft
int vehicleType; //The style of the current aircraft
Int CrashType;//Collision object type, -2 indicates ground, -1 indicates scene static object, 0 indicates no collision, and above 1 indicates ID number of the collided aircraft
```

```
    double runnedTime; //Timestamp of the current aircraft
    float VelE[3]; //Speed of the current aircraft
    float PosE[3]; //Current aircraft position
    Float CrashPos [3];//Coordinates of the collision point
    Float targetPos [3];//the center coordinate of the touched object
    float AngEuler[3]; //Euler angle of the current aircraft
    float MotorRPMS[8]; //Current aircraft motor speed
    float ray[6]; //front, back, left, right, up and down scan lines of the aircraft
    Char CrashedName [20] = { 0 };//Name of the touched object
    }
```

This class has been introduced before. When data return is enabled, RflySim3D will send reqVeCrashData for all Copters (send when data changes, once per second).

4) CameraData with length of 56 bytes:

```
    struct CameraData { //56
    int checksum = 0;//1234567891
    int SeqID; //camera serial number
    Int TypeID;//camera type
    Int Data Height;//pixel height
    Int Data Width;//pixel width
    Float Camera FOV;//camera field angle
    float PosUE[3]; //Camera center position
    Float angEuler [3];//camera Euler angle
    Double times TMP;//timestamp
    };
```

RflySim3D returns the structure regularly according to the command sent by the reqCamCoptObj function, and the program will store it in the self. CamDataVect after receiving it.

5) CoptReqData with length of 64 bytes:

```
    struct CoptReqData { //64
    int checksum = 0; //123456 78 91 as check
    Int CopterID;//Aircraft ID
    float PosUE[3]; //Object center position (specified during artificial 3D modeling, the
attitude coordinate axis is not necessarily at the geometric center)
    Float angEuler [3];//Euler angle of object
    Float boxOrigin [3];//object geometric center coordinate
    Float Box Extent [3];//half of the length, width and height of the object frame
    Double times TMP;//timestamp
    };
```

RflySim3D returns the structure regularly according to the command sent by the reqCamCoptObj function, and the program will store it in the self. CoptDataVect after receiving it.

6) ObjReqData of length 96 bytes:

```
    struct ObjReqData { //96
    int checksum = 0; //123456 78 91 as check
    int seqID = 0;
    float PosUE[3]; //Object center position (specified during artificial 3D modeling, the
attitude coordinate axis is not necessarily at the geometric center)
    Float angEuler [3];//Euler angle of object
    Float boxOrigin [3];//object geometric center coordinate
    Float Box Extent [3];//half of the length, width and height of the object frame
    Double times TMP;//timestamp
    Char ObjName [32] = { 0 };//Name of object touched
    };
```

RflySim3D returns the structure regularly according to the command sent by the reqCamCoptObj function, and the program will store it in the self. ObjDataVect after receiving it.

The received data can be retrieved using the getCamCoptObj function.

5.3.6. Get CamCoptObj (get the specified data from RflySim3D and store the monitored three types in three lists)

```
def getCamCoptObj(self,type=1,objName=1)
```

一、Explanation of parameters:

1) Type: 0 for camera, 1 for airplane, 2 for object

二、Function interpretation:

Obtain the specified data from RflySim3D. The UE4MsgRecLoop function enables the monitoring of RflySim3D messages and stores the monitored three types in three lists. This function searches the data in these lists. Therefore, you need to use the reqCamCoptObj function to request the relevant data from RflySim3D.

5.3.7. reqCamCoptObj (Request data for objects in the scene (does not create new objects, but gets data for existing objects))

```
def reqCamCoptObj(self,type=1,objName=1,windowID=0)
```

一、Explanation of parameters:

1) Type: 0 for camera, 1 for airplane, 2 for object
2) ObjName: type represents camera seqID; when s represents aircraft, objName corresponds to CopterID; when s represents object, objName corresponds to object name
3) WindowID: Indicates which RflySim3D you want to send a message to. The default is window 0. (Do not send this data to all RflySim3D, because the value returned is the same, and there is no need to consume additional performance.)

二、Function interpretation:

Send a data request to RflySim3D to request data for objects in the scene (not to create new objects, but to get data for existing objects). It can be a visual sensor, a Copter, and a common object in the scene. See CoptReqData class, ObjReqData class and CameraData class for details of the obtained data.

5.3.8. sendUE4Cmd (To RflySim3D Send a Byte String Command)

```
def sendUE4Cmd(self, cmd, windowID=-1)
```

一、Explanation of parameters:

1) Cmd: Byte string of the command sent
2) WindowsID: affects the sending target port, which can be taken as -1 during normal multicast. If the designated RflySim3D on the designated computer is required to receive, it can be set according to the target computer IP and the window title number of the RflySim3D program. (This serial number corresponds to the 20 ports from 20010 to 20029 mentioned above. For example, RflySim3D-0 means that it is

monitoring the 20010, and RflySim3D-3 means that it is monitoring the 20013.)

二、Function interpretation:

This function is to send a "command" to RflySim3D, where cmd is a byte string, representing the command sent to RflySim3D, and the structure sent out is as follows:

```
struct Ue4CMD{
    int checksum;
    char data[52];
}
```

For example

```
mav = PX4MavCtrl.PX4MavCtrler(20100)
mav.sendUE4Cmd(b'RflyChangeMapbyName Grasslands')
```

It sends a command, "RflyChangeMapbyName Grasslands", "which tells RflySim3D to change the map to something called" Grasslands. ".

For other commands and their functions, please refer to "OneDrive \ RflySimDocs \ Advanced Version Manuscript \ Development Version \ Chapter 3-3D Scene Modeling and Simulation \ Demo _ Resources _ 3 \ E0 \ E0 _ 2 \ RflySim3D Command Interface.docx". Here they are listed and briefly introduced:

1. RflyShowTextTime (String txt, float time) \ \ Make the UE display txt for time seconds
2. RflyShowText (String txt) \ \ Make the UE display txt for 5 seconds
3. RflyChangeMapbyID (int ID) \ \ Switch the RflySim3D scene map according to the map ID
4. RflyChangeMapbyName (String txt) \ \ Switch the RflySim3D scene map based on the map name
5. RflyChangeView KeyCmd (String key, int num) \ \ Consistent with pressing a key + num in RflySim3D
6. RflyCameraPosAngAdd (float X, float y, float Z, float roll, float pitch, float yaw) \ \ Adds an offset value to the camera's position and angle
7. RflyCameraPosAng (float X, float y, float Z, float roll, float pitch, float yaw) \ \ Sets the position and angle of the camera (world coordinates of the UE)
8. RflyCameraFovD egrees (float degrees) \ \ Sets the camera's view volume FOV angle
9. RflyChange3D Model (int CopterID, int veTypes = 0) \ \ Modify a UAV model style
10. RflyChangeVehicle Size (int CopterID, float size = 0) \ \ Modify the zoom size of a UAV
11. RflyMoveVehiclePosAng(int CopterID, int isFitGround, float x, float y, float z, float roll, float pitch, Float yaw) \ \ Set an offset value for the position and angle of the drone, and isFitGround sets whether the drone fits the ground
12. RflySetVehiclePosAng(int CopterID, int isFitGround, float x, float y, float z, float roll, float pitch, Float yaw) \ \ Sets the position and angle of the drone
13. RflyScanTerrainH(float xLeftBottom(m), float yLeftBottom(m), float xRightTop(m), float yRightTop(m), Float scanHeight (m), float scanInterval (m)) \ \ Scan the terrain to generate a PNG height map and txt, which is needed by the CopterSim program to know what terrains the UE has and their elevations
14. RflyCesium OriPos (double lat, double lon, double Alt) \ \ Modifies the origin position of Cesium based on latitude and longitude
15. RflyClearCapture () \ \ Clear captured images
16. RflySetActuatorPWMs(int CopterID, float pwm1, float pwm2, float pwm3, float pwm4, float pwm5, float pwm6, float pwm7, float pwm8); \ \ Pass in 8 values and trigger the interface function of the target UAV's blueprint

17. RflySetActuatorPWMsExt(int CopterID, float pwm9, float pwm10, float pwm11, float pwm12, float pwm13, float pwm14, float pwm15, float pwm16, float pwm17, float pwm18, float pwm19, float pwm20, float pwm21, float pwm22, float pwm23, Float PWM 24); \ \ Pass in 16 values and trigger the blueprint interface function of the target UAV. This function requires the full version to be functional
18. RflyReqVehicleData(FString isEnable); If the 'isEnable' is not 0, RflySim3D starts sending the data of all Copters (reqVeCrashData as described above).
19. RflySetPosScale (float scale); \ \ Scale of global position
20. RflyLoad3DFile (FString FileName); \ \ Load and execute the TXT script file under the path
21. RflyReqObjData (int opFlag, FString objName, FString colorStr); \ \ Request to get data of objects in 3D scene
22. RflySetIDLabel (int CopterID, FString Text, FString colorStr, float size); \ \ Set the top ID display content of a Copter (default display CopterID)
23. RflySetMsgLabel(int CopterID, FString Text, FString colorStr, float size, float time, int flag); \ \ Set the contents of the Message display on the top of a Copter
24. RflyDelVehicles (FString CopterIDList); \ \ Remove some Copters (comma is a separator)
25. RflyDisableVe Move (FString CopterIDList, int disable); \ \ Rejects the message of the Copter that receives the specified ID (comma is the delimiter)
26. In addition, there are some built-in commands of the UE that can be used, for example, the'stat FPS 'can display the current frame rate, and the't. Maxfps 60' can set the maximum frame rate to 60.

5.4. Python programs and other Ubuntu Inside ROS Node communication

The interface file (VisionCaptureAPI. Py) supports running in the ROS/ROS2 environment by adding the following statement:

```
# Enable ROS Publish Mode
import VisionCaptrueApi
VisionCaptureApi.isEnableRosTrans =True
vis = VisionCaptureApi.VisionCaptureApi()
```

Instructions: The above statement only runs under the Linux system, and does not support the ROS environment under the windows system at present. After running the above code (ROS runs roscore first to confirm that rosmaster can use rosrun or directly python3 XXX. Py instruction to run the code), You can publish the topic corresponding to the config. JSON. Refer to 错误!未找到引用源。

5.5. Application of RflySim platform time stamp alignment with other platform

5.5.1. RflySim Timestamp, Remote System Timestamp, ROS Timestamp

5.5.1.1. RflySim timestamp

RflySim time is assigned based on the system timestamp of its running platform. If RflySim is running on a Windows system, then the reference timestamp is the Windows system timestamp. If RflySim is running on a Linux system, then the reference timestamp is the Windows system timestamp. When CopterSim is used to load the model, timestamps like images, point clouds, and IMU outputs are based on relative timestamps from the moment the aircraft takes off. When CopterSim is not used and only RflySim3D is used, the image time is real-time system timestamp

5.5.1.2. Remote System Timestamp

Usually the RflySim platform runs on Windows systems, and we need to do algorithms and distributed deployments on Ubuntu systems, so each distributed host computer has its own system timestamp, although each system timestamp is defined according to the standard (from 0: Elapsed seconds), but the timestamp for each system will not be the same at the same time.

5.5.1.3. ROS timestamp

Each Node in the distributed system should run under the same timestamp. Therefore, in order to solve the timestamp alignment of different systems, ROS2 sets a ROS timestamp, which has the same reference time point between subsystems. Therefore, at the same time, the RflySim timestamp ≠ the remote system timestamp, the remote system timestamp and the ROS timestamp are equal only when there is only one subsystem in the distributed system, and they are not equal in other cases.

5.5.2. Timestamp alignment across system

As for the timestamp definition view RflySim Timestamp, Remote System Timestamp, ROS Timestamp of each system, the requirement of applying the RflySim platform to the ROS distribution is to obtain the timestamp generated by the image and other sensor data with the ROS timestamp as the reference. To do this, you need to get the timestamp of the start of the simulation of the RflySim platform on the remote side, as well as the relative timestamp of the image and other sensors under this timestamp. If the remote system timestamp is not considered, and only the ROS timestamp and the RflySim platform timestamp are considered, a difference between the ROS timestamp and the RflySim platform timestamp is calculated. Assuming that this difference is DT (both positive and negative are possible), then the time under the ROS timestamp is data _ ROS _ time = RflySim _ start _ time + RflySim _ data _ stmp + DT. The RflySim _ start _ time is the time stamp when the RflySim platform starts simulation recording, while the RflySim _ data _ stmp is the time when the sensor data generates the relative RflySim _ start _ time. About the Interface File (VisionCaptureApi. Py) Implementing Reference Functions 错误!未找到引用源。and Functions 错误!未找到引用源。

6. RflySim3D control interface UE4CtrlAPI.py

6.1. Scene control interface (send command to control RflySim3D)

6.1.1. sendUE4Pos (Set the pose, type, etc. To the object of the specified copter _ ID in RflySim3D)

```
    def sendUE4Pos(self,copterID=1,vehicleType=3,MotorRPMSMean=0,PosE=[0, 0, 0],AngEuler=[0, 0, 0],windowID=-1)
```

一、Explanation of parameters:

1. CopterID: ID of the set Copter
2. VehicleType: The style of the Copter set (determined in the XML)
3. MotorRPMSMean: Indicates the average value of the 8-bit actuator data (the same value for 8 actuators)
4. PosE: indicates the set position of the Copter (m, NE)
5. AngEuler: Represents the Euler angles (radians, roll, pitch, yaw) set for this Copter
6. WindowID: affects the sending target port, which can be taken as -1 during normal multicast. If the designated RflySim3D on the designated computer is required to receive, it can be set according to the target computer IP and the window title number of the RflySim3D program. (This serial number corresponds to the 20 ports from 20010 to 20029 mentioned above. For example, RflySim3D-0 means that it is monitoring the 20010, and RflySim3D-3 means that it is monitoring the 20013.)

二、Function interpretation:

Send the data of a Copter to all RflySim3D in the LAN. If there is no object with the Copter ID, it will Create an object like this. . Where vehicleType denotes the style of the Copter, PosE denotes the position of the Copter (m, NE), AngEuler denotes the Euler angles of the Copters (radians, roll, pitch, yaw), MotorRPMSMean represents the average of the 8-bit actuator data (the same value for 8 actuators).

6.1.2. sendUE4Cmd (To RflySim3D Send a "command")

```
def sendUE4Cmd(self, cmd, windowID=-1)
```

一、Explanation of parameters:

1) Cmd: Byte string of the command sent
2) WindowsID: affects the sending target port, which can be taken as -1 during normal multicast. If the designated RflySim3D on the designated computer is required to receive, it can be set according to the target computer IP and the window title number of the RflySim3D program. (This serial number corresponds to the 20 ports from 20010 to 20029 mentioned above. For example, RflySim3D-0 means that it is monitoring the 20010, and RflySim3D-3 means that it is monitoring the 20013.)

二、Function interpretation:

This function is to send a "command" to RflySim3D, where cmd is a byte string, representing the command sent to RflySim3D, and the structure sent out is as follows:

```
struct Ue4CMD{
    int checksum;
    char data[52];
}
```

For example

```
mav = PX4MavCtrl.PX4MavCtrler(20100)
mav.sendUE4Cmd(b'RflyChangeMapbyName Grasslands')
```

It sends a command, "RflyChangeMapbyName Grasslands", "which tells RflySim3D to change the map to something called" Grasslands. ".

For other commands and their functions, please refer to "OneDrive \ RflySimDocs \ Advanced Version Manuscript \ Development Version \ Chapter 3-3D Scene Modeling and Simulation \ Demo _ Resources _ 3 \ E0 \ E0 _ 2 \ RflySim3D Command Interface.docx". Here they are listed and briefly introduced:

1. RflyShowTextTime (String txt, float time) \ \ Make the UE display txt for time seconds
2. RflyShowText (String txt) \ \ Make the UE display txt for 5 seconds
3. RflyChangeMapbyID (int ID) \ \ Switch the RflySim3D scene map according to the map ID
4. RflyChangeMapbyName (String txt) \ \ Switch the RflySim3D scene map based on the map name
5. RflyChangeView KeyCmd (String key, int num) \ \ Consistent with pressing a key + num in RflySim3D
6. RflyCameraPosAngAdd (float X, float y, float Z, float roll, float pitch, float yaw) \ \ Adds an offset value to the camera's position and angle
7. RflyCameraPosAng (float X, float y, float Z, float roll, float pitch, float yaw) \ \ Sets the position and angle of the camera (world coordinates of the UE)
8. RflyCameraFovD egrees (float degrees) \ \ Sets the camera's view volume FOV angle
9. RflyChange3D Model (int CopterID, int veTypes = 0) \ \ Modify a UAV model style
10. RflyChangeVehicle Size (int CopterID, float size = 0) \ \ Modify the zoom size of a UAV
11. RflyMoveVehiclePosAng(int CopterID, int isFitGround, float x, float y, float z, float roll, float pitch, Float yaw) \ \ Set an offset value for the position and angle of the drone, and isFitGround

sets whether the drone fits the ground

12. RflySetVehiclePosAng(int CopterID, int isFitGround, float x, float y, float z, float roll, float pitch, Float yaw) \ \ Sets the position and angle of the drone
13. RflyScanTerrainH(float xLeftBottom(m), float yLeftBottom(m), float xRightTop(m), float yRightTop(m), Float scanHeight (m), float scanInterval (m)) \ \ Scan the terrain to generate a PNG height map and txt, which is needed by the CopterSim program to know what terrains the UE has and their elevations
14. RflyCesium OriPos (double lat, double lon, double Alt) \ \ Modifies the origin position of Cesium based on latitude and longitude
15. RflyClearCapture () \ \ Clear captured images
16. RflySetActuatorPWMs(int CopterID, float pwm1, float pwm2, float pwm3, float pwm4, float pwm5, float pwm6, float pwm7, float pwm8); \ \ Pass in 8 values and trigger the interface function of the target UAV's blueprint
17. RflySetActuatorPWMsExt(int CopterID, float pwm9, float pwm10, float pwm11, float pwm12, float pwm13, float pwm14, float pwm15, float pwm16, float pwm17, float pwm18, float pwm19, float pwm20, float pwm21, float pwm22, float pwm23, Float PWM 24); \ \ Pass in 16 values and trigger the blueprint interface function of the target UAV. This function requires the full version to be functional
18. RflyReqVehicleData(FString isEnable); If the 'isEnable' is not 0, RflySim3D starts sending the data of all Copters (reqVeCrashData as described above).
19. RflySetPosScale (float scale); \ \ Scale of global position
20. RflyLoad3DFile (FString FileName); \ \ Load and execute the TXT script file under the path
21. RflyReqObjData (int opFlag, FString objName, FString colorStr); \ \ Request to get data of objects in 3D scene
22. RflySetIDLabel (int CopterID, FString Text, FString colorStr, float size); \ \ Set the top ID display content of a Copter (default display CopterID)
23. RflySetMsgLabel(int CopterID, FString Text, FString colorStr, float size, float time, int flag); \ \ Set the contents of the Message display on the top of a Copter
24. RflyDelVehicles (FString CopterIDList); \ \ Remove some Copters (comma is a separator)
25. RflyDisableVe Move (FString CopterIDList, int disable); \ \ Rejects the message of the Copter that receives the specified ID (comma is the delimiter)
26. In addition, there are some built-in commands of the UE that can be used, for example, the'stat FPS 'can display the current frame rate, and the't. Maxfps 60' can set the maximum frame rate to 60.

6.2. Data request interface (get all objects B ounding B Properties such as ox ）

6.2.1. CoptReqData (aircraft data returned by RflySim3D, information of a Copter sent)

I. _ _ init _ _.

```
 def __init__(self):
Self. Checksum = 0 # 1234567891 as check
Self. CopterID = 0 # Aircraft ID
       self.PosUE = [0,0,0]
       self.angEuler = [0,0,0]
       self.boxOrigin = [0,0,0]
       self.BoxExtent = [0,0,0]
       self.timestmp = 0
       self.hasUpdate=True
```

For the aircraft data returned by RflySim3D, after the PX4MavCtrler.reqCamCoptObj () function calls the "RflyReqObjData" command of RflySim3D, RflySim3d, according to the parameters of the command, The information of a Copter sent (the ID of the Copter is attached when sending the command to RflySim3D).

This is the constructor for this class, where:

1. Check sum: is the check bit of the data, which is used to check whether an exception occurs during data transmission.
2. CopterID: indicates the copter ID corresponding to the message.
3. PosUE: indicates the location of the Copter (m, NE)
4. AngEuler: represents the angle of this Copter (radians, Roll, Pitch, Yaw)
5. BoxOrigin: Geometric center of bounding box (m, NE)
6. Box Extent: Half of the side length of the bounding box (meters, X axis (forward), Y axis (right), Z axis (up))
7. Times TMP: timestamp
8. HasUpdate: If the data can be obtained normally, set it to True to provide external calls. If it is True, it means that the data can be read, and then set it to False.

6.2.2. ObjReqData (data returned by RflySim3D, information about an object sent)

二. __init__

```
 def __init__(self):
Self. Checksum = 0 # 1234567891 as check
       self.seqID = 0
       self.PosUE = [0,0,0]
```

```
        self.angEuler = [0,0,0]
        self.boxOrigin = [0,0,0]
        self.BoxExtent = [0,0,0]
        self.timestmp = 0
        self.ObjName=''
        self.hasUpdate=True
```

This class encapsulates the data returned by RflySim3D. After the PX4MavCtrler.reqCamCoptObj () function calls the "RflyReqObjData" command of RflySim3D, The information of an object sent (the Name of the object is attached when sending the command to RflySim3D).

This is the constructor for this class, where:

1. Check sum: is the check bit of the data, which is used to check whether an exception occurs during data transmission.
2. SeqID: always 0. (Obj is not a Copter, it has no ID to return)
3. PosUE: indicates the position of the object (m, NE)
4. AngEuler: represents the angle of this Obj (radians, Roll, Pitch, Yaw)
5. BoxOrigin: Geometric center of bounding box (m, NE)
6. Box Extent: Half of the side length of the bounding box (meters, X axis (forward), Y axis (right), Z axis (up))
7. Timestmp: Time the current scene has been in existence (seconds, the time the scene has been in existence (because it is not a Copter, there is no timestamp))
8. ObjName: the name or ID of the object;
9. HasUpdate: If the data can be obtained normally, set it to True to provide external calls. If it is True, it means that the data can be read, and then set it to False.

6.2.3. CameraData (data returned by RflySim3D, information of a certain camera (vision sensor) sent)

One __init__

```
    def __init__(self):
    Self. Checksum = 0 # 1234567891 as check
        self.SeqID = 0
        self.TypeID = 0
        self.DataHeight = 0
        self.DataWidth = 0
        self.CameraFOV = 0
        self.PosUE = [0,0,0]
        self.angEuler = [0,0,0]
        self.timestmp = 0
        self.hasUpdate=True
```

This class encapsulates the data returned by RflySim3D. After the PX4MavCtrler.reqCamCoptObj () function calls the "RflyReqObjData" command of RflySim3D, The information of a camera (vision sensor) sent (the SeqID of the camera (vision sensor) is attached when the command is sent to RflySim3D).

This is the constructor for this class, where:

1. Check sum: is the check bit of the data, which is used to check whether an exception

occurs during data transmission.

2. SeqID: ID of the vision sensor
3. TypeID: The type of visual sensor (RPG, depth map, etc.)
4. DataHeight: Data height (pixels)
5. DataWidth: Data width (pixels)
6. CameraFOV: Camera field of view (in degrees)
7. PosUE: indicates the position of the object (m, NE)
8. AngEuler: indicates the angle of the Camera (radian, Roll, Pitch, Yaw)
9. Times TMP: timestamp
10. HasUpdate: If the data can be obtained normally, set it to True to provide external calls. If it is True, it means that the data can be read, and then set it to False.

6.2.4. reqCamCoptObj (In the request RflySim3D scene)

```
def reqCamCoptObj(self,type=1,objName=1,windowID=0)
```

一、Explanation of parameters:

1) Type: 0 for camera, 1 for airplane, 2 for object
2) ObjName: when representing an airplane, objName corresponds to CopterID; when representing an object, objName corresponds to the object name or the ID of the created object
3) WindowID: Indicates which RflySim3D you want to send a message to. The default is window 0. (Do not send this data to all RflySim3D, because the value returned is the same, and there is no need to consume additional performance.)

二、Function interpretation:

Send a data request to RflySim3D to request data for objects in the scene (not to create new objects, but to get data for existing objects). It can be a visual sensor, a Copter, and a common object in the scene. See CoptReqData class, ObjReqData class and CameraData class for details of the obtained data.

6.2.5. UE4MsgRecLoop (loop listening to data from RflySim3D)

```
def UE4MsgRecLoop(self)
```

一、Explanation of parameters:

二、Function interpretation:

It monitors 224.0.0.10: 20006 and its own 20006 port. It is used to process the messages returned by RflySim3D or CopterSim. There are 6 kinds of messages in total:

1) CopterSimCrash with length of 12 bytes:

```
struct CopterSimCrash {
int checksum;
int CopterID;
int TargetID;
}
```

For the collision data returned by RflySim3D, RflySim3D will start the collision detection mode when the P key is pressed in RflySim3D. If a collision occurs, the data will be returned. The

P + number can select the sending mode (0 local sending, 1 LAN sending, 2 LAN sending only in case of collision). The default is local sending.

2) PX4SILIntFloat of length 120 bytes:

```
    struct PX4SILIntFloat{
    int checksum;//1234567897
    int CopterID;
    int inSILInts[8];
    float inSILFLoats[20];
    };
```

3) ReqVeCrashData of length 160 bytes:

```
    struct reqVeCrashData {
    int checksum; //Packet check code 1234567897.
    int copterID; //ID number of the current aircraft
    int vehicleType; //The style of the current aircraft
    Int CrashType;//Collision object type, -2 indicates ground, -1 indicates scene static
object, 0 indicates no collision, and above 1 indicates ID number of the collided aircraft
    double runnedTime; //Timestamp of the current aircraft
    float VelE[3]; //Speed of the current aircraft
    float PosE[3]; //Current aircraft position
    Float CrashPos [3];//Coordinates of the collision point
    Float targetPos [3];//the center coordinate of the touched object
    float AngEuler[3]; //Euler angle of the current aircraft
    float MotorRPMS[8]; //Current aircraft motor speed
    float ray[6]; //front, back, left, right, up and down scan lines of the aircraft
    Char CrashedName [20] = { 0 };//Name of the touched object
    }
```

This class has been introduced before. When data return is enabled, RflySim3D will send reqVeCrashData for all Copters (send when data changes, once per second).

4) CameraData with length of 56 bytes:

```
    struct CameraData { //56
    int checksum = 0;//1234567891
    int SeqID; //camera serial number
    Int TypeID;//camera type
    Int Data Height;//pixel height
    Int Data Width;//pixel width
    Float Camera FOV;//camera field angle
    float PosUE[3]; //Camera center position
    Float angEuler [3];//camera Euler angle
    Double times TMP;//timestamp
    };
```

RflySim3D returns the structure regularly according to the command sent by the reqCamCoptObj function, and the program will store it in the self. CamDataVect after receiving it.

5) CoptReqData with length of 64 bytes:

```
    struct CoptReqData { //64
    int checksum = 0; //123456 78 91 as check
    Int CopterID;//Aircraft ID
    float PosUE[3]; //Object center position (specified during artificial 3D modeling, the
attitude coordinate axis is not necessarily at the geometric center)
    Float angEuler [3];//Euler angle of object
    Float boxOrigin [3];//object geometric center coordinate
    Float Box Extent [3];//half of the length, width and height of the object frame
    Double times TMP;//timestamp
    };
```

RflySim3D returns the structure regularly according to the command sent by the reqCamCoptObj function, and the program will store it in the self. CoptDataVect after receiving it.

6) ObjReqData of length 96 bytes:

```
    struct ObjReqData { //96
    int checksum = 0; //123456 78 91 as check
    int seqID = 0;
    float PosUE[3]; //Object center position (specified during artificial 3D modeling, the
attitude coordinate axis is not necessarily at the geometric center)
    Float angEuler [3];//Euler angle of object
    Float boxOrigin [3];//object geometric center coordinate
    Float Box Extent [3];//half of the length, width and height of the object frame
    Double times TMP;//timestamp
    Char ObjName [32] = { 0 };//Name of object touched
    };
```

RflySim3D returns the structure regularly according to the command sent by the reqCamCoptObj function, and the program will store it in the self. ObjDataVect after receiving it. The received data can be retrieved using the getCamCoptObj function.

6.2.6. Get CamCoptObj (get the attributes of the specified target from RflySim3D)

```
    def getCamCoptObj(self,type=1,objName=1)
```

一、Explanation of parameters:

1) Type: 0 for camera, 1 for airplane, 2 for object

二、Function interpretation:

Obtain the specified data from RflySim3D. The UE4MsgRecLoop function enables the monitoring of RflySim3D messages and stores the monitored three types in three lists. This function searches the data in these lists. Therefore, you need to use the reqCamCoptObj function to request the relevant data from RflySim3D.

6.2.7. reqCamCoptObj (Request for properties of the specified object in RflySim3D)

```
    def reqCamCoptObj(self,type=1,objName=1,windowID=0)
```

一、Explanation of parameters:

1) Type: 0 for camera, 1 for airplane, 2 for object
2) ObjName: type represents camera seqID; when s represents aircraft, objName corresponds to CopterID; when s represents object, objName corresponds to object name
3) WindowID: Indicates which RflySim3D you want to send a message to. The default is window 0. (Do not send this data to all RflySim3D, because the value returned is the same, and there is no need to consume additional performance.)

二、Function interpretation:

Send a data request to RflySim3D to request data for objects in the scene (not to create new objects, but to get data for existing objects). It can be a visual sensor, a Copter, and a common object in the scene. See CoptReqData class, ObjReqData class and CameraData class for details of the obtained data.

6.3. Interface such as collision detection

6.3.1. ReqVeCrashData (the structure sent by RflySim3D, which mainly contains data related to collision)

I. _ _ init _ _.

```
def __init__(self):
self.checksum = 1234567897
self.copterID = 0
self.vehicleType = 0
self.CrashType = 0
self.runnedTime = 0
self.VelE = [0, 0, 0]
self.PosE = [0, 0, 0]
self.CrashPos = [0, 0, 0]
self.targetPos = [0, 0, 0]
self.AngEuler = [0, 0, 0]
self.MotorRPMS = [0, 0, 0, 0, 0, 0, 0, 0]
self.ray = [0, 0, 0, 0, 0, 0]
self.CrashedName = ""
```

This class is the struct sent by RflySim3D. It is the struct that will be sent when RflySim3D turns on the data return mode (using the "RflyReqVehicleData 1" command). It mainly contains data related to collision, which will be sent to multicast IP "224.0.0.10: 20006".

1. CopterID: indicates the data of which copter the struct is.
2. VehicleType: Indicates the style of this Copter
3. CrashType: indicates the collision object type, -2 indicates the ground, -1 indicates the scene static object, 0 indicates no collision, and above 1 indicates the ID number of the collided aircraft
4. RunnedTime: Time stamp of the current aircraft
5. VelE: Current aircraft speed (m/s)
6. PosE: Current aircraft position (NE, in meters)
7. CrashPos: Coordinates of the impact point (NE, in meters)
8. TargetPos: coordinates of the center of the impacted object (northeast, unit: m)
9. AngEuler: Euler angle of the current aircraft (Roll, Pitch, Yaw, radians)
10. MotorRPMS: Current aircraft motor speed
11. Ray: Front, back, left, right, up and down scan lines of the aircraft
12. CrashedName: The name of the touched object

6.3.2. UE4MsgRecLoop (used to process messages returned by RflySim3D or CopterSim)

```
def UE4MsgRecLoop(self)
```

一、Explanation of parameters:

二、Function interpretation:

It monitors 224.0.0.10: 20006 and its own 20006 port. It is used to process the messages returned by RflySim3D or CopterSim. There are 6 kinds of messages in total:

1) CopterSimCrash with length of 12 bytes:

```
struct CopterSimCrash {
int checksum;
int CopterID;
int TargetID;
}
```

For the collision data returned by RflySim3D, RflySim3D will start the collision detection mode when the P key is pressed in RflySim3D. If a collision occurs, the data will be returned. The P + number can select the sending mode (0 local sending, 1 LAN sending, 2 LAN sending only in case of collision). The default is local sending.

2) PX4SILIntFloat of length 120 bytes:

```
struct PX4SILIntFloat{
int checksum;//1234567897
int CopterID;
int inSILInts[8];
float inSILFLoats[20];
};
```

3) ReqVeCrashData of length 160 bytes:

```
struct reqVeCrashData {
int checksum; //Packet check code 1234567897.
int copterID; //ID number of the current aircraft
int vehicleType; //The style of the current aircraft
Int CrashType;//Collision object type, -2 indicates ground, -1 indicates scene static object, 0 indicates no collision, and above 1 indicates ID number of the collided aircraft
double runnedTime; //Timestamp of the current aircraft
float VelE[3]; //Speed of the current aircraft
float PosE[3]; //Current aircraft position
Float CrashPos [3];//Coordinates of the collision point
Float targetPos [3];//the center coordinate of the touched object
float AngEuler[3]; //Euler angle of the current aircraft
float MotorRPMS[8]; //Current aircraft motor speed
float ray[6]; //front, back, left, right, up and down scan lines of the aircraft
Char CrashedName [20] = { 0 };//Name of the touched object
}
```

This class has been introduced before. When data return is enabled, RflySim3D will send reqVeCrashData for all Copters (send when data changes, once per second).

4) CameraData with length of 56 bytes:

```
struct CameraData { //56
int checksum = 0;//1234567891
int SeqID; //camera serial number
Int TypeID;//camera type
```

```
    Int Data Height;//pixel height
    Int Data Width;//pixel width
    Float Camera FOV;//camera field angle
    float PosUE[3]; //Camera center position
    Float angEuler [3];//camera Euler angle
    Double times TMP;//timestamp
    };
```

RflySim3D returns the structure regularly according to the command sent by the reqCamCoptObj function, and the program will store it in the self. CamDataVect after receiving it.

5) CoptReqData with length of 64 bytes:

```
    struct CoptReqData { //64
    int checksum = 0; //123456 78 91 as check
    Int CopterID;//Aircraft ID
    float PosUE[3]; //Object center position (specified during artificial 3D modeling, the attitude coordinate axis is not necessarily at the geometric center)
    Float angEuler [3];//Euler angle of object
    Float boxOrigin [3];//object geometric center coordinate
    Float Box Extent [3];//half of the length, width and height of the object frame
    Double times TMP;//timestamp
    };
```

RflySim3D returns the structure regularly according to the command sent by the reqCamCoptObj function, and the program will store it in the self. CoptDataVect after receiving it.

6) ObjReqData of length 96 bytes:

```
    struct ObjReqData { //96
    int checksum = 0; //123456 78 91 as check
    int seqID = 0;
    float PosUE[3]; //Object center position (specified during artificial 3D modeling, the attitude coordinate axis is not necessarily at the geometric center)
    Float angEuler [3];//Euler angle of object
    Float boxOrigin [3];//object geometric center coordinate
    Float Box Extent [3];//half of the length, width and height of the object frame
    Double times TMP;//timestamp
    Char ObjName [32] = { 0 };//Name of object touched
    };
```

RflySim3D returns the structure regularly according to the command sent by the reqCamCoptObj function, and the program will store it in the self. ObjDataVect after receiving it. The received data can be retrieved using the getCamCoptObj function.

6.4. Terrain acquisition interface

6.4.1. sendUE4Cmd (To RflySim3D Send command)

```
    def sendUE4Cmd(self, cmd, windowID=-1)
```

一、Explanation of parameters:

1) Cmd: Byte string of the command sent
2) WindowsID: affects the sending target port, which can be taken as -1 during normal multicast. If the designated RflySim3D on the designated computer is required to

receive, it can be set according to the target computer IP and the window title number of the RflySim3D program. (This serial number corresponds to the 20 ports from 20010 to 20029 mentioned above. For example, RflySim3D-0 means that it is monitoring the 20010, and RflySim3D-3 means that it is monitoring the 20013.)

二、 Function interpretation:

This function is to send a "command" to RflySim3D, where cmd is a byte string, representing the command sent to RflySim3D, and the structure sent out is as follows:

For example

```
struct Ue4CMD{
    int checksum;
    char data[52];
}
mav = PX4MavCtrl.PX4MavCtrler(20100)
mav.sendUE4Cmd(b'RflyChangeMapbyName Grasslands')
```

It sends a command, "RflyChangeMapbyName Grasslands", "which tells RflySim3D to change the map to something called" Grasslands. ".

For other commands and their functions, please refer to "OneDrive \ RflySimDocs \ Advanced Version Manuscript \ Development Version \ Chapter 3-3D Scene Modeling and Simulation \ Demo _ Resources _ 3 \ E0 \ E0 _ 2 \ RflySim3D Command Interface.docx". Here they are listed and briefly introduced:

1. RflyShowTextTime (String txt, float time) \ \ Make the UE display txt for time seconds
2. RflyShowText (String txt) \ \ Make the UE display txt for 5 seconds
3. RflyChangeMapbyID (int ID) \ \ Switch the RflySim3D scene map according to the map ID
4. RflyChangeMapbyName (String txt) \ \ Switch the RflySim3D scene map based on the map name
5. RflyChangeView KeyCmd (String key, int num) \ \ Consistent with pressing a key + num in RflySim3D
6. RflyCameraPosAngAdd (float X, float y, float Z, float roll, float pitch, float yaw) \ \ Adds an offset value to the camera's position and angle
7. RflyCameraPosAng (float X, float y, float Z, float roll, float pitch, float yaw) \ \ Sets the position and angle of the camera (world coordinates of the UE)
8. RflyCameraFovD egrees (float degrees) \ \ Sets the camera's view volume FOV angle
9. RflyChange3D Model (int CopterID, int veTypes = 0) \ \ Modify a UAV model style
10. RflyChangeVehicle Size (int CopterID, float size = 0) \ \ Modify the zoom size of a UAV
11. RflyMoveVehiclePosAng(int CopterID, int isFitGround, float x, float y, float z, float roll, float pitch, Float yaw) \ \ Set an offset value for the position and angle of the drone, and isFitGround sets whether the drone fits the ground
12. RflySetVehiclePosAng(int CopterID, int isFitGround, float x, float y, float z, float roll, float pitch, Float yaw) \ \ Sets the position and angle of the drone
13. RflyScanTerrainH(float xLeftBottom(m), float yLeftBottom(m), float xRightTop(m), float yRightTop(m), Float scanHeight (m), float scanInterval (m)) \ \ Scan the terrain to generate a PNG height map and txt, which is needed by the CopterSim program to know what terrains the UE has and their elevations
14. RflyCesium OriPos (double lat, double lon, double Alt) \ \ Modifies the origin position of Cesium based on latitude and longitude
15. RflyClearCapture () \ \ Clear captured images

16. RflySetActuatorPWMs(int CopterID, float pwm1, float pwm2, float pwm3, float pwm4, float pwm5, float pwm6, float pwm7, float pwm8); \ \ Pass in 8 values and trigger the interface function of the target UAV's blueprint
17. RflySetActuatorPWMsExt(int CopterID, float pwm9, float pwm10, float pwm11, float pwm12, float pwm13, float pwm14, float pwm15, float pwm16, float pwm17, float pwm18, float pwm19, float pwm20, float pwm21, float pwm22, float pwm23, Float PWM 24); \ \ Pass in 16 values and trigger the blueprint interface function of the target UAV. This function requires the full version to be functional
18. RflyReqVehicleData(FString isEnable); If the 'isEnable' is not 0, RflySim3D starts sending the data of all Copters (reqVeCrashData as described above).
19. RflySetPosScale (float scale); \ \ Scale of global position
20. RflyLoad3DFile (FString FileName); \ \ Load and execute the TXT script file under the path
21. RflyReqObjData (int opFlag, FString objName, FString colorStr); \ \ Request to get data of objects in 3D scene
22. RflySetIDLabel (int CopterID, FString Text, FString colorStr, float size); \ \ Set the top ID display content of a Copter (default display CopterID)
23. RflySetMsgLabel(int CopterID, FString Text, FString colorStr, float size, float time, int flag); \ \ Set the contents of the Message display on the top of a Copter
24. RflyDelVehicles (FString CopterIDList); \ \ Remove some Copters (comma is a separator)
25. RflyDisableVe Move (FString CopterIDList, int disable); \ \ Rejects the message of the Copter that receives the specified ID (comma is the delimiter)
26. In addition, there are some built-in commands of the UE that can be used, for example, the'stat FPS 'can display the current frame rate, and the't. Maxfps 60' can set the maximum frame rate to 60.

7. Vision sensor configuration protocol Config. JSON.

7.1. Config. General introduction to JSON protocol

7.1.1. Config. JSON File Parameter Description

Different sensors have different parameter configuration methods. For specific sensor configuration methods Supported sensor list and configuration method , general configuration parameters are described below

SeqID: The serial number of the sensor in RflySim3D. Multiple sensors are incremented from 0. It is required to ensure that the seqid is incremented from 0 and is unique no matter it is a distributed cluster or a single machine;

TypeID: RflySim3D distinguishes the specific sensor type from the TypeID value, such as 1: RGB image, 2: depth image, etc., for detailed reference Supported sensor list and configuration method ;

Target Copter: This value indicates which control carrier the currently configured sensor should be attached to. The carrier can be an aircraft, a car, or even a tree, a grass, a stone, etc., as long as there is a "copter _ ID" "in RflySim3D.

Target MountType: reference coordinate system selection, 0: fixed on the aircraft (relative to the geometric center) 1: fixed on the aircraft (relative to the bottom center). 2: fixed on the ground; 3: fixed on the aircraft, but the camera attitude does not change with the aircraft (ground coordinate system); 4 is fixed on a certain sensor. Currently, only ranging sensors are supported. (If this value is used, Trarge tCopter should be the value of the SeqID of the fixed sensor. If the currently set sensor is fixed on a sensor with a SeqID of 0, this value should be 0.) It is usually set to 0, and the default setting of the pod is 3. It should be noted that if the value is 4, the parameter configuration of the target sensor should be placed before the current sensor. As the saying goes, "If there is no skin, how can there be hair". Therefore, the target sensor must be created first. If the B sensor needs to be installed on the A sensor, the A sensor must be created first.

Note: 3 and 4 are not supported in older versions

DataHeight: The value is set according to the type of the specific sensor. For example, the image indicates the height of the image, and the mechanical laser radar indicates the number of laser beams in the vertical direction. For details, refer to Supported sensor list and configuration method ;

DataWidth: This value is set according to the type of the specific sensor. For example, the image indicates the width of the image, and the mechanical laser radar indicates the number of laser beam points in the horizontal direction. For details, refer to Supported sensor list and configuration method ;

DataCheckFreq: Set the data frequency, which is affected by the RflySim3D refresh rate. The RflySim3D refresh rate must be greater than or equal to the maximum data frequency of all configured sensors.

SendProtocol: communication protocol configuration, SendProtocol [0] value 0: shared memory (the free version only supports shared memory), 1: UDP direct PNG compression, 2: UDP direct image compression, 3: UDP direct IPG compression. If the lidar data is only 0 or 1 (shared memory and UDP network transmission) SendProtocol [1-41]: IP address: SendProtocol [5] port number

Note: The port number under the same destination address is unique.

CameraFOV: This value is set according to the type of the specific sensor. For example, the image data indicates the field angle of the camera, and the mechanical laser radar indicates the horizontal angle range. Refer to Supported sensor list and configuration method this for details;

SensorPosXYZ: the FRD coordinate system of the sensor in terms of its position in meters relative to the center of the carrier

SensorAngEular: sensor installation angle (unit degree), the Euler angle sequence is roll, pitch, yaw, relative to the carrier FRD coordinate system;

OtherParams: specific reference Supported sensor list and configuration method ;

Addition of the following new protocols:

EulerOrOuat: This is the representation of the installation angle. 0 means to use the Euler angle, that is, SensorAngEular. 1 means to use the quaternion SensorAng Quat.

SensorAngEular: refers to the mounting angle of the sensor, and the unit can be changed.

SensorAng Quat: is the sensor mounting angle, expressed as a quaternion.

7.1.2. jsonLoad (Increase vision sensor by reading configuration file)

```
def jsonLoad(self, ChangeMode=-1, jsonPath="")
```

一、Explanation of parameters:

1) Change Mode: overrides the SendProtocol [0] transfer mode in JSON
2) Json Path: Relative to the configuration file path, the actually read directory is "Interface File Directory"/JSON path. If the JSON path is empty, the Config. JSON under the interface file directory will be read.

二、Function interpretation:

In the method Increase the vision sensor by reading the configuration file , after the sensor parameters are added, a corresponding sensor can be applied to the UE for addition through the sendReqToUE4 method. The effect is the same as 1addVisSensor, but multiple vision sensor parameters can be saved in the configuration file.

7.1.3. addVisSensor (Object to add a vision sensor parameter)

```
def addVisSensor(self, vsr=VisionSensorReq())
```

一、Explanation of parameters:

1) Vs R: Object for vision sensor parameters

二、Function interpretation:

According to the method, an object of a visual sensor parameter is added, and after the sensor parameter is added, a user equipment (UE) can apply for adding a corresponding sensor through a sendReqToUE4 method. If the VSR parameter passed is not of type VisionSensorReq, an exception is thrown.

7.2. Transmitting and receiving method

7.2.1. sendUpdateUEImage (Sent a vision sensor request, R flySim3D This sensor will be created or updated when received)

```
def sendUpdateUEImage(self, vs=VisionSensorReq(), windID=0, IP="127.0.0.1")
```

一、Explanation of parameters:

1) Vs: VisionSensorReq () instance sent
2) WindID: affects the sending target port, and -1 is enough for normal multicast. If the designated RflySim3D on the designated computer is required to receive, it can be set according to the target computer IP and the window title number of the RflySim3D

program. (This serial number corresponds to the 20 ports from 20010 to 20029 mentioned above. For example, RflySim3D-0 means that it is monitoring the 20010, and RflySim3D-3 means that it is monitoring the 20013. You can see that this winID defaults to 0, which means that it defaults to unicast to the first RflySim3D program on the target computer.

3) IP: IP address of the sending destination

二、Function interpretation:

This function A vision sensor request is sent and RflySim3D receives it to create or update the sensor (updated according to SeqID), this function is quite special, as mentioned in the introduction of VisionSensorReq class, it is not UDP multicast but unicast. This is to prevent repeated calculations (because the results obtained by the visual sensors on RflySim3D on the same LAN are obviously the same, there is no need for each computer to calculate once and then send it to the target computer, so that each computer can calculate different visual sensors to improve performance and efficiency), and through this function, the properties of the sensors can be changed in real time. Uch as field of view angle, rotation angle and the like.

7.2.2. sendReqToUE4 (Send a set of VisionSen sorReq)

```
def sendReqToUE4(self, windID=0, IP="")
```

一、Explanation of parameters:

1) WindID: affects the sending target port, and -1 can be taken during normal multicast. If the designated RflySim3D on the designated computer is required to receive, it can be set according to the target computer IP and the window title number of the RflySim3D program. (This serial number corresponds to the 20 ports from 20010 to 20029 mentioned above. For example, RflySim3D-0 means that it is monitoring the 20010, and RflySim3D-3 means that it is monitoring the 20013
2) IP: IP address of the sending destination
3) Although there are no more formal parameters in the function, the function actually refers to some member variables in the class:
4) Self. VisSensor: List of VisionSensorReqs. List of VisionSensorReqs sent by this function
5) Self.isUE4 DirectUDP: bool type, whether to use UDP to send
6) Self.mm0: The address of the shared memory (do not need to change it). When using this function to send, it will also create a shared memory on the local machine and assign it a value. It can only read the return information (the width and height of the image) given by RflySim3D of the local machine, and then output a message indicating whether the request is successful or not.

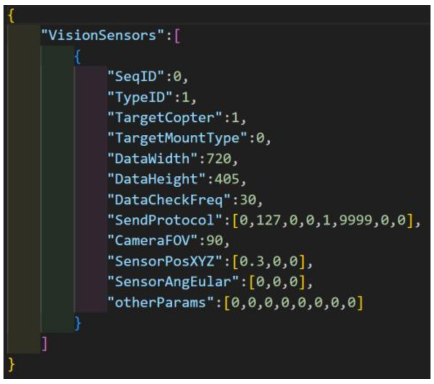
二、Function interpretation:

It is to send a set of VisionSensorReqs, which need to be placed in the self. VisSensor, and assign the value of self.isUE4 DirectUDP as appropriate.

7.3. Supported sensor list and configuration method

Sensor configuration for the RflySim simulation platform is set up through the Sensor Configuration File Config. JSON file. Camera, depth camera, infrared, laser radar and other sensors can be set.

7.3.1. Visible RGB image

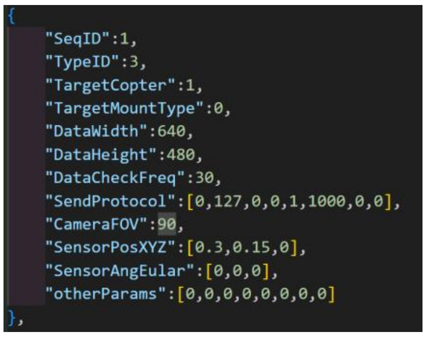


```
{
    "VisionSensors":[
        {
            "SeqID":0,
            "TypeID":1,
            "TargetCopter":1,
            "TargetMountType":0,
            "DataWidth":720,
            "DataHeight":405,
            "DataCheckFreq":30,
            "SendProtocol":[0,127,0,0,1,9999,0,0],
            "CameraFOV":90,
            "SensorPosXYZ":[0.3,0,0],
            "SensorAngEular":[0,0,0],
            "otherParams":[0,0,0,0,0,0,0,0]
        }
    ]
}
```

"SeqID" represents the sensor number. This represents the 1st sensor (the free version only supports 2 graphs).
"TypeID" represents the sensor type ID. 1: RGB map, 2: depth map, 3: grayscale map. Refer to PPT for more configurations.
"The ID of the target aircraft loaded by the Target Copter" sensor can be changed. ".
"TargetMountType" stands for the type of coordinates, 0: fixed on the aircraft (relative to the geometric center), 1: fixed on the aircraft (relative to the bottom center), 2: on the fixed ground (monitoring) also variable, 3: pod camera relative to ground coordinate system, fixed on aircraft, but camera attitude does not change with aircraft (ground coordinate system), 4: Attach a sensor to another sensor, when MountType = 4, TargetCopter = SeqID in the Config. JSON (because MountType = 4 is to attach a sensor to a sensor, So TargetCopter is used to give the vehicle ID, but it's not used at this time, it's used to set the sensor ID, that is, SeqID.
DataWidth is the data or image width, here 640. DataHeight is the data or image height, here 480. The Data CheckFreq "check data update frequency is 30 HZ here.
"SendProtocol [8]" is the transmission mode and address, and the value of SendProtocol [0] is 0: shared memory (the free version only supports shared memory), 1: UDP direct transmission PNG compression, 2: UDP direct transmission image is not compressed, 3: UDPdirect transmission JPG compression; SendProtocol [1-4]: IP address; SendProtocol [5] port number.
CameraFOV is the camera field of view (vision-type sensors only) and can also be changed in degrees.
"EulerOrOuat" is the representation of the installation angle, 0 means to use Euler angle, that is, SensorAngEular, and 1 means to use quaternion SensorAng Quat.
"SensorPosXYZ [3]" is the installation position of the sensor, and the unit can also be changed.
"SensorAngEular [3]" is the mounting angle of the sensor, and the unit can be changed.

"SensorAng Quat [4]" is the sensor mounting angle expressed in quaternions.

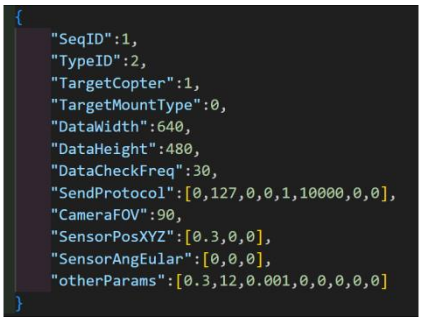
7.3.2. Grayscale image



```
{
    "SeqID":1,
    "TypeID":3,
    "TargetCopter":1,
    "TargetMountType":0,
    "DataWidth":640,
    "DataHeight":480,
    "DataCheckFreq":30,
    "SendProtocol":[0,127,0,0,1,1000,0,0],
    "CameraFOV":90,
    "SensorPosXYZ":[0.3,0.15,0],
    "SensorAngEular":[0,0,0],
    "otherParams":[0,0,0,0,0,0,0,0]
},
```

"SeqID" represents the sensor number. This represents the first sensor.
"TypeID" 1: RGB map, 2: depth map, 3: grayscale map. Refer to PPT for more configurations.
"The ID of the target aircraft loaded by the Target Copter" sensor can be changed. ".
"TargetMountType" stands for the type of coordinates, 0: fixed on the aircraft (relative to the geometric center), 1: fixed on the aircraft (relative to the bottom center), 2: on the fixed ground (monitoring) also variable, 3: pod camera relative to ground coordinate system, fixed on aircraft, but camera attitude does not change with aircraft (ground coordinate system), 4: Attach a sensor to another sensor, when MountType = 4, TargetCopter = SeqID in the Config. JSON (because MountType = 4 is to attach a sensor to a sensor, So TargetCopter is used to give the vehicle ID, but it's not used at this time, it's used to set the sensor ID, that is, SeqID.
DataWidth is the data or image width, here 640. DataHeight is the data or image height, here 480. The Data CheckFreq "check data update frequency is 30 HZ here.
"SendProtocol [8]" is the transmission mode and address, and the value of SendProtocol [0] is 0: shared memory (the free version only supports shared memory), 1: UDP direct transmission PNG compression, 2: UDP direct transmission image is not compressed, 3: UDPdirect transmission JPG compression; SendProtocol [1-4]: IP address; SendProtocol [5] port number.
CameraFOV is the camera field of view (vision-type sensors only) and can also be changed in degrees.
"EulerOrOuat" is the representation of the installation angle, 0 means to use Euler angle, that is, SensorAngEular, and 1 means to use quaternion SensorAng Quat.
"SensorPosXYZ [3]" is the installation position of the sensor, and the unit can also be changed.
"SensorAngEular [3]" is the mounting angle of the sensor, and the unit can also be changed.
"SensorAng Quat [4]" is the sensor mounting angle expressed in quaternions.

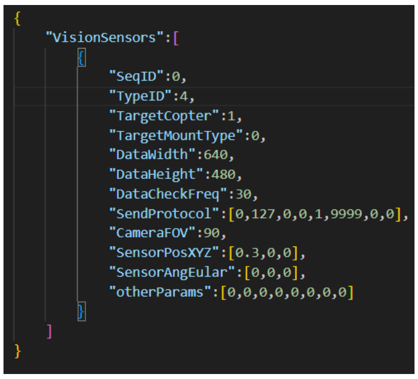
7.3.3. Depth image



```
{
    "SeqID":1,
    "TypeID":2,
    "TargetCopter":1,
    "TargetMountType":0,
    "DataWidth":640,
    "DataHeight":480,
    "DataCheckFreq":30,
    "SendProtocol":[0,127,0,0,1,10000,0,0],
    "CameraFOV":90,
    "SensorPosXYZ":[0.3,0,0],
    "SensorAngEular":[0,0,0],
    "otherParams":[0.3,12,0.001,0,0,0,0,0]
}
```

“ SeqID ” Represents the number of the sensor. This represents the first sensor.
"TypeID" 1: RGB map, 2: depth map, 3: grayscale map, refer to PPT for more configuration
"The ID of the target aircraft loaded by the Target Copter" sensor can be changed. ".
"TargetMountType" represents the coordinate type, 0: fixed on the aircraft (relative to the geometric center), 1: fixed on the aircraft (relative to the bottom center), 2: fixed on ground (monitoring) can also be changed.
DataWidth is the data or image width, here 640. DataHeight is the data or image height, here 480.“ DataCheckFreq ”Check that the data update frequency is 30 HZ here.
"SendProtocol [8]" is the transmission mode and address, and the value of SendProtocol [0] is 0: shared memory (the free version only supports shared memory), 1: UDP direct transmission PNG compression, 2: UDP direct transmission image is not compressed, 3: UDPdirect transmission JPG compression; SendProtocol [1-4]: IP address; SendProtocol [5] port number.
CameraFOV is the camera field of view (vision-type sensors only) and can also be changed in degrees.
"EulerOrOuat" is the representation of the installation angle, 0 means to use Euler angle, that is, SensorAngEular, and 1 means to use quaternion SensorAng Quat.
"SensorPosXYZ [3]" is the installation position of the sensor, and the unit can also be changed.
"SensorAngEular [3]" is the mounting angle of the sensor, and the unit can also be changed.
"SensorAng Quat [4]" is the sensor mounting angle expressed in quaternions.
The output data of depth camera is stored and transmitted in uinit16, and its data range is 0 to 65535. By default, one unit represents 1mm (controlled by otherParams [2]), which means that the maximum range is 0 to 65.535 meters. However, the data range does not represent the actual detection distance of the camera. You also need otherParams [0] to set the minimum detection distance and otherParams [1] to set the maximum detection distance. OtherParams [0]: The minimum recognition distance of the depth camera (unit: m). If the depth distance is less than this value, the 65535 corresponding to NaN will be output. OtherParams [1]: The maximum recognition distance of the depth camera (unit: m). If the depth distance is greater than this value, the 65535 corresponding to NaN will be output. OtherParams [2]: The scale unit (in meters) of the output value of the depth camera uint16. By default, the depth value is in millimeters, so 0.001 is required. Note: If the default value is 0, it will be replaced by otherParams [2] = 0.001. Actual depth value (in meters) = depth picture value (uint16 range) * otherParams [2].

7.3.4. Partition map

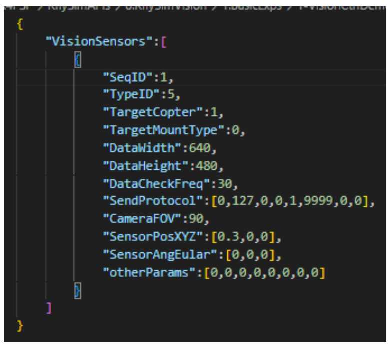


```
{
    "VisionSensors":[
        {
            "SeqID":0,
            "TypeID":4,
            "TargetCopter":1,
            "TargetMountType":0,
            "DataWidth":640,
            "DataHeight":480,
            "DataCheckFreq":30,
            "SendProtocol":[0,127,0,0,1,9999,0,0],
            "CameraFOV":90,
            "SensorPosXYZ":[0.3,0,0],
            "SensorAngEular":[0,0,0],
            "otherParams":[0,0,0,0,0,0,0,0]
        }
    ]
}
```

"SeqID" represents the sensor number. This represents the 1st sensor (the free version only supports 2 graphs).
"TypeID" represents the sensor type ID. 1: RGB map, 2: depth map, 3: grayscale map. Refer to PPT for more configurations.
"The ID of the target aircraft loaded by the Target Copter" sensor can be changed. ".
"TargetMountType" stands for the type of coordinates, 0: fixed on the aircraft (relative to the geometric center), 1: fixed on the aircraft (relative to the bottom center), 2: on the fixed ground (monitoring) also variable, 3: pod camera relative to ground coordinate system, fixed on aircraft, but camera attitude does not change with aircraft (ground coordinate system), 4: Attach a sensor to another sensor, when MountType = 4, TargetCopter = SeqID in the Config. JSON (because MountType = 4 is to attach a sensor to a sensor, So TargetCopter is used to give the vehicle ID, but it's not used at this time, it's used to set the sensor ID, that is, SeqID.
DataWidth is the data or image width, here 640. DataHeight is the data or image height, here 480. The Data CheckFreq "check data update frequency is 30 HZ here.
"SendProtocol [8]" is the transmission mode and address, and the value of SendProtocol [0] is 0: shared memory (the free version only supports shared memory), 1: UDP direct transmission PNG compression, 2: UDP direct transmission image is not compressed, 3: UDPdirect transmission JPG compression; SendProtocol [1-4]: IP address; SendProtocol [5] port number.
CameraFOV is the camera field of view (vision-type sensors only) and can also be changed in degrees.
"EulerOrOuat" is the representation of the installation angle, 0 means to use Euler angle, that is, SensorAngEular, and 1 means to use quaternion SensorAng Quat.
"SensorPosXYZ [3]" is the installation position of the sensor, and the unit can also be changed.
"SensorAngEular [3]" is the mounting angle of the sensor, and the unit can also be changed.
"SensorAng Quat [4]" is the sensor mounting angle expressed in quaternions.

7.3.5. Ranging sensor

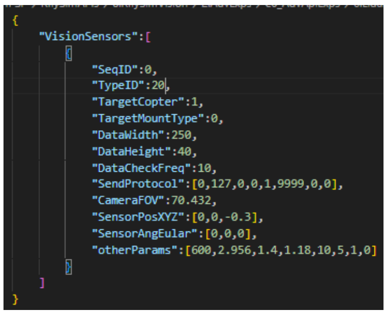
Ranging sensor, emits a ray to the current sensor orientation (if it is attached to the sensor, that is, MountType = = 4 and PosXYZ and AngEular are both zero, the image center target distance is returned). For the ranging sensor, the two parameters DataWidth and DataHeight are invalid, and otherParams [0] represents the maximum distance that the ranging sensor can measure. It should be noted that when TargetMountType is 4, it means that it is installed on other sensors. The Config. JSON file needs to be configured on the installed sensor first, and "SensorPosXYZ" and "SensorAngEular" are both set to 0.



```
{
    "VisionSensors":[
        {
            "SeqID":1,
            "TypeID":5,
            "TargetCopter":1,
            "TargetMountType":0,
            "DataWidth":640,
            "DataHeight":480,
            "DataCheckFreq":30,
            "SendProtocol":[0,127,0,0,1,9999,0,0],
            "CameraFOV":90,
            "SensorPosXYZ":[0.3,0,0],
            "SensorAngEular":[0,0,0],
            "otherParams":[0,0,0,0,0,0,0,0]
        }
    ]
}
```

"SeqID" represents the sensor number. This represents the 1st sensor (the free version only supports 2 graphs).
"TypeID" represents the sensor type ID, and the ranging sensor is 5. Refer to PPT for more configurations.
"The ID of the target aircraft loaded by the Target Copter" sensor can be changed. ". (If it is installed on other sensors, it needs to be changed to sensor ID).
"TargetMountType" stands for the type of coordinates, 0: fixed on the aircraft (relative to the geometric center), 1: fixed on the aircraft (relative to the bottom center), 2: on the fixed ground (monitoring) also variable, 3: pod camera relative to ground coordinate system, fixed on aircraft, but camera attitude does not change with aircraft (ground coordinate system), 4: Attach a sensor to another sensor, when MountType = 4, TargetCopter = SeqID in the Config. JSON (because MountType = 4 is to attach a sensor to a sensor, So TargetCopter is used to give the vehicle ID, but it is not used at this time. It is used to set the ID of the sensor, that is, SeqID, and the sensor to be installed needs to be configured first.
DataWidth and DataHeight are not valid in a ranging sensor.
"SendProtocol [8]" provides the transport address. SendProtocol [1-4]: IP address; SendProtocol [5]: port number.
"EulerOrOuat" is the representation of the installation angle, 0 means to use Euler angle, that is, SensorAngEular, and 1 means to use quaternion SensorAng Quat.
"SensorPosXYZ [3]" is the installation position of the sensor, and the unit can also be changed.
"SensorAngEular [3]" is the mounting angle of the sensor, and the unit can also be changed.
"SensorAng Quat [4]" is the sensor mounting angle expressed in quaternions.

7.3.6. Mechanical scanning laser radar



```
{
    "VisionSensors":[
        {
            "SeqID":0,
            "TypeID":20,
            "TargetCopter":1,
            "TargetMountType":0,
            "DataWidth":250,
            "DataHeight":40,
            "DataCheckFreq":10,
            "SendProtocol":[0,127,0,0,1,9999,0,0],
            "CameraFOV":70.432,
            "SensorPosXYZ":[0,0,-0.3],
            "SensorAngEular":[0,0,0],
            "otherParams":[600,2.956,1.4,1.18,10,5,1,0]
        }
    ]
}
```

"SeqID" represents the sensor number.
Values of "TypeID" are 20, 21, 22; 20: the output point cloud is the lidar coordinate system; 21: the output point cloud is the world coordinate system; 22: livox lidar
"The ID of the target aircraft loaded by the Target Copter" sensor can be changed. ".
"TargetMountType" stands for the type of coordinates, 0: fixed on the aircraft (relative to the geometric center), 1: fixed on the aircraft (relative to the bottom center), 2: on the fixed ground (monitoring) also variable, 3: pod camera relative to ground coordinate system, fixed on aircraft, but camera attitude does not change with aircraft (ground coordinate system), 4: Attach a sensor to another sensor, when MountType = 4, TargetCopter = SeqID in the Config. JSON (because MountType = 4 is to attach a sensor to a sensor, So TargetCopter is used to give the vehicle ID, but it's not used at this time, it's used to set the sensor ID, that is, SeqID.
"DataWidth" is the number of point clouds within a ring of the laser radar, and "DataHeight" is the number of laser radar harnesses.
The Data CheckFreq "Point Cloud Publish Frequency (Hz) is 30 HZ here.
"SendProtocol [8]" is the transmission mode and address. The value of SendProtocol [0] is 0, indicating the shared memory output mode, and the value is 1, indicating the UDP direct transmission mode.
"CameraFOV": No effect on the LIDAR sensor.
"EulerOrOuat" is the representation of the installation angle, 0 means to use Euler angle, that is, SensorAngEular, and 1 means to use quaternion SensorAng Quat.
"SensorPosXYZ [3]" is the installation position of the sensor, and the unit can also be changed.
"SensorAngEular [3]" is the mounting angle of the sensor, and the unit can also be changed.
"SensorAng Quat [4]" is the sensor mounting angle expressed in quaternions.
OtherParams: [maximum laser distance (m), precision (m), lower limit of horizontal scanning angle (degree), upper limit of horizontal scanning angle (degree), lower limit of vertical scanning angle (degrees), upper limit of vertical scanning angle [degrees], reservation, reservation];
The horizontal resolution of the lidar is represented by the DataWidth and the horizontal scanning angle range, and the vertical resolution is represented by the vertical scanning angle (as shown in the figure, horizontal resolution = 90/900, vertical resolution = 40/32). The above angle values are expressed by degree.

7.3.7. Fancy scanning laser radar

下图（图 1.2.1）所示为不同积分时间内（分别为 0.1s，0.2s，0.5s 和 1s）Livox Mid-70 的点云图。

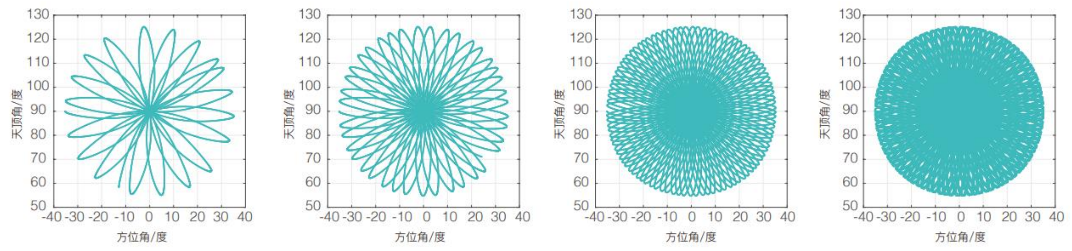
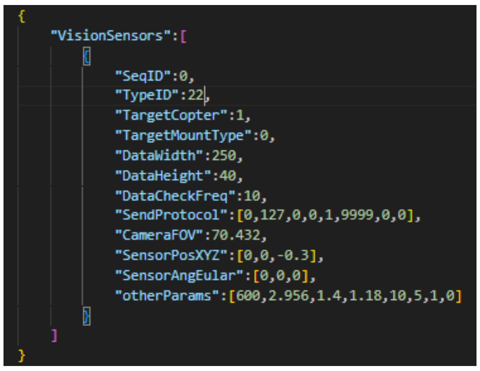


图 1.2.1 Livox Mid-70 不同积分时间内点云效果图

Reference Routine 8.RflySimVision \ 2.AdvExps \ E0 _ AdvApiExps \ 6.LidarLivoxDemo

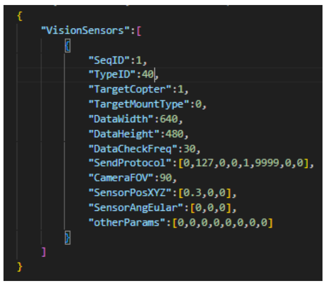


```
{
    "VisionSensors":[
        {
            "SeqID":0,
            "TypeID":22,
            "TargetCopter":1,
            "TargetMountType":0,
            "DataWidth":250,
            "DataHeight":40,
            "DataCheckFreq":10,
            "SendProtocol":[0,127,0,0,1,9999,0,0],
            "CameraFOV":70.432,
            "SensorPosXYZ":[0,0,-0.3],
            "SensorAngEular":[0,0,0],
            "otherParams":[600,2.956,1.4,1.18,10,5,1,0]
        }
    ]
}
```

"SeqID" represents the sensor number. This represents the first sensor.
Values of "TypeID" are 20, 21, 22; 20-: represents that the output point cloud is the lidar coordinate system; 21: represents that the output point cloud is the world coordinate system; 22: represents livox lidar.
"The ID of the target aircraft loaded by the Target Copter" sensor can be changed. ".
"TargetMountType" stands for the type of coordinates, 0: fixed on the aircraft (relative to the geometric center), 1: fixed on the aircraft (relative to the bottom center), 2: on the fixed ground (monitoring) also variable, 3: pod camera relative to ground coordinate system, fixed on aircraft, but camera attitude does not change with aircraft (ground coordinate system), 4: Attach a sensor to another sensor, when MountType = 4, TargetCopter = SeqID in the Config. JSON (because MountType = 4 is to attach a sensor to a sensor, So TargetCopter is used to give the vehicle ID, but it's not used at this time, it's used to set the sensor ID, that is, SeqID.
Number of points for the "Data Width" half petal (corresponding to the red area above)
DataHeight indicates how many petal halves need to be scanned in a frame.
"DataCheckFreq" represents the number of data frames sent per second.
"SendProtocol [8]" is the transmission mode and address. The value of SendProtocol [0] is 0, indicating the shared memory output mode, and the value is 1, indicating the UDP direct transmission mode.
2 times the length of the half petal of "CameraFOV". Unit degree.
"EulerOrOuat" is the representation of the installation angle, 0 means to use Euler angle, that is, SensorAngEular, and 1 means to use quaternion SensorAng Quat.

"SensorPosXYZ [3]" is the installation position of the sensor, and the unit can also be changed.
"SensorAngEular [3]" is the mounting angle of the sensor, and the unit can also be changed.
"SensorAng Quat [4]" is the sensor mounting angle expressed in quaternions.
OtherParams [0]: The maximum scanning distance of the lidar (in meters), for example, 600 meters here. Note
The point cloud data is finally published in int16 format, and this value is coupled with the otherParams [0] parameter.
For example, X _ float is the x-axis floating point value of the point cloud measured by the lidar (in meters, the value is less than
OtherParams [0]), then X _ int = X _ float/otherParams [0] * 32767. So, get.
After the point cloud data, you need to make a reverse mapping according to the farthest distance otherParams [0]: X _ float = X _ int
/ 32767 * otherParams[0].
OtherParams [1]: corresponds to the petal height (unit degree) in the above figure. This reflects the bending of the scanning petals.
Degree.
OtherParams [1]: corresponds to the petal height (unit degree) in the above figure. This reflects the bending of the scanning petals.
Degree.
OtherParams [3]: calibration index of the arc to the left of the red petal, 1.18 by default.
OtherParams [4]: Several figures of eight (a figure of eight is two symmetrical petals with four arcs) form a flower
Flower, as shown in the figure above, is the 0.1s data of DJI lidar, that is, 10 eight characters (20 petals) form a flower.
Flower. Thus otherParams [4] = 10. That is, the angle between two adjacent petals is 180/10 = 18 degrees.

7.3.8. Infrared grayscale image

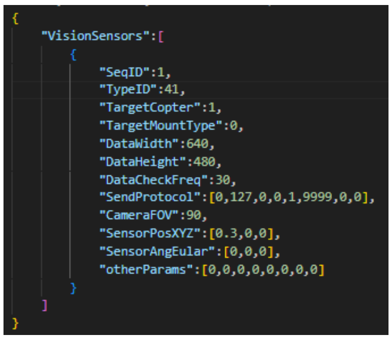


```
{
    "VisionSensors":[
        {
            "SeqID":1,
            "TypeID":40,
            "TargetCopter":1,
            "TargetMountType":0,
            "DataWidth":640,
            "DataHeight":480,
            "DataCheckFreq":30,
            "SendProtocol":[0,127,0,0,1,9999,0,0],
            "CameraFOV":90,
            "SensorPosXYZ":[0.3,0,0],
            "SensorAngEular":[0,0,0],
            "otherParams":[0,0,0,0,0,0,0,0]
        }
    ]
}
```

"SeqID" represents the sensor number. This represents the first sensor.
"TypeID" 40 indicates an infrared grayscale map
"The ID of the target aircraft loaded by the Target Copter" sensor can be changed. ".
"TargetMountType" stands for the type of coordinates, 0: fixed on the aircraft (relative to the geometric center), 1: fixed on the aircraft (relative to the bottom center), 2: on the fixed ground (monitoring) also variable, 3: pod camera relative to ground coordinate system, fixed on aircraft, but camera attitude does not change with aircraft (ground coordinate system), 4: Attach a sensor

to another sensor, when MountType = 4, TargetCopter = SeqID in the Config. JSON (because MountType = 4 is to attach a sensor to a sensor, So TargetCopter is used to give the vehicle ID, but it's not used at this time, it's used to set the sensor ID, that is, SeqID.
DataWidth is the data or image width, here 640. DataHeight is the data or image height, here 480. The Data CheckFreq "check data update frequency is 30 HZ here.
"SendProtocol [8]" is the transmission mode and address, and the value of SendProtocol [0] is 0: shared memory (the free version only supports shared memory), 1: UDP direct transmission PNG compression, 2: UDP direct transmission image is not compressed, 3: UDPdirect transmission JPG compression; SendProtocol [1-4]: IP address; SendProtocol [5] port number.
CameraFOV is the camera field of view (vision-type sensors only) and can also be changed in degrees.
"EulerOrOuat" is the representation of the installation angle, 0 means to use Euler angle, that is, SensorAngEular, and 1 means to use quaternion SensorAng Quat.
"SensorPosXYZ [3]" is the installation position of the sensor, and the unit can also be changed.
"SensorAngEular [3]" is the mounting angle of the sensor, and the unit can also be changed.
"SensorAng Quat [4]" is the sensor mounting angle expressed in quaternions.

7.3.9. Thermodynamic color map



```
{
    "VisionSensors":[
        {
            "SeqID":1,
            "TypeID":41,
            "TargetCopter":1,
            "TargetMountType":0,
            "DataWidth":640,
            "DataHeight":480,
            "DataCheckFreq":30,
            "SendProtocol":[0,127,0,0,1,9999,0,0],
            "CameraFOV":90,
            "SensorPosXYZ":[0.3,0,0],
            "SensorAngEular":[0,0,0],
            "otherParams":[0,0,0,0,0,0,0,0]
        }
    ]
}
```

"SeqID" represents the sensor number. This represents the first sensor.
"TypeID" uses 41 for thermal color map
"The ID of the target aircraft loaded by the Target Copter" sensor can be changed. ".
"TargetMountType" stands for the type of coordinates, 0: fixed on the aircraft (relative to the geometric center), 1: fixed on the aircraft (relative to the bottom center), 2: on the fixed ground (monitoring) also variable, 3: pod camera relative to ground coordinate system, fixed on aircraft, but camera attitude does not change with aircraft (ground coordinate system), 4: Attach a sensor to another sensor, when MountType = 4, TargetCopter = SeqID in the Config. JSON (because MountType = 4 is to attach a sensor to a sensor, So TargetCopter is used to give the vehicle ID, but it's not used at this time, it's used to set the sensor ID, that is, SeqID.
DataWidth is the data or image width, here 640. DataHeight is the data or image height, here 480. The Data CheckFreq "check data update frequency is 30 HZ here.
"SendProtocol [8]" is the transmission mode and address, and the value of SendProtocol [0] is 0: shared memory (the free version only supports shared memory), 1: UDP direct transmission PNG compression, 2: UDP direct transmission image is not compressed, 3: UDPdirect transmission JPG compression; SendProtocol [1-4]: IP address; SendProtocol [5] port number.
CameraFOV is the camera field of view (vision-type sensors only) and can also be changed in degrees.
"EulerOrOuat" is the representation of the installation angle, 0 means to use Euler angle, that is, SensorAngEular, and 1 means to use quaternion SensorAng Quat.
"SensorPosXYZ [3]" is the installation position of the sensor, and the unit can also be changed.

"SensorAngEular [3]" is the mounting angle of the sensor, and the unit can also be changed. "SensorAng Quat [4]" is the sensor mounting angle expressed in quaternions.

8. The Python visual interface is VisionCaptureApi. Py.

8.1. Related classes and data structures

The VisionCaptureApi. Py is the interface file of the platform, including JSON loading, image request, image forwarding, etc. It contains a number of classes related to vision.

- Queue: This class uses a python list to abstract a first-in, first-out queue.

```
def enqueue(self, item):
```

一、Explanation of parameters:

1) Item: pending enqueue item

二、Function interpretation:

This method inserts the item into the head of the queue.

```
def dequeue(self):
```

一、Explanation of parameters:

二、Function interpretation:

This method removes the element at the end of the queue and returns, throwing an IndexError exception if the queue is empty.

```
def is_empty(self):
```

一、Explanation of parameters:

二、Function interpretation:

This method is used to determine whether the queue is empty. It is usually called before the dequeue method to avoid dequeuing exceptions when the queue is empty.

```
def size(self):
```

一、Explanation of parameters:

二、Function interpretation:

This method reads and returns the queue length. Can be used to control the maximum and minimum length of the queue.

- RflyTimeStmp: This class encapsulates the message sent to the port 20005 of the specified remote computer, mainly stores the timestamp of the terminal computer and the simulation, and detects whether the connection between the terminal computer and the simulation program is normal through heartbeat.

```
 def __init__(self):
self.checksum = 1234567897
self.copterID = 0
self.SysStartTime = 0
self.SysCurrentTime = 0
self.HeartCount = 0
```

1. Check sum: is the check bit of the data, which is used to check whether an exception occurs during data transmission.
2. CopterID: indicates the copter ID corresponding to the message.

3. SysStartTime: is the timestamp to start the simulation under Windows, in milliseconds, using the Greenwich standard starting point.
4. SysCurrentTime: The current timestamp under Windows, in milliseconds, using the Greenwich Mean starting point.
5. Heart Count: The counter of the heartbeat packet.

➢ Class VisionSensorReq: This class is a struct sent to the UE. RflySim3D will create a vision sensor after receiving it and start to output image data. This struct is usually not sent by multicast. Moreover, a RflySim3D window will be designated to receive the structure (that is to say, only one RflySim3D on one computer receives the structure). The assignment of the structure has different configuration methods according to different sensors. Please refer to Supported sensor list and configuration method for details.

```
def __init__(self):
      self.checksum = 12345
      self.SeqID = 0
      self.TypeID = 1
      self.TargetCopter = 1
      self.TargetMountType = 0
      self.DataWidth = 0
      self.DataHeight = 0
      self.DataCheckFreq = 0
      self.SendProtocol = [0, 0, 0, 0, 0, 0, 0, 0]
      self.CameraFOV = 90
      self.SensorPosXYZ = [0, 0, 0]
      self.SensorAngEular = [0, 0, 0]
      self.otherParams = [0, 0, 0, 0, 0, 0, 0, 0]
```

This is the initialization function where the individual parameters determine the state of the sensor.

1. SeqID: is the ID of the sensor, which must be unique in the LAN. Each computer can have at most 32 visual sensors (if there are n computers, then 32 * n sensors can be realized). The sensor IDs on the same computer must belong to the same group (for example, the ID of the sensor on computer A is 0 ~ 31, and the ID on computer B is 32 ~ 63).
2. TypeID: indicates the type of sensor (determines what kind of data RflySim3D returns), which can be [1,6], 1: RGB map (the free version only supports RGB map), 2: depth map, 3: grayscale map, 4: point cloud relative to the body coordinate system, 5: point cloud relative to the geodetic coordinate system, 6: Petal scanning point cloud relative to the body coordinate system.
3. Target Copter: ID of the Copter on which the sensor is mounted. //variable
4. Target MountType: coordinate type, 0: fixed on aircraft (relative to geometric center), 1: fixed on aircraft (relative to bottom center), 2: fixed on ground (monitoring)//variable
5. DataWidth: Data or image width
6. DataHeight: Data or image height
7. Data CheckFreq: Check the data update frequency (unit: Hz)
8. SendProtocol [8]: transmission mode and address, where SendProtocol [0] takes the value of 0: shared memory (the free version only supports shared memory), 1: UDP direct PNG compression, 2: UDP direct image compression, 3: UDP JPG compression. SendProtocol

[1-4]: IP address, SendProtocol [5] port number. The IP address here refers to the destination address of the image data sent by RflySim3D.

9. CameraFOV: Camera field of view (vision sensor only), unit degree//changeable
10. SensorPosXYZ [3]: mounting position of sensor, unit: m//changeable
11. SensorAngEular [3]: sensor installation angle, unit ° ° //changeable
12. OtherParams [8]: parameter configuration reserved for different sensors, refer to Supported sensor list and configuration method for details.

➢ Class imuDataCopter: This class encapsulates the IMU packets returned by CopterSim and provides translation to ROS messages.

```
 def __init__(self, imu_name="/rflysim/imu", node=None):
global isEnableRosTrans
global is_use_ros1
self.checksum = 1234567898
self.seq = 0
self.timestmp = 0
self.acc = [0, 0, 0]
self.rate = [0, 0, 0]
if isEnableRosTrans:
self.time_record = -1
Self. IsUseTime Align = True # Whether to publish the data in the same way as the image
time.
if len(imu_name) == 0:
imu_name = "/rflysim/imu"
if is_use_ros1:
self.imu_pub = rospy.Publisher(imu_name, sensor.Imu, queue_size=1)
self.rostime = rospy.Time.now()
else:
self.rostime = node.get_clock().now()
self.imu_pub = node.create_publisher(sensor.Imu, imu_name, 1)
self.time_queue = Queue()
self.newest_time_img = -1
self.test_imu_time = 0
self.test_sum = 0
self.count = 0
self.ros_imu = sensor.Imu()
self.imu_frame_id = "imu"
self.ros_imu.header.frame_id = self.imu_frame_id
```

1. Check sum: data check bit.
2. Seq: indicates the sequence number of the message.
3. Times TMP: Message timestamp.
4. ACC: indicates IMU acceleration
5. Rate: represents the angular velocity of the IMU

➢ Class SensorReqCopterSim: This class encapsulates the data packets sent to the CopterSim request sensor to request various sensor data.

```
def __init__(self):
self.checksum = 12345
self.sensorType = 0
self.updateFreq = 100
self.port = 9998
self.IP = [127, 0, 0, 1]
self.Params = [0, 0, 0, 0, 0, 0]
```

1. Check sum: check digit.
2. SensorType: sensor types 0 ~ 6 respectively represent Imu, RGB map, depth map, grayscale map, laser point cloud in radar coordinate system, laser point cloud in world coordinate system, and livox point cloud.
3. Update Freq: Frequency of the requested sensor data.
4. Port: The port number to receive the data.
5. IP: The IP address to receive the data.
6. Params: Reserved parameters.

➢ Class reqVeCrashData: This class is the struct sent by RflySim3D. It is the struct that will be sent when RflySim3D enables the data return mode (using the "RflyReqVehicleData 1" command). It mainly contains data related to collision, which will be sent to multicast IP "224.0.0.10: 20006".

```
def __init__(self):
self.checksum = 1234567897
self.copterID = 0
self.vehicleType = 0
self.CrashType = 0
self.runnedTime = 0
self.VelE = [0, 0, 0]
self.PosE = [0, 0, 0]
self.CrashPos = [0, 0, 0]
self.targetPos = [0, 0, 0]
self.AngEuler = [0, 0, 0]
self.MotorRPMS = [0, 0, 0, 0, 0, 0, 0, 0]
self.ray = [0, 0, 0, 0, 0, 0]
self.CrashedName = ""
```

1. CopterID: indicates the data of which copter the struct is
2. VehicleType: Indicates the style of this Copter
3. CrashType: indicates the collision object type, -2 indicates the ground, -1 indicates the scene static object, 0 indicates no collision, and above 1 indicates the ID number of the collided aircraft
4. RunnedTime: Time stamp of the current aircraft
5. VelE: Current aircraft speed (m/s)
6. PosE: Current aircraft position (NE, in meters)
7. CrashPos: Coordinates of the impact point (NE, in meters)
8. TargetPos: coordinates of the center of the impacted object (northeast, unit: m)
9. AngEuler: Euler angle of the current aircraft (Roll, Pitch, Yaw, radians)
10. MotorRPMS: Current aircraft motor speed
11. Ray: Front, back, left, right, up and down scan lines of the aircraft
12. CrashedName: The name of the touched object

➢ Class PX4 SILIntFloat: This class encapsulates the SILInts and SILFloats data output to the CopterSim DLL model.

```
 def __init__(self):
self.checksum = 0
```

```
    self.CopterID = 0
    self.inSILInts = [0, 0, 0, 0, 0, 0, 0, 0]
    self.inSILFLoats = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

    def __init__(self, iv):
    self.checksum = iv[0]
    self.CopterID = iv[1]
    self.inSILInts = iv[2:10]
    self.inSILFLoats = iv[10:30]
```

1. Check sum: check digit
2. CopterID: Copter ID corresponding to the message
3. InSILInts: SILInts data output to the CopterSim DLL model
4. InSILFLoats: SILFoats data output to the CopterSim DLL model

➢ Class VisionCaptureApi: This class is the main body of the whole file, and all the functions are written in it. Here is the content related to UE.
For UE-related functions, please refer to "OneDrive \ RflySimDocs \ Advanced Version Manuscript \ Development Version \ Chapter 3-3D Scene Modeling and Simulation \ Demo _ Resources _ 3 \ E0 \ E0 _ 4 \ Python Interface Related to RflySim3D.docx". Functions, but also prepared a number of routines.
Begin with a brief introduction to the network of RflySim3D:

1. It mainly receives UDP data, and its main listening address is 224.0.0.10: the multicast address of the 20009, and the unicast address of its own 20009 port.
2. It can also accept multicast address 224.0.0.11: 20008 and its own unicast port 20008, but this multicast only accepts data from the local loopback address.
3. It can also receive 20010 ~ 20029 one of these 20 ports, because if multiple RflySim3D programs are opened on the same computer, they will compete for unicast ports, so according to the order of creation, they will allocate these unicast ports in turn.
4. These ports handle data in exactly the same way.

8.2. Transmitting and receiving image related interface

8.2.1. sendReqToUE4 (Send one group VisionSensorReq ）

```
    def sendReqToUE4(self, windID=0, IP="")
```

一、Explanation of parameters:

1) WindID: affects the sending target port, and -1 can be taken during normal multicast. If the designated RflySim3D on the designated computer is required to receive, it can be set according to the target computer IP and the window title number of the RflySim3D program. (This serial number corresponds to the 20 ports from 20010 to 20029 mentioned above. For example, RflySim3D-0 means that it is monitoring the 20010, and RflySim3D-3 means that it is monitoring the 20013
2) IP: IP address of the sending destination

3) Although there are no more formal parameters in the function, the function actually refers to some member variables in the class:
4) Self. VisSensor: List of VisionSensorReqs. List of VisionSensorReqs sent by this function
5) Self.isUE4 DirectUDP: bool type, whether to use UDP to send
6) Self.mm0: The address of the shared memory (do not need to change it). When using this function to send, it will also create a shared memory on the local machine and assign it a value. It can only read the return information (the width and height of the image) given by RflySim3D of the local machine, and then output a message indicating whether the request is successful or not.

二、Function interpretation:

That is Send a set of VisionSen sorReq , these VisionSensorReqs need to be placed in the self. VisSensor, with the value self.isUE4DirectUDP assigned as appropriate.

8.2.2. jsonLoad (Increase vision sensor by reading configuration file)

```
def jsonLoad(self, ChangeMode=-1, jsonPath="")
```

一、Explanation of parameters:

1) Change Mode: overrides the SendProtocol [0] transfer mode in JSON
2) Json Path: Relative to the configuration file path, the actually read directory is "Interface File Directory"/JSON path. If the JSON path is empty, the Config. JSON under the interface file directory will be read.

二、Function interpretation:

According to the method, a visual sensor is added by reading a configuration file, and after a sensor parameter is added, a user equipment (UE) can be applied for adding a corresponding sensor through a sendReqToUE4 method. The effect is the same as 1addVisSensor, but multiple vision sensor parameters can be saved in the configuration file.

8.2.3. sendReqToCopterSim (Send U DP Data to C opterSim Request sensor data. E.g. IMU)

```
def sendReqToCopterSim(self, srcs=SensorReqCopterSim(), copterID=1)
```

一、Explanation of parameters:

1) Srcs: SensorReqCopterSim instance sent, refer to 3.3.5
2) CopterID: The requested CopterSim index

二、Function interpretation:

This method requests sensor data from CopterSim by sending UDP data. Uch as requesting Imu, RGB image data, and the like.

8.2.4. img_mem_thrd (Image of receiving RflySim3D return by reading shared memory)

```
def img_mem_thrd(self, idxList)
```

一、Explanation of parameters:

1) IdxList: Sensor serial number list

二、Function interpretation:

This method The image data returned by CopterSim is received by reading the shared memory and is parsed into a numpy format, also fills in the parsed data in the built-in variables hasData and Img, through which the user program can access the image. At the same time, when the global variable isEnableRosTrans is True, the image content will be published in the form of ROS message.

8.2.5. startImgCap (Turn on the thread that grabs images in shared memory)

```
def startImgCap(self, isRemoteSend=False)
```

一、Explanation of parameters:

1) IsRemoteSend: If true, then Images retrieved from shared memory are forwarded to UDP

二、Function interpretation:

Start to capture images, according to the transmission mode indicated in the self. VisSensor list, start the receiving thread of UDP "self. IMG _ UDP _ thrdNew" "or Start the thread to grab the image in the shared memory " self. IMG _ mem _ thrd "". The receiving thread for this UDP will open multiple, one for each sensor.

8.2.6. sendImgUDPNew (Pass the picture returned by RflySim3D through u dp Send to the specified ip And port number)

```
def sendImgUDPNew(self, idx)
```

一、Explanation of parameters:

1) Idx: serial number of sensor

二、Function interpretation:

This method sends the image returned by CopterSim to the specified IP and port number through UDP. The setting of IP and port number refers to SendProtocol of VisionSensorReq class. Or set it in jsonLoad's configuration file.

8.2.7. img_udp_thrdNew (Receive the image data returned by RflySim3D)

```
def img_udp_thrdNew(self, udpSok, idx, typeID)
```

一、Explanation of parameters:

1) UpdSok: UDP socket
2) Idx: serial number of sensor
3) TypeID: sensor type

二、Function interpretation:

This method receives the image data returned by CopterSim through UDP and parses it into numpy format. Fill in the parsed data in the built-in variables hasData and Img, and the user program can access the image through these two variables. For example

```
for i in range(len(vis.hasData)):
if vis.hasData[i]:
cv2.imshow('Img'+str(i),vis.Img[i])
cv2.waitKey(1)
```

At the same time, when the global variable isEnableRosTrans is True, the image content will be published in the form of ROS message.

8.3. IMU obtains relevant interface

8.3.1. sendImuReqCopterSim (By sending S ensorReqCopterSim Instance to request I mu Data

```
def sendImuReqCopterSim(self,copterID=1,IP="", port=31000, freq=200)
```

一、Explanation of parameters:

1) CopterID: The requested CopterSim index
2) IP: The IP of the data receiving end. The default is the local IP in the null time.
3) Port: The port number on which CopterSim sends data
4) Freq: The frequency at which data is requested

二、Function interpretation:

This method requests Imu data by sending a SensorReqCopterSim instance, and then starts a thread to monitor and process the Imu data returned by CopterSim.

8.3.2. sendImuReqClient (Send S ensorReqCopterSim Instance to request I mu Data

```
    def sendImuReqClient(self, copterID=1, IP="", port=31000, freq=200)
```

一、Explanation of parameters:

1) CopterID: The requested CopterSim index
2) IP: The IP of the data receiving end. The default is the local IP in the null time.
3) Port: The port number on which CopterSim sends data
4) Freq: The frequency at which data is requested

二、Function interpretation:

This method requests Imu data by sending a SensorReqCopterSim instance.

8.3.3. sendImuReqServe (Receive C in thread opterSim Back to pass Imu Data

```
    def sendImuReqClient(self, copterID=1, IP="", port=31000, freq=200)
```

一、Explanation of parameters:

1) CopterID: CopterSim index of Imu data returned
2) Port: The port number on which CopterSim sends data

二、Function interpretation:

This method creates a thread in which the Imu data returned by CopterSim is received without blocking the main thread.

8.3.4. AlignTime (Guarantee I mu Maintain stable frequency during data release (release)

```
    def AlignTime(self, img_time):
```

一、Explanation of parameters:

1) IMG _ time: Timestamp

二、Function interpretation:

This method is used to ensure the stable frequency release in the process of Imu data release, and is not disturbed by the unstable frequency of image release. This is done by aligning the time before the program starts.

8.4. ROS-related interfaces

8.4.1. Imu2ros （C opterSim Back to pass imu Message to Ros Built-in I mu Message format conversion)

```
def Imu2ros(self, node=None)
```

一、Explanation of parameters:

1) Node: ROS node

二、Function interpretation:

This method provides conversion of IMU messages returned by CopterSim to the ROS built-in IMU message format. When isEnableRosTrans is True, this method converts the IMU message received from CopterSim into a ROS message, and publishes it to the topic named IMU _ name through the IMU _ pub, which is convenient for docking with mature modules in ROS.

8.4.2. getIMUDataLoop (Will be Imu Data conversion to ROS Imu And send it to the specified topic)

```
def getIMUDataLoop(self)
```

一、Explanation of parameters:

二、Function interpretation:

This method is used to process the received Imu data. If isEnableRosTrans is True, it will also convert the Imu data to the Imu message in ROS and send it to the specified topic.

8.4.3. img_mem_thrd (When the global variable isEnableRosTrans is True, the image content will be published in the form of ROS message.)

```
def img_mem_thrd(self, idxList)
```

一、Explanation of parameters:

1) IdxList: Sensor serial number list

二、Function interpretation:

This method receives the image data returned by CopterSim by reading the shared memory and parses it into numpy format, and fills the parsed data in the built-in variables hasData and Img. The user program can access the image through these two variables. Method is consistent with 3.3.8.23. At the same time, when the global variable isEnableRosTrans is True, the image content will be published in the form of ROS message.

8.4.4. img_udp_thrdNew (Global variable I sEnableRosTrans T rue Is used, the R os Publish the picture content in the form of a message)

```
def img_udp_thrdNew(self, udpSok, idx, typeID)
```

一、Explanation of parameters:

1) UpdSok: UDP socket
2) Idx: serial number of sensor
3) TypeID: sensor type

二、Function interpretation:

This method receives the image data returned by CopterSim through UDP and parses it into numpy format. Fill in the parsed data in the built-in variables hasData and Img, and the user program can access the image through these two variables. For example

```
for i in range(len(vis.hasData)):
if vis.hasData[i]:
cv2.imshow('Img'+str(i),vis.Img[i])
cv2.waitKey(1)
```

At the same time, when the global variable isEnableRosTrans is True, the image content will be published in the form of ROS message.

8.5. Other intrinsic functions

8.5.1. sendImgBuffer (Encapsulate the data to be sent into one or more messages according to the fixed size, and send the message through u. dp Send to Specified I p And port number)

```
def sendImgBuffer(self, idx, data)
```

一、Explanation of parameters:

1) Idx: serial number of sensor
2) Data: data to be sent

二、Function interpretation:

This method Ncapsulate that data to be sent into one or more messages accord to a fixed size, and sending the messages to a specified IP and port number through UDP .

8.5.2. initUE4MsgRec (Start a t 4(self.UE4MsgRecLoop) Start listening 2 24.0.0.10：20006）

```
def initUE4MsgRec(self)
```

一、Explanation of parameters:

二、Function interpretation:

Initialize the self. UDP _ socket UE4 Start a thread T4 (self.UE4MsgRecLoop) to start listening to 224.0.0.10: 20006. and its own 20006 port.

8.5.3. endUE4MsgRec (Stop thread t 4(self.UE4MsgRecLoop) Monitoring of)

```
def endUE4MsgRec(self)
```

一、Explanation of parameters:

二、Function interpretation:

Stop listening for thread T4 (self.UE4MsgRecLoop).

8.5.4. UE4MsgRecLoop (For processing R fly Sim3D Or C opterSim Message returned)

```
def UE4MsgRecLoop(self)
```

一、Explanation of parameters:

二、Function interpretation:

It monitors 224.0.0.10: 20006 and its own 20006 port. It is used to process messages returned by RflySim3D or CopterSim. There are three types of messages:

1) CopterSimCrash with length of 12 bytes:

```
struct CopterSimCrash {
int checksum;
int CopterID;
int TargetID;
}
```

For the collision data returned by RflySim3D, RflySim3D will start the collision detection mode when the P key is pressed in RflySim3D. If a collision occurs, the data will be returned. The P + number can select the sending mode (0 local sending, 1 LAN sending, 2 LAN sending only in case of collision). The default is local sending.

2) PX4SILIntFloat of length 120 bytes:

```
struct PX4SILIntFloat{
int checksum;//1234567897
int CopterID;
int inSILInts[8];
float inSILFLoats[20];
};
```

3) ReqVeCrashData of length 160 bytes:

```
    struct reqVeCrashData {
    int checksum; //Packet check code 1234567897.
    int copterID; //ID number of the current aircraft
    int vehicleType; //The style of the current aircraft
    Int CrashType;//Collision object type, -2 indicates ground, -1 indicates scene static object, 0 indicates no collision, and above 1 indicates ID number of the collided aircraft
    double runnedTime; //Timestamp of the current aircraft
    float VelE[3]; //Speed of the current aircraft
    float PosE[3]; //Current aircraft position
    Float CrashPos [3];//Coordinates of the collision point
    Float targetPos [3];//the center coordinate of the touched object
    float AngEuler[3]; //Euler angle of the current aircraft
    float MotorRPMS[8]; //Current aircraft motor speed
    float ray[6]; //front, back, left, right, up and down scan lines of the aircraft
    Char CrashedName [20] = { 0 };//Name of the touched object
    }
```

This class has been introduced before. When data return is enabled, RflySim3D will send reqVeCrashData for all Copters (send when data changes, once per second).

8.5.5. getUE4Pos (Obtain Co pter At Rf lySim3D Position in the)

```
    def getUE4Pos(self, CopterID=1)
```

一、Explanation of parameters:

1) CopterID: Indicates which copter's location data is obtained.

二、Function interpretation:

This function gets the position of the Copter in RflySim3D.

8.5.6. getUE4Data (Obtain C opter In R flySim3D Data in)

```
    def getUE4Data(self, CopterID=1)
```

一、Explanation of parameters:

1) CopterID: Indicates which copter's data is obtained.

二、Function interpretation:

This function can obtain the data of Copter in RflySim3D, which is also the reqVeCrashData structure.

8.5.7. TimeStmploop (Receive C opterSim The returned heartbeat and timestamp data ensures that the terminal and C. opterSim The connection of is normal)

```
    def TimeStmploop(self)
```

一、Explanation of parameters:

二、Function interpretation:

The method is used for receiving the heartbeat and timestamp data returned by the CopterSim to ensure that the connection between the terminal and the CopterSim is normal. And the assignment operation of each system for time alignment

8.5.8. StartTimeStmplisten (Receive C in child thread opterSim Heartbeat and timestamp data returned to ensure C opterSim The connection is normal

```
def StartTimeStmplisten(self, cpID=1)
```

一、Explanation of parameters:

1) CpID: CopterSim index

二、Function interpretation:

The method binds a 20005 port of a local machine, receives heartbeat and timestamp data returned by the CopterSim in a child thread, ensures that a terminal is normally connected with the CopterSim, and does not block a main thread. Where cpID denotes the ID of CopterSim.

8.5.9. sendUE4Attatch (Other C opter Attach to other C opter Plus, "append" means to follow the move)

```
def sendUE4Attatch(self, CopterIDs, AttatchIDs, AttatchTypes, windowID=-1)
```

一、Explanation of parameters:

1) CopterIDs: The ID of a Copter attached to another Copter, either a value or a list of up to 25 values
2) AttatchIDs: The ID of a Copter that is attached to another Copter, either a value or a list of up to 25 values
3) AttatchTypes: The value of each group of attachment methods is [0, 3], where 0 indicates normal mode (no attachment), 1 indicates that only the position is attached and the attitude is not modified, 2 indicates that the position and yaw angle are attached, and 3 indicates that both the position and attitude are attached.
4) WindowsID: affects the sending target port, which can be taken as -1 during normal multicast. If the designated RflySim3D on the designated computer is required to receive, it can be set according to the target computer IP and the window title number of the RflySim3D program. (This serial number corresponds to the 20 ports from 20010 to 20029 mentioned above. For example, RflySim3D-0 means that it is monitoring the 20010, and RflySim3D-3 means that it is monitoring the 20013.)
5) The number of CopterIDs, AttatchIDs, and AttatchTypes must be consistent. They mean "attach CopterIDs [I] to AttatchIDs [I] in the manner of AttatchTypes [I], up to 25 groups."

二、Function interpretation:

This function sends a structure like this to RflySim3D, with three parameters corresponding to three members in the structure:

```
    # struct VehicleAttatch25 {
    # int checksum;//1234567892
    # int CopterIDs[25];
    # int AttatchIDs[25];
    # int AttatchTypes [25];//0: normal mode, 1: relative position not relative attitude, 2: 
relative position + yaw (not relative pitch and roll), 3: relative position + full attitude 
(pitch roll yaw)
           # }
```

It can attach up to 25 copters at a time to other copters. "Attaching" means that it will follow the movement. If A is attached to B, then when B moves, A will move with it. The document mentioned at the beginning has already written a routine to test its effect.

8.5.10. sendUE4PosScalePwm20 (Set the status attributes of 20 groups of UAVs and set the motor data)

```
    def sendUE4PosScalePwm20(self,copterID,vehicleType,PosE,AngEuler ,Scale,PWMs,isFitGround=
False,windowID=-1)
```

一、Explanation of parameters:

1) CopterID: ID of the set copter, which must be a list with a length of 20
2) VehicleType: The style of the Copter set (determined in the XML), which must be a list of length 20
3) PosE: indicates the set position of the Copter (m, NE), which must be a list of length 60 (each Copter needs three values XYZ to indicate the position)
4) AngEuler: Represents the Euler angles (radians, roll, pitch, yaw) set for this Copter, which must be a list of length 60 (each Copter requires three values XYZ to represent the pose)
5) Scale: indicates the set scaling factor of the Copter, which is (1, 1, 1) by default. The Copter can be zoomed in or out along each axis. (Dimensionless, representing the X, y, Z axes of the UAV's local coordinates, respectively), which must be a list of length 60 (each Copter requires three values XYZ to represent scaling)
6) PWMs: represent 8-bit actuator data. , which must be a list of length 160 (each Copter requires 8 values of motor data)
7) IsFitGround: Indicates whether the UAV is close to the ground. It must be a list of length 20
8) WindowID: affects the sending target port, which can be taken as -1 during normal multicast. If the designated RflySim3D on the designated computer is required to receive, it can be set according to the target computer IP and the window title number of the RflySim3D program. (This serial number corresponds to the 20 ports from 20010 to 20029 mentioned above. For example, RflySim3D-0 means that it is monitoring the 20010, and RflySim3D-3 means that it is monitoring the 20013

二、Function interpretation:

Send the data of 100 copters. RflySim3D will update the Copter after receiving it. Each set of data in the list constitutes the data of a UAV. There are 20 Group in total. It can set 8 motor data for each Copter.

8.5.11. sendUE4Pos (To all in the LAN R f lySim3D Send a C opter If it does not exist, one will be created)

```
    def sendUE4Pos(self,copterID=1,vehicleType=3,MotorRPMSMean=0,PosE=[0, 0, 0],AngEuler=
[0, 0, 0],windowID=-1)
```

一、Explanation of parameters:

1) CopterID: ID of the set Copter
2) VehicleType: The style of the Copter set (determined in the XML)
3) MotorRPMSMean: Indicates the average value of the 8-bit actuator data (the same value for 8 actuators)
4) PosE: indicates the set position of the Copter (m, NE)
5) AngEuler: Represents the Euler angles (radians, roll, pitch, yaw) set for this Copter
6) WindowID: affects the sending target port, which can be taken as -1 during normal multicast. If the designated RflySim3D on the designated computer is required to receive, it can be set according to the target computer IP and the window title number of the RflySim3D program. (This serial number corresponds to the 20 ports from 20010 to 20029 mentioned above. For example, RflySim3D-0 means that it is monitoring the 20010, and RflySim3D-3 means that it is monitoring the 20013.)

二、Function interpretation:

Send the data of a Copter to all RflySim3D in the LAN. If there is no object with the Copter ID, one will be created. Where vehicleType denotes the style of the Copter, PosE denotes the position of the Copter (m, NE), AngEuler denotes the Euler angles of the Copters (radians, roll, pitch, yaw), MotorRPMSMean represents the average of the 8-bit actuator data (the same value for 8 actuators).

8.5.12. sendUE4PosScale100 (Set 100 groups of UAV status attributes)

```
    def sendUE4PosScale100(self,copterID,vehicleType,PosE,AngEuler,MotorRPMSMean,Scale,
isFitGround=False,windowID=-1)
```

一、Explanation of parameters:

1) CopterID: ID of the set copter, which must be a list with a length of 100
2) VehicleType: The style of the Copter set (determined in the XML), which must be a list of length 100
3) PosE: indicates the set position of the Copter (m, NE), which must be a list of length 300 (each Copter needs three values XYZ to indicate the position)
4) AngEuler: Represents the Euler angles (radians, roll, pitch, yaw) set for this Copter, which must be a list of length 300 (each Copter requires three values XYZ to represent the angle)
5) MotorRPMSMean: Indicates the average value of 8-bit actuator data (the value of 8 actuators is the same). It must be a list of length 100.
6) Scale: indicates the set scaling factor of the Copter, which is (1, 1, 1) by default. The Copter can be zoomed in or out along each axis. (Dimensionless, representing the X, y, Z axes of the UAV's local coordinates, respectively), which must be a list of length 300 (each

Copter requires three values XYZ to represent scaling)

7) IsFitGround: Indicates whether the UAV is close to the ground. It must be a list of length 100
8) WindowID: affects the sending target port, which can be taken as -1 during normal multicast. If the designated RflySim3D on the designated computer is required to receive, it can be set according to the target computer IP and the window title number of the RflySim3D program. (This serial number corresponds to the 20 ports from 20010 to 20029 mentioned above. For example, RflySim3D-0 means that it is monitoring the 20010, and RflySim3D-3 means that it is monitoring the 20013

二、Function interpretation:

Send the data of 100 copters. RflySim3D will update the Copter after receiving it. Each group of data in the list constitutes the data of a UAV. There are 100 groups in total.

8.5.13. sendUE4PosScale (Send a C opter The data for, R flySim3D Update the Co upon receipt pter , with a scaling factor)

```
   def sendUE4PosScale(self,copterID=1,vehicleType=3,MotorRPMSMean=0,PosE=[0, 0, 0],AngEuler=
[0, 0, 0],Scale=[1, 1, 1],windowID=-1)
```

一、Explanation of parameters:

1) CopterID: ID of the set Copter
2) VehicleType: The style of the Copter set (determined in the XML)
3) MotorRPMSMean: Indicates the average value of the 8-bit actuator data (the same value for 8 actuators)
4) PosE: indicates the set position of the Copter (m, NE)
5) AngEuler: Represents the Euler angles (radians, roll, pitch, yaw) set for this Copter
6) Scale: indicates the set scaling factor of the Copter, which is (1, 1, 1) by default. The Copter can be zoomed in or out along each axis. (Dimensionless, representing the local X, y, Z axes of the UAV, respectively)
7) WindowID: affects the sending target port, which can be taken as -1 during normal multicast. If the designated RflySim3D on the designated computer is required to receive, it can be set according to the target computer IP and the window title number of the RflySim3D program. (This serial number corresponds to the 20 ports from 20010 to 20029 mentioned above. For example, RflySim3D-0 means that it is monitoring the 20010, and RflySim3D-3 means that it is monitoring the 20013

二、Function interpretation:

Send the data of a Copter, and RflySim3D will update the Copter after receiving it, but with a parameter of scaling factor.

8.5.14. sendUE4Pos2Ground (Send a C opter The data for, RflySim3D The Co will be updated upon receipt pter , but it can make C opter Close to the ground)

```
    def sendUE4Pos2Ground(self,copterID=1,vehicleType=3,MotorRPMSMean=0,PosE=[0,0, 0],
AngEuler=[0, 0, 0],windowID=-1)
```

一、Explanation of parameters:

1) CopterID: ID of the set Copter
2) VehicleType: The style of the Copter set (determined in the XML)
3) MotorRPMSMean: Indicates the average value of the 8-bit actuator data (the same value for 8 actuators)
4) PosE: indicates the set position of the Copter (m, NE)
5) AngEuler: Represents the Euler angles (radians, roll, pitch, yaw) set for this Copter
6) WindowID: affects the sending target port, which can be taken as -1 during normal multicast. If the designated RflySim3D on the designated computer is required to receive, it can be set according to the target computer IP and the window title number of the RflySim3D program. (This serial number corresponds to the 20 ports from 20010 to 20029 mentioned above. For example, RflySim3D-0 means that it is monitoring the 20010, and RflySim3D-3 means that it is monitoring the 20013.)

二、Function interpretation:

Send the data of a Copter to all RflySim3D in the LAN. If there is no object with the Copter ID, one will be created. Where vehicleType denotes the style of the Copter, PosE denotes the position of the Copter (m, NE), AngEuler denotes the Euler angles of the Copters (radians, roll, pitch, yaw), MotorRPMSMean represents the average of the 8-bit actuator data (the same value for 8 actuators). The main difference with other functions that send Copter data is that the Z coordinate (i.e. PosE [2]) will not work and will keep the Copter close to the ground until there is new Copter data.

8.5.15. sendUE4PosScale2Ground (Send a C opter , update the Co pter , with a zoom factor, and set the aircraft to ground contact)

```
    def sendUE4PosScale2Ground(self,copterID=1,vehicleType=3,MotorRPMSMean=0,PosE=[0,0,0],
AngEuler=[0, 0, 0],Scale=[1, 1, 1],windowID=-1)
```

一、Explanation of parameters:

1) CopterID: ID of the set Copter
2) VehicleType: The style of the Copter set (determined in the XML)
3) MotorRPMSMean: Indicates the average value of the 8-bit actuator data (the same value for 8 actuators)
4) PosE: indicates the set position of the Copter (m, NE)
5) AngEuler: Represents the Euler angles (radians, roll, pitch, yaw) set for this Copter
6) Scale: indicates the set scaling factor of the Copter, which is (1, 1, 1) by default. The

Copter can be zoomed in or out along each axis. (Dimensionless, representing the local X, y, Z axes of the UAV, respectively)

7) WindowID: affects the sending target port, which can be taken as -1 during normal multicast. If the designated RflySim3D on the designated computer is required to receive, it can be set according to the target computer IP and the window title number of the RflySim3D program. (This serial number corresponds to the 20 ports from 20010 to 20029 mentioned above. For example, RflySim3D-0 means that it is monitoring the 20010, and RflySim3D-3 means that it is monitoring the 20013

二、Function interpretation:

Send the data of a Copter, and RflySim3D will update the Copter after receiving it, but with a parameter of scaling factor, and the aircraft is set to be close to the ground.

8.5.16. sendUE4PosNew (Send the data of a Copter, and RflySim3D will update the Copter after receiving it)

```
    def sendUE4PosNew(self,copterID=1,vehicleType=3,PosE=[0,0,0],AngEuler=[0,0,0],
VelE=[0,0,0],PWMs=[0]*8,runnedTime=-1,windowID=-1)
```

一、Explanation of parameters:

1) CopterID: ID of the set Copter
2) VehicleType: The style of the Copter set (determined in the XML)
3) PosE: indicates the set position of the Copter (m, NE)
4) AngEuler: Represents the Euler angles (radians, roll, pitch, yaw) set for this Copter
5) VelE: represents the speed of the UAV (m/s, NE)
6) PWMs: represent 8-bit actuator data
7) RunnedTime: indicates the time stamp of the UAV (no actual effect at present)
8) WindowID: affects the sending target port, which can be taken as -1 during normal multicast. If the designated RflySim3D on the designated computer is required to receive, it can be set according to the target computer IP and the window title number of the RflySim3D program. (This serial number corresponds to the 20 ports from 20010 to 20029 mentioned above. For example, RflySim3D-0 means that it is monitoring the 20010, and RflySim3D-3 means that it is monitoring the 20013

二、Function interpretation:

Send the data of a Copter, and RflySim3D will update the Copter after receiving it, but the parameters are different.

8.5.17. sendUE4PosSimple (Send the data of a Copter, and RflySim3D will update the Copter after receiving it)

```
    def sendUE4PosSimple(self,copterID,vehicleType,PWMs,VelE,PosE,AngEuler,runnedTime=-1,
windowID=-1)
```

一、Explanation of parameters:
1) CopterID: ID of the set Copter
2) VehicleType: The style of the Copter set (determined in the XML)
3) PWMs: represent 8-bit actuator data
4) VelE: represents the speed of the UAV (m/s, NE)
5) PosE: indicates the set position of the Copter (m, NE)
6) AngEuler: Represents the Euler angles (radians, roll, pitch, yaw) set for this Copter
7) RunnedTime: indicates the time stamp of the UAV (no actual effect at present)
8) WindowID: affects the sending target port, which can be taken as -1 during normal multicast. If the designated RflySim3D on the designated computer is required to receive, it can be set according to the target computer IP and the window title number of the RflySim3D program. (This serial number corresponds to the 20 ports from 20010 to 20029 mentioned above. For example, RflySim3D-0 means that it is monitoring the 20010, and RflySim3D-3 means that it is monitoring the 20013

二、Function interpretation:

Send the data of a Copter, and RflySim3D will update the Copter after receiving it, but the parameters are different.

8.5.18. sendUE4PosFull (Send a C opter The data for, R flySim3D The Co will be updated upon receipt pter)

```
    def sendUE4PosFull(self, copterID, vehicleType, MotorRPMS, VelE, PosE, RateB, AngEuler,
windowID=-1)
```

一、Explanation of parameters:
1) CopterID: ID of the set Copter
2) VehicleType: The style of the Copter set (determined in the XML)
3) MotorRPMS: represents 8-bit actuator data
4) VelE: represents the speed of the UAV (m/s, NE)
5) PosE: indicates the set position of the Copter (m, NE)
6) RateB: represents the angular velocity of the drone in radians per second
7) AngEuler: Represents the Euler angles (radians, roll, pitch, yaw) set for this Copter
8) WindowID: affects the sending target port, which can be taken as -1 during normal multicast. If the designated RflySim3D on the designated computer is required to receive, it can be set according to the target computer IP and the window title number of the RflySim3D program. (This serial number corresponds to the 20 ports from 20010 to 20029 mentioned above. For example, RflySim3D-0 means that it is monitoring the 20010, and RflySim3D-3 means that it is monitoring the 20013

二、Function interpretation:

Send the data of a Copter, and RflySim3D will update the Copter after receiving it Just a few more parameters to set.

8.5.19. sendUE4ExtAct (To target C opter Send a set of data to be processed by the user-defined function of the target to achieve various user-defined effects)

```
    def sendUE4ExtAct(self,copterID=1,ActExt=[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 
0],windowID=-1)
```

一、Explanation of parameters:

1) CopterID: ID of the set Copter
2) ActExt: a 16-dimensional blueprint interface parameter whose value is related to the target UAV's blueprint (the target UAV must have implemented the relevant blueprint function).
3) WindowID: affects the sending target port, which can be taken as -1 during normal multicast. If the designated RflySim3D on the designated computer is required to receive, it can be set according to the target computer IP and the window title number of the RflySim3D program. (This serial number corresponds to the 20 ports from 20010 to 20029 mentioned above. For example, RflySim3D-0 means that it is monitoring the 20010, and RflySim3D-3 means that it is monitoring the 20013

二、Function interpretation:

A set of data is sent to the target Copter and processed by the target's user-defined function to achieve various user-defined effects.

9. Python Aircraft Control Interface PX4MavCtrlV4.py

The following shows a piece of code that connects to CopeterSim through the PX4MavCtrlV4.py interface and then controls the aircraft to take off.

```
import PX4MavCtrlV4 as PX4MavCtrl

mav = PX4MavCtrl.PX4MavCtrler(20100, '255.255.255.255')
time.sleep(2)
mav.InitMavLopp()
time.sleep(2)
mav. initOffboard()
time.sleep(2)
mav.SendMavArm(True)
mav.SendPosNED(0, 0, -1, 0)
```

9.1. Initialize

9.1.1. InitMavLoop (Enable MAVLink to monitor CopterSim data and update it in real time)

```
def InitMavLoop(self,UDPMode=2)
```

一、Explanation of parameters:

1) UDPMode: UDP Mode

二、Function interpretation:

Enable MAVLink to monitor CopterSim data and update it in real time. Where the UDP Model value is:

0: Corresponding to UDP _ Full mode, Python transmits complete UDP data to CopterSim, and the amount of transmitted data is small; after receiving the data, CopterSim converts it into Mavlink and transmits it to PX4 flight control; it is suitable for simulation of small and medium-sized clusters (the number is less than 10).

1: Corresponding to UDP _ Simple mode, the packet size and sending frequency are smaller than UDP _ Full mode; suitable for large-scale cluster simulation, the number of UAVs is less than 100.

2. Corresponding to the Mavlink _ Full mode (default mode), Python directly sends MAVLink message to CopterSim, and then forwards it to PX4. The large amount of data is suitable for single machine control; it is suitable for single machine or a small number of aircraft simulation, and the number of UAVs is less than 4;

3. Corresponding to the Mavlink _ Simple mode, part of MAVLink message packets will be shielded, and the data frequency will be reduced. The amount of data sent is much smaller than that in the MAVLink _ Full, which is suitable for multi-aircraft cluster control; it is suitable for small-scale cluster simulation, and the number of UAVs is less than 8.

4: Corresponding to the Mavlink _ NoSend mode, CopterSim will not send MAVLink data to the outside. This mode needs to cooperate with hardware-in-the-loop simulation + data transmission serial communication. MAVLink is transmitted by wired mode. The data volume in the LAN of this mode is the smallest. It is suitable for distributed vision hardware-in-the-loop simulation. The number of UAVs is not limited.

9.1.2. endMavLoop (Stop receiving CopterSim data, consistent with stopRun)

```
def endMavLoop(self)
```

一、Explanation of parameters:

二、Function interpretation:

Stop receiving CopterSim data, consistent with stopRun.

9.1.3. stopRun (Quit MAVLink Data reception mode)

```
def stopRun(self)
```

一、Explanation of parameters:

二、Function interpretation:

Exit the MAVLink data receiving mode. The effect is the same as that of endMavLoop.

9.2. The offboard exits at the end

9.2.1. initOffboard (On offboard Mode, and start the thread to send o in a loop ffboard News

```
def initOffboard(self)
```

一、Explanation of parameters:

二、Function interpretation:

Turn on the offboard mode, start the thread, and send the offboard message in a loop.

9.2.2. initOffboard2 (On offboard Mode, and start the thread to send o in a loop ffboard News

```
def initOffboard2(self)
```

一、Explanation of parameters:

二、Function interpretation:

Turn on the offboard mode, start the thread, and send the offboard message in a loop.

9.2.3. OffboardSendMode (Send offboard data cyclically)

```
def OffboardSendMode(self)
```

一、Explanation of parameters:

二、Function interpretation:

Send offboard data in a loop.

9.2.4. endOffboard (Send MAV _ CMD _ NAV _ GUIDED _ ENABLE command to exit offboard mode and stop sending offboard messages)

```
def endOffboard(self)
```

一、Explanation of parameters:

二、Function interpretation:

Send the MAV _ CMD _ NAV _ GUIDED _ ENABLE command to exit offboard mode and stop sending offboard messages.

9.3. Unlock

9.3.1. SendMavArm (Unlock or lock the UAV)

```
def endOffboard(self)
```

一、Explanation of parameters:

1) IsArm: 0: locked; 1: unlocked

二、Function interpretation:

Send the MAV _ CMD _ COMPONENT _ ARM _ DISARM command to unlock or lock.

9.4. Aircraft Pointing Flight Interface

9.4.1. SendPosNED (Send the target position and heading angle interface in the northeast earth coordinate system)

```
def SendPosNED(self,x=0,y=0,z=0,yaw=0)
```

一、Explanation of parameters:

1) X: X coordinate (m)
2) Y: y coordinate (m)
3) Z: Z coordinate (m), when the aircraft is above the ground, Z is less than 0
4) Yaw: course angle (radian)

二、Function interpretation:

Sending the target position and course angle in the northeast earth coordinate system To PX4.

9.4.2. SendPosNEDNoYaw (Sending the target position in the North-East coordinate system)

```
def SendPosNEDNoYaw(self,x=0,y=0,z=0)
```

一、Explanation of parameters:

1) X: X coordinate (m)
2) Y: y coordinate (m)
3) Z: Z coordinate (m), when the aircraft is above the ground, Z is less than 0

二、Function interpretation:

Send the target position in the NE coordinate system to PX4.

9.4.3. SendPosNEDExt (Select Northeasterly or lower right forebody coordinate system to send target position to PX4)

```
def SendPosNEDExt(self,x=0,y=0,z=0,mode=3,isNED=True)
```

一、Explanation of parameters:

1) X: X coordinate (m)
2) Y: y coordinate (m)
3) Z: z-coordinate (m)
4) Mode: mode flag bit
5) IsNED: when True, the coordinate system is northeast, and when False, it is the lower right front coordinate system of the airframe.

二、Function interpretation:

Send the target position to PX4.

9.4.4. SendPosGlobal (Earth coordinate system (latitude, longitude, altitude) with yaw pointing flight interface)

```
def SendPosGlobal(self,lat=0,lon=0,alt=0,yawValue=0,yawType=0)
```

一、Explanation of parameters:

1) Lat: Longitude
2) Lon: Latitude
3) Alt: Height
4) YawValue: course angle or course angular velocity
5) YawType: flag bit (0: no course angle and course angular velocity; 1: course angle control; 2: course angular velocity control)

二、Function interpretation:

Send the target position and heading in the northeast earth coordinate system to PX4.

9.4.5. sendMavLand (Send Three-dimensional coordinates 、MAV_CMD_NAV_LAND Command the aircraft to land at the target position)

```
def sendMavLand(self,xM,yM,zM)
```

一、Explanation of parameters:

1) XM: X coordinate (m)
2) YM: y coordinate (m)
3) ZM: z-coordinate (m)

二、Function interpretation:

Send the MAV _ CMD _ NAV _ LAND command to land the aircraft at the target position.

9.4.6. sendMavTakeOffLocal (Send 3D coordinates, course angle, pitch angle, take-off speed MAV_CMD_NAV_TAKEOFF_LOCAL Command the aircraft to fly to the target position)

```
def sendMavTakeOffLocal(self,xM=0,yM=0,zM=0,YawRad=0,PitchRad=0,AscendRate=2)
```

一、Explanation of parameters:

1) XM: X coordinate (m)
2) YM: y coordinate (m)
3) ZM: z-coordinate (m)
4) YawRad: Heading angle (radians)
5) PitchRad: Pitch angle (radians)
6) AscendRate： Takeoff speed

二、Function interpretation:

Send the MAV _ CMD _ NAV _ TAKEOFF _ LOCAL command to fly the aircraft to the target position.

9.4.7. sendMavTakeOff (Send 3D coordinates, course angle, pitch angle, MAV_CMD_NAV_TAKEOFF Command the aircraft to take off to the target position)

```
def sendMavTakeOff(self,xM=0,yM=0,zM=0,YawRad=0,PitchRad=0)
```

一、Explanation of parameters:

1) XM: X coordinate (m, NE)
2) YM: y coordinate (m, NE)
3) ZM: Z coordinate (m, NE)

4) YawRad: Heading angle (radians)
5) PitchRad: Pitch Angle (radians)

二、Function interpretation:

Send the MAV _ CMD _ NAV _ TAKEOFF command to take off the aircraft to the target position.

9.4.8. F RD Pointing flight interface in coordinate system

9.4.8.1. SendPosFRD (target position and heading angle in airframe coordinate system (lower right front))

```
def SendPosFRD(self,x=0,y=0,z=0,yaw=0)
```

一、Explanation of parameters:

1) X: X coordinate (m)
2) Y: y coordinate (m)
3) Z: z-coordinate (m)
4) Yaw: course angle (radian)

二、Function interpretation:

Send Target position and heading angle in airframe coordinate system (lower right front) to PX4.

9.4.8.2. SendPosFRDNoYaw (target position in airframe coordinate system (lower right front) to PX4)

```
def SendPosFRDNoYaw(self,x=0,y=0,z=0)
```

一、Explanation of parameters:

1) X: X coordinate (m)
2) Y: y coordinate (m)
3) Z: z-coordinate (m)

二、Function interpretation:

Send Target position in airframe coordinate system (lower right front) to PX4 .

9.4.8.3. SendPosNEDExt (select the North-East or the lower right front coordinate system to send the target position to PX4)

```
def SendPosNEDExt(self,x=0,y=0,z=0,mode=3,isNED=True)
```

一、Explanation of parameters:

1) X: X coordinate (m)
2) Y: y coordinate (m)
3) Z: z-coordinate (m)
4) Mode: mode flag bit

5) IsNED: when True, the coordinate system is northeast, and when False, it is the lower right front coordinate system of the airframe.

二、Function interpretation:

Send the target position to PX4.

9.4.9. F RD Pointing flight interface without yaw angle in coordinate system

9.4.9.1. SendPosFRDNoYaw (target position in airframe coordinate system (lower right front) to PX4)

```
def SendPosFRDNoYaw(self,x=0,y=0,z=0)
```

一、Explanation of parameters:

1) X: X coordinate (m)
2) Y: y coordinate (m)
3) Z: z-coordinate (m)

二、Function interpretation:

Send the target position in the airframe coordinate system (lower right front) to PX4.

9.4.9.2. SendPosNEDExt (Select Northeasterly or lower right forebody coordinate system to send target position to PX4)

```
def SendPosNEDExt(self,x=0,y=0,z=0,mode=3, isNED=True)
```

一、Explanation of parameters:

1) X: X coordinate (m)
2) Y: y coordinate (m)
3) Z: z-coordinate (m)
4) Mode: mode flag bit
5) IsNED: when True, the coordinate system is northeast, and when False, it is the lower right front coordinate system of the airframe.

二、Function interpretation:

Send the target position to PX4.

9.5. Aircraft Speed Control Interface

9.5.1. F RD Speed control interface in coordinate system

9.5.1.1. SendVelFRD (target velocity and course angular velocity in airframe coordinate system (lower right front))

```
def SendVelFRD(self,vx=0,vy=0,vz=0,yawrate=0)
```

一、Explanation of parameters:

1) VX: X direction velocity (m/s, front lower right)
2) Vy: velocity in y direction (m/s, lower right front)
3) VZ: velocity in Z direction (m/s, lower right front), VZ is less than 0 when flying upward
4) Yawrate: course angular velocity (radians/second)

二、Function interpretation:

Send the target speed and course angular velocity in the airframe coordinate system (lower right front).

9.5.1.2. SendVelNoYaw (airframe target velocity)

```
def SendVelNoYaw(self,vx,vy,vz)
```

一、Explanation of parameters:

1) VX: velocity in X direction (m/s)
2) Vy: velocity in y direction (m/s)
3) VZ: velocity in Z direction (m/s), when flying upward, VZ is less than 0

二、Function interpretation:

Send Body coordinate system the target speed under (lower right front) to PX4.

9.5.2. Speed control interface for uncontrolled aircraft yaw

9.5.2.1. SendVelNEDNoYaw (speed control in North East coordinate system)

```
def SendVelNEDNoYaw(self,vx,vy,vz)
```

一、Explanation of parameters:

1) VX: velocity in X direction (m/s, NE)
2) Vy: velocity in y direction (m/s, NE)
3) VZ: velocity in Z direction (m/s, NE), VZ is less than 0 when flying upward

二、Function interpretation:

Send the speed control in the northeast coordinate system.

9.5.2.2. Send Ground Speed (send a command to modify the GND _ SPEED _ TRIM and set the ground speed of the aircraft)

```
def SendGroundSpeed(self,Speed=0)
```

一、Explanation of parameters:

1) Speed: Ground speed (m/s)

二、Function interpretation:

Send commands to modify the GND _ SPEED _ TRIM and set the ground speed of the aircraft.

9.5.2.3. SendCopterSpeed (send a command to modify the MPC _ XY _ VEL _ MAX and set the maximum speed of the aircraft)

```
def SendCopterSpeed(self,Speed=0)
```

一、Explanation of parameters:

1) Speed: Maximum speed (m/s)

二、Function interpretation:

Send commands to modify the MPC _ XY _ VEL _ MAX and set the maximum speed of the aircraft.

9.5.2.4. SendCruiseSpeed (send a command to modify the FW _ AIRSPD _ TRIM parameters and set the cruising speed of the aircraft)

```
def SendCruiseSpeed(self,Speed=0)
```

一、Explanation of parameters:

1) Speed: cruising speed (m/s)

二、Function interpretation:

Send commands to modify the FW _ AIRSPD _ TRIM parameters and set the cruising speed of the aircraft.

9.5.3. With yaw rate and altitude control interface

9.5.3.1. SendVelYawAlt (Target speed, course angle and altitude to PX4 in northeast coordinate system)

```
def SendVelYawAlt(self,vel=10,yaw=6.28,alt=-100)
```

一、Explanation of parameters:

1) Vel: Speed
2) Yaw: course angle
3) Alt: Height

二、Function interpretation:

Send the target speed, course angle and altitude in the northeast coordinate system to PX4.

9.5.3.2. SendPosGlobal (course angular velocity and altitude control in northeast earth coordinate system)

```
def SendPosGlobal(self,lat=0,lon=0,alt=0,yawValue=0,yawType=0)
```

一、Explanation of parameters:

1) Lat: Longitude
2) Lon: Latitude
3) Alt: Height
4) YawValue: course angle or course angular velocity
5) YawType: flag bit (0: no course angle and course angular velocity; 1: course angle control; 2: course angular velocity control)

二、Function interpretation:

Send the target position and heading in the northeast earth coordinate system to PX4.

9.5.3.3. SendMavLandGPS (sends a Longitude, latitude, altitude, MAV _ CMD _ NAV _ LAND command to land the aircraft to the target position)

```
def sendMavLandGPS(self,lat,lon,alt)
```

一、Explanation of parameters:

1) Lat: Longitude
2) Lon: Latitude
3) Alt: Height

二、Function interpretation:

Send the MAV _ CMD _ NAV _ LAND command to land the aircraft at the target position.

9.5.3.4. SendMavTakeOffGPS (send commands of longitude, latitude, altitude, heading angle, pitch angle and MAV _ CMD _ NAV _ TAKEOFF to fly the aircraft to the target position)

```
def sendMavTakeOffGPS(self,lat,lon,alt,yawDeg=0,pitchDeg=15)
```

一、Explanation of parameters:

1) Lat: Longitude
2) Lon: Latitude
3) Alt: Height
4) YawDeg: heading angle (degrees)
5) PitchDeg: Pitch angle (degree)

二、Function interpretation:

Send MAV_CMD_NAV_TAKEOFF Order the aircraft to fly to the target position 。

9.5.3.5. SendTakeoffMode (send the takeoff altitude command to change the flight control to takeoff mode to make the aircraft take off)

```
def sendTakeoffMode(self,alt=0)
```

一、Explanation of parameters:

1) Alt: takeoff altitude

二、Function interpretation:

Send the command to change the flight control to takeoff mode to make the aircraft take off.

9.6. Aircraft attitude control interface

9.6.1. SendAttPX4 （F RD Attitude Control Interface in Coordinate System ）

```
def SendAttPX4(self,att=[0,0,0,0],thrust=0.5,CtrlFlag=0,AltFlg=0)
```

一、Explanation of parameters:

1) Att: Attitude input
2) Thrust: Throttle
3) CtrlFlag: control flag (0: att is 3D Euler angle in degrees, 1: att is 3D Euler angle in radians, 2: att is a quaternion, 3: attis 3D angular velocity in radians/second, 4: attis 3D angular velocity in degrees/second)
4) AltFlg: height flag

二、Function interpretation:

Send the target attitude in the airframe coordinate system (lower right front) to PX4.

9.7. Aircraft Acceleration Control Interface

9.7.1. SendAccPX4 (Send target acceleration to P X4 ）

```
def SendAccPX4(self,afx=0,afy=0,afz=0,yawValue=0,yawType=0,frameType=0)
```

一、Explanation of parameters:

1) Afx: x-direction acceleration
2) Afy: y-direction acceleration
3) Afz: acceleration in Z direction
4) YawValue: course angle or course angular velocity
5) YawType: acceleration control mode (0: no course angle and course angular velocity;

1: course angle control; 2: course angular velocity control)

6) FrameType: coordinate system (0: north east; 1: lower right front of airframe)

二、Function interpretation:

Send target acceleration to PX4 。

9.8. Carrier, carrier vision sensor, target and other attribute interfaces

9.8.1. Request Carrier Aircraft Attribute

9.8.1.1. GetUE4Data (get Copter data in RflySim3D)

```
def SendAccPX4(self,afx=0,afy=0,afz=0,yawValue=0,yawType=0,frameType=0)
```

一、Explanation of parameters:

1) CopterID: Indicates which copter's data is obtained.

二、Function interpretation:

This function can obtain the data of Copter in RflySim3D, which is also the reqVeCrashData structure.

9.8.1.2. GetUE4Pos (get Copter's position in RflySim3D)

```
def getUE4Pos(self, CopterID=1)
```

一、Explanation of parameters:

1) CopterID: Indicates which copter's location data is obtained.

二、Function interpretation:

The function can Get Co pter At Rf lySim3D Location in

9.8.2. Request carrier vision sensor interface

9.8.2.1. CameraData (data returned by RflySim3D, information of a certain camera (vision sensor) sent)

One __init__

```
def __init__(self):
Self. Checksum = 0 # 1234567891 as check
       self.SeqID = 0
       self.TypeID = 0
       self.DataHeight = 0
       self.DataWidth = 0
```

```
        self.CameraFOV = 0
        self.PosUE = [0,0,0]
        self.angEuler = [0,0,0]
        self.timestmp = 0
        self.hasUpdate=True
```

For the data returned by RflySim3D, after the PX4MavCtrler.reqCamCoptObj () function calls the "RflyReqObjData" command of RflySim3D, Rflymim3d, according to the parameters of the command, The information of a camera (vision sensor) sent (the SeqID of the camera (vision sensor) is attached when the command is sent to RflySim3D).

This is the constructor for this class, where:

11. Check sum: is the check bit of the data, which is used to check whether an exception occurs during data transmission.
12. SeqID: ID of the vision sensor
13. TypeID: The type of visual sensor (RPG, depth map, etc.)
14. DataHeight: Data height (pixels)
15. DataWidth: Data width (pixels)
16. CameraFOV: horizontal field of view of camera (unit: degree)
17. PosUE: indicates the position of the object (m, NE)
18. AngEuler: indicates the angle of the Camera (radian, Roll, Pitch, Yaw)
19. Times TMP: timestamp
20. HasUpdate: This value is not sent by RflySim3D

9.8.3. Request intra-scene target data interface

9.8.3.1. Get CamCoptObj (get the specified data from RflySim3D and store the monitored three types in three lists)

```
def getCamCoptObj(self,type=1,objName=1)
```

一、Explanation of parameters:

1) Type: 0 for camera, 1 for airplane, 2 for object

二、Function interpretation:

Obtain the specified data from RflySim3D. The UE4MsgRecLoop function enables the monitoring of RflySim3D messages and stores the monitored three types in three lists. This function searches the data in these lists. Therefore, you need to use the reqCamCoptObj function to request the relevant data from RflySim3D.

9.8.3.2. ReqCamCoptObj (requests data for objects in the scene (does not create new objects, but gets data for existing objects))

```
def reqCamCoptObj(self,type=1,objName=1,windowID=0)
```

一、Explanation of parameters:

1) Type: 0 for camera, 1 for airplane, 2 for object

2) ObjName: type represents camera seqID; when s represents aircraft, objName corresponds to CopterID; when s represents object, objName corresponds to object name
3) WindowID: Indicates which RflySim3D you want to send a message to. The default is window 0. (Do not send this data to all RflySim3D, because the value returned is the same, and there is no need to consume additional performance.)

二、Function interpretation:

Send a data request to RflySim3D to request data for objects in the scene (not to create new objects, but to get data for existing objects). It can be a visual sensor, a Copter, and a common object in the scene. See CoptReqData class, ObjReqData class and CameraData class for details of the obtained data.

9.9. Other intrinsic functions

9.9.1. sendStartMsg (Wake up all aircraft)

```
def sendStartMsg(self,copterID=-1)
```

一、Explanation of parameters:

1) CopterID: Copter ID to be awakened

二、Function interpretation:

Send a message to "224.0.0.10" through UDP, "20007" to wake up and execute waitForStartMSG, when copterID is -1, all aircraft will be woken up, message format:

```
struct startSignal{
int checksum; // set to 1234567890 to verify the data
int isStart; // should start to run
int copterID; // the copter's ID to start
       }
```

9.9.2. waitForStartMsg Program blocks until received sendStartMsg Message of)

```
def waitForStartMsg(self)
```

一、Explanation of parameters:

二、Function interpretation:

The program blocks until it receives a message from sendStartMsg.

9.9.3. initPointMassModel (Create a particle drone model)

```
def initPointMassModel(self,intAlt=0,intState=[0,0,0])
```

一、Explanation of parameters:

1) IntAlt: initial height
2) IntState: Initial X (m), Y (m), Yaw (angle)

二、Function interpretation:

Create a particle UAV model, set the initial ground height, XY position and yaw angle, and create a thread to listen for position and velocity commands. The particle UAV model can provide similar dynamic effects of UAV in hardware and software, but greatly reduce the occupancy of computer performance and improve the stability of aircraft.

9.9.4. EndPointMassModel (End particle model)

```
def EndPointMassModel(self)
```

一、Explanation of parameters:

二、Function interpretation:

Ends the particle model and terminates the thread created in initPointMassModel.

9.9.5. yawSat (Back- PI To P I The heading angle between)

```
def yawSat(self,yaw)
```

一、Explanation of parameters:

1) Yaw: course angle

二、Function interpretation:

Return-Heading angle from PI to PI.

9.9.6. PointMassModelLoop (An endless loop in which the particle model processes messages)

```
def PointMassModelLoop(self)
```

一、Explanation of parameters:

二、Function interpretation:

The particle model processes the endless loop of messages, calculates the motion of the aircraft through the target position or velocity, and sends the position and attitude messages to the UE to achieve the effect of particle model simulation.

9.9.7. InitTrueDataLoop (Start two threads, each receiving C opterSim Real data and PX4 Data

```
def InitTrueDataLoop(self)
```

一、Explanation of parameters:

二、Function interpretation:

Start two threads to receive the real data from CopterSim and the PX4 data.

9.9.8. EndTrueDataLoop (Stop I nitTrueDataLoop Thread listening for)

```
def EndTrueDataLoop(self)
```

一、Explanation of parameters:

二、Function interpretation:

Stop thread listening for InitTrueDataLoop.

9.9.9. initUE4MsgRec (Start a t 4(self.UE4MsgRecLoop) Start listening 2 24.0.0.10：20006）

```
def initUE4MsgRec(self)
```

三、Explanation of parameters:

四、Function interpretation:

Initialize the self. UDP _ socketUE4, and start a thread T4 (self.UE4MsgRecLoop) to listen to the 224.0.0.10 20006 and its own 20006 port.

9.9.10. endUE4MsgRec (Stop thread t 4(self.UE4MsgRecLoop) Monitoring of)

```
def endUE4MsgRec(self)
```

三、Explanation of parameters:

四、Function interpretation:

Stop listening for thread T4 (self.UE4MsgRecLoop).

9.9.11. UE4MsgRecLoop (used to process messages returned by RflySim3D or CopterSim)

```
def UE4MsgRecLoop(self)
```

三、Explanation of parameters:

四、Function interpretation:

It monitors 224.0.0.10: 20006 and its own 20006 port. It is used to process the messages returned by RflySim3D or CopterSim. There are 6 kinds of messages in total:

7) CopterSimCrash with length of 12 bytes:

```
struct CopterSimCrash {
int checksum;
int CopterID;
int TargetID;
}
```

For the collision data returned by RflySim3D, RflySim3D will start the collision detection mode when the P key is pressed in RflySim3D. If a collision occurs, the data will be returned. The P + number can select the sending mode (0 local sending, 1 LAN sending, 2 LAN sending only in case of collision). The default is local sending.

8) PX4SILIntFloat of length 120 bytes:

```
struct PX4SILIntFloat{
int checksum;//1234567897
int CopterID;
int inSILInts[8];
float inSILFLoats[20];
};
```

9) ReqVeCrashData of length 160 bytes:

```
    struct reqVeCrashData {
    int checksum; //Packet check code 1234567897.
    int copterID; //ID number of the current aircraft
    int vehicleType; //The style of the current aircraft
    Int CrashType;//Collision object type, -2 indicates ground, -1 indicates scene static
object, 0 indicates no collision, and above 1 indicates ID number of the collided aircraft
    double runnedTime; //Timestamp of the current aircraft
    float VelE[3]; //Speed of the current aircraft
    float PosE[3]; //Current aircraft position
    Float CrashPos [3];//Coordinates of the collision point
    Float targetPos [3];//the center coordinate of the touched object
    float AngEuler[3]; //Euler angle of the current aircraft
    float MotorRPMS[8]; //Current aircraft motor speed
    float ray[6]; //front, back, left, right, up and down scan lines of the aircraft
    Char CrashedName [20] = { 0 };//Name of the touched object
    }
```

This class has been introduced before. When data return is enabled, RflySim3D will send reqVeCrashData for all Copters (send when data changes, once per second).

10) CameraData with length of 56 bytes:

```
struct CameraData { //56
int checksum = 0;//1234567891
int SeqID; //camera serial number
Int TypeID;//camera type
Int Data Height;//pixel height
Int Data Width;//pixel width
Float Camera FOV;//camera field angle
float PosUE[3]; //Camera center position
Float angEuler [3];//camera Euler angle
Double times TMP;//timestamp
};
```

RflySim3D returns the structure regularly according to the command sent by the reqCamCoptObj function, and the program will store it in the self. CamDataVect after receiving it.

11) CoptReqData with length of 64 bytes:

```
    struct CoptReqData { //64
    int checksum = 0; //123456 78 91 as check
    Int CopterID;//Aircraft ID
    float PosUE[3]; //Object center position (specified during artificial 3D modeling, the
attitude coordinate axis is not necessarily at the geometric center)
    Float angEuler [3];//Euler angle of object
    Float boxOrigin [3];//object geometric center coordinate
    Float Box Extent [3];//half of the length, width and height of the object frame
    Double times TMP;//timestamp
    };
```

RflySim3D returns the structure regularly according to the command sent by the reqCamCoptObj function, and the program will store it in the self. CoptDataVect after receiving it.

12) ObjReqData of length 96 bytes:

```
    struct ObjReqData { //96
    int checksum = 0; //123456 78 91 as check
    int seqID = 0;
    float PosUE[3]; //Object center position (specified during artificial 3D modeling, the
attitude coordinate axis is not necessarily at the geometric center)
    Float angEuler [3];//Euler angle of object
    Float boxOrigin [3];//object geometric center coordinate
    Float Box Extent [3];//half of the length, width and height of the object frame
    Double times TMP;//timestamp
    Char ObjName [32] = { 0 };//Name of object touched
    };
```

RflySim3D returns the structure regularly according to the command sent by the reqCamCoptObj function, and the program will store it in the self. ObjDataVect after receiving it. The received data can be retrieved using the getCamCoptObj function.

9.9.12. sat (InPWM clipping)

```
    def sat(self,inPwm=0,thres=1):
```

一、Explanation of parameters:

1) In Pwm: Enter a value
2) Thres: Upper absolute value

二、Function interpretation:

If inPwm is less than -thres, it returns -thres; if inPwm is greater than thres, it returns thres; the rest of the case returns inpwm.

9.9.13. SendMavCmdLong (Send M AVLINK The C of the message ommand_long Message, which may request the drone to perform some action)

```
    def  SendMavCmdLong(self,  command,  param1=0,  param2=0,  param3=0,  param4=0,  param5=0,
param6=0, param7=0):
```

一、Explanation of parameters:

1) Command: command type
2) Param 1: command argument 1
3) Param2: Command parameter 2
4) Param3: Command parameter 3
5) Param4: Command parameter 4
6) Param5: Command parameter 5
7) Param 6: command argument 6
8) Param 7: command argument 7

二、Function interpretation:

The command _ long message that sends the MAVLINK message can request the UAV to perform certain operations. Detailed introduction refers to:

https://mavlink.io/en/messages/common.html#COMMAND_LONG

https://mavlink.io/en/messages/common.html#MAV_CMD

9.9.14. send MavOffboardCmd (Send o ffboard Command to Flight Control)

```
    def sendMavOffboardCmd(self,type_mask,coordinate_frame, x, y, z, vx, vy, vz, afx, afy, afz,
yaw, yaw_rate):
```

一、Explanation of parameters:

1) Type _ mask: a mask indicating which items should be ignored
2) Coordinate _ frame: coordinate system
3) X: X coordinate (m, NE)
4) Y: y coordinate (m, NE)
5) Z: Z coordinate (m, NE)
6) VX: velocity in X direction (m/s, NE)
7) Vy: velocity in y direction (m/s, NE)
8) VZ: velocity in Z direction (m/s, NE)
9) Afx: x-direction acceleration or force (m/s ^ 2 or N, NE)
10) Afy: y-direction acceleration or force (m/s ^ 2 or N, NE)
11) Afz: acceleration or force in the Z direction (m/s ^ 2 or N, NE)
12) Yaw: course angle (radian)
13) Yaw _ rate: course angular velocity (rad/sec)

二、Function interpretation:

Send the offboard command to the flight control to enter the offboard mode. Detailed reference:

https://mavlink.io/en/messages/common.html#SET_POSITION_TARGET_LOCAL_NED

9.9.15. TypeMask (Get o ffboard The message requires typemask ）

```
    def TypeMask(self,EnList)
```

一、Explanation of parameters:

1) Enlist: a list of valid items, 0 ~ 5 represent position, speed, acceleration, force, course angle and course angular velocity respectively.

二、Function interpretation:

Get the typemask needed for the offboard message to determine what information is valid. Reference:

https://mavlink.io/en/messages/common.html#POSITION_TARGET_TYPEMASK

9.9.16. sendM a vOffboardAPI (Update off board Data of the message)

```
    def sendMavOffboardAPI(self,type_mask=0,coordinate_frame=0,pos=[0,0,0],vel=[0,0,0],acc=
[0,0,0],yaw=0,yawrate=0):
```

一、Explanation of parameters:

1) Type _ mask: a mask indicating which items should be ignored
2) Coordinate _ frame: coordinate system
3) Pos: Location
4) Vel: Speed
5) ACC: Acceleration
6) Yaw: course angle
7) Yawrate: course angular velocity

二、Function interpretation:

Update the data of the offboard message (the data will be sent at a certain frequency).

9.9.17. sendUDPSimpData (Set control mode and control quantity)

```
    def sendUDPSimpData(self,ctrlMode,ctrls):
```

一、Explanation of parameters:

1) Ctrl Mode: Control Mode
2) Ctrls: control quantity

二、Function interpretation:

Sends a UDP message, where IP and port are specified when creating a new instance of this class. The message structure is:

```
    # struct inOffboardShortData{
    # int checksum;
    # int ctrlMode;
    # float controls[4];
        # };
```

9.9.18. sendMavSetParam (Send command to P X4 Modify the specified parameter)

```
def sendMavSetParam(self,param_id, param_value, param_type):
```

一、Explanation of parameters:

1) Param _ ID: parameter ID
2) Param _ value: parameter values
3) Param _ type: parameter type

二、Function interpretation:

Send the command to PX4 to modify the specified parameters. Refer to the parameter introduction:

https://mavlink.io/en/messages/common.html#PARAM_SET

9.9.19. SendSetMode (Send M AV_CMD_DO_SET_MOD Set airplane mode)

```
def SendSetMode(self,mainmode,cusmode=0)
```

一、Explanation of parameters:

1) Main mode: main mode
2) Cusmode: Submode

二、Function interpretation:

Send the MAV _ CMD _ DO _ SET _ MOD to set the airplane mode. Refer to:
https://mavlink.io/en/messages/common.html#MAV_CMD_DO_SET_MODE

9.9.20. getTrueDataMsg (Loop listens to real data and updates built-in state)

```
def getTrueDataMsg(self)
```

一、Explanation of parameters:

二、Function interpretation:

Loop listens to the real data and updates the built-in state.

9.9.21. getPX4DataMsg (Loop monitor PX4 Data, and update the built-in status)

```
def getPX4DataMsg(self)
```

一、Explanation of parameters:

二、Function interpretation:

Loop listens to the PX4 data and updates the built-in status.

9.9.22. getMavMsg (Cyclic update M AVLink Received data)

```
def getMavMsg(self):
```

一、Explanation of parameters:

二、Function interpretation:

The data received by MAVLink is cyclically updated.

9.9.23. sendCustomData (Send a 1 6 Dimension data to the specified port. This interface can be used with the S imulink Communication)

```
def sendCustomData(self,CopterID,data=[0]*16,checksum=123456,port=50000,IP='127.0.0.1'):
```

一、Explanation of parameters:

1) copterID：copter ID
2) Data: data content
3) Check sum: check digit
4) Port: port number
5) IP: IP addr

二、Function interpretation:

Sends a 16-dimensional data to a designated port. This interface can be used to communicate with Simulink. The corresponding data structure is:

```
# struct CustomData{
# int checksum;
# int CopterID;
# double data
# } ii16d 136 package length
```

2.3.12.64 2.3.12.65 2.3.12.67 2.3.12.68 2.3.12.69 2.3.12.70 2.3.12.71 2.3.12.72 2.3.12.79 2.3.12.85

9.10. Aircraft status information

9.10.1. uavAngEular Euler angle returned by flight control (obtained from flight control and can be used for real experiment)

9.10.2. uavAngRate Angular velocity returned by flight control (obtained from flight control and can be used for real experiment)

9.10.3. uavPosNED The estimated local position (NED coordinate system) of the PX4 relative to the takeoff position returned by the flight control (obtained from the flight control and can be used for real experiments)

9.10.4. uavVelNED Estimated local velocity (NED coordinate system) returned by the flight control (derived from the flight control for real experiments)

9.10.5. isVehicleCrash Is it colliding with another vehicle?

9.10.6. IsVehicleCrashIDThe number of the vehicle involved in the crash

9.10.7. uavPosGPS GPS position returned from PX4, using the NED coordinate system, including latitude, longitude, altitude, relative altitude, velocity component, heading angle (derived from flight control for real experiments)

9.10.8. uavPosGPSHome GPS origin (takeoff) position returned from PX4, using the NED coordinate system (derived from flight control for real experiments)

9.10.9. uavGlobalPos The global position returned from PX4 is converted into UE4 map coordinate system (obtained from flight control and can be used for real experiments).

9.10.10. uavAngQuatern Euler angle quaternion returned from PX4 (derived from flight control for real experiments)

9.10.11. uavMotorRPMS Motor PWM signal returned from PX4 (obtained from flight control and can be used for real experiment)

9.10.12. uavAccB = [0, 0, 0] Acceleration returned from PX4 (derived from flight control for real experiments)

9.10.13. UavGyro = [0,0,0] Gyro data returned from PX4 (obtained from flight control and can be used for real experiments)

9.10.14. uavMag = [0, 0, 0] Magnetometer data returned from PX4 (obtained from flight control for real experiments)

9.10.15. uavVibr = [0, 0, 0] Vibration data returned from PX4 (obtained from flight control and can be used for real experiment)

9.10.16. trueAngEular Real Euler angles simulated in CopterSim's DLL model (simulation data)

9.10.17. trueAngRate Real angular rate simulated in CopterSim's DLL model (simulation data)

9.10.18. truePosNED Real position simulated in the DLL model of CopterSim (in relation to the UE4 map center), using the NED coordinate system (simulation data)

9.10.19. trueVelNED Real speed of simulation in DLL model of CopterSim, using NED coordinate system (simulation data)

9.10.20. trueAngQuatern Real Euler angle quaternions simulated in CopterSim's DLL model (simulation data)

9.10.21. trueMotorRPMS Real motor speed simulated in CopterSim's DLL model (simulation data)

9.10.22. trueAccB Real acceleration simulated in CopterSim's DLL model (simulation data)

10. Related interfaces of mavros/ROS required in development

10.1. Introduction of Common Communication Modes in ROS

10.1.1. Message topic communication

ROS is a multi-process robot operating system. Each process is usually called ROS node, and the data communication between nodes can be through the communication mode of message topics. Topic communication is based on a publish-subscribe model that allows nodes to send and receive messages in an asynchronous manner.

Publisher: a ROS node can act as a publisher, responsible for publishing data to one or more Topics. A publisher publishes a message to a specific topic, and other nodes can receive the message by subscribing to the topic.

Subscriber: Another ROS node can act as a subscriber, which can subscribe to one or more topics. Subscribers receive messages sent from publishers by listening to topics. When new messages are posted to a topic, subscribers receive and process them asynchronously.

Topic: a topic is a message communication channel used to publish and subscribe to messages. Topics have unique names by which ROS nodes can identify and connect to them. Publishers post messages to topics, and subscribers receive messages from topics.

Message: a message is a structured data unit used to transmit data in the ROS. ROS uses message definitions to describe the structure and data types of messages. The same type of message is shared between the publisher and the subscriber to ensure data consistency.

ROS Master: The ROS Master is a core component in an ROS system that manages communication between nodes. It maintains a registry of all nodes, topics, and services. Publishers and subscribers use ROS Master to discover and connect to each other.

Publisher-subscriber relationship: a publisher node publishes a message on a specific topic, and a subscriber node registers a subscription relationship by telling the ROS Master which topics it is interested in. When a new message is posted on a topic of interest to a subscriber, ROS Master routes the message to the appropriate subscriber.

Basically, the principle of ROS topic communication is that a publisher publishes a message to a topic, and a subscriber listens to the topic to receive the message. The advantage of this publish-subscribe model is that it allows asynchronous communication between nodes, which improves the flexibility and scalability of the system. Nodes can publish or subscribe to different topics as needed to meet various communication needs.

10.1.2. Service type communication

Service communication in ROS (Robot Operating System) is a communication mechanism for requesting and providing stateful operations or services between nodes. Unlike topic communication, service communication is a request-response pattern in which one node requests another node to perform a specific task and waits for a response.

Service Provider: a service provider is a ROS node that implements one or more services. The service provider is responsible for receiving service requests from other nodes and performing the corresponding tasks. Typically, the service provider registers the services it provides when the node is started.

Service Requester: a service requester is also a ROS node that needs to perform a specific task and send a service request to the service provider. The service requester initiates the request by invoking the client interface of the service.

Service: a service is an ROS communication mechanism that defines request and response messages. The service consists of two parts: Request Message and Response Message. The request message is used to pass the parameters of the service request, and the response message is used to return the result of the service execution.

ROS Master: The ROS Master also plays an important role in service communication and is responsible for service registration and discovery. Service providers register their services with the ROS Master at startup, and service requesters find and connect to the required services through the ROS Master.

The principle of service communication is to establish a communication pattern of request and response, allowing stateful interaction between nodes. Service communication is suitable for situations where tasks need to be performed cooperatively between nodes, such as the request and processing of sensor data, the adjustment of motion control, or other tasks requiring synchronous processing.

10.2. Application of ROS on RflySim platform

10.2.1. Interface for obtaining visual sensor data through topic

The RflySim platform supports the acquisition of RGB images, depth images, grayscale images, laser point clouds, infrared images, segmentation maps, etc Supported sensor list and configuration method . In the corresponding topic name/rflysim/sensro */type, sensor * in the topic name, where * is the seq _ ID in the Config. JSON file, and type is the data type. For example, the RGB image corresponds to the IMG _ RGB, the similar depth corresponds to the IMG _ depth, and the grayscale image corresponds to the IMG _ gray. For details, please refer to Python VisionAPI \ 1-APIUsageDemos \ 12-VisCapture MergeROSAPI

10.2.2. Interface for obtaining IMU data through topic

The default published topic name for RflySim is/rflysim/IMU, which makes it possible to use IMU data published by mavros (/mavros/IMU/data _ raw) if mavros emulation is used. When using the IMU data published by RflySim, there is a corresponding interface reference **错误!未定义书签。错误!未找到引用源。**in the VisionCaptureAPI. Py interface. The corresponding routine is in Python VisionAPI \ 1-APIUsageDemos \ 12-VisCaptureMergeROSAPI

10.3. MavROS common interface

10.3.1. Topic/mavros/state Obtain flight control status information

Through this topic, we can check the flight control status information, such as whether the flight control is successfully connected, the current mode of flight control, whether the flight control has been unlocked, etc. Usually, we send instructions to the value of flight control, and often need to monitor the data of this topic, such as flight mode switching, unlocking command. The service communication has two fields, request and feedback. Generally, if the feedback is not successful, it means that the command is not executed, but the successful feedback does not mean that the command is executed, so we need to judge the state of flight control after sending the command.

10.3.2. Topic/mavros/local _ position/pose Get flight control pose data

Through this topic, the pose of the flight control system in the world coordinate system "map" can be obtained, including position information and attitude information. Therefore, if the attitude of the aircraft needs to be obtained, the IMU data can be used (through the topic/mavros/IMU/data _ raw). Generally, it is more direct to use the IMU data to obtain the pose.

10.3.3. Topic/mavros/IMU/data _ raw Obtain flight control IMU data

For the original IMU data, the coordinate system is related to the setting of flight control parameters, and the FLU coordinate system is generally used. Related to this is the/mavros/IMU/data topic, which is the filtered data.

10.3.4. Topic/mavros/setpoint _ raw/local Send position, speed, acceleration, control command interface

Send the interface of position control, speed control and acceleration control to the flight control. You can select the control mode through the type _ mask variable. View in detail.**错误!未找到引用源。**

10.3.5. Topic mavros/setpoint _ raw/attitude Send Attitude Control Interface

Flight control by means of attitude control Detailed Reference Attitude loop control

10.3.6. Service/mavros/set _ mode flight control mode switching interface

Switch the flight mode of the flight control through this service name. Of course, whether the mode switching is successful or not is limited by the flight control settings. For example, some flight controls shield the switching of some modes. For details about mode switching, please refer to Switch flight control Flight mode

10.3.7. Service/mavros/cmd/arming flight control unlocking interface

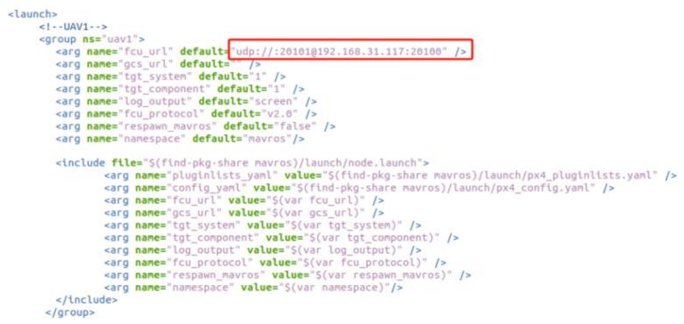
The flight control can be unlocked through the service name. Of course, unlocking is also subject to the constraints of flight control settings. Refer to for details. Unlock the flight control

10.4. Application of MavROS on RflySim Platform

10.4.1. MavROS Software in the Loop

Used in the RflySim platform MavROS ， Configuration of software bat script and MavROS startup script is required. Since MavROS is running Linux virtual machine or on-board computer in, first set the communication port number SET/a UDP _ START _ PORT = 20100 in the software in-the-loop script, Then look at the IP address of the Linux virtual machine for example 192.168.31.155 and change the setting in the script SET IS _ BROADCAST = 192.168.31.155. The 20100 is the external communication port provided by CopterSim, and the 20101 is the external communication port provided by MavROS.

The corresponding port settings are also made in the MavROS startup script as follows:



```
<launch>
    <!--UAV1-->
    <group ns="uav1">
      <arg name="fcu_url" default="udp://:20101@192.168.31.117:20100" />
      <arg name="gcs_url" default="" />
      <arg name="tgt_system" default="1" />
      <arg name="tgt_component" default="1" />
      <arg name="log_output" default="screen" />
      <arg name="fcu_protocol" default="v2.0" />
      <arg name="respawn_mavros" default="false" />
      <arg name="namespace" default="mavros"/>

      <include file="$(find-pkg-share mavros)/launch/node.launch">
              <arg name="pluginlists_yaml" value="$(find-pkg-share mavros)/launch/px4_pluginlists.yaml" />
              <arg name="config_yaml" value="$(find-pkg-share mavros)/launch/px4_config.yaml" />
              <arg name="fcu_url" value="$(var fcu_url)" />
              <arg name="gcs_url" value="$(var gcs_url)" />
              <arg name="tgt_system" value="$(var tgt_system)" />
              <arg name="tgt_component" value="$(var tgt_component)" />
              <arg name="log_output" value="$(var log_output)" />
              <arg name="fcu_protocol" value="$(var fcu_protocol)" />
              <arg name="respawn_mavros" value="$(var respawn_mavros)" />
              <arg name="namespace" value="$(var namespace)"/>
      </include>
     </group>
```

Where//: The middle of the symbol should be the IP address in the Linux virtual machine or on-board computer, but since MavROS is started in it, it defaults to its corresponding IP address, but it can also be added manually. The port 20101 is the external communication port of MavROS, and

192.168.31.117 is the IP address under the platform window.

10.4.2. MavROS Hardware-in-the-Loop

When MavROS is used for hardware-in-the-loop in the RflySim platform, the corresponding hardware-in-the-loop script shall be used. The flight control shall be connected to the computer through the USB port using the USB data line for communication with CopterSim, and connected to the computer through the TELEM1 using the USB to TTL line for communication between the computer and the flight control. Then set the corresponding port in the startup script of MavROS as follows:

```
<!--UAV1-->
<group ns="uav1">
   <arg name="fcu_url" default="/dev/ttyACM0" />
   <arg name="gcs_url" default="" />
   <arg name="tgt_system" default="1" />
   <arg name="tgt_component" default="1" />
   <arg name="log_output" default="screen" />
   <arg name="fcu_protocol" default="v2.0" />
   <arg name="respawn_mavros" default="false" />
   <arg name="namespace" default="mavros"/>

   <include file="$(find-pkg-share mavros)/launch/node.launch">
           <arg name="pluginlists_yaml" value="$(find-pkg-share mavros)/launch/px4_pluginlists.yaml" />
           <arg name="config_yaml" value="$(find-pkg-share mavros)/launch/px4_config.yaml" />
           <arg name="fcu_url" value="$(var fcu_url)" />
           <arg name="gcs_url" value="$(var gcs_url)" />
           <arg name="tgt_system" value="$(var tgt_system)" />
           <arg name="tgt_component" value="$(var tgt_component)" />
           <arg name="log_output" value="$(var log_output)" />
           <arg name="fcu_protocol" value="$(var fcu_protocol)" />
           <arg name="respawn_mavros" value="$(var respawn_mavros)" />
           <arg name="namespace" value="$(var namespace)"/>
   </include>
 </group>
```

10.4.3. MavROS and Flight Control Communication

10. 4. 3. 1. Obtain the flight control status

```
state_sub = rospy.Subscriber('/mavros/state', State, state_callback)
```

In mavros, to get the status of the Flight Control Unit (FCU), you can use the mavros/state topic to get the current flight status information. The mavros/state topic is used to publish the status information of the current flight controller. The message type for this topic is The mavros _ msgs/State contains information such as flight mode, connection status, and unlock status. The flight control status listener is defined as above.

10. 4. 3. 2. Obtain the aircraft attitude

```
rospy.Subscriber('/mavros/imu/data', Imu, imu_callback)
```

Obtain the attitude information of the aircraft in mavros, which can be obtained by using mavros/imu/data topics. These topics provide the attitude data of the aircraft, including information such as angle and angular velocity. The topic published the inertial measurement unit (IMU) data of the aircraft, including the angle and angular velocity. You can use ROS to subscribe to this topic to get attitude information. The message type for this topic is the sensor _ msgs/MSG/Imu message type. The aircraft attitude monitor is defined as above.

10. 4. 3. 3. Acquire the flight control pose

Local pose:

```
    local_pose_sub=rospy.Subscriber('/mavros/local_position/pose',PoseStamped,local_pose_call
back)
```

In mavros, to get the pose information of the flight controller, you can use the mavros/local _ position/pose topic to get the local position and pose information of the current aircraft. The topic is used to publish the local pose information of the current flight controller, including the position coordinates (such as the position in the northeast sky coordinate system) and the attitude information (such as the attitude represented by Euler angles or quaternions). The message type for this topic is geometry _ msgs/PoseStamped.

Global pose:

```
    global_pose_sub=rospy.Subscriber('/mavros/global_position/global',NavSatFix,global_pose_c
allback)
```

In mavros, to get the global pose information of the flight controller (usually expressed in longitude and latitude coordinates), you can use the mavros/global _ position/global topic to get the global position information of the current aircraft. This topic is used to publish the global position information of the current flight controller, including longitude and latitude coordinates and altitude. The message type for this topic is the sensor _ msgs/NavSatFix message type.

10. 4. 3. 4. Switch flight control Flight mode

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

In mavros, to set the flight mode of the flight controller, you can use the mavros/set _ mode service. The mavros/set _ mode service is used to request changes to the flight mode of the flight controller. By calling this service, you can request to switch the aircraft to different flight modes, such as manual mode, self-stabilization mode, positioning mode, etc. You can use the ROS service client to call the mavros/set _ mode service to request a change in flight mode. You need to create a service request message, usually using the mavros _ msgs/SetMode message type. In particular, the flight controller needs to send a target command before switching the offboard mode. For detailed examples, refer to the official website http://docs.px4.io/main/zh/ros/mavros_offboard_python.html.

10. 4. 3. 5. Unlock the flight control

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

In mavros, to unlock the flight control unit (FCU), the commonly used interface is a service named mavros/cmd/arming service, which allows you to control the arming and disarming of the flight control unit. Unlocking means that the motor of the flight controller is enabled so that it is ready to fly, while locking means that the motor is disabled so that it stops working. To invoke the mavros/cmd/arming service, you need to create a service request message, typically using the mavros _ msgs/CommandBool message type. Usually the real plane flies, for the sake of safety, we use the remote control to send the unlocking command, and only use the service type to unlock the flight control during the simulation. For unlocking examples, refer to the http://docs.px4.io/main/zh/ros/mavros_offboard_python.html on the PX4 official website.

10.4.4. Flight control mode of MavROS

There are many topics of mavros control and flight control, but it is enough to learn to use two of them. We only need to publish one or a combination of these two topics to achieve the control of the aircraft. Other mavros control topics of the aircraft are used by disassembling the functions of these two topics, so we will not describe each sub-function topic here. Note that the control frequency of each topic should be above 5 Hz, and if you need to combine multiple topics to control the effect, you'd better understand the underlying code of flight control.

Topic name	Explain
/mavros/setpoint_raw/local	Position control, speed control, acceleration control, or combination control can be achieved through this topic (this topic has a disassembly topic for the entire control function, which will not be introduced here. Interested friends can read the official documents).
/mavros/setpoint_raw/attitude	Attitude control can be achieved through this topic (this topic has a topic of disassembly of the entire control function, which will not be introduced here. Interested friends can read the official documentation).

10. 4. 4. 1. Attitude loop control

```
attitude_pub=rospy.Publisher('/mavros/setpoint_raw/attitude',AttitudeTarget,queue_size=10)
```

Content description for mavros _ msgs/AttitudeTarget message type:

```
uint8 IGNORE_ROLL_RATE=1
uint8 IGNORE_PITCH_RATE=2
uint8 IGNORE_YAW_RATE=4
uint8 IGNORE_THRUST=64
uint8 IGNORE_ATTITUDE=128
std_msgs/Header header
  uint32 seq
  time stamp
```

string frame_id
uint8 type_mask
geometry_msgs/Quaternion orientation
    float64 x
    float64 y
    float64 z
    float64 w
geometry_msgs/Vector3 body_rate
    float64 x
    float64 y
    float64 z
float32 thrust

Variable name	Explain	How to use
type_mask	The type mask is used to mask whether to use the desired function. Its value can be directly used in the message type. The value is not a static constant value. The value description is as follows: 1: angular velocity of shield roll 2: angular velocity of shield pitch 4: angular velocity of shielding yaw 64: Shielding throttle amount control 128: Shielding attitude amount control	The value can be directly assigned to the typt _ mask, and the combination of values (binary, hexadecimal, and decimal) can be used. It is recommended to use the constant name to directly add and assign the value. If the AttitudeTarget variable att is defined, this method is recommended. att.type_mask = att. IGNORE_ROLL_RATE + att.IGNORE_PITCH_RATE + ... Using this method is more intuitive and easier to read and use than binary or hexadecimal. Of course, you can also redefine variables for combined values;
orientation	The values needed to control the attitude of the aircraft are represented here by quaternions.	att.orientation.x = 'value' ....
body_rate	Controls the angular velocity of each component of the attitude, where X, y, Z correspond to roll _ rate, pitch _ rate, yaw _ rate	att.body_rate.x = 'value' ....
thrust	A given amount of throttle	Value range (0 ~ 1). Usually, when the throttle value is 0.6, the aircraft keeps hovering.

It should be noted that when giving flight control commands, it is often necessary to set the time, generally using (ROS: time: now () in C + +; Rospy. Time. Now ()) in Python, and then use the publishing topic/mavros/setpoint _ raw/attitude to publish the set attitude control data.

10.4.4.2. Acceleration loop control

```
accel_pub=rospy.Publisher('/mavros/setpoint_raw/local',PositionTarget ,queue_size=10)
```

In mavros, you can use the/mavros/setpoint _ raw/local topic for flight control Acceleration Loop control Publish 。 This topic allows you to send the desired Acceleration command on X, y, Z axes For flight control, so as to achieve Acceleration Control. The message type for this topic is PositionTarget Message type The details are as follows:

```
uint8 FRAME_LOCAL_NED =1
uint8 FRAME_LOCAL_OFFSET_NED=7
uint8 FRAME_BODY_NED=8
uint8 FRAME_BODY_OFFSET_NED=9
uint16 IGNORE_PX=1
uint16 IGNORE_PY=2
uint16 IGNORE_PZ=4
uint16 IGNORE_VX=8
uint16 IGNORE_VY=16
uint16 IGNORE_VZ=32
uint16 IGNORE_AFX=64
uint16 IGNORE_AFY=128
uint16 IGNORE_AFZ=256
uint16 FORCE=512
uint16 IGNORE_YAW=1024
uint16 IGNORE_YAW_RATE=2048
std_msgs/Header header
  uint32 seq
  time stamp
  string frame_id
uint8 coordinate_frame
uint16 type_mask
geometry_msgs/Point position
  float64 x
  float64 y
  float64 z
geometry_msgs/Vector3 velocity
  float64 x
  float64 y
  float64 z
geometry_msgs/Vector3 acceleration_or_force
  float64 x
  float64 y
  float64 z
float32 yaw
float32 yaw_rate
```

Variable name	Explain	How to use
coordinate_frame	Select what coordinate system to control based on. The optional values	Because only one of the coordinate systems on which

	are as follow 1: Although the static variable name is FRAME _ LOCAL _ NED, it actually uses the global ENU coordinate system, which has a yaw angle of 90 degrees with the coordinate system in RflySim3D. For details, please refer to Problems needing attention in using RflySim and mavros 2: FRAME _ LOCAL _ OFFSET _ NED this value is not supported by PX4 3: FRAME _ BODY _ NED Also although NED is indicated here, FLU is actually used relative to the airframe coordinate system; 4: FRAME _ BODY _ OFFSET _ NED this value is not supported by px4 Note: The coordinate system definition of the ROS version melodic is different from that of the previous version melodic, so I will not introduce it here.	the control quantity is based can be used, numerical assignment can be used, and the following method can also be used. tar. coordinate_frame = tar. FRAME _ LOCAL _ NED, speed control, or acceleration control generally uses the body coordinate system, or tar. coordinate_frame = tar. FRAME_BODY_NED

type_mask	Select the control type, which can be based on position control, based on speed control, based on acceleration control, or can be used in combination. When used in combination, the effect is the combination effect of each control quantity. Interested readers can read the underlying source code of flight control by themselves (this is necessary for smooth control).
	The optional values are:
	1: Shielding position control coordinate point X component
	2: Shield position control coordinate point y component
	4: Shield the Z component of the position control coordinate point
	8: Shielding speed control X direction component
	16: shield velocity control y-direction component
		The assignment of type _ mask is generally in the form of numeric value, such as (binary, hexadecimal). However, it is recommended to use static variable name assignment, which is more intuitive. The following assignment method is recommended:
		If it is a single control mode:
		Tar. Type _ mask = ~ (0);//set all to 1, shield all
		//Then release the shield for a single control mode, such as acceleration control only here
		tar.type_mask = ~tar. IGNORE_AFX & ~tar. IGNORE_AFY & ~tar. IGNORE_AFZ & ~ tar.FORCE; //Don't forget to block Force at the end.
		If multiple control modes are used in combination
		The control mode is assumed to be a combination of acceleration and velocity.
		Tar. Type _ mask = 0//set all to 0 to enable

	32: Shielding velocity controls z-direction component
	64: Shielding acceleration or thrust control X direction component
	128: shield acceleration or thrust control y-direction component
	256: Shield acceleration or thrust control z-direction component
	512: Select to use thrust control
	1024: Shield yaw control
	2048: Shield yaw _ rate control
		all controls, except Force. Froce is 0, which means masking. Then mask bit value control with pertinence;
		tar.type_mask = tar. IGNORE_PX |
		tar.IGNORE_PY | tar.IGNORE_PZ;

velocity	Speed control, where X, y, Z correspond to the values of the respective components based on the coordinate _ frame coordinate system	
acceleration_or_force	Acceleration or thrust control, where X, y, and Z correspond to the values of each component based on the coordinate _ frame coordinate system. When the Force flag in the type _ mask is 1, it indicates that thrust control is used, and when it is 0, it indicates that acceleration control is used. Not all flight controls support force control.	
yaw	The yaw angle control quantity is valid only when the IIGNORE _ YAW of the flag bit in the type _ mask is 0	
yaw_rate	The yaw-rate control quantity is valid only when the IIGNORE _ YAW _ RATE of the flag bit in the type _ mask is 0	

Similarly, in addition to controlling the quantity assignment, you also need to take the timestamp with you, generally using (ROS: Time: now () in C + +; Rospy. Time. Now ()) in Python, and then use the publishing topic/mavros/setpoint _ raw/local to publish the set acceleration control data.

10. 4. 4. 3. Speed loop control

```
vel_pub=rospy.Publisher('/mavros/setpoint_raw/local',PositionTarget ,queue_size=10)
```

Speed control and the use of acceleration and position control are the same topic control. The topic combines speed control, acceleration control and position control mode, and is selected by type _ mask. The Position Target data type is described in detail below:

uint8 FRAME_LOCAL_NED =1

```
uint8 FRAME_LOCAL_OFFSET_NED=7
uint8 FRAME_BODY_NED=8
uint8 FRAME_BODY_OFFSET_NED=9
uint16 IGNORE_PX=1
uint16 IGNORE_PY=2
uint16 IGNORE_PZ=4
uint16 IGNORE_VX=8
uint16 IGNORE_VY=16
uint16 IGNORE_VZ=32
uint16 IGNORE_AFX=64
uint16 IGNORE_AFY=128
uint16 IGNORE_AFZ=256
uint16 FORCE=512
uint16 IGNORE_YAW=1024
uint16 IGNORE_YAW_RATE=2048
std_msgs/Header header
  uint32 seq
  time stamp
  string frame_id
uint8 coordinate_frame
uint16 type_mask
geometry_msgs/Point position
  float64 x
  float64 y
  float64 z
geometry_msgs/Vector3 velocity
  float64 x
  float64 y
  float64 z
geometry_msgs/Vector3 acceleration_or_force
  float64 x
  float64 y
  float64 z
float32 yaw
float32 yaw_rate
```

Variable name	Explain	How to use
coordinate_frame	Select what coordinate system to control based on. The optional values are as follow 1: Although the static variable name is FRAME _ LOCAL _ NED, it actually uses the global ENU coordinate system,	Because only one of the coordinate systems on which the control quantity is based can be used, numerical assignment can be used, and the following method can also

	which has a yaw angle of 90 degrees with the coordinate system in RflySim3D. For details, please refer to Problems needing attention in using RflySim and mavros 7: FRAME _ LOCAL _ OFFSET _ NED that this value is temporarily not supported by PX4 8: FRAME _ BODY _ NED Also although NED is indicated here, FLU is actually used relative to the airframe coordinate system; 9: FRAME _ BODY _ OFFSET _ NED this value is not supported by px4 Note: The coordinate system definition of the ROS version melodic is different from that of the previous version melodic, so I will not introduce it here.	be used. tar. coordinate_frame = tar. FRAME _ LOCAL _ NED, speed control, or acceleration control generally uses the body coordinate system, or tar. coordinate_frame = tar. FRAME_BODY_NED

type_mask	Select the control type, which can be based on position control, based on speed control, based on acceleration control, or can be used in combination. When used in combination, the effect is the combination effect of each control quantity. Interested readers can read the underlying source code of flight control by themselves (this is necessary for smooth control).
	The optional values are:
	1: Shielding position control coordinate point X component
	2: Shield position control coordinate point y component
	4: Shield the Z component of the position control coordinate point
	8: Shielding speed control X direction component
	16: shield velocity control y-direction component
	32: Shielding velocity controls z-direction component
	64: Shielding acceleration or thrust control X direction component
		The assignment of type _ mask is generally in the form of numeric value, such as (binary, hexadecimal). However, it is recommended to use static variable name assignment, which is more intuitive. The following assignment method is recommended:
		If it is a single control mode:
		Tar. Type _ mask = ~ (0);//set all to 1, shield all
		//then release the mask for a single control mode, such as speed control only here
		tar.type_mask = ~tar. IGNORE_VX & ~tar. IGNORE_VX & ~tar. IGNORE_VX & ~tar.FORCE
		If multiple control modes are used in combination
		The control mode is assumed to be a combination of acceleration and velocity.
		Tar. Type _ mask = 0//set all to 0 to enable all controls; and targeted mask bit value control;
		tar.type_mask = tar. IGNORE_PX | tar.IGNORE_PY | tar.IGNORE_PZ;

	128: shield acceleration or thrust control y-direction component 256: Shield acceleration or thrust control z-direction component 512: Select to use thrust control 1024: Shield yaw control 2048: Shield yaw _ rate control	
velocity	Speed control, where X, y, Z correspond to the values of the respective components based on the coordinate _ frame coordinate system	
acceleration_or_force	Acceleration or thrust control, where X, y, and Z correspond to the values of each component based on the coordinate _ frame coordinate system. When the Force flag in the type _ mask is 1, it indicates that thrust control is used, and when it is 0, it indicates that acceleration control is used. Not all flight controls support force control.	
yaw	The yaw angle control quantity is valid only when the IIGNORE _ YAW of the flag bit in the type _ mask is 0	
yaw_rate	The yaw-rate control quantity is valid only when the IIGNORE _ YAW _ RATE of the flag bit in the type _ mask is 0	

Similarly, in addition to controlling the quantity assignment, you also need to take the timestamp with you, generally using (ROS: Time: now () in C + +; Rospy. Time. Now ()) in Python, and then use the publishing topic/mavros/setpoint _ raw/local to publish the set speed control data.

10. 4. 4. 4. Position loop control

```
Pose_pub=rospy.Publisher('/mavros/setpoint_position/local',PoseStamped,queue_size=10)
```

In mavros, you can use the mavros/setpoint _ position/local topic to publish the position loop control of the flight control. This topic allows you to send the desired position (X, y, Z) command in the NED coordinate system to the flight control to achieve position control. The message type for this topic is the geometry _ msgs/PoseStamped message type. The position loop control issuer is defined as above.

This is common because of its simplicity, but it is more recommended to use the topic/mavros/setpoint _ raw/local to control the location

```
Pose_pub=rospy.Publisher('/mavros/setpoint_raw/local ', PositionTarget,queue_size=10)
```

Position control, speed control, acceleration control/thrust control can be performed through topic/mavros/setpoint _ raw/local. Therefore, efficient control requires multiple combinations of controls. The following describes the data type PositionTarget and the position control method in detail

```
uint8 FRAME_LOCAL_NED =1
uint8 FRAME_LOCAL_OFFSET_NED=7
uint8 FRAME_BODY_NED=8
uint8 FRAME_BODY_OFFSET_NED=9
uint16 IGNORE_PX=1
uint16 IGNORE_PY=2
uint16 IGNORE_PZ=4
uint16 IGNORE_VX=8
uint16 IGNORE_VY=16
uint16 IGNORE_VZ=32
uint16 IGNORE_AFX=64
uint16 IGNORE_AFY=128
uint16 IGNORE_AFZ=256
uint16 FORCE=512
uint16 IGNORE_YAW=1024
uint16 IGNORE_YAW_RATE=2048
std_msgs/Header header
  uint32 seq
  time stamp
  string frame_id
uint8 coordinate_frame
uint16 type_mask
geometry_msgs/Point position
  float64 x
  float64 y
  float64 z
geometry_msgs/Vector3 velocity
  float64 x
  float64 y
  float64 z
geometry_msgs/Vector3 acceleration_or_force
  float64 x
  float64 y
  float64 z
float32 yaw
float32 yaw_rate
```

Variable name	Explain	How to use
coordinate_frame	Select what coordinate system to control based on. The optional values are as follow	Because only one of the coordinate systems on which the control quantity is based

	1: Although the static variable name is FRAME _ LOCAL _ NED, it actually uses the global ENU coordinate system, which has a yaw angle of 90 degrees with the coordinate system in RflySim3D. For details, please refer to Problems needing attention in using RflySim and mavros 7: FRAME _ LOCAL _ OFFSET _ NED that this value is temporarily not supported by PX4 8: FRAME _ BODY _ NED Also although NED is indicated here, FLU is actually used relative to the airframe coordinate system; 9: FRAME _ BODY _ OFFSET _ NED this value is not supported by px4 Note: The coordinate system definition of the ROS version melodic is different from that of the previous version melodic, so I will not introduce it here.	can be used, numerical assignment can be used, and the following method can also be used. tar. coordinate_frame = tar. FRAME _ LOCAL _ NED, speed control, or acceleration control generally uses the body coordinate system, or tar. coordinate_frame = tar. FRAME_BODY_NED

type_mask	Select the control type, which can be based on position control, based on speed control, based on acceleration control, or can be used in combination. When used in combination, the effect is the combination effect of each control quantity. Interested readers can read the underlying source code of flight control by themselves (this is necessary for smooth control).
	The optional values are:
	1: Shielding position control coordinate point X component
	2: Shield position control coordinate point y component
	4: Shield the Z component of the position control coordinate point
	8: Shielding speed control X direction component
	16: shield velocity control y-direction component
	32: Shielding velocity controls z-
		The assignment of type _ mask is generally in the form of numeric value, such as (binary, hexadecimal). However, it is recommended to use static variable name assignment, which is more intuitive. The following assignment method is recommended:
		If it is a single control mode:
		Tar. Type _ mask = ~ (0);//set all to 1, shield all
		//then release the shield for a single control mode, such as position control only here
		tar.type_mask = ~tar. IGNORE_PX & ~tar. IGNORE_PY & ~tar. IGNORE_PZ & ~tar.FORCE
		If multiple control modes are used in combination
		The control mode is assumed to be a combination of position and velocity
		Tar. Type _ mask = 0//set all to 0, enabling all controls; and targeted shielding of

	direction component
	64: Shielding acceleration or thrust control X direction component
	128: shield acceleration or thrust control y-direction component
	256: Shield acceleration or thrust control z-direction component
	512: Select to use thrust control
	1024: Shield yaw control
	2048: Shield yaw _ rate control
		acceleration control;
		tar.type_mask = tar. IGNORE_AFX | tar.IGNORE_AFY | tar.IGNORE_AFZ;

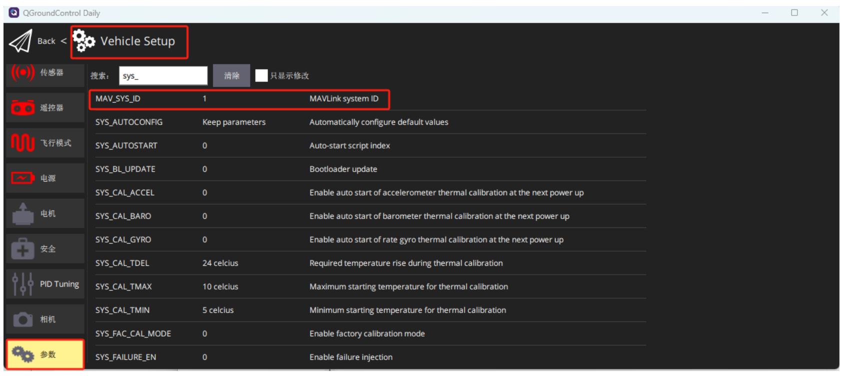
velocity	Speed control, where X, y, Z correspond to the values of the respective components based on the coordinate _ frame coordinate system	
acceleration_or_force	Acceleration or thrust control, where X, y, and Z correspond to the values of each component based on the coordinate _ frame coordinate system. When the Force flag in the type _ mask is 1, it indicates that thrust control is used, and when it is 0, it indicates that acceleration control is used. Not all flight controls support force control.	
yaw	The yaw angle control quantity is valid only when the IIGNORE _ YAW of the flag bit in the type _ mask is 0	
yaw_rate	The yaw-rate control quantity is valid only when the IIGNORE _ YAW _ RATE of the flag bit in the type _ mask is 0	

Similarly, in addition to controlling the quantity assignment, you also need to take the timestamp with you, generally using (ROS: Time: now () in C + +; Rospy. Time. Now ()) in Python, and then use the publishing topic/mavros/setpoint _ raw/local to publish the set location control data.

10.5. RflySim Multi-machine ROS Distributed Deployment

10.5.1. Configuration of flight control parameters

Connect each hardware flight control with NX through USB cable and open QGroundControl software to set ID number. This is for the CopterSimID to be consistent with the Flight Control ID. The settings are as follows:



MAV_SYS_ID	1	MAVLink system ID
SYS_AUTOCONFIG	Keep parameters	Automatically configure default values
SYS_AUTOSTART	0	Auto-start script index
SYS_BL_UPDATE	0	Bootloader update
SYS_CAL_ACCEL	0	Enable auto start of accelerometer thermal calibration at the next power up
SYS_CAL_BARO	0	Enable auto start of barometer thermal calibration at the next power up
SYS_CAL_GYRO	0	Enable auto start of rate gyro thermal calibration at the next power up
SYS_CAL_TDEL	24 celcius	Required temperature rise during thermal calibration
SYS_CAL_TMAX	10 celcius	Maximum starting temperature for thermal calibration
SYS_CAL_TMIN	5 celcius	Minimum starting temperature for thermal calibration
SYS_FAC_CAL_MODE	0	Enable factory calibration mode
SYS_FAILURE_EN	0	Enable failure injection

10.5.2. MavROS parameter configuration

For the MAVROS node running on each machine, the following parameters need to be configured:

```
 <group ns="uav1">
    <arg name="fcu_url" default="/dev/ttyACM0" />
    <arg name="gcs_url" default="" />
    <arg name="tgt_system" default="1" />
    <arg name="tgt_component" default="1" />
    <arg name="log_output" default="screen" />
    <arg name="fcu_protocol" default="v2.0" />
    <arg name="respawn_mavros" default="false" />
    <arg name="namespace" default="mavros"/>

    <include file="$(find-pkg-share mavros)/launch/node.launch">
            <arg name="pluginlists_yaml" value="$(find-pkg-share mavros)/launch/px4_pluginlists.yaml" />
            <arg name="config_yaml" value="$(find-pkg-share mavros)/launch/px4_config.yaml" />
            <arg name="fcu_url" value="$(var fcu_url)" />
            <arg name="gcs_url" value="$(var gcs_url)" />
            <arg name="tgt_system" value="$(var tgt_system)" />
            <arg name="tgt_component" value="$(var tgt_component)" />
            <arg name="log_output" value="$(var log_output)" />
            <arg name="fcu_protocol" value="$(var fcu_protocol)" />
            <arg name="respawn_mavros" value="$(var respawn_mavros)" />
            <arg name="namespace" value="$(var namespace)"/>
    </include>
  </group>
```

The parameter FCU _ URL is the connection mode of the flight control, the parameter GCS _ URL is that the IP of the host where the QGC is located has been defaulted, The parameter TGT _ system is the ID number of the flight control, the parameter TGT _ component specifies that the target component ID of the MAVLink message is the autopilot, and the parameter log _ output is that the log output mode is configured. The parameter FCU _ protocol is used to specify the MAVLink protocol version used to communicate with the flight controller, and the parameter namespace specifies the namespace of the node.

10.5.3. TF tree building

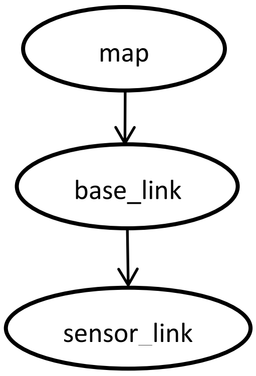
10. 5. 3. 1. Brief description of the TF tree

The TF tree is used to manage the coordinate transformation between the robot joint and the two links connected to it in the ROS. The joint can be fixed or float. Taking the robot arm as an example, if the position of the fingertip is known, the position of the fingertip to the base coordinate system should be known. Multiple transformation matrices need to be multiplied. If each joint needs to know that the fingertip is in the link coordinate system of the current joint connection, it needs to be multiplied, which is too cumbersome. Therefore, ROS introduces TF tree to manage this complex conversion relationship. At any time, we can directly obtain the coordinates in the desired coordinate system through link. In this technology, ROS has introduced a series of convenient debugging tools, such as Rviz, TF _ graph, etc. So is it necessary to deliberately build a TF tree? This depends on the requirements. In many algorithms, the TF tree is not used for coordinate conversion. The operation of coordinate conversion is completed inside the algorithm, and then the debugging visualization uses a local TF tree frame. In this case, it is not necessary to build the entire TF tree.

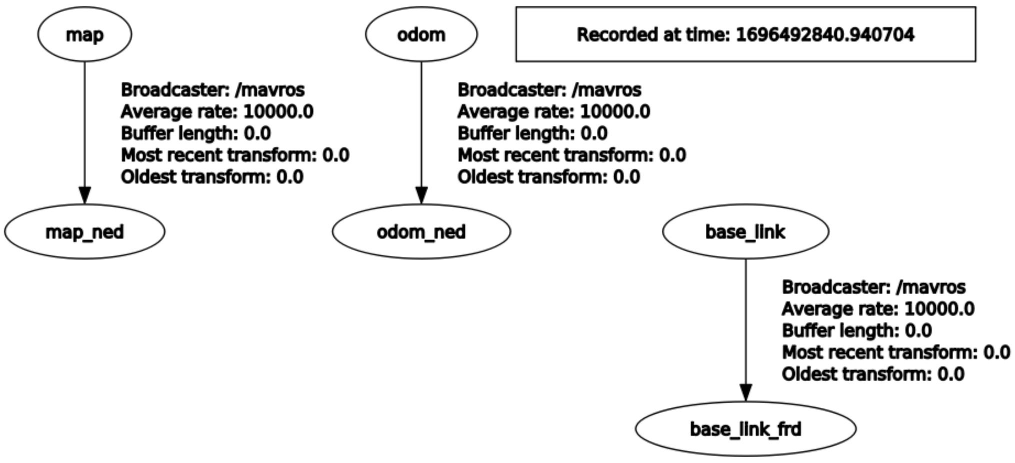
10. 5. 3. 2. Application of TF tree between RflySim and mavros

We know that there are three basic coordinate systems in ROS, map coordinate system, body coordinate system and sensor coordinate system. Usually, the corresponding frame _ ID in the TF tree are map, base _ link, (lidar _ link, camera _ link..) and so on. Often a complex system will introduce other coordinate systems between the three, such as Odom, base _ link _ foot, etc.

The sensors provided by RflySim use the frame _ ID of Map by default in order to facilitate visualization. Users can configure the frame _ ID of each sensor data by referring to the routine RflySimAPIs \ Python VisionAPI \ 1-APIUsageDemos \ 18-ConfigROSTFAPIDemo. Usually, the sensor coordinate system needs to be converted to the body coordinate system (base _ link), and the base _ link needs to be converted to the world coordinate system (map) for algorithm processing such as control, while the conversion from base _ link to map coordinate system needs to be given by the positioning system. So the simplest TF tree is as follows:



We run mavros to connect the flight control and then use the rqt _ TF (command: rosrun rqt _ graph rqt _ graph) tool to see what the current TF tree looks like



At this point, the TF tree is fragmentary, and we can connect base _ link to map through the external location source, or connect base _ link to Odom and then Odom to map. Form a complete Tf tree. Of course, we generally do not use the NED coordinate system, so we do not need to consider XXX _ Ned.

There are three common ways to build a TF tree:

1. The Tf tree can be built by loading the prepared urdf file, referring urdf/XML/link - ROS Wiki to how to build the URDF file, and then loading the file in the following way, as called in the launch file:

```
<launch>
<param name="robot_description" textfile="$(find mbot_description)/urdf/mbot_base.urdf" />
<!-- 运行 joint_state_publisher 节点，发布机器人的关节状态 -->
<node name="joint_state_publisher" pkg="joint_state_publisher" type="joint_state_publisher" />
<!-- 运行 robot_state_publisher 节点，发布 tf -->
<node name="robot_state_publisher" pkg="robot_state_publisher" type="robot_state_publisher" />
</launch>
```

Use the node joint _ state _ publisher to load the TF data in the urdf file, and use the node robot _ state _ publisher to publish the TF tree. This method is suitable for constructing the TF tree of a complex system. It is very convenient to construct all TF data structures in the urdf file.

2. For the simple TF tree structure, of course, it is only for those that are static, that is, the coordinate systems are rigidly connected. It is only necessary to add such a line of content in the launch startup file. For example, to build a TF tree result from the lidar _ link to the base _ link, it is only necessary to do the following:

```
<node pkg="tf" type="static_transform_publisher" name="base_link_to_laser"
    args="0.14 0.0 0.0 0.0 0.0 0.0 base_link laser_link 40" />
```

Args: amount of translation (X, y, Z), quaternion rotation Q (offset _ X, offset _ y, offset _ Z, Q _ X, Q _ y, Q _ Z, Q _ w), frame _ ID, child _ frame _ ID, frequency;

3. For two coordinate systems that are not rigidly connected, it is necessary to publish the

corresponding conversion relationship in real time through the code, such as:

```
    #include <ros/ros.h>
    # include < TF/transform _ broadcaster. H >//Add the library header file of TF transformation
    #include <geometry_msgs/PoseStamped.h>

void poseCallback(const geometry_msgs::PoseStamped::ConstPtr& msg){
   static tf::TransformBroadcaster br;
   tf::Transform transform;
transform.setOrigin(tf::Vector3(msg->pose.position.x,msg->pose.position.y,msg->pose.position.z)
);
   //tf::Quaternion quaternion;
    transform.setRotation(   tf::Quaternion(msg->pose.orientation.x,   msg->pose.orientation.y,
msg->pose.orientation.z, msg->pose.orientation.w) );
   br.sendTransform(tf::StampedTransform(transform, ros::Time::now(), "map", "base_link"));
}
int main(int argc, char** argv){
   ros::init(argc, argv, "my_tf_broadcaster");
   ros::NodeHandle node;
   ros::Subscriber sub = node.subscribe("/mavros/vision_pose/pose", 10, &poseCallback);
   ros::spin();
   return 0;
};
```

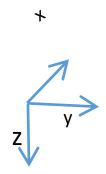
The above code establishes the connection from map to base _ link by subscribing to the visual location source from outside the flight control, because the carrier will move in real time, so the connection between base _ link and map cannot be rigid.

11. Introduction to coordinate systems such as MavROS and RflySim

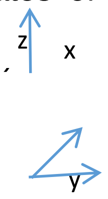
11.1. RflySim3D coordinate system

11.1.1. RflySim3D Map coordinate system

RflySim3D coordinate system is the FRD or NED of North East (X North, Y East, Z Down).



The coordinates of the CopterSim interface are the North-East coordinate system (NEU



11.1.2. Carrier coordinate system

Only when the coordinate system of the carrier is determined, the installation position and angle of the sensor can be determined. It should be noted that the object in the Rflysim 3D model can be scaled, so generally speaking, the installation position of the sensor also needs to be scaled according to the requirements. For example, if the model is magnified by 10 times, the sensor before magnification is (1,0,0) at the head position in the carrier coordinate system. If the sensor is still required to be at the head position after magnification, You need to change the sensor configuration file, setting (10,0,0).

The carrier coordinate system is also NED (coordinate system) in RflySim3D, so the installation position and attitude of the sensor need to be configured with reference to this coordinate system.

11.1.3. RflySim IMU coordinate system

RflySim outputs the FRD (X Forward, Y Right, Z Down) coordinate system by default, but in order to unify with mavros, it is currently changed to the FLU (X Forward, Y Left, Z Up) coordinate system

11.1.4. Sensor coordinate system

The coordinate system of the camera is: (UWN), that is, the Z axis points to the north, the X axis points to the sky, and the Y axis point to the west

Lidar coordinate system: FLU (X Forward, Y Left, Z Up)

11.2. MavROS coordinate system

11.2.1. MavROS IMU coordinate system

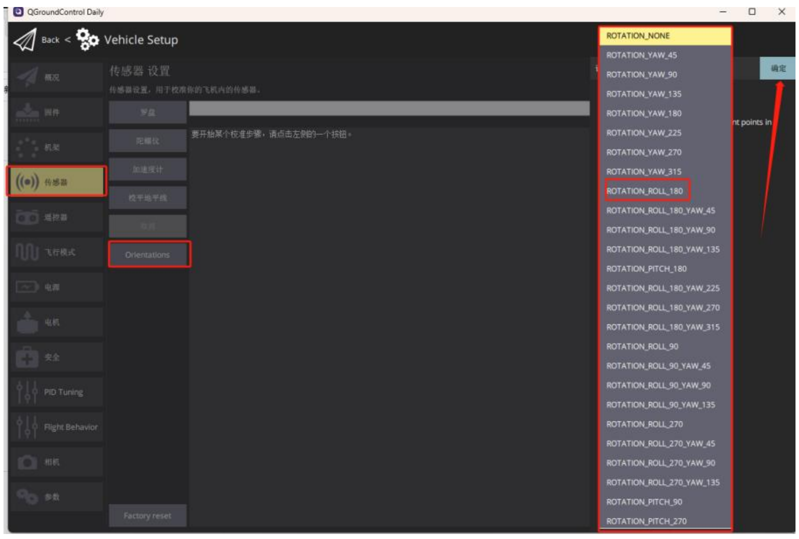
11.2.1.1. Software-in-the-loop simulation

The IMU coordinate system output by mavros is FLU by default. Note: In order to unify the coordinate system, the IMU coordinate system output by the new version of VisionCaptureAPI interface no longer uses the previous FRD (X Forward, Y Right, Z Down) coordinate system. Use FLU consistently with MavROS.

11.2.1.2. Hardware-in-the-loop simulation

Before using the IMU of mavros, first determine the IMU coordinate system. If it is the northeast earth, then change the coordinate system to the northeast sky. The specific steps are as follows:

1) Open QGC (QGround Control) and connect the flight control;
2) Select the Gear icon (Body Settings) in the toolbar, then select Sensor in the sidebar
3) Select the Set Orientations or Orientations button, as shown in the following figure, and select the coordinate system option of rotating to the north and east according to the coordinate system output by IMU;



Then click OK, and remember that the flight control needs to be powered on again to take effect.

11.2.2. MavROS map coordinate system

11.2.2.1. Problems needing attention in using RflySim and mavros

When using RflySim simulation combined with mavros, it should be noted that the map coordinate system used in ROS is also ENU coordinate system, but in the process of development based on mavros, it will be found that the coordinate system in RflySim3D is NED, except that the Z axis is different, there is a 90-degree yaw angle between the two. So when we send the coordinates via/mavros/setpoint _ position/local or via/mavros/setpoint _ raw/local to take off, the aircraft will rotate 90 degrees clockwise to the XY plane during takeoff. So if we want to directly use the coordinates of a certain point in RflySim3D to send through mavros, we need to interchange X and y, X is negative, and Z is negative. If we use attitude control, we also need to pay attention to this difference. Another detail to note here is that if our plane is flying in a corridor, according to custom, we want the nose direction to point to the other end of the corridor, while the X direction in RflySim3D is parallel to the corridor wall, and the X direction goes straight to the other end of the corridor. We control the plane from (0,0,0) to the other end of the corridor (100,0,0) by mavros, and we take off the plane. And send a take-off target point (0,0,2) via the topic/mavros/setpoint _ position/local or/mavros/setpoint _ raw/local. When you take off, you will find that the plane is facing the left wall. This is not the desired effect, so we need to send the target punctuation with a 90-degree yaw (position: 0, 0, 2; Orientation: (0.707, 0, 0, 0.707)), a yaw angle needs to be added every time the target point is sent. Of course, there is no problem to use this way to point the flight, but if we control the attitude, this method can not be used, at this time, we need to start the bat script initialization angle of RflySim to a 90-degree value "SET/a ORIGIN _ YAW = 90", through the description of the above example, When we use mavros to simulate in RflySim, we need to pay attention to the following points: determine the control mode (pose control, speed control, attitude control), the direction of the nose in the scene (for example, in the above example, if the target point does not give a yaw angle, it is not impossible for the aircraft to "face the wall and think"). If the corridor direction in the scene is the y direction of RflySim3D, then after the aircraft takes off, the nose direction is just the corridor direction, so it needs to be set flexibly according to the needs. In particular, when applied to the SLAM scenario, the initialization angle in the bat script for starting RflySim should be set to SET/a ORIGIN _ YAW = 90, because if this value is not set, it may cause a large error in the initialization of the SLAM algorithm due to excessive angular velocity during takeoff rotation.