RflySim 平台视觉接口文档

目录

1. RflySim 视觉架构简介 7
2. 视觉功能开发环境配置 8
    2.1. Windows 开发环境 9
        2.1.1. 平台自带 VS Code 开发环境 9
        2.1.2. Pycharm 开发环境 10
    2.2. Ubuntu 虚拟机开发环境 12
    2.3. Ubuntu 开发环境 12
3. 视觉算法开发依赖开发工具简介 12
    3.1. MAVLink/mavros/pymavlink 12
    3.2. ROS/ROS2 13
    3.3. OpenCV 14
4. RflySim 平台视觉配置步骤 15
    4.1. bat 脚本配置方法（针对视觉的配置） 15
        4.1.1. bat 脚本模板选择（根据需求选择已有脚本模板） 15
        4.1.2. START_INDEX（CopterSim 中 CopterID 的起始编号） 15
        4.1.3. UDP_START_PORT （CopterSim 中 MavLink UDP 通信的端口号） 15
        4.1.4. SimMode （硬件/软件仿真模式选择） 16
        4.1.5. UE4_MAP （指定 RflySim3D 中场景地图） 16
        4.1.6. IS_BROADCAST（选择 CopterSim MavLink 消息通信方式） 16
        4.1.7. UDPSIMMODE （MavLink 消息通信 UDP 仿真模式设置） 16
        4.1.8. TOTOAL_COPTER （飞机的总数量） 17
        4.1.9. VEHICLE_INTERVAL （初始化飞机位置间隔） 17
        4.1.10. 飞控初始化位姿设置(Pos,Yaw，以及分布式多机设置) 17
        4.1.11. IsSysID（硬件在环特有参数，决定 CopterID 是否使用飞控的 sys_id） 17
        4.1.12. 更多 bat 脚本模板说明（如独立设置每架飞机的位姿等） 17
    4.2. PX4 软件在环仿真步骤 17
        4.2.1. Bat 脚本设置 17
    4.3. PX4 硬件在环仿真步骤（飞控需要还原官方固件） 18
    4.4. PX4 软件在环+虚拟机在环 19
        4.4.1. Bat 脚本配置 19
    4.5. PX4 软件在环+NX 硬件在环 19
        4.5.1. Bat 脚本配置 19
        4.5.2. NX 端配置 19
    4.6. PX4 硬件在环+虚拟机在环 19
        4.6.1. Bat 脚本配置 20
        4.6.2. 虚拟机配置 20
    4.7. PX4 硬件在环+NX 硬件在环 21
        4.7.1. Bat 脚本配置 21
        4.7.2. NX 配置 22
    4.8. 多 PX4+多 NX 硬件在环 22
        4.8.1. Bat 脚本配置 23

4.8.2. 飞控参数设置 23
4.8.3. NX 配置 23
5. 视觉相关通信端口介绍 24
5.1. 视觉接口与 CopterSim，RflySim 进行 UDP 相互通信端口说明 24
5.1.1. 端口 20010（默认单个 RflySim3D 窗口，视觉接口与 RflySim3D 通信请求的端口，如碰撞检测，目标属性，相机属性等） 24
5.1.2. 端口 20100（CopterSim 接受重新启动某架飞机的仿真端口） 24
5.1.3. 端口 30100（CopterSim 单架飞机默认接受外部请求 IMU 数据的端口） 24
5.1.4. 端口 31000 （视觉接口配置当前系统默认接受 IMU 数据的端口） 24
5.1.5. 端口 20005 （视觉接口向 CopterSim 请求飞机起飞的时间戳） 25
5.1.6. 端口 20006（监听来自 RflySim3D 的数据） 25
5.1.7. 通信相关数据结构说明 25
5.2. Python 程序与 CopterSim 通信接口 29
5.2.1. PX4SILIntFloat（输出到 CopterSim DLL 模型的 SILInts 和 SILFloats 数据） 29
5.2.2. InitTrueDataLoop（启动两个线程，分别接收 CopterSim 的真实数据和 PX4 数据） 30
5.2.3. InitMavLoop（开启 MAVLink 监听 CopterSim 数据，并实时更新） 30
5.2.4. TimeStmploop（循环监听指定 CopterSim 的时间戳数据） 30
5.2.5. StartTimeStmplisten（线程循环监听指定 CopterSim 的时间戳数据，并不阻塞主线程） 31
5.2.6. endMavLoop（停止接收 CopterSim 数据，和 stopRun 效果一致） 31
5.3. Python 程序与 RflySim3D 通信接口 31
5.3.1. reqVeCrashData（由 RflySim3D 发送的结构体，主要是与碰撞相关的数据） 31
5.3.2. CoptReqData（由 RflySim3D 返回的飞机数据，发送的某个 Copter 的信息） 32
5.3.3. ObjReqData（由 RflySim3D 返回的数据，发送的某个物体的信息） 33
5.3.4. CameraData（由 RflySim3D 返回的数据，发送的某个相机（视觉传感器）的信息） 33
5.3.5. UE4MsgRecLoop（用于处理 RflySim3D 或 CopterSim 返回的消息，一共有 6 种消息） 34
5.3.6. getCamCoptObj（从 RflySim3D 获得指定的数据，并将监听到的三种存放到了 3 个列表中） 36
5.3.7. reqCamCoptObj（请求场景中物体的数据（并不能创建新的物体，而是获得已存在物体的数据）） 36
5.3.8. sendUE4Cmd（向 RflySim3D 发送一个“字节字符串命令”） 38
5.4. Python 程序与其他 Ubuntu 里 ROS 节点通信 38
5.5. RflySim 平台与其他平台时间戳对齐应用 38
5.5.1. RflySim 时间戳，远程系统时间戳与 ROS 时间戳 38
5.5.2. 各系统间时间戳对齐 39
6. RflySim3D 控制接口 UE4CtrlAPI.py 40
6.1. 场景控制接口（发送命令控制 RflySim3D） 40
6.1.1. sendUE4Pos（向 RflySim3D 中给指定 copter_id 的对象设置其位姿，类型等） 40

6.1.2. sendUE4Cmd（向 RflySim3D 发送一个“命令”） 40
6.2. 数据请求接口（获取所有物体 BoundingBox 等属性） 42
6.2.1. CoptReqData（由 RflySim3D 返回的飞机数据，发送的某个 Copter 的信息） 42
6.2.2. ObjReqData（由 RflySim3D 返回的数据，发送的某个物体的信息） 43
6.2.3. CameraData（由 RflySim3D 返回的数据，发送的某个相机（视觉传感器）的信息） 43
6.2.4. reqCamCoptObj（请求 RflySim3D 场景中） 44
6.2.5. UE4MsgRecLoop（循环监听来自 RflySim3D 的数据） 44
6.2.6. getCamCoptObj（从 RflySim3D 获得指定的目标的属性） 46
6.2.7. reqCamCoptObj（请求获得 RflySim3D 中指定物体的属性） 46
6.3. 碰撞检测之类接口 47
6.3.1. reqVeCrashData（由 RflySim3D 发送的结构体，里面主要是与碰撞相关的数据） 47
6.3.2. UE4MsgRecLoop（用于处理 RflySim3D 或 CopterSim 返回的消息） 47
6.4. 地形获取接口 49
6.4.1. sendUE4Cmd（向 RflySim3D 发送命令） 49
7. 视觉传感器配置协议 Config.json 51
7.1. Config. json 协议总体介绍 51
7.1.1. Config.json 文件参数说明 51
7.1.2. jsonLoad（通过读取配置文件来增加视觉传感器） 52
7.1.3. addVisSensor（添加一个视觉传感器参数的对象） 52
7.2. 发送与接收方法 53
7.2.1. sendUpdateUEImage（发送了一个视觉传感器请求，RflySim3D 收到后会创建或更新这个传感器） 53
7.2.2. sendReqToUE4（发送一组 VisionSensorReq） 53
7.3. 支持的传感器列表及配置方法 54
7.3.1. 可见光 RGB 图像 54
7.3.2. 灰度图像 55
7.3.3. 深度图像 55
7.3.4. 分割图 56
7.3.5. 测距传感器 57
7.3.6. 机械式扫描激光雷达 58
7.3.7. 花式扫描激光雷达 59
7.3.8. 红外灰度图像 60
7.3.9. 热力彩色图 61
8. Python 视觉接口 VisionCaptureApi.py 61
8.1. 相关类及数据结构 61
8.2. 发送与接收图像相关接口 66
8.2.1. sendReqToUE4（发送一组 VisionSensorReq） 66
8.2.2. jsonLoad（通过读取配置文件来增加视觉传感器） 66
8.2.3. sendReqToCopterSim（发送 UDP 数据向 CopterSim 请求传感器数据。如：IMU） 66
8.2.4. img_mem_thrd（通过读取共享内存来接收 RflySim3D 回传的图） 67

8.2.5. startImgCap（开启抓取共享内存中图像的线程） 67
8.2.6. sendImgUDPNew（将 RflySim3D 回传的图片通过 udp 发送到指定 ip 和端口号） 67
8.2.7. img_udp_thrdNew（接收 RflySim3D 回传的图像数据） 67
8.3. IMU 获取相关接口 68
8.3.1. sendImuReqCopterSim（通过发送 SensorReqCopterSim 实例来请求 Imu 数据） 68
8.3.2. sendImuReqClient（发送 SensorReqCopterSim 实例来请求 Imu 数据） 68
8.3.3. sendImuReqServe（在线程中接收 CopterSim 回传的 Imu 数据） 69
8.3.4. AlignTime（保证 Imu 数据发布过程中保持稳定频率发布） 69
8.4. ROS 相关接口 69
8.4.1. Imu2ros（CopterSim 回传的 imu 消息到 ros 内置 imu 消息格式的转换） 69
8.4.2. getIMUDataLoop（将 Imu 数据转换到 ros 中 Imu 的消息，并发送到指定话题中） 69
8.4.3. img_mem_thrd（全局变量 isEnableRosTrans 为 True 时，将以 ros 消息的形式发布图片内容） 70
8.4.4. img_udp_thrdNew（全局变量 isEnableRosTrans 为 True 时，将以 ros 消息的形式发布图片内容） 70
8.5. 其他内部函数 71
8.5.1. sendImgBuffer（将待发送数据按固定大小封装成一个或多个报文，通过 udp 发送到指定 ip 和端口号） 71
8.5.2. initUE4MsgRec（启动一个 t4(self.UE4MsgRecLoop)开始监听 224.0.0.10：20006） 71
8.5.3. endUE4MsgRec（停止线程 t4(self.UE4MsgRecLoop)的监听） 71
8.5.4. UE4MsgRecLoop（用于处理 RflySim3D 或 CopterSim 返回的消息） 71
8.5.5. getUE4Pos（获得 Copter 在 RflySim3D 中的位置） 72
8.5.6. getUE4Data（获得 Copter 在 RflySim3D 中的数据） 72
8.5.7. TimeStmploop（接收 CopterSim 回传的心跳和时间戳数据，确保终端和 CopterSim 的连接正常） 73
8.5.8. StartTimeStmplisten（子线程中接收 CopterSim 回传的心跳和时间戳数据，确保 CopterSim 连接正常 73
8.5.9. sendUE4Attatch（其他 Copter 附加到其他 Copter 上，“附加”的意思是会跟随移动） 73
8.5.10. sendUE4PosScalePwm20（设置 20 组无人机的状态属性，并设置电机数据） 74
8.5.11. sendUE4Pos(向局域网内所有 RflySim3D 发送一个 Copter 的数据，如果不存在那么会创建一个) 75
8.5.12. sendUE4PosScale100（设置 100 组无人机状态属性） 75
8.5.13. sendUE4PosScale（发送一个 Copter 的数据，RflySim3D 收到后更新该 Copter，带上了一个缩放系数） 76
8.5.14. sendUE4Pos2Ground（发送一个 Copter 的数据，RflySim3D 收到后会更新该 Copter，但它可以让 Copter 贴近地面） 76
8.5.15. sendUE4PosScale2Ground（发送一个 Copter 的数据，更新该 Copter，带上了一个缩放系数，并设置飞机贴地） 77

8.5.16. sendUE4PosNew（发送一个 Copter 的数据，RflySim3D 收到后会更新该 Copter） 77
8.5.17. sendUE4PosSimple（发送一个 Copter 的数据，RflySim3D 收到后会更新该 Copter） 78
8.5.18. sendUE4PosFull（发送一个 Copter 的数据，RflySim3D 收到后会更新该 Copter） 78
8.5.19. sendUE4ExtAct（向目标 Copter 发送一组数据，由目标的自定义函数进行处理，可以实现各种自定义的效果） 79
9. Python 飞机控制接口 PX4MavCtrlV4.py 79
9.1. 初始化 80
9.1.1. InitMavLoop（开启 MAVLink 监听 CopterSim 数据，并实时更新） 80
9.1.2. endMavLoop（停止接收 CopterSim 数据，和 stopRun 效果一致） 80
9.1.3. stopRun（退出 MAVLink 数据接收模式） 80
9.2. offboard 结束退出 81
9.2.1. initOffboard（开启 offboard 模式，并启动线程，循环发送 offboard 消息） 81
9.2.2. initOffboard2（开启 offboard 模式，并启动线程，循环发送 offboard 消息） 81
9.2.3. OffboardSendMode（循环发送 offboard 数据） 81
9.2.4. endOffboard（发送 MAV_CMD_NAV_GUIDED_ENABLE 命令退出 offboard 模式，并停止发送 offboard 消息） 81
9.3. 解锁 82
9.3.1. SendMavArm（无人机解锁或上锁） 82
9.4. 飞机指点飞行接口 82
9.4.1. SendPosNED（发送北东地坐标系下的目标位置和航向角接口） 82
9.4.2. SendPosNEDNoYaw（发送北东地坐标系下的目标位置） 82
9.4.3. SendPosNEDExt（选择北东地或机体前右下坐标系发送目标位置到 PX4） 82
9.4.4. SendPosGlobal（地球坐标系（经纬度，高度）带偏航角指点飞行接口） 83
9.4.5. sendMavLand（发送三维坐标、MAV_CMD_NAV_LAND 命令使飞机降落到目标位置） 83
9.4.6. sendMavTakeOffLocal（发送三维坐标、航向角、俯仰角、起飞速度 MAV_CMD_NAV_TAKEOFF_LOCAL 命令使飞机飞行到目标位置） 84
9.4.7. sendMavTakeOff（发送三维坐标、航向角、俯仰角、MAV_CMD_NAV_TAKEOFF 命令使飞机起飞到目标位置） 84
9.4.8. FRD 坐标系下的指点飞行接口 84
9.4.9. FRD 坐标系下无偏航角的指点飞行接口 85
9.5. 飞机速度控制接口 86
9.5.1. FRD 坐标系下的速度控制接口 86
9.5.2. 不控飞机偏航角的速度控制接口 86
9.5.3. 带偏航角速度以及高度控制接口 87
9.6. 飞机姿态控制接口 89
9.6.1. SendAttPX4（FRD 坐标系下的姿态控制接口） 89
9.7. 飞机加速度控制接口 89
9.7.1. SendAccPX4（发送目标加速度到 PX4） 89
9.8. 载体，载体视觉传感器，目标等属性接口 90

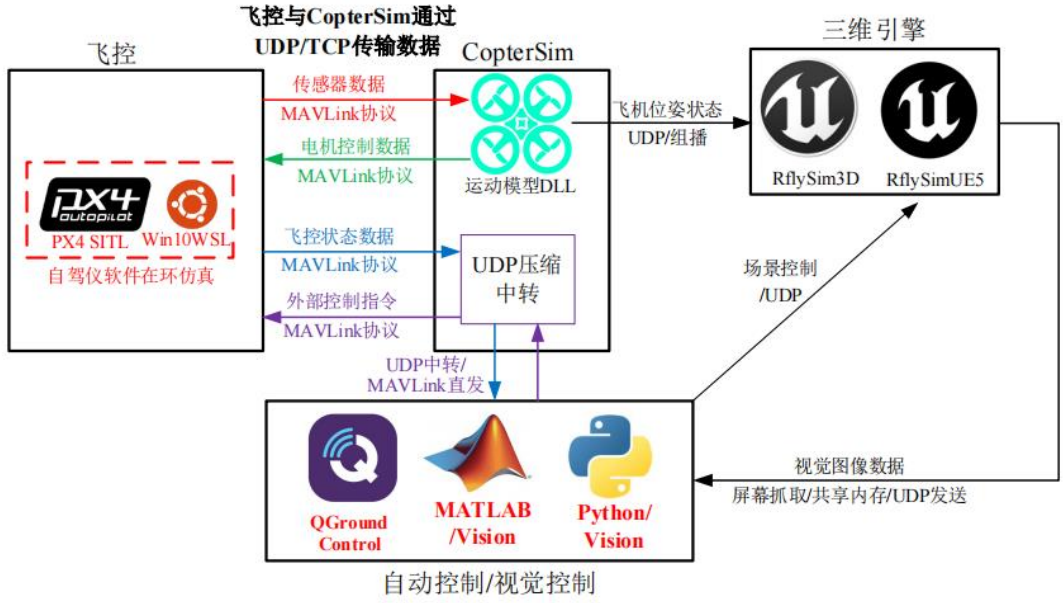
9.8.1. 请求载体飞机属性 90
9.8.2. 请求载体视觉传感器接口 90
9.8.3. 请求场景内目标数据接口 91
9.9. 其他内部函数 92
9.9.1. sendStartMsg（唤醒所有飞机） 92
9.9.2. waitForStartMsg（程序阻塞直到收到 sendStartMsg 的消息） 92
9.9.3. initPointMassModel（创建一个质点无人机模型） 92
9.9.4. EndPointMassModel（结束质点模型） 92
9.9.5. yawSat（返回-PI 到 PI 之间的航向角） 93
9.9.6. PointMassModelLoop （质点模型处理消息的死循环） 93
9.9.7. InitTrueDataLoop（启动两个线程，分别接收 CopterSim 的真实数据和 PX4 数据） 93
9.9.8. EndTrueDataLoop（停止 InitTrueDataLoop 的线程监听） 93
9.9.9. initUE4MsgRec（启动一个 t4(self.UE4MsgRecLoop)开始监听 224.0.0.10：20006） 93
9.9.10. endUE4MsgRec（停止线程 t4(self.UE4MsgRecLoop)的监听） 94
9.9.11. UE4MsgRecLoop（用于处理 RflySim3D 或 CopterSim 返回的消息） 94
9.9.12. sat（inPWM 限幅） 95
9.9.13. SendMavCmdLong（发送 MAVLINK 消息的 command_long 消息，可以请求无人机执行某些操作 ） 96
9.9.14. sendMavOffboardCmd（发送 offboard 命令到飞控） 96
9.9.15. TypeMask（获取 offboard 消息需要的 typemask） 97
9.9.16. sendMavOffboardAPI（更新 offboard 消息的数据） 97
9.9.17. sendUDPSimpData（设置控制模式、控制量） 97
9.9.18. sendMavSetParam（发送命令到 PX4 修改指定参数） 98
9.9.19. SendSetMode（发送 MAV_CMD_DO_SET_MOD 设置飞机模式） 98
9.9.20. getTrueDataMsg（循环监听真实数据，并更新内置状态） 98
9.9.21. getPX4DataMsg（循环监听 PX4 数据，并更新内置状态） 98
9.9.22. getMavMsg（循环更新 MAVLink 接收的数据） 98
9.9.23. sendCustomData（发送一个 16 维数据到指定端口，本接口可用于与 Simulink 通信） 99
10. 开发中需要用到的 mavros/ROS 相关接口 99
10.1. ROS 中常用通信方式介绍 99
10.1.1. 消息话题通信 99
10.1.2. 服务类型通信 100
10.2. ROS 在 RflySim 平台上应用介绍 100
10.2.1. 通过话题获取视觉传感器数据接口 100
10.2.2. 通过话题获取 IMU 数据接口 102
10.3. MavROS 常用接口 102
10.3.1. 话题 /mavros/state 获得飞控状态信息 102
10.3.2. 话题 /mavros/local_position/pose 获得飞控位姿数据 102
10.3.3. 话题 /mavros/imu/data_raw 获取飞控 IMU 数据 102
10.3.4. 话题 /mavros/setpoint_raw/local 发送位置、速度、加速度、控制指令接口 103

10.3.5. 话题 mavros/setpoint_raw/attitude 发送姿态控制接口 103
10.3.6. 服务 /mavros/set_mode 飞控模式切换接口 103
10.3.7. 服务 /mavros/cmd/arming 飞控解锁接口 103
10.4. MavROS 在 RflySim 平台上的应用 103
10.4.1. MavROS 软件在环 103
10.4.2. MavROS 硬件在环 104
10.4.3. MavROS 与飞控通信 104
10.4.4. MavROS 对飞控控制方式 106
10.5. RflySim 多机 ROS 分布式部署 115
10.5.1. 飞控参数配置 115
10.5.2. MavROS 参数配置 115
10.5.3. TF 树搭建 116
11. MavROS，RflySim 等坐标系介绍 118
11.1. RflySim3D 坐标系 118
11.1.1. RflySim3D Map 坐标系 118
11.1.2. 载体坐标系 119
11.1.3. RflySim IMU 坐标系 119
11.1.4. 传感器坐标系 119
11.2. MavROS 坐标系 120
11.2.1. MavROS IMU 坐标系 120
11.2.2. MavROS map 坐标系 121

1. RflySim 视觉架构简介

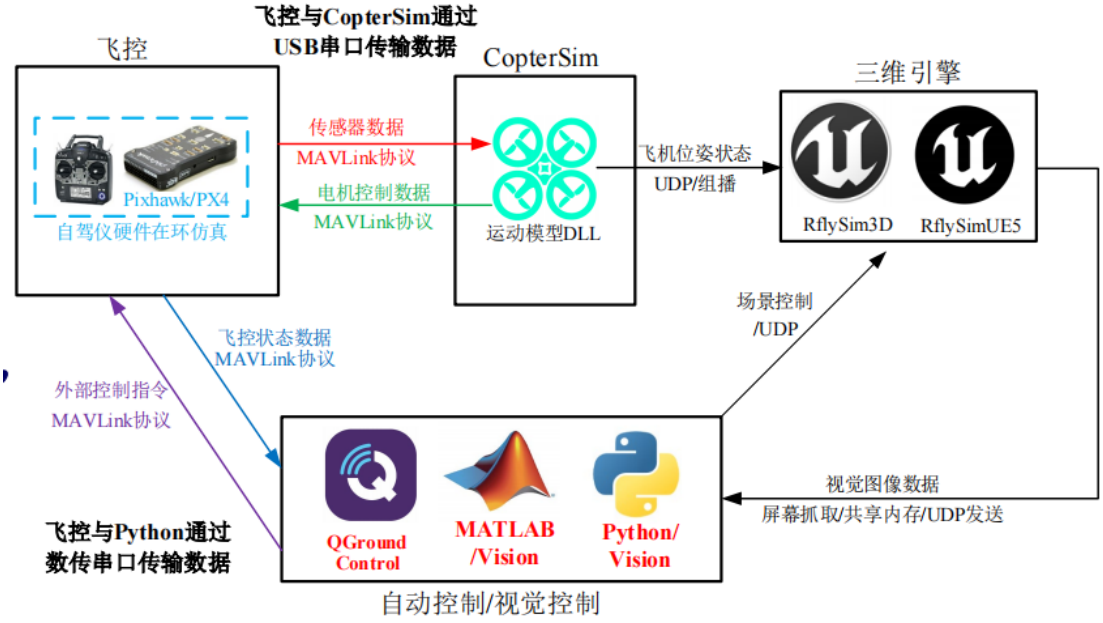
RflySim 视觉架构分为软件在环(SITLRun 通信框架)和硬件在环(HITL+数传通信框架)两部分。

在 SITL 软件在环仿真过程中,PX4 飞控完整运行于 Win10WSL 虚拟机中，使用 px4_sitl 固件。PX4 飞控完整运行于 Win10WSL 虚拟机中，使用 px4_sitl 固件。PX4 与 CopterSim 直接通过网络 MAVLink 协议通信，然后 CopterSim 通过 20100/20101 端口实现外部 Python 消息的收发。Python 程序通过 UDP 直接与 RflySim3D 程序通信，并通过共享内存/UDP 发送等方式获取获取视觉图像。



图一 SITLRun 软件在环仿真通信构架

在 HITL 硬件在环仿真中，PX4 飞控算法运行与 Pixhawk 硬件中，使用 px4_fmu-v5 固件。PX4 与 CopterSim 直接通过 USB 串口的 MAVLink 协议通信，然后 CopterSim 通过数传串口实现与外部 Python 程序的消息收发。Python 程序通过 UDP 直接与 RflySim3D 程序通信，并通过共享内存/UDP 发送等方式获取获取视觉图像。



图二 HITLRun 硬件在环仿真通信构架

2. 视觉功能开发环境配置

RflySim 平台的视觉功能开发环境分为 Windows 开发环境和 Ubuntu 虚拟机开发环境两

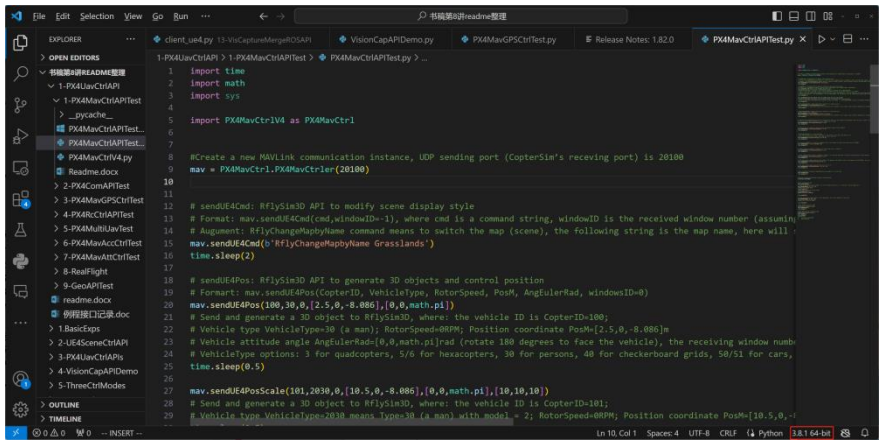
部分。Windows 开发环境用于利用 RflySim 平台的各个接口进行对无人机的运动控制、场景控制、视觉图像数据获取，Ubuntu 虚拟机开发环境则为了适应 Linux 系统环境中 ros 的开发需求进行与 RflySim 平台的适配。

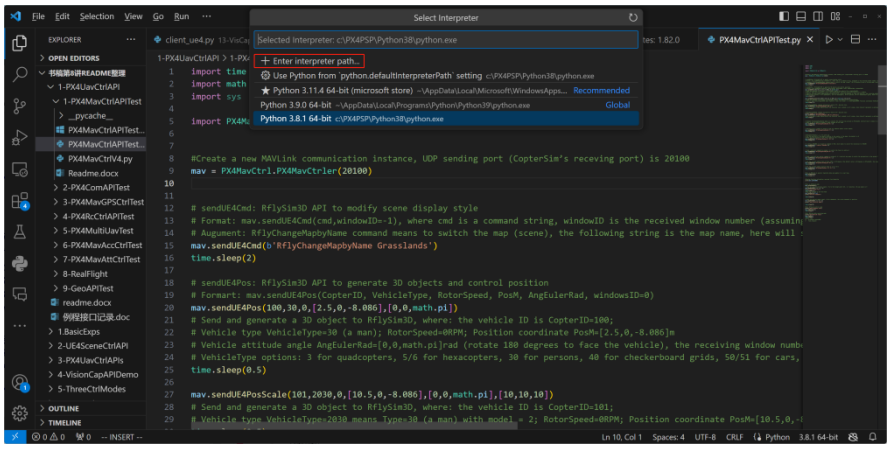
2.1. Windows 开发环境

2.1.1. 平台自带 VS Code 开发环境

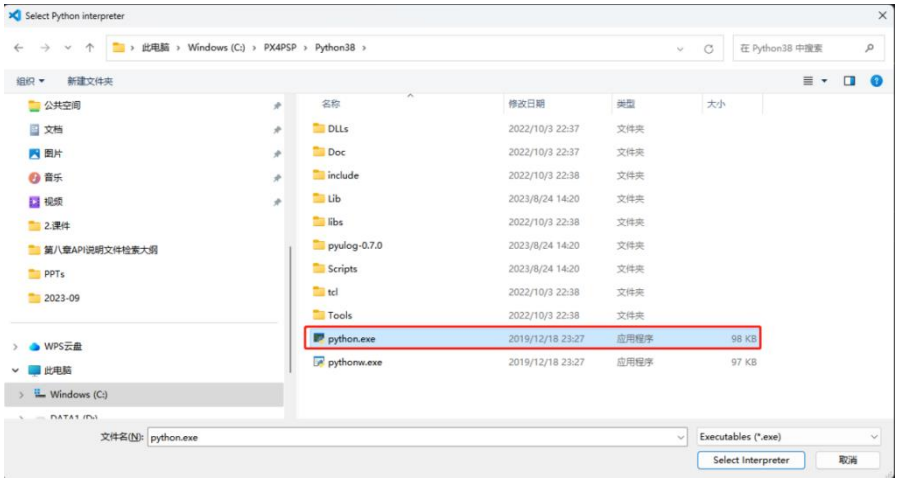
在运行了一键平台安装脚本后，会安装好 RflySim 平台接口所依赖的 python 环境，默认路径为 C:\PX4PSP\Python38\python.exe。安装好 VS Code 软件后，运行一个软件在环例子，在 VS Code 中将此环境进行配置，然后就可以使用 RflySim 平台的接口。配置如下：

1）点击环境路径键，然后点击 Enter interpreter path...，然后点击 Find：





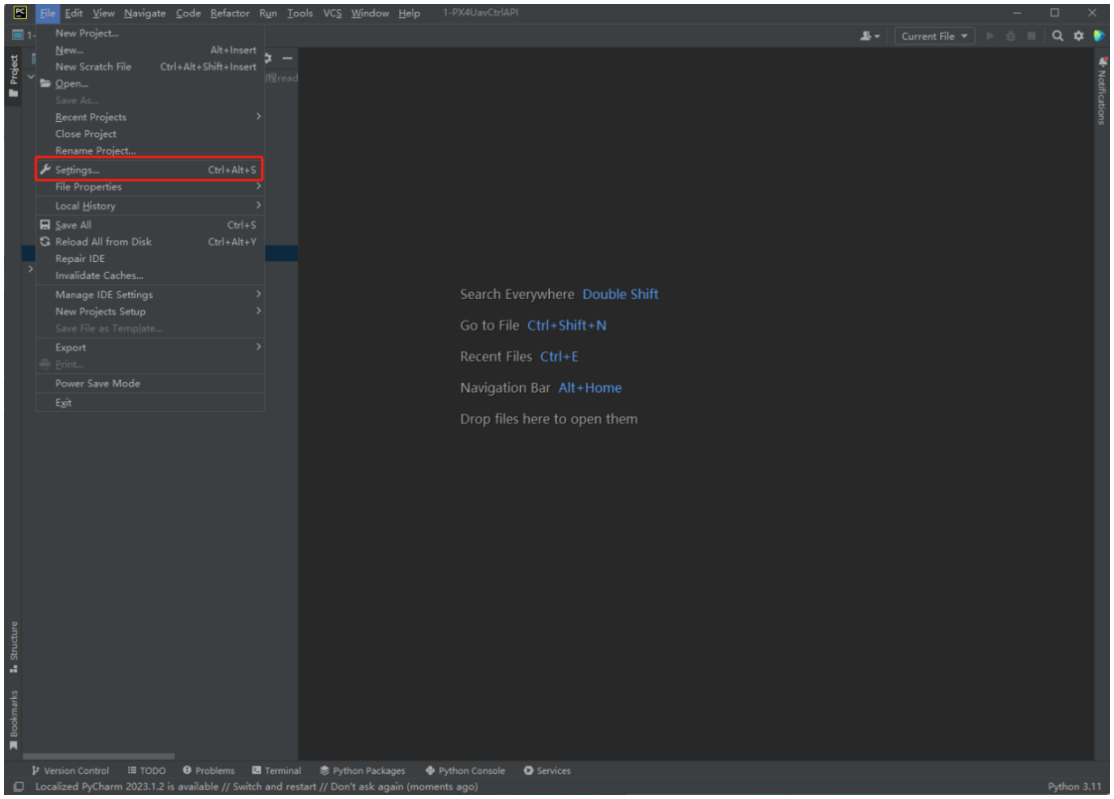
2）选择相应 python 环境：

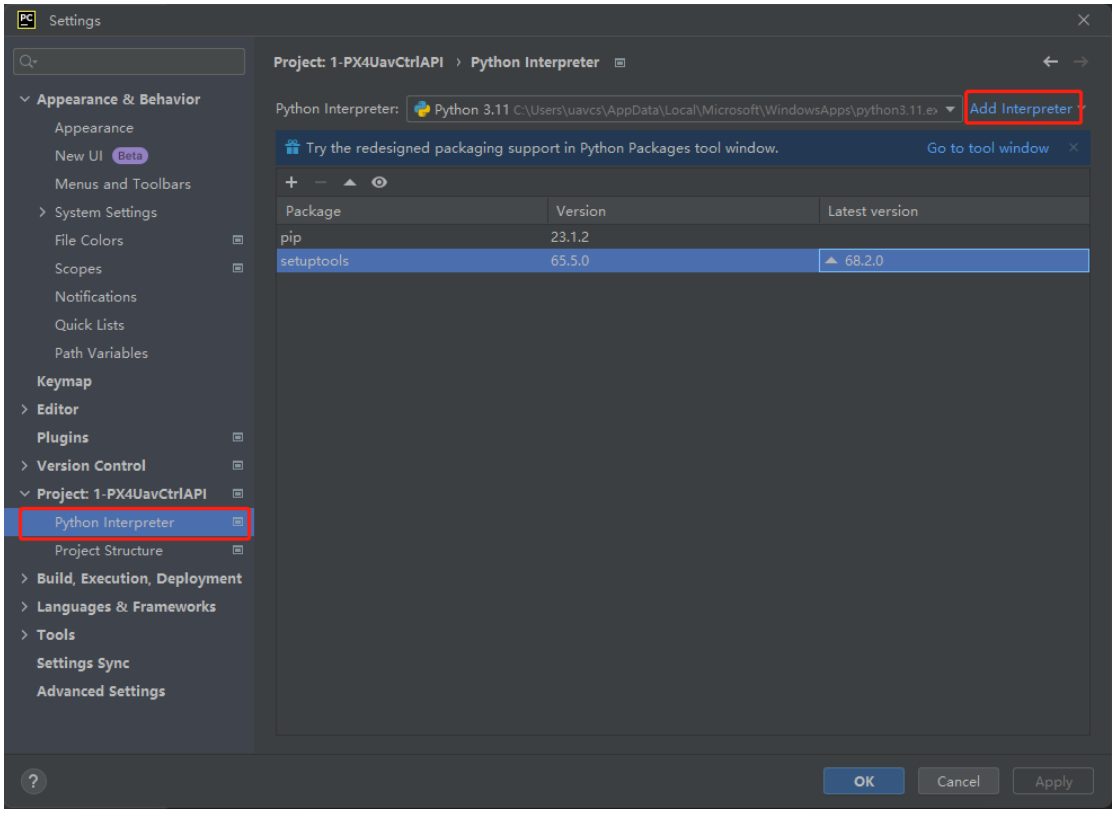


2.1.2. Pycharm 开发环境

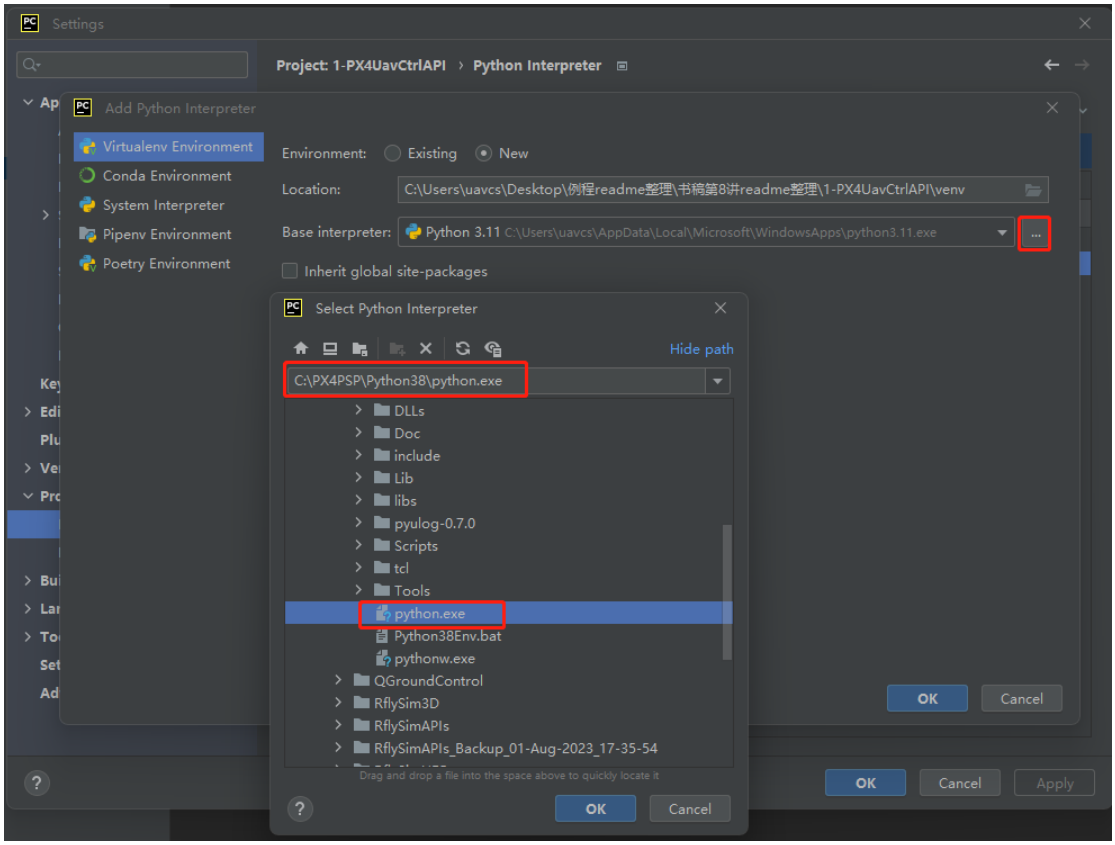
在 Pycharm 下进行 RflySim 平台的使用，一样需要进行 python 环境的配置。配置环境如下：

1)打开 pycharm 点 file--settings，然后点击 profect 下的 Python Interpeter，并点击 Add Interpreter。





2)选择平台安装时的 python 路径。



2.2. Ubuntu 虚拟机开发环境

在使用 Ubuntu 虚拟机前，首先需要下载虚拟机 VMware 软件，然后将 Ubuntu 虚拟机镜像在 VMware 软件上运行即可。虚拟机与主机网络通信推荐使用 NAT 模式，这就需要主机连接了外部路由器，这样可以给虚拟机里面分配物理 IP 地址，这样虚拟机也可访问外部网。如果没有外部路由时，主机使用专用网络连接连接的方式连接虚拟机，需要注意的是，有些电脑使用专用网络通信时，会阻止部分 CopterSim 与虚拟机的通信，当然专用网络通信，极少数情况下会出现这样的问题。

2.3. Ubuntu 开发环境

RflySim 平台视觉算法开发在 Ubuntu 开发环境上是为了 RflySim 平台与 Linux 系统环境进行一个良好的交互操作。进行 RflySim 平台的 Windows 系统与 Linux 系统的联合开发。也是为了和 Linux 系统下的 ROS 和 ROS2 环境以及 mavros 进行紧密联系。

如果需要使用 GPU，则还需要搭建 cuda 库，以及 tensorRT 加速。

3. 视觉算法开发依赖开发工具简介

RflySim 视觉算法开发依赖工具只要分为两部分：仿真部分以及算法依赖库，仿真部分主要是 RflySim 平台的应用相关如（软硬件在环）以及通信部分（pymavlink, mavros，ros,ros2）；算法部分依赖如公共库（OpenCV，Eigen 等），还有一些基于 AI 目标训练与检测的库（cuda, torch，trensorRT 等等）。

RflySim 视觉架构分为软件在环(SITLRun 通信框架)和硬件在环(HITL+数传通信框架)两部分。软件在环开发只需要 RflySim 平台相关接口以及如下相关依赖开发工具，如 MAVLink、mavros、pymavlink、ROS、ROS2 就可以进行软件在环开发。但是硬件在环开发除了上述软件外，还需要 pixhawk 飞控硬件或者如有需求还需要遥控器。

AI 方面需要 GPU 的 cuda 库，以及对应的 tensorRT 库。

3.1. MAVLink/mavros/pymavlink

MAVLink 是一种轻量级、高效的无人机通信协议，用于无人机、地面站和其他无人机相关系统之间的通信。它提供了一种在实时数据传输和控制方面进行高度可靠和灵活的方

法。MAVLink 支持多种通信介质，包括串口、UDP、TCP 和 CAN 等。它定义了各种消息类型、命令和状态信息，用于获取传感器数据、发送控制指令等。更多学习资料请参考：https://mavlink.io/en/。

mavros 是一个开源的 ROS（Robot Operating System）软件包，用于实现 MAVLink 与 ROS 之间的桥接。它提供了 ROS 节点和服务，使开发者能够通过 ROS 与无人机进行通信和控制。mavros 充当无人机系统和 ROS 之间的中间层，可以使用 ROS 功能包对机器人进行状态监测、任务指派和控制等。它提供了许多方便的功能，如飞行模式转换、航点导航、姿态控制等，以及对各种传感器数据的接收和发布。更多学习资料请参考：http://wiki.ros.org/mavros。

pymavlink 是一个用于 Python 开发的 MAVLink 库。它提供了对 MAVLink 协议的解析和生成功能，使开发者能够以 Python 编程语言与无人机进行通信。pymavlink 可以用于编写接收和发送 MAVLink 消息的 Python 脚本，用于处理传感器数据、发送控制指令等。它提供了对 MAVLink 协议的高级封装，使开发者能够更方便地操作和交互无人机系统。更多学习资料请参考：https://mavlink.io/zh/mavgen_python/。

3.2. ROS/ROS2

ROS（Robot Operating System）和 ROS2（Robot Operating System 2）是非常常见和强大的工具。它们是开源的机器人软件开发平台，为机器人系统提供了一种模块化、可重用和分布式的方式来构建软件。

1、ROS（Robot Operating System）： ROS 是一个灵活且可扩展的软件开发框架，旨在支持机器人系统的开发和运行。它提供了一系列工具、库和约定，用于处理机器人的硬件抽象、设备驱动、库集成、共享消息传递、包管理和可视化等方面。ROS 使用节点之间的发布-订阅模式（Publish-Subscribe）进行通信，允许模块化的软件组件通过消息传递进行交互。这种架构使得开发人员能够将系统功能划分为独立的节点，并以可组合和可重用的方式构建机器人应用。更多学习资料请参考：http://wiki.ros.org/cn。

2、ROS2（Robot Operating System 2）： ROS2 是 ROS 的下一代版本，改进了一些 ROS 中的限制和缺点。它是一个多语言、跨平台、实时性和可扩展性更强的机器人软件开发框架。ROS2 引入了 DDS（Data Distribution Service）作为底层通信协议，支持更高级别的实时通信和安全性。它还提供了更好的实时行为、分布式系统支持、多传感器融合和更好的封装等功能。ROS2 的开发重点是提高性能、可靠性和可扩展性，使其适用于更广泛的机器人应用和规模。更多学习资料请参考：Https://www.ros.org。

3.3. OpenCV

OpenCV（Open Source Computer Vision Library）是一个开源的计算机视觉和机器学习库，被广泛应用于视觉算法开发和图像处理任务中。它提供了丰富的函数和工具，可用于处理图像和视频数据，执行各种计算机视觉任务，如特征提取、对象检测、图像分割以及图像和视频的处理与分析。

广泛的功能：OpenCV 提供了超过 2500 个优化的算法函数，包括图像处理、特征检测、计算机视觉、机器学习、深度学习和图像转换等。它支持多种编程语言，如 C++、Python、Java 和 MATLAB，使开发人员可以根据自己的喜好和需求选择适合的语言进行开发。

跨平台支持：OpenCV 是跨平台的，在多个操作系统上都可以运行，包括 Windows、Linux、macOS、Android 和 iOS。这使得开发者可以在不同环境中使用相同的接口和代码进行开发，方便了跨平台应用的开发和部署。

高效的图像处理和计算：OpenCV 使用优化的算法和底层实现，能够高效地处理图像和计算机视觉操作。它利用硬件加速和并行计算，以提高算法的性能和效率。此外，OpenCV 还提供了功能强大的矩阵和向量操作，简化了图像处理和计算的编程。

数百个图像处理函数和算法：OpenCV 提供了丰富的图像处理函数和算法，包括滤波、边缘检测、形态学操作、直方图处理、颜色空间转换、图像修复和增强等。这些函数可以用于处理和操作图像的不同方面，使开发者能够轻松实现各种图像处理任务。

机器学习和深度学习支持：OpenCV 还提供了用于机器学习和深度学习的模块和函数。它支持常见的机器学习算法和技术，如支持向量机（SVM）、k 最近邻（KNN）、随机森林（Random Forest）等。此外，OpenCV 还集成了深度学习框架，如 TensorFlow 和 PyTorch，以实现深度学习模型的构建和推理。更多学习资料请参考：https://opencv.org/

4. RflySim 平台视觉配置步骤

4.1. bat 脚本配置方法（针对视觉的配置）

4.1.1. bat 脚本模板选择（根据需求选择已有脚本模板）

4.1.1.1. SITLRun.bat 软件在环仿真基础脚本

4.1.1.2. SITLPos.bat 初始化位置控制软件在环脚本

4.1.1.3. SITLPosStr.bat 字符串的形式初始化位置控制软件在环脚本

4.1.1.4. HITLRun.bat 硬件在环仿真基础脚本

4.1.1.5. HITLPos.bat 初始化位置控制硬件在环脚本

4.1.1.6. HITLPosStr.bat 字符串的形式初始化位置控制硬件在环脚本

4.1.1.7. HITLPosSysID.bat 配置初始位置序列硬件在环脚本

4.1.1.8. HITLPosSysIDStr.bat 通过 SysID 确定 CopterID 取值，且支持配置初始位置序列

4.1.2. START_INDEX（CopterSim 中 CopterID 的起始编号）

起始飞机序号，本脚本生成的飞机的 CopterID，以此 START_INDEX 为初始值，依次递增 1。此选项对于多台计算机的模拟非常有用。

4.1.3. UDP_START_PORT（CopterSim 中 MavLink UDP 通信的端口号）

设置 SIMULINK/OFFBOARD API 的起始 UDP 端口，这个选项不应该被修改为集群模拟，接收外部控制数据的 UDP 通信接口，与 CopterID 对应会自动加 2 为 20100+CopterID*2，这里通常不需要修改，仅在电脑端口被占用时才修改。

4.1.4. SimMode （硬件/软件仿真模式选择）

设置 CopterSim 的仿真模式，可以用数字或名字标识，设置为 0 或 PX4_HITL，表示硬件在环仿真，设置为 2 或 PX4_SITL_RFLY，表示软件在环仿真。

4.1.5. UE4_MAP （指定 RflySim3D 中场景地图）

设置地图，在 CopterSim 上使用地图的索引或名称，通过指定场景地图名进行对 RflySim3D 场景的控制。使用 LowGPU 场景，来保证低配电脑能运行平台。

4.1.6. IS_BROADCAST（选择 CopterSim MavLink 消息通信方式）

该变量用于设置是否广播到其他电脑，如果 IS_BROADCAST=0 则为只进行本地传输。IS_BROADCAST=1 则为局域网内广播传输。或者使用 IP 地址来提高速度。注意：在填写 IP 地址模式时，IS_BROADCAST=0 相当于 IS_BROADCAST=127.0.0.1，IS_BROADCAST=1 相当于 IS_BROADCAST=255.255.255.255。你也可以用 IP 地址序列用英文“，”或“；”分隔开指定发送。例如：127.0.0.1,192.168.1.4,192.168.1.5， 注意这个影响的只是 CopterSim 通信，并不影响 RflySim3D 通信，换句话说，不管 IS_BROADCAST 是什么值，不影响从 RflySim3D 里面获取传感器数据

4.1.7. UDPSIMMODE （MavLink 消息通信 UDP 仿真模式设置）

UDPSIMMODE 值选择	说明
0	表示使用自定义 UDP_FULL 通信模式（完整模式，数据最全，但传输数据量较大）
1	表示使用自定义 UDP_Simple 通信模式（输入输出精简了，UDP 传输的结构体也精简，因此没有 UDP_FULL 模式中的其他信息，此外，MAVLink 通信针对多电脑视觉硬件在环仿真优化
2	Mavlink_Full 模式（直接使用 MAVLink 数据流进行 Python 与 PX4 的通信（经过 CopterSim 中转），数据量大、占用带宽高，但是和真机更贴近且功能完善，在大规模集群时易阻塞宽带）
3	Mavlink_Simple 模式（直接使用 MAVLink 数据流进行 Python 与 PX4 的通信（经过 CopterSim 中转），但数据量上得到了精简）
4	对应 Mavlink_NoSend 模式，CopterSim 不会向外发送 MAVLink 数据，此模式需要配合硬件在环仿真+数传串口通信，通过有线方式传输 MAVLink，此模式局域网内数据量最小，适合分布式视觉硬件在环仿真，无人机数量不限制。
5	无 GPS 定位的 mavlink_full 模式

4.1.8. TOTOAL_COPTER （飞机的总数量）

无人机总数，自动排列位置。

4.1.9. VEHICLE_INTERVAL （初始化飞机位置间隔）

设置两架无人机间的间隔距离，单位为米。

4.1.10. 飞控初始化位姿设置(Pos,Yaw，以及分布式多机设置)

设置无人机在 RflySim3D 中的原点 x，y 坐标位置单位是米，以及偏角单位是度。也可以将坐标位置以及偏角分别以列表的形式给出，用英文“，”分隔开，分别设置多架无人机的位姿

4.1.11. IsSysID（硬件在环特有参数，决定 CopterID 是否使用飞控的 sys_id）

如果进行硬件在环实验，CopterSim 中将会使用飞控的 ID 号作为 CopterID，进行仿真。

4.1.12. 更多 bat 脚本模板说明（如独立设置每架飞机的位姿等）

可以在脚本中对多架无人机的位姿进行设置，如设置 PosXStr、PosYStr、YawStr 的用“，”分隔的列表。如 SET PosXStr=1.1,2.2，SET PosYStr=1,2，SET YawStr=0,0 就可以设置两架无人机的位姿。

4.2. PX4 软件在环仿真步骤

4.2.1. Bat 脚本设置

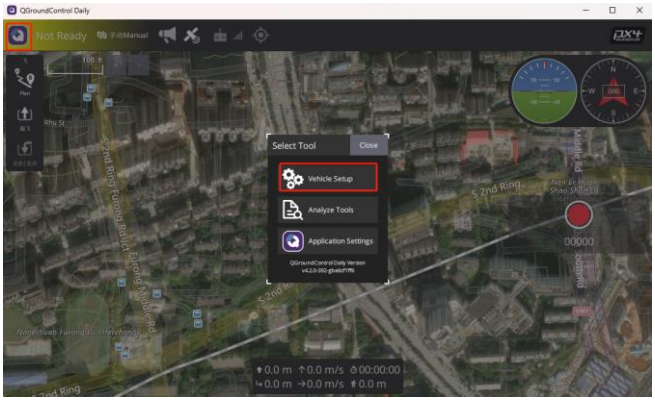
请在 SITLRun.bat 软件在环仿真脚本基础上修改，软件在环仿真首先设置仿真模式 SimMode=2 或 SimMode=PX4_SITL_RFLY。然后设置仿真机架为相应机架，例如四旋翼 set PX4SitlFrame=iris。并且要设置成 SET IS_BROADCAST=1。

4.3. PX4 硬件在环仿真步骤（飞控需要还原官方固件）

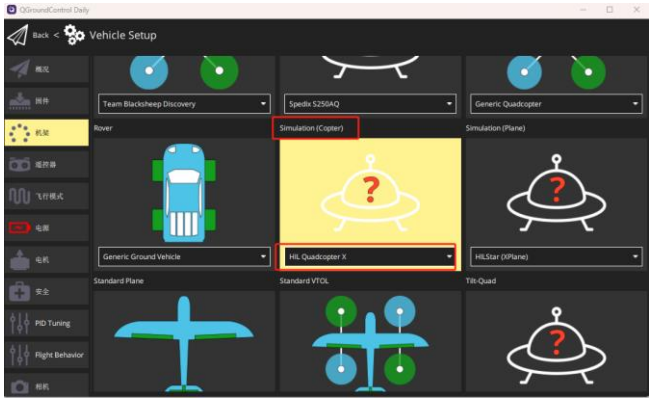
请在 HITLRun.bat 硬件在环仿真脚本基础上修改，硬件在环仿真首先设置仿真模式 SimMode=0 或 SimMode=PX4_HITL。设置 SET /a IsSysID=1 开启通过 SysID 自动计算 CopterID 的功能。

然后将飞控连接电脑上，打开 QGC 地面站进行机架设置，如下图所示：

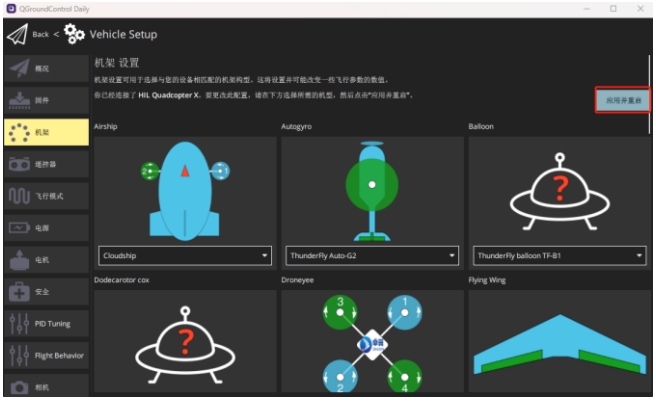
1、选择载具设置：



2、选择机架设置、然后选择 HIL



3、选择完毕后，点击应用并重启，机架设置才会成功。



4.4. PX4 软件在环+虚拟机在环

4.4.1. Bat 脚本配置

请在 SITLRun.bat 软件在环仿真脚本基础上修改，软件在环仿真首先设置仿真模式 SimMode=2 或 SimMode=PX4_SITL_RFLY。设置仿真机架为相应机架，例如四旋翼 set PX4SitlFrame=iris。并且要设置成 SET IS_BROADCAST=1 虚拟机相关配置。

创建一个或多个无人机时在 python 程序中进行指定端口号和 ip 地址，PX4MavCtrler(port,ip)例如单架无人机为 PX4MavCtrler(20100,'127.0.0.1')，多架无人机为 PX4MavCtrler(20100+ID*2,'127.0.0.1')。其中 ip 是 127.0.0.1 为本机通信，ip 是 255.255.255.255 为本地局域网通信。也可指定固定 ip 地址。

4.5. PX4 软件在环+NX 硬件在环

4.5.1. Bat 脚本配置

请在 SITLRun.bat 软件在环仿真脚本基础上修改，软件在环仿真首先设置仿真模式 SimMode=2 或 SimMode=PX4_SITL_RFLY。然后设置仿真机架为相应机架，例如四旋翼 set PX4SitlFrame=iris。并且要设置成 SET IS_BROADCAST=1。

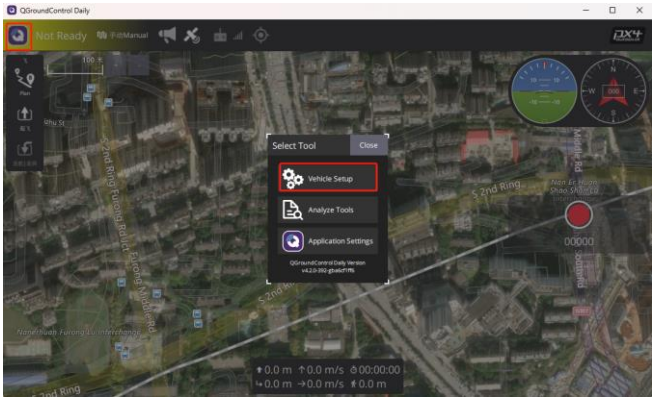
4.5.2. NX 端配置

创建一个或多个无人机时在 python 程序中进行指定端口号和 ip 地址，PX4MavCtrler(port,ip)例如单架无人机为 PX4MavCtrler(20100,'127.0.0.1')，多架无人机为 PX4MavCtrler(20100+ID*2,'127.0.0.1')。其中 ip 是 127.0.0.1 为本机通信，ip 是 255.255.255.255 为本地局域网通信。也可指定固定 ip 地址。

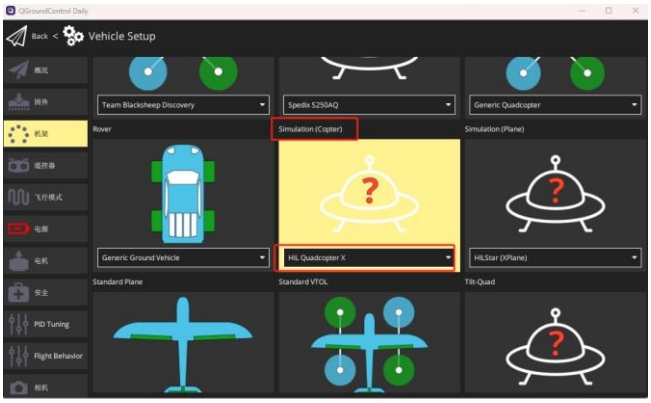
4.6. PX4 硬件在环+虚拟机在环

首先将飞控连接电脑上，打开 QGC 地面站进行机架设置，如下图所示：

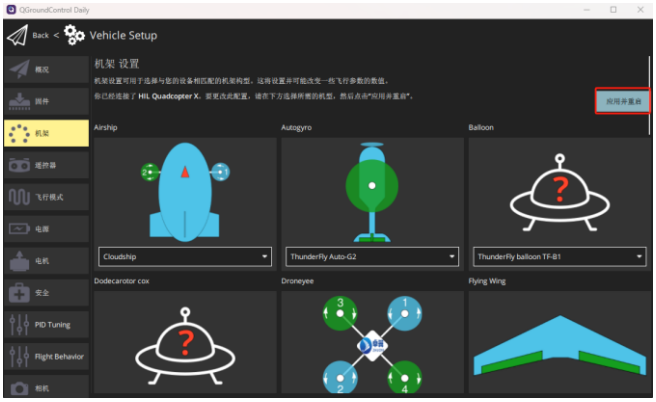
1、选择载具设置：



4、选择机架设置、然后选择 HIL



5、选择完毕后，点击应用并重启，机架设置才会成功。



4.6.1. Bat 脚本配置

请在 HITLRun.bat 硬件在环仿真脚本基础上修改，硬件在环仿真首先设置仿真模式 SimMode=0 或 SimMode=PX4_HITL。设置 SET /a IsSysID=1 开启通过 SysID 自动计算 CopterID 的功能。

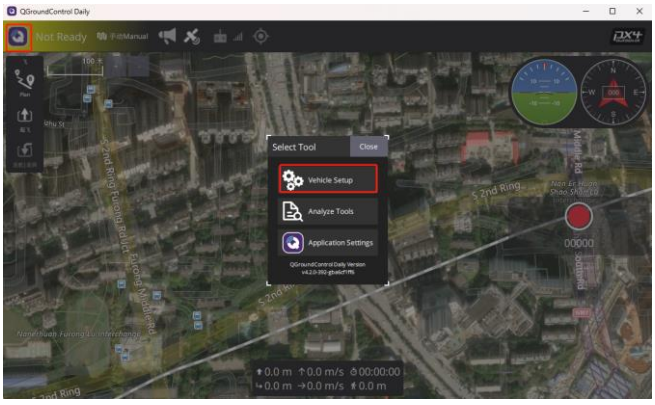
4.6.2. 虚拟机配置

识别出硬件飞控在电脑中的端口号，通过在 python 程序中进行指定端口号或者端口号加波特率，例如 PX4MavCtrler('COM3')或例如 PX4MavCtrler('COM3:57600')。

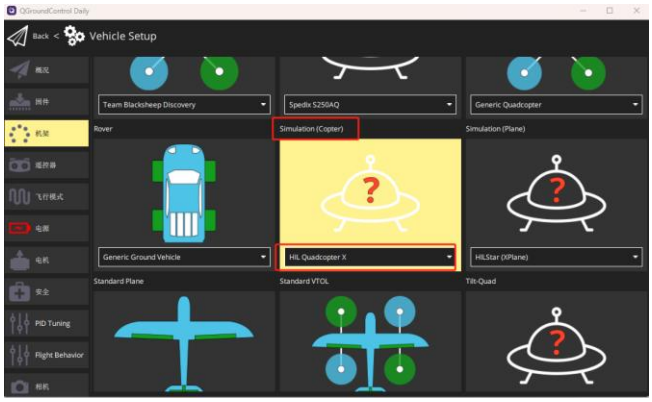
4.7. PX4 硬件在环+NX 硬件在环

首先将飞控连接电脑上，打开 QGC 地面站进行机架设置，如下图所示：

1、选择载具设置：



2、选择机架设置、然后选择 HIL



3、选择完毕后，点击应用并重启，机架设置才会成功。

4.7.1. Bat 脚本配置

请在 HITLRun.bat 硬件在环仿真脚本基础上修改，硬件在环仿真首先设置仿真模式 SimMode=0 或 SimMode=PX4_HITL。设置 SET /a IsSysID=1 开启通过 SysID 自动计算 CopterID 的功能。

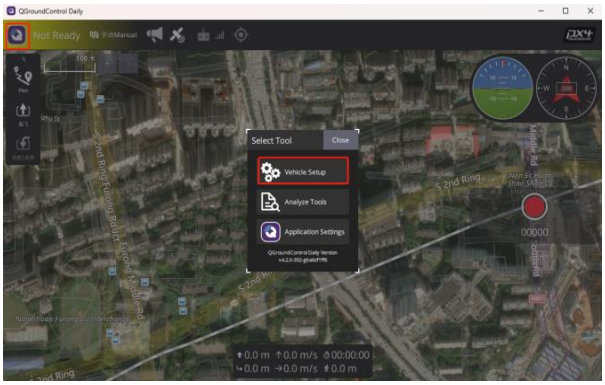
4.7.2. NX 配置

识别出硬件飞控在 NX 内 Ubuntu 系统中的端口号例如/dev/ttyACM0，然后在终端中输入命令 “chmod +x /dev/ttyACM0”将该端口进行赋权执行，其他端口也是相应的操作。然后通过在 python 程序中进行指定端口号或者端口号加波特率，例如 PX4MavCtrler(‘/dev/ttyACM0’)或例如 PX4MavCtrler(‘/dev/ttyACM0:57600’)。

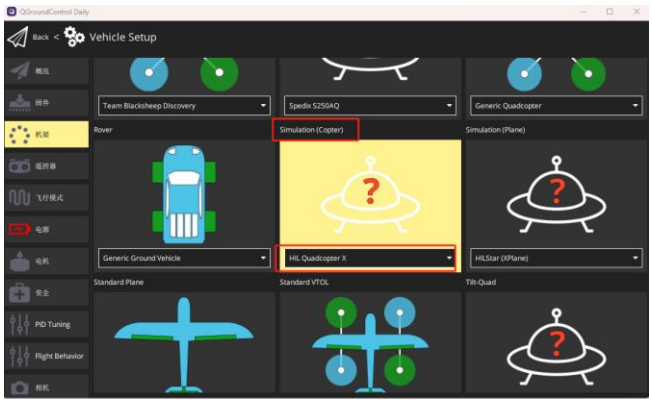
4.8. 多 PX4+多 NX 硬件在环

首先将飞控连接电脑上，打开 QGC 地面站进行机架设置，如下图所示：

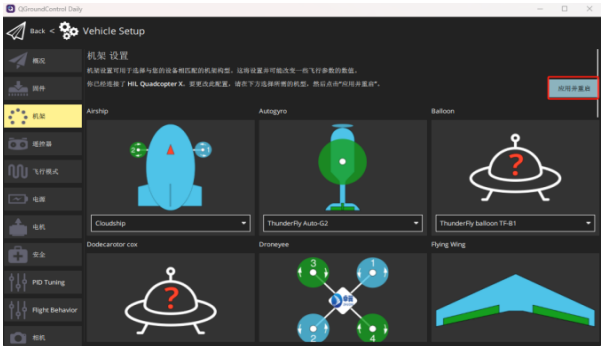
1、选择载具设置：



2、选择机架设置、然后选择 HIL



3、选择完毕后，点击应用并重启，机架设置才会成功。

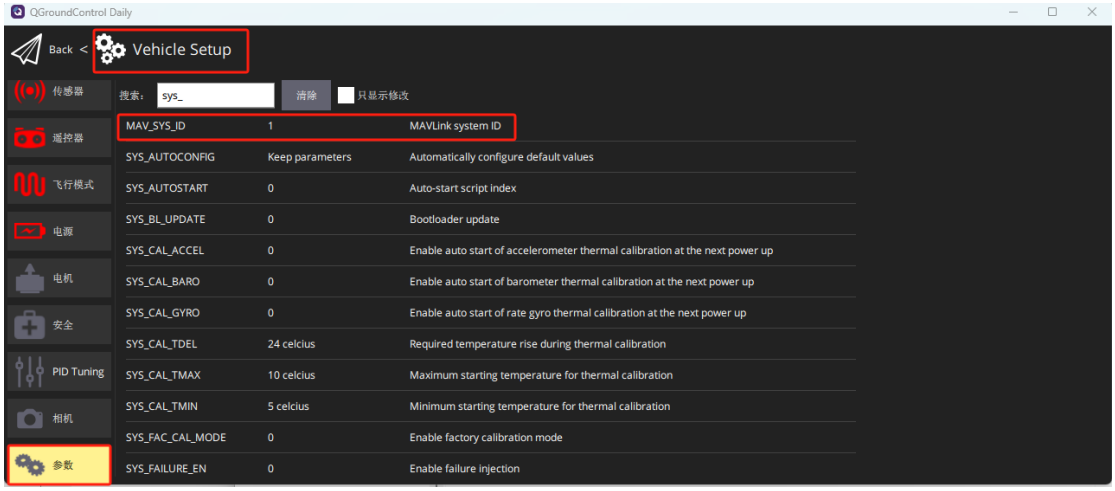


4.8.1. Bat 脚本配置

请在 HITLRun.bat 硬件在环仿真脚本基础上修改，硬件在环仿真首先设置仿真模式 SimMode=0 或 SimMode=PX4_HITL。设置 SET /a IsSysID=1 开启通过 SysID 自动计算 CopterID 的功能。可以进行各个无人机的初始位姿的设置。

4.8.2. 飞控参数设置

将各个硬件飞控通过 USB 线与 NX 连接打开 QGroundControl 软件进行设置 ID 号。这是为了 CopterSimID 与飞控 ID 保持一致。设置如下：



4.8.3. NX 配置

识别出硬件飞控在 NX 内 Ubuntu 系统中的端口号例如/dev/ttyACM0，然后在终端中输入命令 “chmod 777 /dev/ttyACM0”将该端口进行赋权读写权，其他端口也是相应的操作。然后通过在 python 程序中进行指定端口号或者端口号加波特率，例如 PX4MavCtrler(‘/dev/ttyACM0’)或例如 PX4MavCtrler(‘/dev/ttyACM0:57600’)。

5. 视觉相关通信端口介绍

5.1. 视觉接口与 CopterSim，RflySim 进行 UDP 相互通信端口说明

5.1.1. 端口 20010（默认单个 RflySim3D 窗口，视觉接口与 RflySim3D 通信请求的端口，如碰撞检测，目标属性，相机属性等）

相当与指定端口与不同的 RflySim3D 进行通信，20010~20029 这 20 个端口是一一对应的关系，例如 RflySim3D-0 就是指的它在监听 20010，RflySim3D-3 就是指的它在监听 20013

5.1.2. 端口 20100（CopterSim 接受重新启动某架飞机的仿真端口）

CopterSim 对当前飞机给外部提供的 mavlink 通信端口，当然这个值可以是别的，关键是在启动 RflySim 平台的 bat 中的配置。关于 bat 脚本配置参考 bat 脚本配置方法（针对视觉的配置）

5.1.3. 端口 30100（CopterSim 单架飞机默认接受外部请求 IMU 数据的端口）

通过该端口进行各架飞机进行 IMU 数据的请求数据。

5.1.4. 端口 31000 （视觉接口配置当前系统默认接受 IMU 数据的端口）

例如可以通过该端口调用 sendImuReqClient(self, copterID=1, IP="", port=31000, freq=200)接口通过发送 SensorReqCopterSim 实例来请求 Imu 数据。或者调用 sendImuReqServe(self, copterID=1, port=31000)接口创建一个线程，在线程中接收 CopterSim 回传的 Imu 数据，并不阻塞主线程。也可以通过调用 sendImuReqCopterSim(self, copterID=1, IP="", port=31000, freq=200)接口通过发送 SensorReqCopterSim 实例来请求 Imu 数据，然后开启一个线程来监听和处理 CopterSim 回传的 Imu 数据。

5.1.5. 端口 20005 （视觉接口向 CopterSim 请求飞机起飞的时间戳）

该端口主要存储了终端电脑和仿真的时间戳，并通过心跳来检测终端电脑和仿真程序连接是否正常。用于调用 StartTimeStmplisten(self, cpID=1)接口，绑定本机的 20005 端口，在子线程中接收 CopterSim 回传的心跳和时间戳数据，确保终端和 CopterSim 的连接正常，并不阻塞主线程。其中 cpID 表示 CopterSim 的 ID。

5.1.6. 端口 20006（监听来自 RflySim3D 的数据）

该端口是当 RflySim3D 开启数据回传模式时（使用“RflyReqVehicleData 1”命令），会发送出来的结构体。里面主要是与碰撞相关的数据，会被发往组播 ip“224.0.0.10：20006”。也用于在 initUE4MsgRec(self) 接口初始化 self.udp_socketUE4，并且启动一个线程 t4(self.UE4MsgRecLoop)开始监听 224.0.0.10：20006，以及自身的 20006 端口。还可用于在 UE4MsgRecLoop(self)接口监听 224.0.0.10：20006，以及自身的 20006 端口的处理函数，用于处理 RflySim3D 或 CopterSim 返回的消息。

5.1.7. 通信相关数据结构说明

5.1.7.1. RflyTimeStmp (接收到来自 CopterSim 的时间信息）

5.1.7.2. VisionSensorReq（从 Config.json 文件里读取结构体向 RflySim3D 请求传感器配置）

5.1.7.3. imuDataCopter（接受来自 CopterSim 的 IMU 数据结构体）

5.1.7.4. SensorReqCopterSim （向 CopterSim 请求相关数据，如：IMU）

5.1.7.5. reqVeCrashData （监听 RflySim3D 被飞机碰撞到物体属性数据）

5.1.7.6. CoptReqData（向 RflySim3D 请求获取指定飞机的属性）

5.1.7.7. ObjReqData（向 RflySim3D 请求获取指定物体的属性）

5.1.7.8. CameraData（向 RflySim3D 请求获取指定相机的属性）

RflySim 平台的视觉相关的通信分为 Python 程序与 CopterSim 通信、Python 程序与 Rfl

ySim3D 通信、以及 Python 程序与其他 Ubuntu 里 ROS 节点通信。

- fifo:用 python 列表抽象先进先出的队列，并实现 write 和 read 方法，达到类似操作文件的效果。

  1.def write(self, data):

  一、参数解释：

  1) data:待入队项

  二、函数解释：

  这个方法将 data 插入队列头部。

  2.def read(self):

  一、参数解释：

  二、函数解释：

  这个方法将队列末尾的元素删除并返回，如果队列为空，则会抛出 IndexError 异常。

- PX4SILIntFloat:该类封装了输出到 CopterSim DLL 模型的 SILInts 和 SILFloats 数据。

  一. __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]
```

  这个类有两个构造函数，其中：

  1. checksum：校验位
  2. CopterID：消息对应的 Copter ID
  3. inSILInts：输出到 CopterSim DLL 模型的 SILInts 数据
  4. inSILFLoats：输出到 CopterSim DLL 模型的 SILFoats 数据

- reqVeCrashData：这个类是由 RflySim3D 发送的结构体，是当 RflySim3D 开启数据回传模式时（使用“RflyReqVehicleData 1”命令），会发送出来的结构体。里面主要是与碰撞相关的数据，会被发往组播 ip“224.0.0.10：20006”。

  一.__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：表示该结构体是哪个 Copter 的数据。
2. vehicleType：表示该 Copter 的样式
3. CrashType：表示碰撞物体类型，-2 表示地面，-1 表示场景静态物体，0 表示无碰撞，1 以上表示被碰飞机的 ID 号
4. runnedTime：当前飞机的时间戳
5. VelE：当前飞机的速度（米/秒）
6. PosE：当前飞机的位置（北东地，单位米）
7. CrashPos：碰撞点的坐标（北东地，单位米）
8. targetPos：被碰物体的中心坐标（北东地，单位米）
9. AngEuler：当前飞机的欧拉角（Roll，Pitch，Yaw，弧度）
10. MotorRPMS：当前飞机的电机转速
11. Ray：飞机的前后左右上下扫描线
12. CrashedName：被碰物体的名字

- RflyTimeStmp：这个类封装了发送给指定远端电脑端口 20005 的消息，主要存储了终端电脑和仿真的时间戳，并通过心跳来检测终端电脑和仿真程序连接是否正常。

一. __init__

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

这是该类的构造函数，其中：

1. checksum：是数据的校验位，用于检验数据传输过程中是否发生了异常。
2. copterID：表示该消息对应的 copter ID。
3. SysStartTime：是 Windows 下的开始仿真的时间戳，单位毫秒，采用格林尼治标准起点。
4. SysCurrentTime：Windows 下的当前时间戳，单位毫秒，采用格林尼治标准起点。
5. HeartCount：心跳包的计数器。

- PX4ExtMsg：这个类封装了 CopterSim 返回的 pixhawk 状态数据。

一.__init__

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

这是该类的构造函数，其中：

1. checksum：数据校验。
2. CopterID：消息对应的 Copter ID。
3. runnedTime：时间戳。
4. controls：pixhawk 中的状态数据。

- CoptReqData:该类封装了由 RflySim3D 返回的飞机数据，PX4MavCtrler.reqCamCoptObj()函数调用 RflySim3D 的“RflyReqObjData”命令后，RflySim3D 根据该命令的参数，发送的某个 Copter 的信息（向 RflySim3D 发送命令时附带了该 Copter 的 ID）。

一.__init__

```
def __init__(self):
      self.checksum = 0 # 1234567891 作为校验
      self.CopterID = 0 #飞机 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
```

这是该类的构造函数，其中：

1. checksum：是数据的校验位，用于检验数据传输过程中是否发生了异常。
2. CopterID：表示该消息对应的 copter ID。
3. PosUE：表示该 Copter 的位置（米，北东地）
4. angEuler：表示该 Copter 的角度（弧度，Roll、Pitch、Yaw）
5. boxOrigin：包围盒的几何中心（米，北东地）
6. BoxExtent：包围盒的边长的一半（米，X 轴(forward)、Y 轴(right)、Z 轴(up)）
7. Timestmp：时间戳
8. hasUpdate：这个值不是 RflySim3D 发过来的

- ObjReqData：该类封装了由 RflySim3D 返回的数据，PX4MavCtrler.reqCamCoptObj()函数调用 RflySim3D 的“RflyReqObjData”命令后，RflySim3D 根据该命令的参数，发送的某个物体的信息（向 RflySim3D 发送命令时，附带了该物体的 Name）。

一. __init__

```
def __init__(self):
      self.checksum = 0 # 1234567891 作为校验
      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
```

这是该类的构造函数，其中：

1. checksum：是数据的校验位，用于检验数据传输过程中是否发生了异常。
2. seqID：恒为 0。（Obj 不是 Copter，它没有 ID 可供返回）
3. PosUE：表示该物体的位置（米，北东地）
4. angEuler：表示该 Obj 的角度（弧度，Roll、Pitch、Yaw）
5. boxOrigin：包围盒的几何中心（米，北东地）
6. BoxExtent：包围盒的边长的一半（米，X 轴(forward)、Y 轴(right)、Z 轴(up)）
7. Timestmp：当前场景已存在时间（秒，场景存在的时间(因为它不是 Copter，没有时间戳)）
8. ObjName：该物体的名字
9. hasUpdate：这个值不是 RflySim3D 发过来的

- CameraData：该类封装了由 RflySim3D 返回的数据，PX4MavCtrler.reqCamCoptObj()函数调用 RflySim3D 的“RflyReqObjData”命令后，RflySim3D 根据该命令的参数，发送的某个相机（视觉传感器）的信息（向 RflySim3D 发送命令时，附带了该相机（视觉传感器）的 SeqID）。

一. __init__

```
def __init__(self):
        self.checksum = 0 #1234567891 作为校验
        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
```

这是该类的构造函数，其中：

1. checksum：是数据的校验位，用于检验数据传输过程中是否发生了异常。
2. seqID：视觉传感器的 ID
3. TypeID：视觉传感器的类型（RPG、深度图等）
4. DataHeight：数据高度(像素)
5. DataWidth：数据宽度(像素)
6. CameraFOV：相机水平视场角（单位度）
7. PosUE：表示该物体的位置（米，北东地）
8. angEuler：表示该 Camera 的角度（弧度，Roll、Pitch、Yaw）
9. Timestmp：时间戳
10. hasUpdate：这个值不是 RflySim3D 发过来的

5.2. Python 程序与 CopterSim 通信接口

5.2.1. PX4SILIntFloat（输出到 CopterSim DLL 模型的 SILInts 和 SILFloats 数据）

一. __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]
```

这个类有两个构造函数，其中：

1. checksum：校验位
2. CopterID：消息对应的 Copter ID
3. inSILInts：输出到 CopterSim DLL 模型的 SILInts 数据

5.2.2. InitTrueDataLoop（启动两个线程，分别接收 CopterSim 的真实数据和 PX4 数据）

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

一、参数解释：

二、函数解释：

启动两个线程，分别接收 CopterSim 的真实数据和 PX4 数据。

5.2.3. InitMavLoop（开启 MAVLink 监听 CopterSim 数据，并实时更新）

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

一、参数解释：

1) UDPMode：UDP 模式

二、函数解释：

开启 MAVLink 监听 CopterSim 数据，并实时更新。其中 UDPModel 值为：

0：对应 UDP_Full 模式，Python 传输完整的 UDP 数据给 CopterSim，传输数据量小；CopterSim 收到数据后，再转换为 Mavlink 后传输给 PX4 飞控；适合中小规模集群（数量小于 10）仿真。

1：对应 UDP_Simple 模式，数据包大小与发送频率比 UDP_Full 模式小；适合大规模集群仿真，无人机数量小于 100。

2：对应 Mavlink_Full 模式（默认模式），Python 直接发送 MAVLink 消息给 CopterSim，再转发给 PX4，数据量较大适合单机控制；适合单机或少量飞机仿真，无人机数量小于 4；

3: 对应 Mavlink_Simple 模式，会屏蔽部分 MAVLink 消息包，并降低数据频率，发送数据量比 MAVLink_Full 小很多，适合多机集群控制；适合小规模集群仿真，无人机数量小于 8。

4: 对应 Mavlink_NoSend 模式，CopterSim 不会向外发送 MAVLink 数据，此模式需要配合硬件在环仿真+数传串口通信，通过有线方式传输 MAVLink，此模式局域网内数据量最小，适合分布式视觉硬件在环仿真，无人机数量不限制。

5.2.4. TimeStmploop（循环监听指定 CopterSim 的时间戳数据）

```
def TimeStmploop(self)
```

一、参数解释：

二、函数解释：

循环监听指定 CopterSim 的时间戳数据。

5.2.5. StartTimeStmplisten（线程循环监听指定 CopterSim 的时间戳数据，并不阻塞主线程）

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

一、参数解释：

1) cpID：copter ID

二、函数解释：

创建线程循环监听指定 CopterSim 的时间戳数据，并不阻塞主线程。

5.2.6. endMavLoop（停止接收 CopterSim 数据，和 stopRun 效果一致）

```
def endMavLoop(self)
```

一、参数解释：

二、函数解释：

停止接收 CopterSim 数据，和 stopRun 效果一致。

5.3. Python 程序与 RflySim3D 通信接口

5.3.1. reqVeCrashData（由 RflySim3D 发送的结构体，主要是与碰撞相关的数据）

一.__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 = ""
```

这个类是由 RflySim3D 发送的结构体，是当 RflySim3D 开启数据回传模式时（使用“RflyReqVehicleData 1”命令），会发送出来的结构体。里面主要是与碰撞相关的数据，

会被发往组播 ip“224.0.0.10：20006”

1. copterID：表示该结构体是哪个 Copter 的数据。
2. vehicleType：表示该 Copter 的样式
3. CrashType：表示碰撞物体类型，-2 表示地面，-1 表示场景静态物体，0 表示无碰撞，1 以上表示被碰飞机的 ID 号
4. runnedTime：当前飞机的时间戳
5. VelE：当前飞机的速度（米/秒）
6. PosE：当前飞机的位置（北东地，单位米）
7. CrashPos：碰撞点的坐标（北东地，单位米）
8. targetPos：被碰物体的中心坐标（北东地，单位米）
9. AngEuler：当前飞机的欧拉角（Roll，Pitch，Yaw，弧度）
10. MotorRPMS：当前飞机的电机转速
11. Ray：飞机的前后左右上下扫描线
12. CrashedName：被碰物体的名字

5.3.2. CoptReqData（由 RflySim3D 返回的飞机数据，发送的某个 Copter 的信息）

一.__init__

```
def __init__(self):
      self.checksum = 0 # 1234567891 作为校验
      self.CopterID = 0 #飞机 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
```

由 RflySim3D 返回的飞机数据，PX4MavCtrler.reqCamCoptObj()函数调用 RflySim3D 的“RflyReqObjData”命令后，RflySim3D 根据该命令的参数，发送的某个 Copter 的信息（向 RflySim3D 发送命令时附带了该 Copter 的 ID）。

这是该类的构造函数，其中：

1. checksum：是数据的校验位，用于检验数据传输过程中是否发生了异常。
2. CopterID：表示该消息对应的 copter ID。
3. PosUE：表示该 Copter 的位置（米，北东地）
4. angEuler：表示该 Copter 的角度（弧度，Roll、Pitch、Yaw）
5. boxOrigin：包围盒的几何中心（米，北东地）
6. BoxExtent：包围盒的边长的一半（米，X 轴(forward)、Y 轴(right)、Z 轴(up)）
7. Timestmp：时间戳
8. hasUpdate：如果能正常获得数据就置为 True，外面调用它，为 True 表示可以读取数据，然后置为 False

5.3.3. ObjReqData（由 RflySim3D 返回的数据，发送的某个物体的信息）

一. __init__

```
def __init__(self):
      self.checksum = 0 # 1234567891 作为校验
      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
```

由 RflySim3D 返回的数据，PX4MavCtrler.reqCamCoptObj()函数调用 RflySim3D 的“RflyReqObjData”命令后，RflySim3D 根据该命令的参数，发送的某个物体的信息（向 RflySim3D 发送命令时，附带了该物体的 Name）。

这是该类的构造函数，其中：

1. checksum：是数据的校验位，用于检验数据传输过程中是否发生了异常。
2. seqID：恒为 0。（Obj 不是 Copter，它没有 ID 可供返回）
3. PosUE：表示该物体的位置（米，北东地）
4. angEuler：表示该 Obj 的角度（弧度，Roll、Pitch、Yaw）
5. boxOrigin：包围盒的几何中心（米，北东地）
6. BoxExtent：包围盒的边长的一半（米，X 轴(forward)、Y 轴(right)、Z 轴(up)）
7. Timestmp：当前场景已存在时间（秒，场景存在的时间(因为它不是 Copter，没有时间戳)）
8. ObjName：该物体的名字
9. hasUpdate：如果能正常获得数据就置为 True，外面调用它，为 True 表示可以读取数据，然后置为 False

5.3.4. CameraData（由 RflySim3D 返回的数据，发送的某个相机（视觉传感器）的信息）

一. __init__

```
def __init__(self):
      self.checksum = 0 #1234567891 作为校验
      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
```

由 RflySim3D 返回的数据，PX4MavCtrler.reqCamCoptObj()函数调用 RflySim3D 的“RflyReqObjData”命令后，RflySim3D 根据该命令的参数，发送的某个相机（视觉传感

器）的信息（向 RflySim3D 发送命令时，附带了该相机（视觉传感器）的 SeqID）。

这是该类的构造函数，其中：

1. checksum：是数据的校验位，用于检验数据传输过程中是否发生了异常。
2. seqID：视觉传感器的 ID
3. TypeID：视觉传感器的类型（RPG、深度图等）
4. DataHeight：数据高度(像素)
5. DataWidth：数据宽度(像素)
6. CameraFOV：相机视场角（单位度）
7. PosUE：表示该物体的位置（米，北东地）
8. angEuler：表示该 Camera 的角度（弧度，Roll、Pitch、Yaw）
9. Timestmp：时间戳
10. hasUpdate：如果能正常获得数据就置为 True, 外面调用它，为 True 表示可以读取数据，然后置为 False

5.3.5. UE4MsgRecLoop（用于处理 RflySim3D 或 CopterSim 返回的消息，一共有 6 种消息）

```
def UE4MsgRecLoop(self)
```

一、参数解释：

二、函数解释：

它是监听 224.0.0.10：20006，以及自身的 20006 端口的处理函数，用于处理 RflySim3D 或 CopterSim 返回的消息，一共有 6 种消息：

1) 长度为 12 字节的 CopterSimCrash：

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

由 RflySim3D 返回的碰撞数据，RflySim3D 中按 P 键时 RflySim3D 会开启碰撞检测模式，如果发生了碰撞，会传回这个数据，P+数字可以选择发送模式（0 本地发送，1 局域网发送，2 局域网只碰撞时发送），默认为本地发送。

2) 长度为 120 字节的 PX4SILIntFloat：

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

3) 长度为 160 字节的 reqVeCrashData：

```
struct reqVeCrashData {
    int checksum; //数据包校验码 1234567897
    int copterID; //当前飞机的 ID 号
    int vehicleType; //当前飞机的样式
    int CrashType;//碰撞物体类型，-2 表示地面，-1 表示场景静态物体，0 表示无碰撞，1 以上表示被碰飞机的 ID 号
    double runnedTime; //当前飞机的时间戳
    float VelE[3]; // 当前飞机的速度
```

```
    float PosE[3]; //当前飞机的位置
    float CrashPos[3];//碰撞点的坐标
    float targetPos[3];//被碰物体的中心坐标
    float AngEuler[3]; //当前飞机的欧拉角
    float MotorRPMS[8]; //当前飞机的电机转速
    float ray[6]; //飞机的前后左右上下扫描线
    char CrashedName[20] = {0};//被碰物体的名字
}
```

前面介绍过这个类了，在开启数据回传的情况下，RflySim3D 会为所有 Copter 发送 reqVeCrashData（数据变化时发送，每秒一次）。

4) 长度为 56 字节的 CameraData：

```
struct CameraData { //56
    int checksum = 0;//1234567891
    int SeqID; //相机序号
    int TypeID;//相机类型
    int DataHeight;//像素高
    int DataWidth;//像素宽
    float CameraFOV;//相机视场角
    float PosUE[3]; //相机中心位置
    float angEuler[3];//相机欧拉角
    double timestmp;//时间戳
};
```

RflySim3D 根据 reqCamCoptObj 函数发送的命令，定时返回的该结构体，该程序接收到后会存放在 self.CamDataVect 中。

5) 长度为 64 字节的 CoptReqData：

```
struct CoptReqData { //64
    int checksum = 0; //1234567891 作为校验
    int CopterID;//飞机 ID
    float PosUE[3]; //物体中心位置（人为三维建模时指定，姿态坐标轴，不一定在几何中心）
    float angEuler[3];//物体欧拉角
    float boxOrigin[3];//物体几何中心坐标
    float BoxExtent[3];//物体外框长宽高的一半
    double timestmp;//时间戳
};
```

RflySim3D 根据 reqCamCoptObj 函数发送的命令，定时返回的该结构体，该程序接收到后会存放在 self.CoptDataVect 中。

6) 长度为 96 字节的 ObjReqData：

```
struct ObjReqData { //96
    int checksum = 0; //1234567891 作为校验
    int seqID = 0;
    float PosUE[3]; //物体中心位置（人为三维建模时指定，姿态坐标轴，不一定在几何中心）
    float angEuler[3];//物体欧拉角
    float boxOrigin[3];//物体几何中心坐标
    float BoxExtent[3];//物体外框长宽高的一半
    double timestmp;//时间戳
    char ObjName[32] = { 0 };//碰物体的名字
};
```

RflySim3D 根据 reqCamCoptObj 函数发送的命令，定时返回的该结构体，该程序接收到后会存放在 self.ObjDataVect 中。收到的数据可以使用 getCamCoptObj 函数进行获取。

5.3.6. getCamCoptObj（从 RflySim3D 获得指定的数据，并将监听到的三种存放到了 3 个列表中）

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

一、参数解释：

1) Type：0 表示相机，1 表示飞机，2 表示物体

二、函数解释：

从 RflySim3D 获得指定的数据，UE4MsgRecLoop 函数开启了监听 RflySim3D 的消息，并将监听到的三种存放到了 3 个列表中，此函数正是在这个列表中搜索数据，因此需要先使用 reqCamCoptObj 函数向 RflySim3D 请求相关的数据才行。

5.3.7. reqCamCoptObj（请求场景中物体的数据（并不能创建新的物体，而是获得已存在物体的数据））

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

一、参数解释：

1) type：0 表示相机，1 表示飞机，2 表示物体
2) objName：type 表示相机 seqID；s 表示飞机时，objName 对应 CopterID；表示物体时，objName 对应物体名字
3) windowID：表示想往哪个 RflySim3D 发送消息，默认是 0 号窗口。（不要给所有 RflySim3D 发送该数据，因为返回的值都是一样，没必要额外消耗性能）

二、函数解释：

向 RflySim3D 发送一个数据请求，可以请求场景中物体的数据（并不能创建新的物体，而是获得已存在物体的数据）。可以是视觉传感器、Copter、场景中普通的物体。获得的数据详见 CoptReqData 类、ObjReqData 类、CameraData 类。

5.3.8. sendUE4Cmd（向 RflySim3D 发送一个“字节字符串命令”）

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

一、参数解释：

1) Cmd：发送的命令的“字节字符串”
2) windowsID：影响发送目标端口，正常组播时取-1 即可，如果需要指定电脑上的指定的 RflySim3D 进行接收，则可以根据目标电脑 IP 与 RflySim3D 程序的窗口标题序号来设置。（这个序号与上面提到的 20010~20029 这 20 个端口是一一对应的关系，例如 RflySim3D-0 就是指的它在监听 20010，RflySim3D-3 就是指的它在监听 20013）

二、函数解释：

这个函数是向 RflySim3D 发送一个“命令”，其中 cmd 是一个字节字符串，表示发送给 RflySim3D 的命令，发送出去的是这样的结构体：

```
struct Ue4CMD{
   int checksum;
```

```
    char data[52];
}
```

例如：

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

它发送了一条命令“RflyChangeMapbyName Grasslands”，其作用是要求 RflySim3D 改变地图，改成名为“Grasslands”的地图。

至于有哪些其他命令，以及它们的作用，可以参考“\OneDrive\RflySimDocs\高级版书稿\开发版\第 3 章-三维场景建模与仿真\Demo_Resources_3\e0\e0_2\ RflySim3D 命令接口.docx”，这里将它们列出来并简单介绍：

1. RflyShowTextTime(String txt, float time)\\ 让 UE 显示 txt，持续 time 秒
2. RflyShowText(String txt)\\ 让 UE 显示 txt，持续 5 秒
3. RflyChangeMapbyID(int id)\\ 根据地图的 ID 切换 RflySim3D 场景地图
4. RflyChangeMapbyName(String txt)\\ 根据地图名切换 RflySim3D 场景地图
5. RflyChangeViewKeyCmd(String key, int num) \\ 与在 RflySim3D 中按一个 key + num 效果一致
6. RflyCameraPosAngAdd(float x, float y, float z,float roll,float pitch,float yaw) \\ 给摄像机的位置与角度增加一个偏移值
7. RflyCameraPosAng(float x, float y, float z, float roll, float pitch, float yaw) \\ 设置摄像机的位置与角度(UE 的世界坐标)
8. RflyCameraFovDegrees(float degrees) \\ 设置摄像机的视域体 FOV 角度
9. RflyChange3DModel(int CopterID, int veTypes=0) \\ 修改一个无人机的模型样式
10. RflyChangeVehicleSize(int CopterID, float size=0) \\修改一个无人机的缩放大小
11. RflyMoveVehiclePosAng(int CopterID, int isFitGround, float x, float y, float z, float roll, float pitch, float yaw) \\ 给无人机的位置与角度设置一个偏移值，isFitGround 设置无人机是否适应地面
12. RflySetVehiclePosAng(int CopterID, int isFitGround, float x, float y, float z, float roll, float pitch, float yaw) \\设置无人机的位置与角度
13. RflyScanTerrainH(float xLeftBottom(m), float yLeftBottom(m), float xRightTop(m), float yRightTop(m), float scanHeight(m), float scanInterval(m)) \\ 扫描地形，生成一个 png 的高度图与 txt，CopterSim 程序会需要它才知道 UE 有哪些地形、以及它们的高程
14. RflyCesiumOriPos(double lat, double lon, double Alt) \\ 根据经纬度修改 Cesium 的原点位置
15. RflyClearCapture()\\ 清空抓取的图像
16. RflySetActuatorPWMs(int CopterID, float pwm1, float pwm2, float pwm3, float pwm4, float pwm5, float pwm6, float pwm7, float pwm8); \\传入 8 个值，并触发目标无人机的蓝图的接口函数
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 pwm24);\\传入 16 个值，并触发目标无人机的蓝图接口函数。该函数需要完整版才能有作用
18. RflyReqVehicleData(FString isEnable);如果’isEnable’不为 0，则 RflySim3D 会开始发送所有 Copter 的数据（就是前面介绍的 reqVeCrashData）。
19. RflySetPosScale(float scale);\\全局位置的缩放
20. RflyLoad3DFile(FString FileName);\\加载并执行路径下的 TXT 脚本文件

21. RflyReqObjData(int opFlag, FString objName, FString colorStr);\\请求获取三维场景中物体的数据
22. RflySetIDLabel(int CopterID, FString Text, FString colorStr, float size);\\设置一个 Copter 的头顶 ID 显示内容（默认显示 CopterID）
23. RflySetMsgLabel(int CopterID, FString Text, FString colorStr, float size, float time, int flag);\\设置一个 Copter 头顶的 Message 显示的内容
24. RflyDelVehicles(FString CopterIDList);\\删除一些 Copter（逗号是分隔符）
25. RflyDisableVeMove(FString CopterIDList, int disable);\\拒接接收指定 ID 的 Copter 的信息（逗号是分隔符）
26. 除此之外，还有一些 UE 内置的命令可以使用，例如‘stat fps’可以显示当前帧率，‘t.Maxfps 60’可以设置最大帧率为 60。

5.4. Python 程序与其他 Ubuntu 里 ROS 节点通信

接口文件（VisionCaptureAPI.py）支持在 ROS/ROS2 环境下运行，只需要加入下面的语句：

```
# 启用 ROS 发布模式
import VisionCaptrueApi
VisionCaptureApi.isEnableRosTrans =True
vis = VisionCaptureApi.VisionCaptureApi()
```

使用说明：以上语句仅在 linux 系统下运行，目前不支持 windows 系统下的 ros 环境，运行上述代码后（ROS 的先运行 roscore 确定 rosmaster 才能使用 rosrun 或者直接 python3 xxx.py 指令运行代码），即可发布对应 config.json 的话题，参考**错误!未找到引用源。**

5.5. RflySim 平台与其他平台时间戳对齐应用

5.5.1. RflySim 时间戳，远程系统时间戳与 ROS 时间戳

5.5.1.1. RflySim 时间戳

RflySim 时间是基于其运行平台系统时间戳来赋值的，如 RflySim 运行在 Windows 系统上，那么参考时间戳即是 Windows 系统时间戳，如 RflySim 运行在 linux 系统上，那么参考时间戳。当使用了 CopterSim 加载模型的时候，像图像，点云以及 IMU 输出的时间戳都是基于飞机起飞那一刻开始的相对时间戳。当没使用 CopterSim，仅仅使用 RflySim3D 时，图像时间即时系统时间戳

5.5.1.2. 远程系统时间戳

通常 RflySim 平台运行在 Windows 系统上，而我们需要在 ubuntu 系统上去做算法和分布式部署，那么在各分布式的宿主电脑上都有一个其自己的系统时间戳，尽管各每个系统的时间戳都是按照标准来定义的（从 1970 年 1 月 1 日 0 时 0 分 0 秒算起，经过的秒数），但是在同一个时刻每个系统的时间戳都不会相同。

5.5.1.3. ROS 时间戳

在分布式系统内各个 Node 都应该在同一时间戳下运行，因此 ROS2 为了解决不同系统时间戳对齐，定一个 ROS 时间戳，这个时间戳在各个子系统间参考时间点都一样。因此，同一个时刻，RflySim 时间戳≠远端端系统时间戳，远程端系统时间戳与 ROS 时间戳，仅分布式内只有一个子系统的时候相等，其他情况下是不相等的。

5.5.2. 各系统间时间戳对齐

关于各系统的时间戳定义查看 RflySim 时间戳，远程系统时间戳与 ROS 时间戳，在 RflySim 平台应用到 ROS 分布式内的需求是获得在 ROS 时间戳为参考的前提下，图像以及其他传感器数据生成的时间戳。要实现这一功能，需要在远程端获得 RflySim 平台的开始仿真的时间戳，以及在这个时间戳下，图像及其他传感器的相对时间戳。如果不考虑远端系统时间戳，仅仅考虑 ROS 时间戳与 RflySim 平台时间戳，只要计算一个 ROS 的时间戳与 RflySim 平台时间戳的一个差值就行。假定这个差值为 dt(正负皆有可能)，那么 ROS 时间戳下的时间为 data_ros_time = RflySim_start_time + RflySim_data_stmp + dt ，RflySim_start_time 即是 RflySim 平台开始仿真记录的时间戳，而 RflySim_data_stmp 为传感器数据生成相对 RflySim_start_time 的时间。关于接口文件(VisionCaptureApi.py)实现参考函数**错误!未找到引用源。**以及函数**错误!未找到引用源。**

6. RflySim3D 控制接口 UE4CtrlAPI.py

6.1. 场景控制接口（发送命令控制 RflySim3D）

6.1.1. sendUE4Pos（向 RflySim3D 中给指定 copter_id 的对象设置其位姿，类型等）

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

一、参数解释：

1. copterID：设置的 Copter 的 ID
2. vehicleType：设置的 Copter 的样式（在 xml 中确定）
3. MotorRPMSMean：表示 8 位执行器数据的平均值(8 个执行器的值相同)
4. PosE：表示设置的该 Copter 的位置（米，北东地）
5. AngEuler：表示设置的该 Copter 的欧拉角（弧度，roll,pitch,yaw）
6. windowID：影响发送目标端口，正常组播时取-1 即可，如果需要指定电脑上的指定的 RflySim3D 进行接收，则可以根据目标电脑 IP 与 RflySim3D 程序的窗口标题序号来设置。（这个序号与上面提到的 20010~20029 这 20 个端口是一一对应的关系，例如 RflySim3D-0 就是指的它在监听 20010，RflySim3D-3 就是指的它在监听 20013）

二、函数解释：

向局域网内所有 RflySim3D 发送一个 Copter 的数据，如果不存在该 CopterID 的物体，那么会创建一个这样的物体。其中 vehicleType 表示该 Copter 的样式，PosE 表示该 Copter 的位置（米，北东地），AngEuler 表示该 Copter 的欧拉角（弧度，roll,pitch,yaw），MotorRPMSMean 表示 8 位执行器数据的平均值(8 个执行器的值相同)。

6.1.2. sendUE4Cmd（向 RflySim3D 发送一个“命令”）

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

一、参数解释：

1) Cmd：发送的命令的“字节字符串”
2) windowsID：影响发送目标端口，正常组播时取-1 即可，如果需要指定电脑上的指定的 RflySim3D 进行接收，则可以根据目标电脑 IP 与 RflySim3D 程序的窗口标题序号来设置。（这个序号与上面提到的 20010~20029 这 20 个端口是一一对应的关系，例如 RflySim3D-0 就是指的它在监听 20010，RflySim3D-3 就是指的它在监听 20013）

二、函数解释：

这个函数是向 RflySim3D 发送一个“命令”，其中 cmd 是一个字节字符串，表示发送给 RflySim3D 的命令，发送出去的是这样的结构体：

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

例如：

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

它发送了一条命令“RflyChangeMapbyName Grasslands”，其作用是要求 RflySim3D 改变地图，改成名为“Grasslands”的地图。

至于有哪些其他命令，以及它们的作用，可以参考“\OneDrive\RflySimDocs\高级版书稿\开发版\第 3 章-三维场景建模与仿真\Demo_Resources_3\e0\e0_2\RflySim3D 命令接口.docx”，这里将它们列出来并简单介绍：

1. RflyShowTextTime(String txt, float time)\\ 让 UE 显示 txt，持续 time 秒
2. RflyShowText(String txt)\\ 让 UE 显示 txt，持续 5 秒
3. RflyChangeMapbyID(int id)\\ 根据地图的 ID 切换 RflySim3D 场景地图
4. RflyChangeMapbyName(String txt)\\ 根据地图名切换 RflySim3D 场景地图

5. RflyChangeViewKeyCmd(String key, int num) \\ 与在 RflySim3D 中按一个 key + num 效果一致
6. RflyCameraPosAngAdd(float x, float y, float z,float roll,float pitch,float yaw) \\ 给摄像机的位置与角度增加一个偏移值
7. RflyCameraPosAng(float x, float y, float z, float roll, float pitch, float yaw) \\ 设置摄像机的位置与角度(UE 的世界坐标)
8. RflyCameraFovDegrees(float degrees) \\ 设置摄像机的视域体 FOV 角度
9. RflyChange3DModel(int CopterID, int veTypes=0) \\ 修改一个无人机的模型样式
10. RflyChangeVehicleSize(int CopterID, float size=0) \\修改一个无人机的缩放大小
11. RflyMoveVehiclePosAng(int CopterID, int isFitGround, float x, float y, float z, float roll, float pitch, float yaw) \\ 给无人机的位置与角度设置一个偏移值，isFitGround 设置无人机是否适应地面
12. RflySetVehiclePosAng(int CopterID, int isFitGround, float x, float y, float z, float roll, float pitch, float yaw) \\设置无人机的位置与角度
13. RflyScanTerrainH(float xLeftBottom(m), float yLeftBottom(m), float xRightTop(m), float yRightTop(m), float scanHeight(m), float scanInterval(m)) \\ 扫描地形，生成一个 png 的高度图与 txt，CopterSim 程序会需要它才知道 UE 有哪些地形、以及它们的高程
14. RflyCesiumOriPos(double lat, double lon, double Alt) \\ 根据经纬度修改 Cesium 的原点位置
15. RflyClearCapture()\\ 清空抓取的图像
16. RflySetActuatorPWMs(int CopterID, float pwm1, float pwm2, float pwm3, float pwm4, float pwm5, float pwm6, float pwm7, float pwm8); \\传入 8 个值，并触发目标无人机的蓝图的接口函数
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 pwm24);\\传入 16 个值，并触发目标无人机的蓝图接口函数。该函数需要完整版才能有作用
18. RflyReqVehicleData(FString isEnable);如果’isEnable’不为 0，则 RflySim3D 会开始发送所有 Copter 的数据（就是前面介绍的 reqVeCrashData）。
19. RflySetPosScale(float scale);\\全局位置的缩放
20. RflyLoad3DFile(FString FileName);\\加载并执行路径下的 TXT 脚本文件
21. RflyReqObjData(int opFlag, FString objName, FString colorStr);\\请求获取三维场景中物体的数据
22. RflySetIDLabel(int CopterID, FString Text, FString colorStr, float size);\\设置一个 Copter 的头顶 ID 显示内容（默认显示 CopterID）
23. RflySetMsgLabel(int CopterID, FString Text, FString colorStr, float size, float time, int flag);\\设置一个 Copter 头顶的 Message 显示的内容
24. RflyDelVehicles(FString CopterIDList);\\删除一些 Copter（逗号是分隔符）
25. RflyDisableVeMove(FString CopterIDList, int disable);\\拒接接收指定 ID 的 Copter 的信息（逗号是分隔符）
26. 除此之外，还有一些 UE 内置的命令可以使用，例如‘stat fps’可以显示当前帧率，‘t.Maxfps 60’可以设置最大帧率为 60。

6.2. 数据请求接口（获取所有物体 BoundingBox 等属性）

6.2.1. CoptReqData（由 RflySim3D 返回的飞机数据，发送的某个 Copter 的信息）

一.__init__

```
def __init__(self):
      self.checksum = 0 # 1234567891 作为校验
      self.CopterID = 0 #飞机 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
```

由 RflySim3D 返回的飞机数据，PX4MavCtrler.reqCamCoptObj()函数调用 RflySim3D 的“RflyReqObjData”命令后，RflySim3D 根据该命令的参数，发送的某个 Copter 的信息（向 RflySim3D 发送命令时附带了该 Copter 的 ID）。

这是该类的构造函数，其中：

1. checksum：是数据的校验位，用于检验数据传输过程中是否发生了异常。
2. CopterID：表示该消息对应的 copter ID。
3. PosUE：表示该 Copter 的位置（米，北东地）
4. angEuler：表示该 Copter 的角度（弧度，Roll、Pitch、Yaw）
5. boxOrigin：包围盒的几何中心（米，北东地）
6. BoxExtent：包围盒的边长的一半（米，X 轴(forward)、Y 轴(right)、Z 轴(up)）
7. Timestmp：时间戳
8. hasUpdate：如果能正常获得数据就置为 True，提供外部调用，为 True 表示可以读取数据，然后置为 False

6.2.2. ObjReqData（由 RflySim3D 返回的数据，发送的某个物体的信息）

二. __init__

```
def __init__(self):
      self.checksum = 0 # 1234567891 作为校验
      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
```

该类封装了由 RflySim3D 返回的数据，PX4MavCtrler.reqCamCoptObj()函数调用 RflySim3D 的“RflyReqObjData”命令后，RflySim3D 根据该命令的参数，发送的某个物体的信息（向

RflySim3D 发送命令时，附带了该物体的 Name）。

这是该类的构造函数，其中：

1. checksum：是数据的校验位，用于检验数据传输过程中是否发生了异常。
2. seqID：恒为 0。（Obj 不是 Copter，它没有 ID 可供返回）
3. PosUE：表示该物体的位置（米，北东地）
4. angEuler：表示该 Obj 的角度（弧度，Roll、Pitch、Yaw）
5. boxOrigin：包围盒的几何中心（米，北东地）
6. BoxExtent：包围盒的边长的一半（米，X 轴(forward)、Y 轴(right)、Z 轴(up)）
7. Timestmp：当前场景已存在时间（秒，场景存在的时间(因为它不是 Copter，没有时间戳)）
8. ObjName：该物体的名字或者 ID；
9. hasUpdate：如果能正常获得数据就置为 True，提供外部调用，为 True 表示可以读取数据，然后置为 False

6.2.3. CameraData（由 RflySim3D 返回的数据，发送的某个相机（视觉传感器）的信息）

一. __init__

```
def __init__(self):
        self.checksum = 0 #1234567891 作为校验
        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
```

该类封装了由 RflySim3D 返回的数据，PX4MavCtrler.reqCamCoptObj()函数调用 RflySim3D 的“RflyReqObjData”命令后，RflySim3D 根据该命令的参数，发送的某个相机（视觉传 感器）的信息（向 RflySim3D 发送命令时，附带了该相机（视觉传感器）的 SeqID）。

这是该类的构造函数，其中：

1. checksum：是数据的校验位，用于检验数据传输过程中是否发生了异常。
2. seqID：视觉传感器的 ID
3. TypeID：视觉传感器的类型（RPG、深度图等）
4. DataHeight：数据高度(像素)
5. DataWidth：数据宽度(像素)
6. CameraFOV：相机视场角（单位度）
7. PosUE：表示该物体的位置（米，北东地）
8. angEuler：表示该 Camera 的角度（弧度，Roll、Pitch、Yaw）
9. Timestmp：时间戳
10. hasUpdate：如果能正常获得数据就置为 True，提供外部调用，为 True 表示可以读取数据，然后置为 False

6.2.4. reqCamCoptObj（请求 RflySim3D 场景中）

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

一、参数解释：

1) type：0 表示相机，1 表示飞机，2 表示物体
2) objName：表示飞机时，objName 对应 CopterID；表示物体时，objName 对应物体名字或者创建物体的 id
3) windowID：表示想往哪个 RflySim3D 发送消息，默认是 0 号窗口。（不要给所有 RflySim3D 发送该数据，因为返回的值都是一样，没必要额外消耗性能）

二、函数解释：

向 RflySim3D 发送一个数据请求，可以请求场景中物体的数据（并不能创建新的物体，而是获得已存在物体的数据）。可以是视觉传感器、Copter、场景中普通的物体。获得的数据详见 CoptReqData 类、ObjReqData 类、CameraData 类。

6.2.5. UE4MsgRecLoop（循环监听来自 RflySim3D 的数据）

```
def UE4MsgRecLoop(self)
```

一、参数解释：

二、函数解释：

它是监听 224.0.0.10：20006，以及自身的 20006 端口的处理函数，用于处理 RflySim3D 或 CopterSim 返回的消息，一共有 6 种消息：

1) 长度为 12 字节的 CopterSimCrash：

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

由 RflySim3D 返回的碰撞数据，RflySim3D 中按 P 键时 RflySim3D 会开启碰撞检测模式，如果发生了碰撞，会传回这个数据，P+数字可以选择发送模式（0 本地发送，1 局域网发送，2 局域网只碰撞时发送），默认为本地发送。

2) 长度为 120 字节的 PX4SILIntFloat：

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

3) 长度为 160 字节的 reqVeCrashData：

```
struct reqVeCrashData {
    int checksum; //数据包校验码 1234567897
    int copterID; //当前飞机的 ID 号
    int vehicleType; //当前飞机的样式
    int CrashType;//碰撞物体类型，-2 表示地面，-1 表示场景静态物体，0 表示无碰撞，1 以上表示被碰飞机的 ID 号
    double runnedTime; //当前飞机的时间戳
    float VelE[3]; // 当前飞机的速度
    float PosE[3]; //当前飞机的位置
```

```
    float CrashPos[3];//碰撞点的坐标
    float targetPos[3];//被碰物体的中心坐标
    float AngEuler[3]; //当前飞机的欧拉角
    float MotorRPMS[8]; //当前飞机的电机转速
    float ray[6]; //飞机的前后左右上下扫描线
    char CrashedName[20] = {0};//被碰物体的名字
}
```

前面介绍过这个类了，在开启数据回传的情况下，RflySim3D 会为所有 Copter 发送 reqVeCrashData（数据变化时发送，每秒一次）。

4) 长度为 56 字节的 CameraData：

```
struct CameraData { //56
    int checksum = 0;//1234567891
    int SeqID; //相机序号
    int TypeID;//相机类型
    int DataHeight;//像素高
    int DataWidth;//像素宽
    float CameraFOV;//相机视场角
    float PosUE[3]; //相机中心位置
    float angEuler[3];//相机欧拉角
    double timestmp;//时间戳
};
```

RflySim3D 根据 reqCamCoptObj 函数发送的命令，定时返回的该结构体，该程序接收到后会存放在 self.CamDataVect 中。

5) 长度为 64 字节的 CoptReqData：

```
struct CoptReqData { //64
    int checksum = 0; //1234567891 作为校验
    int CopterID;//飞机 ID
    float PosUE[3]; //物体中心位置（人为三维建模时指定，姿态坐标轴，不一定在几何中心）
    float angEuler[3];//物体欧拉角
    float boxOrigin[3];//物体几何中心坐标
    float BoxExtent[3];//物体外框长宽高的一半
    double timestmp;//时间戳
};
```

RflySim3D 根据 reqCamCoptObj 函数发送的命令，定时返回的该结构体，该程序接收到后会存放在 self.CoptDataVect 中。

6) 长度为 96 字节的 ObjReqData：

```
struct ObjReqData { //96
    int checksum = 0; //1234567891 作为校验
    int seqID = 0;
    float PosUE[3]; //物体中心位置（人为三维建模时指定，姿态坐标轴，不一定在几何中心）
    float angEuler[3];//物体欧拉角
    float boxOrigin[3];//物体几何中心坐标
    float BoxExtent[3];//物体外框长宽高的一半
    double timestmp;//时间戳
    char ObjName[32] = { 0 };//碰物体的名字
};
```

RflySim3D 根据 reqCamCoptObj 函数发送的命令，定时返回的该结构体，该程序接收到后会存放在 self.ObjDataVect 中。收到的数据可以使用 getCamCoptObj 函数进行获取。

6.2.6. getCamCoptObj（从 RflySim3D 获得指定的目标的属性）

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

一、参数解释：

1) Type：0 表示相机，1 表示飞机，2 表示物体

二、函数解释：

从 RflySim3D 获得指定的数据，UE4MsgRecLoop 函数开启了监听 RflySim3D 的消息，并将监听到的三种存放到了 3 个列表中，此函数正是在这个列表中搜索数据，因此需要先使用 reqCamCoptObj 函数向 RflySim3D 请求相关的数据才行。

6.2.7. reqCamCoptObj（请求获得 RflySim3D 中指定物体的属性）

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

一、参数解释：

1) type：0 表示相机，1 表示飞机，2 表示物体
2) objName：type 表示相机 seqID；s 表示飞机时，objName 对应 CopterID；表示物体时，objName 对应物体名字
3) windowID：表示想往哪个 RflySim3D 发送消息，默认是 0 号窗口。（不要给所有 RflySim3D 发送该数据，因为返回的值都是一样，没必要额外消耗性能）

二、函数解释：

向 RflySim3D 发送一个数据请求，可以请求场景中物体的数据（并不能创建新的物体，而是获得已存在物体的数据）。可以是视觉传感器、Copter、场景中普通的物体。获得的数据详见 CoptReqData 类、ObjReqData 类、CameraData 类。

6.3. 碰撞检测之类接口

6.3.1. reqVeCrashData（由 RflySim3D 发送的结构体，里面主要是与碰撞相关的数据）

一.__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 = ""
```

这个类是由 RflySim3D 发送的结构体，是当 RflySim3D 开启数据回传模式时（使用“RflyReqVehicleData 1”命令），会发送出来的结构体。里面主要是与碰撞相关的数据，会被发往组播 ip“224.0.0.10：20006”。

1. copterID：表示该结构体是哪个 Copter 的数据。
2. vehicleType：表示该 Copter 的样式
3. CrashType：表示碰撞物体类型，-2 表示地面，-1 表示场景静态物体，0 表示无碰撞，1 以上表示被碰飞机的 ID 号
4. runnedTime：当前飞机的时间戳
5. VelE：当前飞机的速度（米/秒）
6. PosE：当前飞机的位置（北东地，单位米）
7. CrashPos：碰撞点的坐标（北东地，单位米）
8. targetPos：被碰物体的中心坐标（北东地，单位米）
9. AngEuler：当前飞机的欧拉角（Roll，Pitch，Yaw，弧度）
10. MotorRPMS：当前飞机的电机转速
11. Ray：飞机的前后左右上下扫描线
12. CrashedName：被碰物体的名字

6.3.2. UE4MsgRecLoop（用于处理 RflySim3D 或 CopterSim 返回的消息）

```
def UE4MsgRecLoop(self)
```

一、参数解释：

二、函数解释：

它是监听 224.0.0.10：20006，以及自身的 20006 端口的处理函数，用于处理 RflySim3D 或 CopterSim 返回的消息，一共有 6 种消息：

1) 长度为 12 字节的 CopterSimCrash：

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

由 RflySim3D 返回的碰撞数据，RflySim3D 中按 P 键时 RflySim3D 会开启碰撞检测模式，如果发生了碰撞，会传回这个数据，P+数字可以选择发送模式（0 本地发送，1 局域网发送，2 局域网只碰撞时发送），默认为本地发送。

2) 长度为 120 字节的 PX4SILIntFloat：

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

3) 长度为 160 字节的 reqVeCrashData：

```
struct reqVeCrashData {
    int checksum; //数据包校验码 1234567897
    int copterID; //当前飞机的 ID 号
```

```
        int vehicleType; //当前飞机的样式
        int CrashType;//碰撞物体类型，-2 表示地面，-1 表示场景静态物体，0 表示无碰撞，1 以上表示被碰飞机的
ID 号
        double runnedTime; //当前飞机的时间戳
        float VelE[3]; // 当前飞机的速度
        float PosE[3]; //当前飞机的位置
        float CrashPos[3];//碰撞点的坐标
        float targetPos[3];//被碰物体的中心坐标
        float AngEuler[3]; //当前飞机的欧拉角
        float MotorRPMS[8]; //当前飞机的电机转速
        float ray[6]; //飞机的前后左右上下扫描线
        char CrashedName[20] = {0};//被碰物体的名字
    }
```

前面介绍过这个类了，在开启数据回传的情况下，RflySim3D 会为所有 Copter 发送 reqVeCrashData（数据变化时发送，每秒一次）。

4) 长度为 56 字节的 CameraData：

```
    struct CameraData { //56
        int checksum = 0;//1234567891
        int SeqID; //相机序号
        int TypeID;//相机类型
        int DataHeight;//像素高
        int DataWidth;//像素宽
        float CameraFOV;//相机视场角
        float PosUE[3]; //相机中心位置
        float angEuler[3];//相机欧拉角
        double timestmp;//时间戳
    };
```

RflySim3D 根据 reqCamCoptObj 函数发送的命令，定时返回的该结构体，该程序接收到后会存放在 self.CamDataVect 中。

5) 长度为 64 字节的 CoptReqData：

```
    struct CoptReqData { //64
        int checksum = 0; //1234567891 作为校验
        int CopterID;//飞机 ID
        float PosUE[3]; //物体中心位置（人为三维建模时指定，姿态坐标轴，不一定在几何中心）
        float angEuler[3];//物体欧拉角
        float boxOrigin[3];//物体几何中心坐标
        float BoxExtent[3];//物体外框长宽高的一半
        double timestmp;//时间戳
    };
```

RflySim3D 根据 reqCamCoptObj 函数发送的命令，定时返回的该结构体，该程序接收到后会存放在 self.CoptDataVect 中。

6) 长度为 96 字节的 ObjReqData：

```
    struct ObjReqData { //96
        int checksum = 0; //1234567891 作为校验
        int seqID = 0;
        float PosUE[3]; //物体中心位置（人为三维建模时指定，姿态坐标轴，不一定在几何中心）
        float angEuler[3];//物体欧拉角
        float boxOrigin[3];//物体几何中心坐标
        float BoxExtent[3];//物体外框长宽高的一半
        double timestmp;//时间戳
        char ObjName[32] = { 0 };//碰物体的名字
    };
```

RflySim3D 根据 reqCamCoptObj 函数发送的命令，定时返回的该结构体，该程序接收到

后会存放在 self.ObjDataVect 中。收到的数据可以使用 getCamCoptObj 函数进行获取。

6.4. 地形获取接口

6.4.1. sendUE4Cmd（向 RflySim3D 发送命令）

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

一、参数解释：

1) Cmd：发送的命令的“字节字符串”
2) windowsID：影响发送目标端口，正常组播时取-1 即可，如果需要指定电脑上的指定的 RflySim3D 进行接收，则可以根据目标电脑 IP 与 RflySim3D 程序的窗口标题序号来设置。（这个序号与上面提到的 20010~20029 这 20 个端口是一一对应的关系，例如 RflySim3D-0 就是指的它在监听 20010，RflySim3D-3 就是指的它在监听 20013）

二、函数解释：

这个函数是向 RflySim3D 发送一个“命令”，其中 cmd 是一个字节字符串，表示发送给 RflySim3D 的命令，发送出去的是这样的结构体：

例如：

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

它发送了一条命令“RflyChangeMapbyName Grasslands”，其作用是要求 RflySim3D 改变地图，改成名为“Grasslands”的地图。

至于有哪些其他命令，以及它们的作用，可以参考“\OneDrive\RflySimDocs\高级版书稿\开发版\第 3 章-三维场景建模与仿真\Demo_Resources_3\e0\e0_2\ RflySim3D 命令接口.docx”，这里将它们列出来并简单介绍：

1. RflyShowTextTime(String txt, float time)\\ 让 UE 显示 txt，持续 time 秒
2. RflyShowText(String txt)\\ 让 UE 显示 txt，持续 5 秒
3. RflyChangeMapbyID(int id)\\ 根据地图的 ID 切换 RflySim3D 场景地图
4. RflyChangeMapbyName(String txt)\\ 根据地图名切换 RflySim3D 场景地图
5. RflyChangeViewKeyCmd(String key, int num) \\ 与在 RflySim3D 中按一个 key + num 效果一致
6. RflyCameraPosAngAdd(float x, float y, float z,float roll,float pitch,float yaw) \\ 给摄像机的位置与角度增加一个偏移值
7. RflyCameraPosAng(float x, float y, float z, float roll, float pitch, float yaw) \\ 设置摄像机的位置与角度(UE 的世界坐标)
8. RflyCameraFovDegrees(float degrees) \\ 设置摄像机的视域体 FOV 角度
9. RflyChange3DModel(int CopterID, int veTypes=0) \\ 修改一个无人机的模型样式
10. RflyChangeVehicleSize(int CopterID, float size=0) \\修改一个无人机的缩放大小
11. RflyMoveVehiclePosAng(int CopterID, int isFitGround, float x, float y, float z, float roll, float

pitch, float yaw) \\ 给无人机的位置与角度设置一个偏移值，isFitGround 设置无人机是否适应地面

12. RflySetVehiclePosAng(int CopterID, int isFitGround, float x, float y, float z, float roll, float pitch, float yaw) \\设置无人机的位置与角度
13. RflyScanTerrainH(float xLeftBottom(m), float yLeftBottom(m), float xRightTop(m), float yRightTop(m), float scanHeight(m), float scanInterval(m)) \\ 扫描地形，生成一个 png 的高度图与 txt，CopterSim 程序会需要它才知道 UE 有哪些地形、以及它们的高程
14. RflyCesiumOriPos(double lat, double lon, double Alt) \\ 根据经纬度修改 Cesium 的原点位置
15. RflyClearCapture()\\ 清空抓取的图像
16. RflySetActuatorPWMs(int CopterID, float pwm1, float pwm2, float pwm3, float pwm4, float pwm5, float pwm6, float pwm7, float pwm8); \\传入 8 个值，并触发目标无人机的蓝图的接口函数
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 pwm24);\\传入 16 个值，并触发目标无人机的蓝图接口函数。该函数需要完整版才能有作用
18. RflyReqVehicleData(FString isEnable);如果’isEnable’不为 0，则 RflySim3D 会开始发送所有 Copter 的数据（就是前面介绍的 reqVeCrashData）。
19. RflySetPosScale(float scale);\\全局位置的缩放
20. RflyLoad3DFile(FString FileName);\\加载并执行路径下的 TXT 脚本文件
21. RflyReqObjData(int opFlag, FString objName, FString colorStr);\\请求获取三维场景中物体的数据
22. RflySetIDLabel(int CopterID, FString Text, FString colorStr, float size);\\设置一个 Copter 的头顶 ID 显示内容（默认显示 CopterID）
23. RflySetMsgLabel(int CopterID, FString Text, FString colorStr, float size, float time, int flag);\\设置一个 Copter 头顶的 Message 显示的内容
24. RflyDelVehicles(FString CopterIDList);\\删除一些 Copter（逗号是分隔符）
25. RflyDisableVeMove(FString CopterIDList, int disable);\\拒接接收指定 ID 的 Copter 的信息（逗号是分隔符）
26. 除此之外，还有一些 UE 内置的命令可以使用，例如‘stat fps’可以显示当前帧率，‘t.Maxfps 60’可以设置最大帧率为 60。

7. 视觉传感器配置协议 Config.json

7.1. Config. json 协议总体介绍

7.1.1. Config.json 文件参数说明

不同的传感器有不同的参数配置方法，对于具体传感器配置方法参考支持的传感器列表及配

置方法，下面描述通用的配置参数

SeqID: 传感器在 RflySim3D 内的编号，多个传感器从 0 开始递增，不管是分步式集群还是单机都应该保证 seqid 从 0 开始递增且唯一；

TypeID: RflySim3D 从 TypeID 的值区分具体的传感器类型，如 1：代表 RGB 图像，2：深度图像等等，详细参考支持的传感器列表及配置方法；

TargetCopter: 该值表示当前配置的传感器应该依附在哪个控制载体上，载体可以是飞机，汽车，甚至一个树，一株草，一块石头等，只要能在 RflySim3D 内有“copter_id”即可。

TargetMountType: 参考坐标系选择，0:固定飞机上(相对几何中心) 1:固定飞机上(相对底部中心)。2:固定地面上；3:固定飞机上，但相机姿态不随飞机变化 (地面坐标系)；4 固定在某个传感器上，目前仅支持测距传感器，(使用该值是，TrargetCopter 应该为被固定传感器的 SeqID 的值，如把当前设置的传感器固定在 SeqID 为 0 的传感器上，该值应该为 0），通常设置为 0 吊舱默认设置为 3，需要注意的是如果该值为 4 时，被目标的传感器的参数配置应该放在当前传感器之前，所谓“皮之不存，毛将焉附”，所以得先创建目标传感器，如需要把 B 传感器安装在 A 传感器上，得先创建 A 传感器。

注意：老版本不支持 3 和 4

DataHeight：该值得根据具体传感器的类型来设置，如图像则表示图像的高，机械式激光雷达表示其垂直方向上的激光线束数量，具体参考支持的传感器列表及配置方法；

DataWidth: 该值得根据具体传感器的类型来设置，如图像则表示图像的宽，机械式激光雷达表示其水平方向上的激光线束点数，具体参考支持的传感器列表及配置方法；

DataCheckFreq：设置数据频率，该频率受 RflySim3D 刷新频率影响，RflySim3D 刷新频率一定要大于或等于所有配置传感器的最大数据频率。

SendProtocol：通信协议配置，SendProtocol[0]取值 0：共享内存 (免费版只支持共享内存)，1: UDP 直传 png 压缩，2: UDP 直传图片不压缩，3: UDP 直传 ipg 压缩。如果是激光雷达数据只有 0 或 1（共享内存和 UDP 网络传输）SendProtocol[1-41]: IP 地址: SendProtocol[5]端口号

注意：同一个目标地址下的端口号唯一

CameraFOV： 该值得根据具体传感器的类型来设置，如图像数据则表示相机的视场角，机械式激光雷达表示水平角范围，具体参考支持的传感器列表及配置方法；

SensorPosXYZ： 传感器的按照位置(单位米)，相对于载体中心的 FRD 坐标系

SensorAngEular： 传感器安装角度(单位度)，欧拉角度顺序为 roll,pitch,yaw，相对于载体 FRD 坐标系；

otherParams：具体参考支持的传感器列表及配置方法；

下面新协议的新增：

EulerOrOuat：为安装角度的表示方式，0 表示使用欧拉角也就是 SensorAngEular， 1 表示使用四元数 SensorAngQuat。

SensorAngEular：为传感器安装角度，单位度° 也可改变。

SensorAngQuat：为传感器安装角度，用四元数表示。

7.1.2. jsonLoad（通过读取配置文件来增加视觉传感器）

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

一、参数解释：

1) ChangeMode：可覆盖 json 中的 SendProtocol[0]传输模式

2) jsonPath：相对配置文件路径，实际读取的目录为“接口文件目录”/jsonPath，如果 jsonPath 空时，则将读取接口文件所在目录下的 Config.json。

二、函数解释：

该方法通过读取配置文件来增加视觉传感器，在添加完传感器参数后，可通过 sendReqToUE4 方法向 UE 申请添加对应传感。效果和 1addVisSensor 一致，但是配置文件中可以保存多个视觉传感器参数。

7.1.3. addVisSensor（添加一个视觉传感器参数的对象）

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

一、参数解释：

1) vsr：视觉传感器参数的对象

二、函数解释：

该方法添加一个视觉传感器参数的对象，在添加完传感器参数后，可通过 sendReqToUE4 方法向 UE 申请添加对应传感。如果传递的 vsr 参数不是 VisionSensorReq 类型，将会抛出一个异常。

7.2. 发送与接收方法

7.2.1. sendUpdateUEImage（发送了一个视觉传感器请求，RflySim3D 收到后会创建或更新这个传感器）

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

一、参数解释：

1) Vs：发送的 VisionSensorReq()实例
2) windID：影响发送目标端口，正常组播时取-1 即可，如果需要指定电脑上的指定的 RflySim3D 进行接收，则可以根据目标电脑 IP 与 RflySim3D 程序的窗口标题序号来设置。（这个序号与上面提到的 20010~20029 这 20 个端口是一一对应的关系，例如 RflySim3D-0 就是指的它在监听 20010，RflySim3D-3 就是指的它在监听 20013。可以看见这个 winID 默认为 0，也就是说它默认为单播，播往目标计算机的第一个 RflySim3D 程序。
3) IP：发送的目的地的 IP 地址

二、函数解释：

这个函数发送了一个视觉传感器请求，RflySim3D 收到后会创建或更新这个传感器（根据 SeqID 更新），这个函数比较特殊，介绍 VisionSensorReq 类的时候也说了，它并不是 UDP 组播而是单播，这是防止进行重复的计算（因为同局域网的 RflySim3D 上的视觉传感器得到的结果显然是一致的，没必要每个电脑都计算一遍然后发给目标计算机，可以让各电脑各自计算不同的视觉传感器，提升性能与效率），此外通过该函数可以是实时更改传感器的属性，比如视场角，以及旋转角度等等。

7.2.2. sendReqToUE4（发送一组 VisionSensorReq）

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

一、参数解释：

1) windID：影响发送目标端口，正常组播时取-1 即可，如果需要指定电脑上的指定的 RflySim3D 进行接收，则可以根据目标电脑 IP 与 RflySim3D 程序的窗口标题序号来设置。（这个序号与上面提到的 20010~20029 这 20 个端口是一一对应的关系，例如 RflySim3D-0 就是指的它在监听 20010，RflySim3D-3 就是指的它在监听 20013
2) IP：发送的目的地的 IP 地址
3) 虽然函数中没有更多的形参，但其实该函数引用了一些类中的成员变量：
4) self.VisSensor：VisionSensorReq 列表，该函数所发送的 VisionSensorReq 的列表
5) self.isUE4DirectUDP：bool 类型，是否使用 UDP 发送
6) self.mm0：共享内存的地址(不需要改它)，使用该函数发送时，还会在本机上创建一个共享内存赋值给它，可以且仅能读取本机的 RflySim3D 给出的返回信息（图像的宽高），然后输出一条信息表示该请求是否成功。

二、函数解释：

就是发送一组 VisionSensorReq，这些 VisionSensorReq 需要放在 self.VisSensor 中，视情况赋值 self.isUE4DirectUDP。

7.3. 支持的传感器列表及配置方法

RflySim 仿真平台的传感器配置通过传感器配置文件 Config.json 文件进行设置。可进行相机、深度相机、红外、激光雷达等传感器的设置。

7.3.1. 可见光 RGB 图像

```
{
    "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”代表第几个传感器。此处表示第 1 个传感器（免费版只支持 2 个图）。
“TypeID”代表传感器类型 ID，1:RGB 图、，2:深度图，3:灰度图，更多配置参考 PPT。
“TargetCopter”传感器装载的目标飞机的 ID ，可改变。
“TargetMountType”代表坐标类型，0：固定飞机上（相对几何中心），1：固定飞机上（相对底部中心），2：固定地面上（监控）也可变，3: 相对地面坐标系的吊舱相机，固

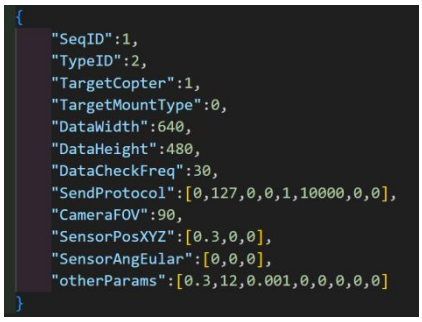
定飞机上，但相机姿态不随飞机变化（地面坐标系），4:将传感器附加到另外一个传感器上，当 MountType == 4 的时候，Config.json 中的 TargetCopter == SeqID（因为 MountType == 4 是将传感器附加到传感器上，所以 TargetCopter 本来是用于给定载具 ID，这时候就没用了，就被用来设定传感器的 ID 了，也就是 SeqID）。
“DataWidth”为数据或图像宽度此处为 640，“DataHeight”为数据或图像高度此处为 480。
“DataCheckFreq”检查数据更新频率此处为 30HZ。
“SendProtocol[8]”为传输方式与地址， SendProtocol[0]取值 0：共享内存（免费版只支持共享内存），1：UDP 直传 png 压缩，2：UDP 直传图片不压缩，3：UDP 直传 jpg 压缩；SendProtocol[1-4] ：IP 地址； SendProtocol[5]端口号。
“CameraFOV”为相机视场角（仅限视觉类传感器），单位度也可改变。
“EulerOrOuat”为安装角度的表示方式，0 表示使用欧拉角也就是 SensorAngEular， 1 表示使用四元数 SensorAngQuat。
“SensorPosXYZ[3]”为传感器安装位置，单位米也可改变。
“SensorAngEular[3]”为传感器安装角度，单位度° 可改变
“SensorAngQuat[4]”为传感器安装角度，用四元数表示。

7.3.2. 灰度图像

“SeqID”代表第几个传感器。此处表示第 1 个传感器。
“TypeID”1:RGB 图，2:深度图，3:灰度图，更多的配置参考 PPT。
“TargetCopter”传感器装载的目标飞机的 ID ，可改变。
“TargetMountType”代表坐标类型，0：固定飞机上（相对几何中心），1：固定飞机上（相对底部中心），2：固定地面上（监控）也可变，3: 相对地面坐标系的吊舱相机，固定飞机上，但相机姿态不随飞机变化（地面坐标系），4:将传感器附加到另外一个传感器上，当 MountType == 4 的时候，Config.json 中的 TargetCopter == SeqID（因为 MountType == 4 是将传感器附加到传感器上，所以 TargetCopter 本来是用于给定载具 ID，这时候就没用了，就被用来设定传感器的 ID 了，也就是 SeqID）。
“DataWidth”为数据或图像宽度此处为 640，“DataHeight”为数据或图像高度此处为 480。
“DataCheckFreq”检查数据更新频率此处为 30HZ。
“SendProtocol[8]”为传输方式与地址， SendProtocol[0]取值 0：共享内存（免费版只支持共享内存），1：UDP 直传 png 压缩，2：UDP 直传图片不压缩，3：UDP 直传 jpg 压缩；SendProtocol[1-4] ：IP 地址； SendProtocol[5]端口号。
“CameraFOV”为相机视场角（仅限视觉类传感器），单位度也可改变。
“EulerOrOuat”为安装角度的表示方式，0 表示使用欧拉角也就是 SensorAngEular， 1 表示使用四元数 SensorAngQuat。
“SensorPosXYZ[3]”为传感器安装位置，单位米也可改变。

“SensorAngEular[3]”为传感器安装角度，单位度° 也可改变。
“SensorAngQuat[4]”为传感器安装角度，用四元数表示。

7.3.3. 深度图像



“SeqID”代表第几个传感器。此处表示第 1 个传感器。
“TypeID”1:RGB 图，2:深度图，3:灰度图，更多的配置参考 PPT
“TargetCopter”传感器装载的目标飞机的 ID ，可改变。
“TargetMountType”代表坐标类型，0：固定飞机上（相对几何中心），1：固定飞机上（相对底部中心），2：固定地面上（监控）也可变。
“DataWidth”为数据或图像宽度此处为 640，“DataHeight”为数据或图像高度此处为 480。
“DataCheckFreq”检查数据更新频率此处为 30HZ。
“SendProtocol[8]”为传输方式与地址， SendProtocol[0]取值 0：共享内存（免费版只支持共享内存），1：UDP 直传 png 压缩，2：UDP 直传图片不压缩，3：UDP 直传 jpg 压缩；SendProtocol[1-4] ：IP 地址； SendProtocol[5]端口号。
“CameraFOV”为相机视场角（仅限视觉类传感器），单位度也可改变。
“EulerOrOuat”为安装角度的表示方式，0 表示使用欧拉角也就是 SensorAngEular， 1 表示使用四元数 SensorAngQuat。
“SensorPosXYZ[3]”为传感器安装位置，单位米也可改变。
“SensorAngEular[3]”为传感器安装角度，单位度° 也可改变。
“SensorAngQuat[4]”为传感器安装角度，用四元数表示。
深度相机输出的数据是以 uinit16 存储和传输的，它的数据范围是 0~65535。默认情况下，一个单位表示 1mm（由 otherParams[2]控制），也就是说最大范围是 0 到 65.535 米。但是，数据范围并不代表相机的实际探测距离，还需要 otherParams[0]设置最小探测距离 otherParams[1]设置最大探测距离。otherParams[0]：深度相机的最小识别距离（单位米），如果深度距离小于本值，那么输出 NaN 对应 65535。otherParams[1]：深度相机的最大识别距离（单位米），如果深度距离大于本值，那么输出 NaN 对应 65535。otherParams[2]：深度相机 uint16 输出值的刻度单位（单位米），默认情况下深度值以毫米为单位，因此需要填 0.001。注，默认值填 0 的话，会被替换为 otherParams[2]=0.001。实际深度值（单位米） = 深度图片值（uint16 范围）* otherParams[2]。

7.3.4. 分割图

“SeqID”代表第几个传感器。此处表示第 1 个传感器（免费版只支持 2 个图）。
“TypeID”代表传感器类型 ID，1:RGB 图、，2:深度图，3:灰度图，更多配置参考 PPT。
“TargetCopter”传感器装载的目标飞机的 ID ，可改变。
“TargetMountType”代表坐标类型，0：固定飞机上（相对几何中心），1：固定飞机上（相对底部中心），2：固定地面上（监控）也可变，3: 相对地面坐标系的吊舱相机，固定飞机上，但相机姿态不随飞机变化（地面坐标系），4:将传感器附加到另外一个传感器上，当 MountType == 4 的时候，Config.json 中的 TargetCopter == SeqID（因为 MountType == 4 是将传感器附加到传感器上，所以 TargetCopter 本来是用于给定载具 ID，这时候就没用了，就被用来设定传感器的 ID 了，也就是 SeqID）。
“DataWidth”为数据或图像宽度此处为 640，“DataHeight”为数据或图像高度此处为 480。
“DataCheckFreq”检查数据更新频率此处为 30HZ。
“SendProtocol[8]”为传输方式与地址， SendProtocol[0]取值 0：共享内存（免费版只支持共享内存），1：UDP 直传 png 压缩，2：UDP 直传图片不压缩，3：UDP 直传 jpg 压缩；SendProtocol[1-4] ：IP 地址； SendProtocol[5]端口号。
“CameraFOV”为相机视场角（仅限视觉类传感器），单位度也可改变。
“EulerOrOuat”为安装角度的表示方式，0 表示使用欧拉角也就是 SensorAngEular， 1 表示使用四元数 SensorAngQuat。
“SensorPosXYZ[3]”为传感器安装位置，单位米也可改变。
“SensorAngEular[3]”为传感器安装角度，单位度° 也可改变。
“SensorAngQuat[4]”为传感器安装角度，用四元数表示。

7.3.5. 测距传感器

测距传感器，向当前传感器朝向发射一条射线（如果是附加在传感器上，也就是 MountType == 4 且 PosXYZ 和 AngEular 都为零的情况 返回图像中心目标距离）。对于测距传感器，DataWidth 以及 DataHeight 两个参数无效，而 otherParams[0]表示测距传感器能够测量的最大距离，需要注意的是当 TargetMountType 为 4 也就是安装在其他传感器上，则 Config.json 文件需要先配置被安装的传感器上，并将” SensorPosXYZ”和” SensorAngEular”全部设置为 0。

“SeqID”代表第几个传感器。此处表示第 1 个传感器（免费版只支持 2 个图）。
“TypeID”代表传感器类型 ID，测距传感器为 5，更多配置参考 PPT。
“TargetCopter”传感器装载的目标飞机的 ID ，可改变。（若安装在其他传感器上则需要改为传感器 ID）。
“TargetMountType”代表坐标类型，0：固定飞机上（相对几何中心），1：固定飞机上（相对底部中心），2：固定地面上（监控）也可变，3: 相对地面坐标系的吊舱相机，固定飞机上，但相机姿态不随飞机变化（地面坐标系），4:将传感器附加到另外一个传感器上，当 MountType == 4 的时候，Config.json 中的 TargetCopter == SeqID（因为 MountType == 4 是将传感器附加到传感器上，所以 TargetCopter 本来是用于给定载具 ID，这时候就没用了，就被用来设定传感器的 ID 了，也就是 SeqID），且被安装的传感器需要先行配置。
“DataWidth”与 “DataHeight”在测距传感器中无效。
“SendProtocol[8]”提供传输地址，SendProtocol[1-4] ：IP 地址； SendProtocol[5]端口号。
“EulerOrOuat”为安装角度的表示方式，0 表示使用欧拉角也就是 SensorAngEular， 1 表示使用四元数 SensorAngQuat。
“SensorPosXYZ[3]”为传感器安装位置，单位米也可改变。
“SensorAngEular[3]”为传感器安装角度，单位度° 也可改变。
“SensorAngQuat[4]”为传感器安装角度，用四元数表示。

7.3.6. 机械式扫描激光雷达

“SeqID”代表第几个传感器。
“TypeID”取值 20，21，22；20:代表输出点云为激光雷达坐标系, 21:代表输出点云为世界坐标系;22:代表 livox 激光雷达
“TargetCopter”传感器装载的目标飞机的 ID ，可改变。
“TargetMountType”代表坐标类型，0：固定飞机上（相对几何中心），1：固定飞机上（相对底部中心），2：固定地面上（监控）也可变，3: 相对地面坐标系的吊舱相机，固定飞机上，但相机姿态不随飞机变化（地面坐标系），4:将传感器附加到另外一个传感器

上，当 MountType == 4 的时候，Config.json 中的 TargetCopter == SeqID（因为 MountType == 4 是将传感器附加到传感器上，所以 TargetCopter 本来是用于给定载具 ID，这时候就没用了，就被用来设定传感器的 ID 了，也就是 SeqID）。
“DataWidth”为激光雷达一个 ring 内的点云个数，“DataHeight”为激光雷达线束数量。
“DataCheckFreq”点云发布频率(hz)此处为 30HZ。
“SendProtocol[8]”为传输方式与地址，SendProtocol[0]取值 0：表示共享内存输出模式，取值 1：表示 UDP 直发模式。
“CameraFOV”：在激光雷达传感器上无作用。
“EulerOrOuat”为安装角度的表示方式，0 表示使用欧拉角也就是 SensorAngEular， 1 表示使用四元数 SensorAngQuat。
“SensorPosXYZ[3]”为传感器安装位置，单位米也可改变。
“SensorAngEular[3]”为传感器安装角度，单位度° 也可改变。
“SensorAngQuat[4]”为传感器安装角度，用四元数表示。
otherParams：[激光最远距离(m),精度(m),水平扫描角度 下限值(度),水平扫描角度上限值(度),垂直扫描角度下限值(度),垂直扫描角度上限值(度),预留,预留];
激光雷达水平分辨通过 DataWidth 和水平扫描角度范围体现，垂直分辨率通过处置扫描角度体现(如图中的水平分辨率=90/900),垂直分辨率=40/32)，以上角度值都是用 degree 表示。

7.3.7. 花式扫描激光雷达

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。

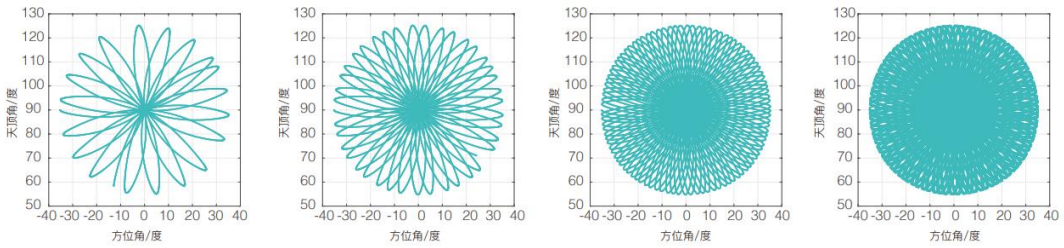


图 1.2.1 Livox Mid-70 不同积分时间内点云效果图

参考例程 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”代表第几个传感器。此处表示第 1 个传感器。
“TypeID”取值 20,21,22；20-:代表输出点云为激光雷达坐标系, 21:代表输出点云为世界坐标系;22:代表 livox 激光雷达

“TargetCopter”传感器装载的目标飞机的 ID ，可改变。
“TargetMountType”代表坐标类型，0：固定飞机上（相对几何中心），1：固定飞机上（相对底部中心），2：固定地面上（监控）也可变，3: 相对地面坐标系的吊舱相机，固定飞机上，但相机姿态不随飞机变化（地面坐标系），4:将传感器附加到另外一个传感器上，当 MountType == 4 的时候，Config.json 中的 TargetCopter == SeqID（因为 MountType == 4 是将传感器附加到传感器上，所以 TargetCopter 本来是用于给定载具 ID，这时候就没用了，就被用来设定传感器的 ID 了，也就是 SeqID）。
“DataWidth” 半边花瓣（对应上图红色区域）的点数
“DataHeight” 表示一帧需要扫描多少个半边花瓣
“DataCheckFreq” 表示每秒发送的数据帧数。
“SendProtocol[8]”为传输方式与地址，SendProtocol[0]取值 0：表示共享内存输出模式，取值 1：表示 UDP 直发模式。
“CameraFOV” 半边花瓣长度的 2 倍。单位度。
“EulerOrOuat”为安装角度的表示方式，0 表示使用欧拉角也就是 SensorAngEular， 1 表示使用四元数 SensorAngQuat。
“SensorPosXYZ[3]”为传感器安装位置，单位米也可改变。
“SensorAngEular[3]”为传感器安装角度，单位度° 也可改变。
“SensorAngQuat[4]”为传感器安装角度，用四元数表示。
otherParams[0]：激光雷达的最远扫描距离（单位米），例如这里是 600 米。注意：点云数据我们最终是以 int16 格式发布，这个值是与 otherParams[0]参数是耦合的。例如，x_float 是激光雷达测量得到的点云的 x 轴浮点数值（单位米，数值小于 otherParams[0]），则 x_int = x_float / otherParams[0] * 32767。因此，拿到点云数据后，需要再根据最远距离 otherParams[0]，做一个逆向映射：x_float = x_int / 32767 * otherParams[0].
otherParams[1]：对应了上图的花瓣高度（单位度）。这里反映了扫描花瓣的弯曲程度。
otherParams[1]：对应了上图的花瓣高度（单位度）。这里反映了扫描花瓣的弯曲程度。
otherParams[3]：红色花瓣左侧圆弧的校准指数，默认为 1.18。
otherParams[4]：几个 8 字（一个八字是两片对称花瓣，4 段弧）组成一朵花，如上图所示是大疆激光雷达 0.1s 的数据，也就是 10 个八字（20 片花瓣）组成了一朵花。这样 otherParams[4]=10。也就是说，相邻两个花瓣之间的角度为 180/10=18 度。

7.3.8. 红外灰度图像

```
{
    "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”代表第几个传感器。此处表示第 1 个传感器。

“TypeID”40 表示红外灰度图
“TargetCopter”传感器装载的目标飞机的 ID ，可改变。
“TargetMountType”代表坐标类型，0：固定飞机上（相对几何中心），1：固定飞机上（相对底部中心），2：固定地面上（监控）也可变，3: 相对地面坐标系的吊舱相机，固定飞机上，但相机姿态不随飞机变化（地面坐标系），4:将传感器附加到另外一个传感器上，当 MountType == 4 的时候，Config.json 中的 TargetCopter == SeqID（因为 MountType == 4 是将传感器附加到传感器上，所以 TargetCopter 本来是用于给定载具 ID，这时候就没用了，就被用来设定传感器的 ID 了，也就是 SeqID）。
“DataWidth”为数据或图像宽度此处为 640，“DataHeight”为数据或图像高度此处为 480。
“DataCheckFreq”检查数据更新频率此处为 30HZ。
“SendProtocol[8]”为传输方式与地址， SendProtocol[0]取值 0：共享内存（免费版只支持共享内存），1：UDP 直传 png 压缩，2：UDP 直传图片不压缩，3：UDP 直传 jpg 压缩；SendProtocol[1-4] ：IP 地址； SendProtocol[5]端口号。
“CameraFOV”为相机视场角（仅限视觉类传感器），单位度也可改变。
“EulerOrOuat”为安装角度的表示方式，0 表示使用欧拉角也就是 SensorAngEular， 1 表示使用四元数 SensorAngQuat。
“SensorPosXYZ[3]”为传感器安装位置，单位米也可改变。
“SensorAngEular[3]”为传感器安装角度，单位度° 也可改变。
“SensorAngQuat[4]”为传感器安装角度，用四元数表示。

7.3.9. 热力彩色图

“SeqID”代表第几个传感器。此处表示第 1 个传感器。
“TypeID”使用 41 表示热力彩色图
“TargetCopter”传感器装载的目标飞机的 ID ，可改变。
“TargetMountType”代表坐标类型，0：固定飞机上（相对几何中心），1：固定飞机上（相对底部中心），2：固定地面上（监控）也可变，3: 相对地面坐标系的吊舱相机，固定飞机上，但相机姿态不随飞机变化（地面坐标系），4:将传感器附加到另外一个传感器上，当 MountType == 4 的时候，Config.json 中的 TargetCopter == SeqID（因为 MountType == 4 是将传感器附加到传感器上，所以 TargetCopter 本来是用于给定载具 ID，这时候就没用了，就被用来设定传感器的 ID 了，也就是 SeqID）。
“DataWidth”为数据或图像宽度此处为 640，“DataHeight”为数据或图像高度此处为 480。
“DataCheckFreq”检查数据更新频率此处为 30HZ。
“SendProtocol[8]”为传输方式与地址， SendProtocol[0]取值 0：共享内存（免费版只支持共享内存），1：UDP 直传 png 压缩，2：UDP 直传图片不压缩，3：UDP 直传 jpg 压缩；SendProtocol[1-4] ：IP 地址； SendProtocol[5]端口号。

“CameraFOV”为相机视场角（仅限视觉类传感器），单位度也可改变。
“EulerOrOuat”为安装角度的表示方式，0 表示使用欧拉角也就是 SensorAngEular， 1 表示使用四元数 SensorAngQuat。
“SensorPosXYZ[3]”为传感器安装位置，单位米也可改变。
“SensorAngEular[3]”为传感器安装角度，单位度°也可改变。
“SensorAngQuat[4]”为传感器安装角度，用四元数表示。

8. Python 视觉接口 VisionCaptureApi.py

8.1. 相关类及数据结构

VisionCaptureApi.py 是本平台的取图接口文件，包含了 json 加载，图像请求，图像转发等。其中包含了多个与视觉相关的类。

- Queue：这个类用 python 列表来抽象先进先出的队列。

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

一、参数解释：

1) item:待入队项

二、函数解释：

这个方法将 item 项插入队列头部。

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

一、参数解释：

二、函数解释：

这个方法将队列末尾的元素删除并返回，如果队列为空，则会抛出 IndexError 异常。

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

一、参数解释：

二、函数解释：

这个方法用于判断队列是否为空，通常在调用 dequeue 方法前先调用该方法，可以避免队列空时出队异常。

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

一、参数解释：

二、函数解释：

这个方法读取并返回队列长度。可用于队列最大和最小长度的控制。

- class RflyTimeStmp：这个类封装了发送给指定远端电脑端口 20005 的消息，主要存储了终端电脑和仿真的时间戳，并通过心跳来检测终端电脑和仿真程序连接是否正常。

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

1. checksum：是数据的校验位，用于检验数据传输过程中是否发生了异常。
2. copterID：表示该消息对应的 copter ID。
3. SysStartTime：是 Windows 下的开始仿真的时间戳，单位毫秒，采用格林尼治标准

起点。

4. SysCurrentTime：Windows 下的当前时间戳，单位毫秒，采用格林尼治标准起点。
5. HeartCount：心跳包的计数器。

- class VisionSensorReq: 这个类是一个发送往 UE 的结构体，RflySim3D 收到后会创建一个视觉传感器，并开始输出图像数据，该结构体通常不是组播发送的，而且会指定一个 RflySim3D 的窗口接收（也就是说只有一台电脑上一个 RflySim3D 收到该结构体），该结构体的赋值根据配置不同的传感器有不同的配置方法，详细参考支持的传感器列表及配置方法。

```
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]
```

这是初始化函数，其中各个参数决定了传感器的状态。

1. SeqID：是传感器的 ID，它必须在局域网内唯一，每台电脑上最多 32 个视觉传感器（如果有 n 台电脑，那么可以实现 32*n 个传感器），同一台电脑上的传感器 ID 必须属于同一组(例如 A 电脑传感器的 ID 为 0~31，B 电脑传感器的 ID 为 32~63)。
2. TypeID：表示传感器的类型（决定 RflySim3D 返回什么样的数据）可以取[1,6]，1:RGB 图（免费版只支持 RGB 图），2:深度图，3:灰度图，4:点云相对于机体坐标系，5:点云相对于大地坐标系，6:相对于机体坐标系下的花瓣式扫描点云。
3. TargetCopter：装载该传感器的 Copter 的 ID。//可变
4. TargetMountType：坐标类型，0:固定飞机上（相对几何中心），1：固定飞机上（相对底部中心），2:固定地面上（监控）//可变
5. DataWidth：数据或图像宽度
6. DataHeight：数据或图像高度
7. DataCheckFreq：检查数据更新频率（单位为 Hz）
8. SendProtocol[8]：传输方式与地址，其中 SendProtocol[0]取值 0：共享内存（免费版只支持共享内存），1：UDP 直传 png 压缩，2：UDP 直传图片不压缩，3：UDP 直传 jpg 压缩。SendProtocol[1-4] ：IP 地址，SendProtocol[5]端口号。这里的 IP 地址指的是 RflySim3D 发送图像数据的目的地址。
9. CameraFOV：相机视场角（仅限视觉类传感器），单位度 //可改变
10. SensorPosXYZ[3] ：传感器安装位置，单位米 //可改变
11. SensorAngEular[3] ：传感器安装角度，单位度° //可改变
12. otherParams[8]：针对不同的传感器预留的参数配置，详细参考支持的传感器列表及配置方法。

- class imuDataCopter:该类封装了 CopterSim 回传的 IMU 数据包，并提供了向 ros 消息的转换。

```
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.isUseTimeAlign = True  # 是否使用与图像时间对其的方式发布数据
            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. checksum：数据校验位。
2. seq：表示消息的序号。
3. timestmp：消息时间戳。
4. acc：表示 IMU 加速度
5. rate：表示 IMU 角速度

- class SensorReqCopterSim:该类封装了发送给 CopterSim 请求传感器的数据包，用于请求各种传感器数据。

```
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. checksum：校验位。
2. sensorType：传感器类型 0~6 分别代表 Imu；RGB 图；深度图；灰度图；雷达坐标系的激光点云；世界坐标系的激光点云；livox 点云。
3. updateFreq：请求的传感器数据频率。
4. port：接收数据的端口号。
5. IP：接收数据的 ip 地址。
6. Params：预留参数。

- class reqVeCrashData:这个类是由 RflySim3D 发送的结构体，是当 RflySim3D 开启数据回传模式时（使用“RflyReqVehicleData 1”命令），会发送出来的结构体。里面主要是与碰撞相关的数据，会被发往组播 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：表示该结构体是哪个 Copter 的数据
2. vehicleType：表示该 Copter 的样式
3. CrashType：表示碰撞物体类型，-2 表示地面，-1 表示场景静态物体，0 表示无碰撞，1 以上表示被碰飞机的 ID 号
4. runnedTime：当前飞机的时间戳
5. VelE：当前飞机的速度（米/秒）
6. PosE：当前飞机的位置（北东地，单位米）
7. CrashPos：碰撞点的坐标（北东地，单位米）
8. targetPos：被碰物体的中心坐标（北东地，单位米）
9. AngEuler：当前飞机的欧拉角（Roll，Pitch，Yaw，弧度）
10. MotorRPMS：当前飞机的电机转速
11. Ray：飞机的前后左右上下扫描线
12. CrashedName：被碰物体的名字

- class PX4SILIntFloat:这个类封装了输出到 CopterSim DLL 模型的 SILInts 和 SILFloats 数据。

```
 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. checksum：校验位
2. CopterID：消息对应的 Copter ID
3. inSILInts：输出到 CopterSim DLL 模型的 SILInts 数据
4. inSILFLoats：输出到 CopterSim DLL 模型的 SILFoats 数据

- class VisionCaptureApi:这个类是整个文件的主体，所有函数都写在里面，这里介绍 UE

相关内容。

这里面的 UE 相关的函数可以参考“\OneDrive\RflySimDocs\高级版书稿\开发版\第 3 章-三维场景建模与仿真\Demo_Resources_3\e0\e0_4\RflySim3D 相关的 Python 接口.docx”里面也介绍了各个函数的用途，还准备了一些例程。

开头稍微介绍下 RflySim3D 的网络情况：

1. 它主要接收 UDP 数据，它最主要的监听地址是 224.0.0.10:20009 这个组播地址，以及自身的 20009 端口的单播地址。
2. 它还可以接收组播地址 224.0.0.11:20008，以及自身的单播端口 20008，但这个组播只接受本机环回地址的数据。
3. 它还能接收 20010~20029 这 20 个端口中的一个，因为如果同一个电脑开启多个 RflySim3D 程序，它们会争夺单播端口，所以根据创建顺序，它们会依次分配这些单播端口。
4. 这些端口对数据的处理方式是完全相同的。

8.2. 发送与接收图像相关接口

8.2.1. sendReqToUE4（发送一组 VisionSensorReq）

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

一、参数解释：

1) windID：影响发送目标端口，正常组播时取-1 即可，如果需要指定电脑上的指定的 RflySim3D 进行接收，则可以根据目标电脑 IP 与 RflySim3D 程序的窗口标题序号来设置。（这个序号与上面提到的 20010~20029 这 20 个端口是一一对应的关系，例如 RflySim3D-0 就是指的它在监听 20010，RflySim3D-3 就是指的它在监听 20013
2) IP：发送的目的地的 IP 地址
3) 虽然函数中没有更多的形参，但其实该函数引用了一些类中的成员变量：
4) self.VisSensor：VisionSensorReq 列表，该函数所发送的 VisionSensorReq 的列表
5) self.isUE4DirectUDP：bool 类型，是否使用 UDP 发送
6) self.mm0：共享内存的地址(不需要改它)，使用该函数发送时，还会在本机上创建一个共享内存赋值给它，可以且仅能读取本机的 RflySim3D 给出的返回信息（图像的宽高），然后输出一条信息表示该请求是否成功。

二、函数解释：

就是发送一组 VisionSensorReq，这些 VisionSensorReq 需要放在 self.VisSensor 中，视情况赋值 self.isUE4DirectUDP。

8.2.2. jsonLoad（通过读取配置文件来增加视觉传感器）

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

一、参数解释：

1) ChangeMode：可覆盖 json 中的 SendProtocol[0]传输模式
2) jsonPath：相对配置文件路径，实际读取的目录为“接口文件目录”/jsonPath，

如果 jsonPath 空时，则将读取接口文件所在目录下的 Config.json。

二、函数解释：

该方法通过读取配置文件来增加视觉传感器，在添加完传感器参数后，可通过 sendReqToUE4 方法向 UE 申请添加对应传感。效果和 1addVisSensor 一致，但是配置文件中可以保存多个视觉传感器参数。

8.2.3. sendReqToCopterSim（发送 UDP 数据向 CopterSim 请求传感器数据。如：IMU）

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

一、参数解释：

1) srcs：发送的 SensorReqCopterSim 实例，参考 3.3.5
2) copterID：请求的 CopterSim 索引

二、函数解释：

该方法通过发送 UDP 数据向 CopterSim 请求传感器数据。例如请求 Imu、RGB 图像数据等。

8.2.4. img_mem_thrd（通过读取共享内存来接收 RflySim3D 回传的图）

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

一、参数解释：

1) idxList：传感器序号列表

二、函数解释：

这个方法通过读取共享内存来接收 CopterSim 回传的图像数据并解析成 numpy 格式，并在内置变量 hasData 和 Img 填写解析后的数据，用户程序可通过这两个变量访问图片。同时全局变量 isEnableRosTrans 为 True 时，将以 ros 消息的形式发布图片内容。

8.2.5. startImgCap（开启抓取共享内存中图像的线程）

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

一、参数解释：

1) isRemoteSend：如果为 true，则在共享内存中拿到的图像会被转发到 UDP 中

二、函数解释：

开始抓取图像，会根据 self.VisSensor 列表中指示的传输方式，开启 UDP 的接收线程“self.img_udp_thrdNew”，或者会开启抓取共享内存中图像的线程“self.img_mem_thrd”。这个 UDP 的接收线程会开启多个，每个传感器一个。

8.2.6. sendImgUDPNew（将 RflySim3D 回传的图片通过 udp 发送到指定 ip 和端口号）

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

一、参数解释：

1) idx：传感器序号

二、函数解释：

这个方法将 CopterSim 回传的图片通过 udp 发送到指定 ip 和端口号，ip 和端口号设置参考 VisionSensorReq 类的 SendProtocol。或者在 jsonLoad 的配置文件中设置。

8.2.7. img_udp_thrdNew（接收 RflySim3D 回传的图像数据）

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

一、参数解释：

1) updSok：udp 套接字
2) idx：传感器序号
3) typeID：传感器类型

二、函数解释：

这个方法通过 udp 接收 CopterSim 回传的图像数据并解析成 numpy 格式。并在内置变量 hasData 和 Img 里填写解析后的数据，用户程序可通过这两个变量访问图片。例如：

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

同时全局变量 isEnableRosTrans 为 True 时，将以 ros 消息的形式发布图片内容。

8.3. IMU 获取相关接口

8.3.1. sendImuReqCopterSim（通过发送 SensorReqCopterSim 实例来请求 Imu 数据）

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

一、参数解释：

1) copterID：请求的 CopterSim 索引
2) IP：数据接收端的 IP，空时默认为本机 ip

3) port：CopterSim 发送数据的端口号
4) freq：请求数据的频率

二、函数解释：

该方法通过发送 SensorReqCopterSim 实例来请求 Imu 数据，然后开启一个线程来监听和处理 CopterSim 回传的 Imu 数据。

8.3.2. sendImuReqClient（发送 SensorReqCopterSim 实例来请求 Imu 数据）

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

一、参数解释：

1) copterID：请求的 CopterSim 索引
2) IP：数据接收端的 IP，空时默认为本机 ip
3) port：CopterSim 发送数据的端口号
4) freq：请求数据的频率

二、函数解释：

该方法通过发送 SensorReqCopterSim 实例来请求 Imu 数据。

8.3.3. sendImuReqServe（在线程中接收 CopterSim 回传的 Imu 数据）

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

一、参数解释：

1) copterID：回传 Imu 数据的 CopterSim 索引
2) port：CopterSim 发送数据的端口号

二、函数解释：

该方法创建一个线程，在线程中接收 CopterSim 回传的 Imu 数据，并不阻塞主线程。

8.3.4. AlignTime（保证 Imu 数据发布过程中保持稳定频率发布）

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

一、参数解释：

1) img_time：时间戳

二、函数解释：

这个方法用于保证 Imu 数据发布过程中保持稳定频率发布，不受图像发布频率不稳定的干扰。采用的方式是在程序启动前对齐时间。

8.4. ROS 相关接口

8.4.1. Imu2ros（CopterSim 回传的 imu 消息到 ros 内置 imu 消息格式的转换）

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

一、参数解释：

1) node：ros 节点

二、函数解释：

这个方法提供了 CopterSim 回传的 imu 消息到 ros 内置 imu 消息格式的转换。当 isEnableRosTrans 为 True 时，该方法将从 CopterSim 接收到的 imu 消息转换成 ros 消息，并通过 imu_pub 发布到名称为 imu_name 的话题，方便和 ros 中成熟的模块对接。

8.4.2. getIMUDataLoop（将 Imu 数据转换到 ros 中 Imu 的消息，并发送到指定话题中）

```
def getIMUDataLoop(self)
```

一、参数解释：

二、函数解释：

该方法用于处理接收到的 Imu 数据，如果 isEnableRosTrans 为 True 时，还会将 Imu 数据转换到 ros 中 Imu 的消息，并发送到指定话题中。

8.4.3. img_mem_thrd（全局变量 isEnableRosTrans 为 True 时，将以 ros 消息的形式发布图片内容）

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

一、参数解释：

1) idxList：传感器序号列表

二、函数解释：

这个方法通过读取共享内存来接收 CopterSim 回传的图像数据并解析成 numpy 格式，并在内置变量 hasData 和 Img 填写解析后的数据，用户程序可通过这两个变量访问图片。方法与 3.3.8.23 一致。同时全局变量 isEnableRosTrans 为 True 时，将以 ros 消息的形式发布图片内容。

8.4.4. img_udp_thrdNew（全局变量 isEnableRosTrans 为 True 时，将以 ros 消息的形式发布图片内容）

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

一、参数解释：

1) updSok：udp 套接字
2) idx：传感器序号
3) typeID：传感器类型

二、函数解释：

这个方法通过 udp 接收 CopterSim 回传的图像数据并解析成 numpy 格式。并在内置变量 hasData 和 Img 里填写解析后的数据，用户程序可通过这两个变量访问图片。例如：

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

同时全局变量 isEnableRosTrans 为 True 时，将以 ros 消息的形式发布图片内容。

8.5. 其他内部函数

8.5.1. sendImgBuffer（将待发送数据按固定大小封装成一个或多个报文，通过 udp 发送到指定 ip 和端口号）

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

一、参数解释：

1) idx：传感器序号
2) data：待发送数据

二、函数解释：

这个方法将待发送数据按固定大小封装成一个或多个报文，并通过 udp 发送到指定 ip 和端口号。

8.5.2. initUE4MsgRec（启动一个 t4(self.UE4MsgRecLoop)开始监听 224.0.0.10：20006）

```
def initUE4MsgRec(self)
```

一、参数解释：

二、函数解释：

初始化 self.udp_socketUE4，并且启动一个线程 t4(self.UE4MsgRecLoop)开始监听

224.0.0.10：20006，以及自身的 20006 端口。

8.5.3. endUE4MsgRec（停止线程 t4(self.UE4MsgRecLoop)的监听）

```
def endUE4MsgRec(self)
```

一、参数解释：

二、函数解释：

停止线程 t4(self.UE4MsgRecLoop)的监听。

8.5.4. UE4MsgRecLoop（用于处理 RflySim3D 或 CopterSim 返回的消息）

```
def UE4MsgRecLoop(self)
```

一、参数解释：

二、函数解释：

它是监听 224.0.0.10：20006，以及自身的 20006 端口的处理函数，用于处理 RflySim3D 或 CopterSim 返回的消息，一共有 3 种消息：

1) 长度为 12 字节的 CopterSimCrash：

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

由 RflySim3D 返回的碰撞数据，RflySim3D 中按 P 键时 RflySim3D 会开启碰撞检测模式，如果发生了碰撞，会传回这个数据，P+数字可以选择发送模式（0 本地发送，1 局域网发送，2 局域网只碰撞时发送），默认为本地发送。

2) 长度为 120 字节的 PX4SILIntFloat：

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

3) 长度为 160 字节的 reqVeCrashData：

```
struct reqVeCrashData {
    int checksum; //数据包校验码 1234567897
    int copterID; //当前飞机的 ID 号
    int vehicleType; //当前飞机的样式
    int CrashType;//碰撞物体类型，-2 表示地面，-1 表示场景静态物体，0 表示无碰撞，1 以上表示被碰飞机的 ID 号
    double runnedTime; //当前飞机的时间戳
    float VelE[3]; // 当前飞机的速度
    float PosE[3]; //当前飞机的位置
    float CrashPos[3];//碰撞点的坐标
    float targetPos[3];//被碰物体的中心坐标
    float AngEuler[3]; //当前飞机的欧拉角
    float MotorRPMS[8]; //当前飞机的电机转速
```

```
    float ray[6]; //飞机的前后左右上下扫描线
    char CrashedName[20] = {0};//被碰物体的名字
}
```

前面介绍过这个类了，在开启数据回传的情况下，RflySim3D 会为所有 Copter 发送 reqVeCrashData（数据变化时发送，每秒一次）。

8.5.5. getUE4Pos（获得 Copter 在 RflySim3D 中的位置）

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

一、参数解释：

1) copterID：表示获取哪个 Copter 的位置数据。

二、函数解释：

该函数可以获得 Copter 在 RflySim3D 中的位置。

8.5.6. getUE4Data（获得 Copter 在 RflySim3D 中的数据）

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

一、参数解释：

1) copterID：表示获取哪个 Copter 的数据。

二、函数解释：

该函数可以获得 Copter 在 RflySim3D 中的数据，这个数据也是 reqVeCrashData 结构的。

8.5.7. TimeStmploop（接收 CopterSim 回传的心跳和时间戳数据，确保终端和 CopterSim 的连接正常）

```
def TimeStmploop(self)
```

一、参数解释：

二、函数解释：

该方法用于接收 CopterSim 回传的心跳和时间戳数据，确保终端和 CopterSim 的连接正常。以及各系统用于时间对齐的赋值操作

8.5.8. StartTimeStmplisten（子线程中接收 CopterSim 回传的心跳和时间戳数据，确保 CopterSim 连接正常

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

一、参数解释：

1) cpID：CopterSim 索引

二、函数解释：

该方法绑定本机的 20005 端口，在子线程中接收 CopterSim 回传的心跳和时间戳数据，确保终端和 CopterSim 的连接正常，并不阻塞主线程。其中 cpID 表示 CopterSim 的

ID。

8.5.9. sendUE4Attatch（其他 Copter 附加到其他 Copter 上，“附加”的意思是会跟随移动）

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

一、参数解释：

1) CopterIDs：附加到其他 Copter 上的 Copter 的 ID，可以是一个值，也可以是最多 25 个值的 list
2) AttatchIDs：被附加到其他 Copter 上的 Copter 的 ID，可以是一个值，也可以是最多 25 个值的 list
3) AttatchTypes：每一组附加的方式取值为[0,3]，0 表示正常模式（不进行附加），1 表示仅附加位置，不修改姿态，2 表示附加位置与偏航角，3 表示位置与姿态都附加。
4) windowsID：影响发送目标端口，正常组播时取-1 即可，如果需要指定电脑上的指定的 RflySim3D 进行接收，则可以根据目标电脑 IP 与 RflySim3D 程序的窗口标题序号来设置。(这个序号与上面提到的 20010~20029 这 20 个端口是一一对应的关系，例如 RflySim3D-0 就是指的它在监听 20010，RflySim3D-3 就是指的它在监听 20013）
5) CopterIDs ，AttatchIDs ，AttatchTypes 的数量必须一致，它们的含义是“将 CopterIDs[i]以 AttatchTypes[i]的方式附加到 AttatchIDs[i]上，最多 25 组”

二、函数解释：

这个函数是向 RflySim3D 发送一个这样的结构体，三个参数对应结构体中三个成员：

```
# struct VehicleAttatch25 {
#   int checksum;//1234567892
#   int CopterIDs[25];
#   int AttatchIDs[25];
#   int AttatchTypes[25];//0：正常模式，1：相对位置不相对姿态，2：相对位置+偏航（不相对俯仰和滚转），3：相对位置+全姿态（俯仰滚转偏航）
# }
```

它每次可以将最多 25 个 Copter 附加到其他 Copter 上，“附加”的意思是会跟随移动，如果 A 附加在 B 身上，那么 B 移动时，A 也会随之移动，开头提及的文档中有已经写好的例程可以测试它的效果。

8.5.10. sendUE4PosScalePwm20（设置 20 组无人机的状态属性，并设置电机数据）

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

一、参数解释：

1) copterID：设置的 Copter 的 ID，它必须是一个长度为 20 的 list
2) vehicleType：设置的 Copter 的样式（在 xml 中确定），它必须是一个长度为 20 的 list

3) PosE：表示设置的该 Copter 的位置（米，北东地），它必须是一个长度为 60 的 list（每个 Copter 需要三个值 xyz 来表示位置）
4) AngEuler：表示设置的该 Copter 的欧拉角（弧度，roll,pitch,yaw），它必须是一个长度为 60 的 list（每个 Copter 需要三个值 xyz 来表示姿态）
5) Scale：表示设置的该 Copter 的缩放系数，默认为（1，1，1），可以沿着各个轴对 Copter 进行放大或者缩小。（无量纲，分别代表无人机的本地坐标 x,y,z 轴），它必须是一个长度为 60 的 list（每个 Copter 需要三个值 xyz 来表示缩放）
6) PWMs：表示 8 位执行器数据。，它必须是一个长度为 160 的 list（每个 Copter 需要 8 个值的电机数据）
7) isFitGround：表示该无人机是否贴合地面。它必须是一个长度为 20 的 list
8) windowID：影响发送目标端口，正常组播时取-1 即可，如果需要指定电脑上的指定的 RflySim3D 进行接收，则可以根据目标电脑 IP 与 RflySim3D 程序的窗口标题序号来设置。(这个序号与上面提到的 20010~20029 这 20 个端口是一一对应的关系，例如 RflySim3D-0 就是指的它在监听 20010，RflySim3D-3 就是指的它在监听 20013

二、函数解释：

发送 100 个 Copter 的数据，RflySim3D 收到后会更新该 Copter，列表中每一组数据都构成一个无人机的数据，一共有 20 组，它可以设置每个 Copter 的 8 个电机数据。

8.5.11. sendUE4Pos(向局域网内所有 RflySim3D 发送一个 Copter 的数据，如果不存在那么会创建一个)

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

一、参数解释：

1) copterID：设置的 Copter 的 ID
2) vehicleType：设置的 Copter 的样式（在 xml 中确定）
3) MotorRPMSMean：表示 8 位执行器数据的平均值(8 个执行器的值相同)
4) PosE：表示设置的该 Copter 的位置（米，北东地）
5) AngEuler：表示设置的该 Copter 的欧拉角（弧度，roll,pitch,yaw）
6) windowID：影响发送目标端口，正常组播时取-1 即可，如果需要指定电脑上的指定的 RflySim3D 进行接收，则可以根据目标电脑 IP 与 RflySim3D 程序的窗口标题序号来设置。（这个序号与上面提到的 20010~20029 这 20 个端口是一一对应的关系，例如 RflySim3D-0 就是指的它在监听 20010，RflySim3D-3 就是指的它在监听 20013）

二、函数解释：

向局域网内所有 RflySim3D 发送一个 Copter 的数据，如果不存在该 CopterID 的物体，那么会创建一个这样的物体。其中 vehicleType 表示该 Copter 的样式，PosE 表示该 Copter 的位置（米，北东地），AngEuler 表示该 Copter 的欧拉角（弧度，roll,pitch,yaw），MotorRPMSMean 表示 8 位执行器数据的平均值(8 个执行器的值相同)。

8.5.12. sendUE4PosScale100（设置 100 组无人机状态属性）

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

一、参数解释：

1) copterID：设置的 Copter 的 ID，它必须是一个长度为 100 的 list
2) vehicleType：设置的 Copter 的样式（在 xml 中确定），它必须是一个长度为 100 的 list
3) PosE：表示设置的该 Copter 的位置（米，北东地），它必须是一个长度为 300 的 list（每个 Copter 需要三个值 xyz 来表示位置）
4) AngEuler：表示设置的该 Copter 的欧拉角（弧度，roll,pitch,yaw），它必须是一个长度为 300 的 list（每个 Copter 需要三个值 xyz 来表示角度）
5) MotorRPMSMean：表示 8 位执行器数据的平均值(8 个执行器的值相同)，它必须是一个长度为 100 的 list
6) Scale：表示设置的该 Copter 的缩放系数，默认为（1，1，1），可以沿着各个轴对 Copter 进行放大或者缩小。（无量纲，分别代表无人机的本地坐标 x,y,z 轴），它必须是一个长度为 300 的 list（每个 Copter 需要三个值 xyz 来表示缩放）
7) isFitGround：表示该无人机是否贴合地面。它必须是一个长度为 100 的 list
8) windowID：影响发送目标端口，正常组播时取-1 即可，如果需要指定电脑上的指定的 RflySim3D 进行接收，则可以根据目标电脑 IP 与 RflySim3D 程序的窗口标题序号来设置。(这个序号与上面提到的 20010~20029 这 20 个端口是一一对应的关系，例如 RflySim3D-0 就是指的它在监听 20010，RflySim3D-3 就是指的它在监听 20013

二、函数解释：

发送 100 个 Copter 的数据，RflySim3D 收到后会更新该 Copter，列表中每一组数据都构成一个无人机的数据，一共有 100 组。

8.5.13. sendUE4PosScale（发送一个 Copter 的数据，RflySim3D 收到后更新该 Copter，带上了一个缩放系数）

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

一、参数解释：

1) copterID：设置的 Copter 的 ID
2) vehicleType：设置的 Copter 的样式（在 xml 中确定）
3) MotorRPMSMean：表示 8 位执行器数据的平均值(8 个执行器的值相同)
4) PosE：表示设置的该 Copter 的位置（米，北东地）
5) AngEuler：表示设置的该 Copter 的欧拉角（弧度，roll,pitch,yaw）
6) Scale：表示设置的该 Copter 的缩放系数，默认为（1，1，1），可以沿着各个轴对 Copter 进行放大或者缩小。（无量纲，分别代表无人机的本地坐标 x,y,z 轴）
7) windowID：影响发送目标端口，正常组播时取-1 即可，如果需要指定电脑上的指定的 RflySim3D 进行接收，则可以根据目标电脑 IP 与 RflySim3D 程序的窗口标题序号来设置。(这个序号与上面提到的 20010~20029 这 20 个端口是一一对应的关系，

例如 RflySim3D-0 就是指的它在监听 20010，RflySim3D-3 就是指的它在监听 20013

二、函数解释：

发送一个 Copter 的数据，RflySim3D 收到后会更新该 Copter，只是带上了一个缩放系数的参数。

8.5.14. sendUE4Pos2Ground（发送一个 Copter 的数据，RflySim3D 收到后会更新该 Copter，但它可以让 Copter 贴近地面）

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

一、参数解释：

1) copterID：设置的 Copter 的 ID
2) vehicleType：设置的 Copter 的样式（在 xml 中确定）
3) MotorRPMSMean：表示 8 位执行器数据的平均值(8 个执行器的值相同)
4) PosE：表示设置的该 Copter 的位置（米，北东地）
5) AngEuler：表示设置的该 Copter 的欧拉角（弧度，roll,pitch,yaw）
6) windowID：影响发送目标端口，正常组播时取-1 即可，如果需要指定电脑上的指定的 RflySim3D 进行接收，则可以根据目标电脑 IP 与 RflySim3D 程序的窗口标题序号来设置。（这个序号与上面提到的 20010~20029 这 20 个端口是一一对应的关系，例如 RflySim3D-0 就是指的它在监听 20010，RflySim3D-3 就是指的它在监听 20013）

二、函数解释：

向局域网内所有 RflySim3D 发送一个 Copter 的数据，如果不存在该 CopterID 的物体，那么会创建一个这样的物体。其中 vehicleType 表示该 Copter 的样式，PosE 表示该 Copter 的位置（米，北东地），AngEuler 表示该 Copter 的欧拉角（弧度，roll,pitch,yaw），MotorRPMSMean 表示 8 位执行器数据的平均值(8 个执行器的值相同)。和其他发送 Copter 数据的函数的主要区别在于，Z 坐标（也就是 PosE[2]）将不起作用，并且使 Copter 一直贴合地面，直到有新的 Copter 数据为止。

8.5.15. sendUE4PosScale2Ground（发送一个 Copter 的数据，更新该 Copter，带上了一个缩放系数，并设置飞机贴地）

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

一、参数解释：

1) copterID：设置的 Copter 的 ID
2) vehicleType：设置的 Copter 的样式（在 xml 中确定）
3) MotorRPMSMean：表示 8 位执行器数据的平均值(8 个执行器的值相同)
4) PosE：表示设置的该 Copter 的位置（米，北东地）
5) AngEuler：表示设置的该 Copter 的欧拉角（弧度，roll,pitch,yaw）
6) Scale：表示设置的该 Copter 的缩放系数，默认为（1，1，1），可以沿着各个轴对 Copter 进行放大或者缩小。（无量纲，分别代表无人机的本地坐标 x,y,z 轴）

7) windowID：影响发送目标端口，正常组播时取-1 即可，如果需要指定电脑上的指定的 RflySim3D 进行接收，则可以根据目标电脑 IP 与 RflySim3D 程序的窗口标题序号来设置。(这个序号与上面提到的 20010~20029 这 20 个端口是一一对应的关系，例如 RflySim3D-0 就是指的它在监听 20010，RflySim3D-3 就是指的它在监听 20013

二、函数解释：

发送一个 Copter 的数据，RflySim3D 收到后会更新该 Copter，只是带上了一个缩放系数的参数，并且设置的飞机是贴地的。

8.5.16. sendUE4PosNew（发送一个 Copter 的数据，RflySim3D 收到后会更新该 Copter）

```
    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、参数解释：

1) copterID：设置的 Copter 的 ID
2) vehicleType：设置的 Copter 的样式（在 xml 中确定）
3) PosE：表示设置的该 Copter 的位置（米，北东地）
4) AngEuler：表示设置的该 Copter 的欧拉角（弧度，roll,pitch,yaw）
5) VelE：表示无人机的速度（米/秒，北东地）
6) PWMs：表示 8 位执行器数据
7) runnedTime：表示该无人机的时间戳（目前无实际效果）
8) windowID：影响发送目标端口，正常组播时取-1 即可，如果需要指定电脑上的指定的 RflySim3D 进行接收，则可以根据目标电脑 IP 与 RflySim3D 程序的窗口标题序号来设置。(这个序号与上面提到的 20010~20029 这 20 个端口是一一对应的关系，例如 RflySim3D-0 就是指的它在监听 20010，RflySim3D-3 就是指的它在监听 20013

二、函数解释：

发送一个 Copter 的数据，RflySim3D 收到后会更新该 Copter，只是参数有所不同。

8.5.17. sendUE4PosSimple（发送一个 Copter 的数据，RflySim3D 收到后会更新该 Copter）

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

一、参数解释：

1) copterID：设置的 Copter 的 ID
2) vehicleType：设置的 Copter 的样式（在 xml 中确定）
3) PWMs：表示 8 位执行器数据
4) VelE：表示无人机的速度（米/秒，北东地）
5) PosE：表示设置的该 Copter 的位置（米，北东地）
6) AngEuler：表示设置的该 Copter 的欧拉角（弧度，roll,pitch,yaw）

7) runnedTime：表示该无人机的时间戳（目前无实际效果）
8) windowID：影响发送目标端口，正常组播时取-1 即可，如果需要指定电脑上的指定的 RflySim3D 进行接收，则可以根据目标电脑 IP 与 RflySim3D 程序的窗口标题序号来设置。(这个序号与上面提到的 20010~20029 这 20 个端口是一一对应的关系，例如 RflySim3D-0 就是指的它在监听 20010，RflySim3D-3 就是指的它在监听 20013

二、函数解释：

发送一个 Copter 的数据，RflySim3D 收到后会更新该 Copter，只是参数有所不同。

8.5.18. sendUE4PosFull（发送一个 Copter 的数据，RflySim3D 收到后会更新该 Copter）

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

一、参数解释：

1) copterID：设置的 Copter 的 ID
2) vehicleType：设置的 Copter 的样式（在 xml 中确定）
3) MotorRPMS：表示 8 位执行器数据
4) VelE：表示无人机的速度（米/秒，北东地）
5) PosE：表示设置的该 Copter 的位置（米，北东地）
6) RateB：表示无人机的角速度（弧度/秒）
7) AngEuler：表示设置的该 Copter 的欧拉角（弧度，roll,pitch,yaw）
8) windowID：影响发送目标端口，正常组播时取-1 即可，如果需要指定电脑上的指定的 RflySim3D 进行接收，则可以根据目标电脑 IP 与 RflySim3D 程序的窗口标题序号来设置。(这个序号与上面提到的 20010~20029 这 20 个端口是一一对应的关系，例如 RflySim3D-0 就是指的它在监听 20010，RflySim3D-3 就是指的它在监听 20013

二、函数解释：

发送一个 Copter 的数据，RflySim3D 收到后会更新该 Copter，只是多了一些参数可以供设置。

8.5.19. sendUE4ExtAct（向目标 Copter 发送一组数据，由目标的自定义函数进行处理，可以实现各种自定义的效果）

```
    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、参数解释：

1) copterID：设置的 Copter 的 ID
2) ActExt：16 维的蓝图接口参数，它的值与目标无人机的蓝图有关（目标无人机必须实现了相关蓝图函数）。
3) windowID：影响发送目标端口，正常组播时取-1 即可，如果需要指定电脑上的指定的 RflySim3D 进行接收，则可以根据目标电脑 IP 与 RflySim3D 程序的窗口标题序号来设置。(这个序号与上面提到的 20010~20029 这 20 个端口是一一对应的关系，例如 RflySim3D-0 就是指的它在监听 20010，RflySim3D-3 就是指的它在监听 20013

二、函数解释：

向目标 Copter 发送一组数据，由目标的自定义函数进行处理，可以实现各种自定义的效果。

9. Python 飞机控制接口 PX4MavCtrlV4.py

下面展示一段通过 PX4MavCtrlV4.py 接口连接 CopeterSim ,然后控制飞机起飞的代码

```
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. 初始化

9.1.1. InitMavLoop（开启 MAVLink 监听 CopterSim 数据，并实时更新）

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

一、参数解释：

1) UDPMode：UDP 模式

二、函数解释：

开启 MAVLink 监听 CopterSim 数据，并实时更新。其中 UDPModel 值为：

0：对应 UDP_Full 模式，Python 传输完整的 UDP 数据给 CopterSim，传输数据量小；CopterSim 收到数据后，再转换为 Mavlink 后传输给 PX4 飞控；适合中小规模集群（数量小于 10）仿真。

1：对应 UDP_Simple 模式，数据包大小与发送频率比 UDP_Full 模式小；适合大规模集群仿真，无人机数量小于 100。

2：对应 Mavlink_Full 模式（默认模式），Python 直接发送 MAVLink 消息给 CopterSim，再转发给 PX4，数据量较大适合单机控制；适合单机或少量飞机仿真，无人机数量小于 4；

3: 对应 Mavlink_Simple 模式，会屏蔽部分 MAVLink 消息包，并降低数据频率，发送数据量比 MAVLink_Full 小很多，适合多机集群控制；适合小规模集群仿真，无人机数量小于 8。

4: 对应 Mavlink_NoSend 模式，CopterSim 不会向外发送 MAVLink 数据，此模式需要配合硬件在环仿真+数传串口通信，通过有线方式传输 MAVLink，此模式局域网内数据量最小，适合分布式视觉硬件在环仿真，无人机数量不限制。

9.1.2. endMavLoop（停止接收 CopterSim 数据，和 stopRun 效果一致）

```
def endMavLoop(self)
```

一、参数解释：

二、函数解释：

停止接收 CopterSim 数据，和 stopRun 效果一致。

9.1.3. stopRun（退出 MAVLink 数据接收模式）

```
def stopRun(self)
```

一、参数解释：

二、函数解释：

退出 MAVLink 数据接收模式，和 endMavLoop 效果一致。

9.2. offboard 结束退出

9.2.1. initOffboard（开启 offboard 模式，并启动线程，循环发送 offboard 消息）

```
def initOffboard(self)
```

一、参数解释：

二、函数解释：

开启 offboard 模式，并启动线程，循环发送 offboard 消息。

9.2.2. initOffboard2（开启 offboard 模式，并启动线程，循环发送 offboard 消息）

```
def initOffboard2(self)
```

一、参数解释：

二、函数解释：

开启 offboard 模式，并启动线程，循环发送 offboard 消息。

9.2.3. OffboardSendMode（循环发送 offboard 数据）

```
def OffboardSendMode(self)
```

一、参数解释：

二、函数解释：

循环发送 offboard 数据。

9.2.4. endOffboard（发送 MAV_CMD_NAV_GUIDED_ENABLE 命令退出 offboard 模式，并停止发送 offboard 消息）

```
def endOffboard(self)
```

一、参数解释：

二、函数解释：

发送 MAV_CMD_NAV_GUIDED_ENABLE 命令退出 offboard 模式，并停止发送 offboard 消息。

9.3. 解锁

9.3.1. SendMavArm（无人机解锁或上锁）

```
def endOffboard(self)
```

一、参数解释：

1) isArm：0：锁定；1：解锁

二、函数解释：

发送 MAV_CMD_COMPONENT_ARM_DISARM 命令解锁或上锁。

9.4. 飞机指点飞行接口

9.4.1. SendPosNED（发送北东地坐标系下的目标位置和航向角接口）

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

一、参数解释：

1) x： x 坐标（米）
2) y： y 坐标（米）
3) z： z 坐标（米），飞机高于地面时，z 小于 0
4) yaw：航向角（弧度）

二、函数解释：

发送北东地坐标系下的目标位置和航向角到 PX4。

9.4.2. SendPosNEDNoYaw（发送北东地坐标系下的目标位置）

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

一、参数解释：

1) x： x 坐标（米）
2) y： y 坐标（米）
3) z： z 坐标（米），飞机高于地面时，z 小于 0

二、函数解释：

发送北东地坐标系下的目标位置到 PX4。

9.4.3. SendPosNEDExt（选择北东地或机体前右下坐标系发送目标位置到 PX4）

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

一、参数解释：

1) x：x 坐标（米）
2) y：y 坐标（米）
3) z：z 坐标（米）
4) mode：模式标志位
5) isNED：True 时坐标系为北东地，False 时为机体前右下坐标系

二、函数解释：

发送目标位置到 PX4。

9.4.4. SendPosGlobal（地球坐标系（经纬度，高度）带偏航角指点飞行接口）

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

一、参数解释：

1) lat：经度
2) lon：纬度
3) alt：高度
4) yawValue：航向角或航向角速度
5) yawType：标志位（0：无航向角和航向角速度；1：航向角控制；2：航向角速度控制）

二、函数解释：

发送北东地坐标系下的目标位置和航向到 PX4。

9.4.5. sendMavLand（发送三维坐标、MAV_CMD_NAV_LAND 命令使飞机降落到目标位置）

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

一、参数解释：

1) xM：x 坐标（米）
2) yM：y 坐标（米）
3) zM：z 坐标（米）

二、函数解释：

发送 MAV_CMD_NAV_LAND 命令使飞机降落到目标位置。

9.4.6. sendMavTakeOffLocal（发送三维坐标、航向角、俯仰角、起飞速度 MAV_CMD_NAV_TAKEOFF_LOCAL 命令使飞机飞行到目标位置）

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

一、参数解释：

1) xM：x 坐标（米）
2) yM：y 坐标（米）
3) zM：z 坐标（米）
4) YawRad：航向角（弧度）
5) PitchRad：俯仰角（弧度）
6) AscendRate：起飞速度

二、函数解释：

发送 MAV_CMD_NAV_TAKEOFF_LOCAL 命令使飞机飞行到目标位置。

9.4.7. sendMavTakeOff（发送三维坐标、航向角、俯仰角、MAV_CMD_NAV_TAKEOFF 命令使飞机起飞到目标位置）

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

一、参数解释：

1) xM：x 坐标（米，北东地）
2) yM：y 坐标（米，北东地）
3) zM：z 坐标（米，北东地）
4) YawRad：航向角（弧度）
5) PitchRad：俯仰角（弧度）

二、函数解释：

发送 MAV_CMD_NAV_TAKEOFF 命令使飞机起飞到目标位置。

9.4.8. FRD 坐标系下的指点飞行接口

9.4.8.1. SendPosFRD（机体坐标系（前右下）下的目标位置和航向角)

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

一、参数解释：

1) x：x 坐标（米）
2) y：y 坐标（米）
3) z：z 坐标（米）
4) yaw：航向角（弧度）

二、函数解释：

发送机体坐标系（前右下）下的目标位置和航向角到 PX4。

9.4.8.2. SendPosFRDNoYaw（机体坐标系（前右下）下的目标位置到 PX4）

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

一、参数解释：

1) x：x 坐标（米）
2) y：y 坐标（米）
3) z：z 坐标（米）

二、函数解释：

发送机体坐标系（前右下）下的目标位置到 PX4。

9.4.8.3. SendPosNEDExt（选择北东地或机体前右下坐标系发送目标位置到 PX4）

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

一、参数解释：

1) x：x 坐标（米）
2) y：y 坐标（米）
3) z：z 坐标（米）
4) mode：模式标志位
5) isNED：True 时坐标系为北东地，False 时为机体前右下坐标系

二、函数解释：

发送目标位置到 PX4。

9.4.9. FRD 坐标系下无偏航角的指点飞行接口

9.4.9.1. SendPosFRDNoYaw（机体坐标系（前右下）下的目标位置到 PX4）

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

一、参数解释：

1) x：x 坐标（米）
2) y：y 坐标（米）
3) z：z 坐标（米）

二、函数解释：

发送机体坐标系（前右下）下的目标位置到 PX4。

9.4.9.2. SendPosNEDExt（选择北东地或机体前右下坐标系发送目标位置到 PX4）

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

一、参数解释：

1) x：x 坐标（米）
2) y：y 坐标（米）
3) z：z 坐标（米）
4) mode：模式标志位
5) isNED：True 时坐标系为北东地，False 时为机体前右下坐标系

二、函数解释：

发送目标位置到 PX4。

9.5. 飞机速度控制接口

9.5.1. FRD 坐标系下的速度控制接口

9.5.1.1. SendVelFRD（机体坐标系（前右下）下的目标速度和航向角速度）

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

一、参数解释：

1) vx：x 方向速度（米/秒，前右下）
2) vy：y 方向速度（米/秒，前右下）
3) vz：z 方向速度（米/秒，前右下），向上飞时，vz 小于 0
4) yawrate：航向角速度（弧度/秒）

二、函数解释：

发送机体坐标系（前右下）下的目标速度和航向角速度。

9.5.1.2. SendVelNoYaw（机体坐标系目标速度）

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

一、参数解释：

1) vx： x 方向速度（米/秒）

2) vy： y 方向速度（米/秒）
3) vz： z 方向速度（米/秒），向上飞时，vz 小于 0

二、函数解释：

发送机体坐标系（前右下）下的目标速度到 PX4。

9.5.2. 不控飞机偏航角的速度控制接口

9.5.2.1. SendVelNEDNoYaw（北东地坐标系下的速度控制）

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

一、参数解释：

1) vx：x 方向速度（米/秒，北东地）
2) vy：y 方向速度（米/秒，北东地）
3) vz：z 方向速度（米/秒，北东地），向上飞时，vz 小于 0

二、函数解释：

发送北东地坐标系下的速度控制。

9.5.2.2. SendGroundSpeed（发送命令修改 GND_SPEED_TRIM，设置飞机的地面速度）

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

一、参数解释：

1) Speed：地面速度（米/秒）

二、函数解释：

发送命令修改 GND_SPEED_TRIM，设置飞机的地面速度。

9.5.2.3. SendCopterSpeed（发送命令修改 MPC_XY_VEL_MAX，设置飞机的最大速度）

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

一、参数解释：

1) Speed：最大速度（米/秒）

二、函数解释：

发送命令修改 MPC_XY_VEL_MAX，设置飞机的最大速度。

9.5.2.4. SendCruiseSpeed（发送命令修改 FW_AIRSPD_TRIM 参数，设置飞机的巡航速度）

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

一、参数解释：

1) Speed：巡航速度（米/秒）

二、函数解释：

发送命令修改 FW_AIRSPD_TRIM 参数，设置飞机的巡航速度。

9.5.3. 带偏航角速度以及高度控制接口

9.5.3.1. SendVelYawAlt（北东地坐标系下目标速度、航向角、高度到 PX4）

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

一、参数解释：

1) vel：速度
2) yaw： 航向角
3) alt： 高度

二、函数解释：

发送北东地坐标系下目标速度、航向角、高度到 PX4。

9.5.3.2. SendPosGlobal（北东地坐标系航向角速度、高度控制）

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

一、参数解释：

1) lat：经度
2) lon：纬度
3) alt：高度
4) yawValue：航向角或航向角速度
5) yawType：标志位（0：无航向角和航向角速度；1：航向角控制；2：航向角速度控制）

二、函数解释：

发送北东地坐标系下的目标位置和航向到 PX4。

9.5.3.3. sendMavLandGPS（发送经度、纬度、高度、MAV_CMD_NAV_LAND 命令使飞机降落到目标位置）

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

一、参数解释：

1) lat：经度
2) lon：纬度
3) alt：高度

二、函数解释：

发送 MAV_CMD_NAV_LAND 命令使飞机降落到目标位置。

9.5.3.4. sendMavTakeOffGPS（发送经度、纬度、高度、航向角、俯仰角、MAV_CMD_NAV_TAKEOFF 命令使飞机飞行到目标位置）

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

一、参数解释：

1) lat：经度
2) lon：纬度
3) alt：高度
4) yawDeg：航向角（度）
5) pitchDeg：俯仰角（度）

二、函数解释：

发送 MAV_CMD_NAV_TAKEOFF 命令使飞机飞行到目标位置。

9.5.3.5. sendTakeoffMode（发送起飞高度命令将飞控改为 takeoff 模式，使飞机起飞)

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

一、参数解释：

1) alt：起飞高度

二、函数解释：

发送命令将飞控改为 takeoff 模式，使飞机起飞。

9.6. 飞机姿态控制接口

9.6.1. SendAttPX4（FRD 坐标系下的姿态控制接口）

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

一、参数解释：

1) att： 姿态输入
2) thrust： 油门
3) CtrlFlag：控制标志（0：att 是 3 维欧拉角，单位度；1：att 是 3 维欧拉角，单位弧度；2：att 是四元数；3：att 是 3 维角速度，单位弧度/秒；4：att 是 3 维角速度，单位度/秒）
4) AltFlg： 高度标志

二、函数解释：

发送机体坐标系（前右下）下的目标姿态到 PX4。

9.7. 飞机加速度控制接口

9.7.1. SendAccPX4（发送目标加速度到 PX4）

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

一、参数解释：

1) afx： x 方向加速度
2) afy： y 方向加速度
3) afz： z 方向加速度
4) yawValue： 航向角或航向角速度
5) yawType：加速度控制模式（0：无航向角和航向角速度；1：航向角控制；2：航向角速度控制）
6) frameType：坐标系（0：北东地；1：机体前右下）

二、函数解释：

发送目标加速度到 PX4。

9.8. 载体，载体视觉传感器，目标等属性接口

9.8.1. 请求载体飞机属性

9.8.1.1. getUE4Data（获得 Copter 在 RflySim3D 中的数据）

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

一、参数解释：

1) copterID：表示获取哪个 Copter 的数据。

二、函数解释：

该函数可以获得 Copter 在 RflySim3D 中的数据，这个数据也是 reqVeCrashData 结构的。

9.8.1.2. getUE4Pos（获得 Copter 在 RflySim3D 中的位置）

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

一、参数解释：

1) copterID：表示获取哪个 Copter 的位置数据。

二、函数解释：

该函数可以获得 Copter 在 RflySim3D 中的位置

9.8.2. 请求载体视觉传感器接口

9.8.2.1. CameraData（由 RflySim3D 返回的数据，发送的某个相机（视觉传感器）的信息）

一. __init__

```
def __init__(self):
        self.checksum = 0 #1234567891 作为校验
        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
```

由 RflySim3D 返回的数据，PX4MavCtrler.reqCamCoptObj()函数调用 RflySim3D 的“RflyReqObjData”命令后，RflySim3D 根据该命令的参数，发送的某个相机（视觉传感器）的信息（向 RflySim3D 发送命令时，附带了该相机（视觉传感器）的 SeqID）。

这是该类的构造函数，其中：

11. checksum：是数据的校验位，用于检验数据传输过程中是否发生了异常。
12. seqID：视觉传感器的 ID
13. TypeID：视觉传感器的类型（RPG、深度图等）
14. DataHeight：数据高度(像素)
15. DataWidth：数据宽度(像素)
16. CameraFOV：相机水平视场角（单位度）
17. PosUE：表示该物体的位置（米，北东地）
18. angEuler：表示该 Camera 的角度（弧度，Roll、Pitch、Yaw）
19. Timestmp：时间戳
20. hasUpdate：这个值不是 RflySim3D 发过来的

9.8.3. 请求场景内目标数据接口

9.8.3.1. getCamCoptObj（从 RflySim3D 获得指定的数据，并将监听到的三种存放到了 3 个列表中）

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

一、参数解释：

1) Type：0 表示相机，1 表示飞机，2 表示物体

二、函数解释：

从 RflySim3D 获得指定的数据，UE4MsgRecLoop 函数开启了监听 RflySim3D 的消息，并

将监听到的三种存放到了 3 个列表中，此函数正是在这个列表中搜索数据，因此需要先使用 reqCamCoptObj 函数向 RflySim3D 请求相关的数据才行。

9.8.3.2. reqCamCoptObj（请求场景中物体的数据（并不能创建新的物体，而是获得已存在物体的数据））

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

一、参数解释：

1) type：0 表示相机，1 表示飞机，2 表示物体
2) objName：type 表示相机 seqID；s 表示飞机时，objName 对应 CopterID；表示物体时，objName 对应物体名字
3) windowID：表示想往哪个 RflySim3D 发送消息，默认是 0 号窗口。（不要给所有 RflySim3D 发送该数据，因为返回的值都是一样，没必要额外消耗性能）

二、函数解释：

向 RflySim3D 发送一个数据请求，可以请求场景中物体的数据（并不能创建新的物体，而是获得已存在物体的数据）。可以是视觉传感器、Copter、场景中普通的物体。获得的数据详见 CoptReqData 类、ObjReqData 类、CameraData 类。

9.9. 其他内部函数

9.9.1. sendStartMsg（唤醒所有飞机）

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

一、参数解释：

1) copterID：待唤醒的 Copter ID

二、函数解释：

通过 udp 发送消息到“224.0.0.10”，“20007”来唤醒执行 waitForStartMSG，copterID 是-1 时，将唤醒所有飞机，消息格式：

```
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（程序阻塞直到收到 sendStartMsg 的消息）

```
def waitForStartMsg(self)
```

一、参数解释：

二、函数解释：

程序阻塞直到收到 sendStartMsg 的消息。

9.9.3. initPointMassModel（创建一个质点无人机模型）

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

一、参数解释：

1) intAlt：初始高度

2) intState：初始 X（米），Y（米），Yaw（角度）

二、函数解释：

创建一个质点无人机模型，设定初始地面高度，XY 位置和偏航角度，并创建线程监听位置和速度指令。质点无人机模型可以提供软硬件在环相近的无人机动态效果，但是极大降低对电脑性能的占用和提升飞机平稳性。

9.9.4. EndPointMassModel（结束质点模型）

```
def EndPointMassModel(self)
```

一、参数解释：

二、函数解释：

结束质点模型，终止 initPointMassModel 中创建的线程。

9.9.5. yawSat（返回-PI 到 PI 之间的航向角）

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

一、参数解释：

1) yaw：航向角

二、函数解释：

返回-PI 到 PI 之间的航向角。

9.9.6. PointMassModelLoop （质点模型处理消息的死循环）

```
def PointMassModelLoop(self)
```

一、参数解释：

二、函数解释：

质点模型处理消息的死循环，通过目标位置或速度计算飞机的运动，并将位置和姿态消息发送到 UE，达到质点模型仿真的效果。

9.9.7. InitTrueDataLoop（启动两个线程，分别接收 CopterSim 的真实数据和 PX4 数据）

```
def InitTrueDataLoop(self)
```

三、参数解释：

四、函数解释：

启动两个线程，分别接收 CopterSim 的真实数据和 PX4 数据。

9.9.8. EndTrueDataLoop（停止 InitTrueDataLoop 的线程监听）

```
def EndTrueDataLoop(self)
```

一、参数解释：

二、函数解释：

停止 InitTrueDataLoop 的线程监听。

9.9.9. initUE4MsgRec（启动一个 t4(self.UE4MsgRecLoop)开始监听 224.0.0.10：20006）

```
def initUE4MsgRec(self)
```

三、参数解释：

四、函数解释：

初始化 self.udp_socketUE4，并且启动一个线程 t4(self.UE4MsgRecLoop)开始监听 224.0.0.10：20006，以及自身的 20006 端口。

9.9.10. endUE4MsgRec（停止线程 t4(self.UE4MsgRecLoop)的监听）

```
def endUE4MsgRec(self)
```

三、参数解释：

四、函数解释：

停止线程 t4(self.UE4MsgRecLoop)的监听。

9.9.11. UE4MsgRecLoop（用于处理 RflySim3D 或 CopterSim 返回的消息）

```
def UE4MsgRecLoop(self)
```

三、参数解释：

四、函数解释：

它是监听 224.0.0.10：20006，以及自身的 20006 端口的处理函数，用于处理 RflySim3D 或 CopterSim 返回的消息，一共有 6 种消息：

7) 长度为 12 字节的 CopterSimCrash：

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

由 RflySim3D 返回的碰撞数据，RflySim3D 中按 P 键时 RflySim3D 会开启碰撞检测模

式，如果发生了碰撞，会传回这个数据，P+数字可以选择发送模式（0 本地发送，1 局域网发送，2 局域网只碰撞时发送），默认为本地发送。

8) 长度为 120 字节的 PX4SILIntFloat：

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

9) 长度为 160 字节的 reqVeCrashData：

```
struct reqVeCrashData {
     int checksum; //数据包校验码 1234567897
     int copterID; //当前飞机的 ID 号
     int vehicleType; //当前飞机的样式
     int CrashType;//碰撞物体类型，-2 表示地面，-1 表示场景静态物体，0 表示无碰撞，1 以上表示被碰飞机的 ID 号
     double runnedTime; //当前飞机的时间戳
     float VelE[3]; // 当前飞机的速度
     float PosE[3]; //当前飞机的位置
     float CrashPos[3];//碰撞点的坐标
     float targetPos[3];//被碰物体的中心坐标
     float AngEuler[3]; //当前飞机的欧拉角
     float MotorRPMS[8]; //当前飞机的电机转速
     float ray[6]; //飞机的前后左右上下扫描线
     char CrashedName[20] = {0};//被碰物体的名字
}
```

前面介绍过这个类了，在开启数据回传的情况下，RflySim3D 会为所有 Copter 发送 reqVeCrashData（数据变化时发送，每秒一次）。

10) 长度为 56 字节的 CameraData：

```
struct CameraData { //56
     int checksum = 0;//1234567891
     int SeqID; //相机序号
     int TypeID;//相机类型
     int DataHeight;//像素高
     int DataWidth;//像素宽
     float CameraFOV;//相机视场角
     float PosUE[3]; //相机中心位置
     float angEuler[3];//相机欧拉角
     double timestmp;//时间戳
};
```

RflySim3D 根据 reqCamCoptObj 函数发送的命令，定时返回的该结构体，该程序接收到后会存放在 self.CamDataVect 中。

11) 长度为 64 字节的 CoptReqData：

```
struct CoptReqData { //64
     int checksum = 0; //1234567891 作为校验
     int CopterID;//飞机 ID
     float PosUE[3]; //物体中心位置（人为三维建模时指定，姿态坐标轴，不一定在几何中心）
     float angEuler[3];//物体欧拉角
     float boxOrigin[3];//物体几何中心坐标
     float BoxExtent[3];//物体外框长宽高的一半
     double timestmp;//时间戳
};
```

RflySim3D 根据 reqCamCoptObj 函数发送的命令，定时返回的该结构体，该程序接收到

后会存放在 self.CoptDataVect 中。

12) 长度为 96 字节的 ObjReqData：

```
struct ObjReqData { //96
    int checksum = 0; //1234567891 作为校验
    int seqID = 0;
    float PosUE[3]; //物体中心位置（人为三维建模时指定，姿态坐标轴，不一定在几何中心）
    float angEuler[3];//物体欧拉角
    float boxOrigin[3];//物体几何中心坐标
    float BoxExtent[3];//物体外框长宽高的一半
    double timestmp;//时间戳
    char ObjName[32] = { 0 };//碰物体的名字
};
```

RflySim3D 根据 reqCamCoptObj 函数发送的命令，定时返回的该结构体，该程序接收到后会存放在 self.ObjDataVect 中。收到的数据可以使用 getCamCoptObj 函数进行获取。

9.9.12. sat（inPWM 限幅）

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

一、参数解释：

1) inPwm：输入值
2) thres：上限绝对值

二、函数解释：

如果 inPwm 小于-thres，则返回-thres；如果 inPwm 大于 thres，则返回 thres；其余情况返回 inPwm。

9.9.13. SendMavCmdLong（发送 MAVLINK 消息的 command_long 消息，可以请求无人机执行某些操作 ）

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

一、参数解释：

1) command：命令类型
2) param1：命令参数 1
3) param2：命令参数 2
4) param3：命令参数 3
5) param4：命令参数 4
6) param5：命令参数 5
7) param6：命令参数 6
8) param7：命令参数 7

二、函数解释：

发送 MAVLINK 消息的 command_long 消息，可以请求无人机执行某些操作。详细介绍参考：

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

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

9.9.14. sendMavOffboardCmd（发送 offboard 命令到飞控）

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

一、参数解释：

1) type_mask：掩码，表示应该忽略哪些项
2) coordinate_frame：坐标系
3) x：x 坐标（米，北东地）
4) y：y 坐标（米，北东地）
5) z：z 坐标（米，北东地）
6) vx：x 方向速度（米/秒，北东地）
7) vy：y 方向速度（米/秒，北东地）
8) vz：z 方向速度（米/秒，北东地）
9) afx：x 方向加速度或力（米/秒^2 或牛，北东地）
10) afy：y 方向加速度或力（米/秒^2 或牛，北东地）
11) afz：z 方向加速度或力（米/秒^2 或牛，北东地）
12) yaw：航向角（弧度）
13) yaw_rate：航向角速度（弧度/秒）

二、函数解释：

发送 offboard 命令到飞控，使其进入 offboard 模式。详细参考：
https://mavlink.io/en/messages/common.html#SET_POSITION_TARGET_LOCAL_NED

9.9.15. TypeMask（获取 offboard 消息需要的 typemask）

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

一、参数解释：

1) Enlist：有效项列表，0~5 分别表示位置、速度、加速度、力、航向角、航向角速度。

二、函数解释：

获取 offboard 消息需要的 typemask，确定哪些信息是有效的。可参考：
https://mavlink.io/en/messages/common.html#POSITION_TARGET_TYPEMASK

9.9.16. sendMavOffboardAPI（更新 offboard 消息的数据）

```
    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、参数解释：

1) type_mask：掩码，表示应该忽略哪些项
2) coordinate_frame：坐标系
3) pos：位置
4) vel：速度
5) acc：加速度
6) yaw：航向角

7) yawrate：航向角速度

二、函数解释：

更新 offboard 消息的数据（该数据会以一定频率发送）。

9.9.17. sendUDPSimpData（设置控制模式、控制量）

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

一、参数解释：

1) ctrlMode：控制模式
2) ctrls：控制量

二、函数解释：

发送 udp 消息，其中 ip 和 port 由新建本类实例时指定。消息结构为：

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

9.9.18. sendMavSetParam（发送命令到 PX4 修改指定参数）

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

一、参数解释：

1) param_id：参数 id
2) param_value：参数值
3) param_type：参数类型

二、函数解释：

发送命令到 PX4 修改指定参数，参数介绍参考：

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

9.9.19. SendSetMode（发送 MAV_CMD_DO_SET_MOD 设置飞机模式）

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

一、参数解释：

1) mainmode：主模式
2) cusmode：子模式

二、函数解释：

发送 MAV_CMD_DO_SET_MOD 设置飞机模式，参考：

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

9.9.20. getTrueDataMsg（循环监听真实数据，并更新内置状态）

```
def getTrueDataMsg(self)
```

一、参数解释：

二、函数解释：

循环监听真实数据，并更新内置状态。

9.9.21. getPX4DataMsg（循环监听 PX4 数据，并更新内置状态）

```
def getPX4DataMsg(self)
```

一、参数解释：

二、函数解释：

循环监听 PX4 数据，并更新内置状态。

9.9.22. getMavMsg（循环更新 MAVLink 接收的数据）

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

一、参数解释：

二、函数解释：

循环更新 MAVLink 接收的数据。

9.9.23. sendCustomData（发送一个 16 维数据到指定端口，本接口可用于与 Simulink 通信）

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

一、参数解释：

1) copterID：copter ID
2) data：数据内容
3) checksum：校验位
4) port：端口号
5) IP：IP 地址

二、函数解释：

发送一个 16 维数据到指定端口，本接口可用于与 Simulink 通信。对应数据结构为：

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

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

10. 开发中需要用到的 mavros/ROS 相关接口

10.1. ROS 中常用通信方式介绍

10.1.1. 消息话题通信

ROS 是多进程运行的机器人操作系统，各进程通常称为 ROS 节点，各节点间进行数据通信交流可以通过消息话题的通信方式。话题通信基于发布-订阅模式，允许节点以异步的方式发送和接收消息。

发布者（Publisher）：一个 ROS 节点可以充当发布者，负责将数据发布到一个或多个话题（Topics）。发布者将消息发布到特定的话题，其他节点可以通过订阅该话题来接收消息。

订阅者（Subscriber）：另一个 ROS 节点可以充当订阅者，它可以订阅一个或多个话题。订阅者通过监听话题来接收从发布者发送的消息。当有新消息发布到话题上时，订阅者会异步地接收并处理这些消息。

话题（Topic）：话题是一种消息通信通道，用于发布和订阅消息。话题具有唯一的名称，ROS 节点可以通过名称来标识并与其连接。发布者将消息发布到话题上，而订阅者从话题上接收消息。

消息（Message）：消息是在 ROS 中用于传输数据的结构化数据单元。ROS 使用消息定义来描述消息的结构和数据类型。发布者和订阅者之间共享相同类型的消息，以确保数据的一致性。

ROS Master：ROS Master 是一个 ROS 系统中的核心组件，负责管理节点之间的通信。它维护一个注册表，记录了所有节点、话题和服务的信息。发布者和订阅者通过 ROS Master 来发现和连接到彼此。

发布者-订阅者关系：发布者节点将消息发布到特定话题上，而订阅者节点通过告知 ROS Master 它对哪些话题感兴趣来注册订阅关系。当有新消息发布到订阅者感兴趣的话题上时，ROS Master 会将消息路由到相应的订阅者。

基本来说，ROS 话题通信的原理是通过发布者将消息发布到话题上，订阅者监听话题以接收消息。这种发布-订阅模式的好处是它允许节点之间进行异步通信，提高了系统的灵活性和可扩展性。节点可以根据需要发布或订阅不同的话题，以满足各种通信需求。

10.1.2. 服务类型通信

ROS（Robot Operating System）中的服务通信是一种用于节点之间请求和提供有状态的操作或服务的通信机制。与话题通信不同，服务通信是一种请求-响应模式，其中一个节点请求另一个节点执行特定的任务，并等待响应。

服务提供者（Service Provider）：服务提供者是一个 ROS 节点，它实现了一个或多个服务。服务提供者负责接收来自其他节点的服务请求，并执行相应的任务。通常，服务提供者会在节点启动时注册它所提供的服务。

服务请求者（Service Requester）：服务请求者也是一个 ROS 节点，它需要执行特定的

任务，并向服务提供者发送服务请求。服务请求者通过调用服务的客户端接口来发起请求。

服务（Service）：服务是一种定义了请求和响应消息的 ROS 通信机制。服务由两个部分组成：请求消息（Request Message）和响应消息（Response Message）。请求消息用于传递服务请求的参数，而响应消息用于返回服务执行的结果。

ROS Master：ROS Master 在服务通信中也扮演重要角色，它负责服务的注册和发现。服务提供者在启动时向 ROS Master 注册其提供的服务，而服务请求者通过 ROS Master 查找并连接到所需的服务。

服务通信的原理是建立了一种请求和响应的通信模式，允许节点之间进行有状态的交互。服务通信适用于需要节点之间协作执行任务的情况，例如传感器数据的请求和处理，运动控制的调整，或其他需要同步处理的任务。

10.2. ROS 在 RflySim 平台上应用介绍

10.2.1. 通过话题获取视觉传感器数据接口

RflySim 平台支持获取 RGB 图像，深度图像，灰度图像，激光点云，红外图像，分割图等详细参考支持的传感器列表及配置方法，对应的话题名/rflysim/sensro*/type，话题名中 sensor*，此处的*为 Config.json 文件中的 seq_id，而 type 为数据类型，如 RGB 图对应的即是 img_rgb，类似深度对应的是 img_depth， 灰度图对应的是 img_gray，详细参考例程 PythonVisionAPI\1-APIUsageDemos\12-VisCaptureMergeROSAPI

10.2.2. 通过话题获取 IMU 数据接口

RflySim 默认发布的话题名为/rflysim/imu,让如果使用 mavros 仿真也可使用 mavros 发布的 imu 数据（/mavros/imu/data_raw）。使用 RflySim 发布的 imu 数据时，在 VisionCaptureAPI.py 接口中有对应的接口参考**错误!未定义书签。错误!未找到引用源。**，对应的例程在 PythonVisionAPI\1-APIUsageDemos\12-VisCaptureMergeROSAPI

10.3. MavROS 常用接口

10.3.1. 话题 /mavros/state 获得飞控状态信息

通过这个话题可以查看飞控状态信息，如是否成功连接飞控，飞控当前的模式，飞控是否已经解锁等，通常我们再发送指令给飞控值，往往需要监听这个话题的数据，比如飞行模式切换，解锁命令。服务通信有两个字段，请求和反馈。一般如果反馈没成功说明指令没有执行，但是反馈成功并不能代表命令就执行了，所以我们需要再发送指令后需要判断飞控的状态。

10.3.2. 话题 /mavros/local_position/pose 获得飞控位姿数据

通过这话题能获取飞控在世界坐标系“map”下的位姿，有位置信息，也有姿态信息，因此如果需要获取飞机的姿态，可以使用 imu 数据(通过话题/mavros/imu/data_raw)，通常使用 imu 数据获取位姿更为直接。

10.3.3. 话题 /mavros/imu/data_raw 获取飞控 IMU 数据

原始 imu 数据数据，坐标系与飞控参数设置有关，一般使用 FLU 坐标系。与之相关的还有一个/mavros/imu/data 的话题，该话题是进行滤波后的数据。

10.3.4. 话题 /mavros/setpoint_raw/local 发送位置、速度、加速度、控制指令接口

给飞控发送，位置控制，速度控制，加速度控制的接口，可以通过其中 type_mask 变量选择控制的方式，详细查看错误!未找到引用源。

10.3.5. 话题 mavros/setpoint_raw/attitude 发送姿态控制接口

使用姿态控制的方式控制飞控详细参考**姿态环控制**

10.3.6. 服务 /mavros/set_mode 飞控模式切换接口

通过该服务名进行飞控的飞行模式切换，当然状态能否模式切换成功受飞控设置限制，如有些飞控屏蔽了某些模式的切换等，关于模式切换，详细参考**切换飞控飞行模式**

10.3.7. 服务 /mavros/cmd/arming 飞控解锁接口

通过该服务名可以对飞控进行解锁看。当然，解锁也收到飞控的设置的约束，详细参考**解锁飞控**

10.4. MavROS 在 RflySim 平台上的应用

10.4.1. MavROS 软件在环

在 RflySim 平台中使用 MavROS，需要进行软件 bat 脚本和 MavROS 启动脚本的配置。由于 MavROS 运行在 Linux 虚拟机或机载电脑中所以在软件在环脚本中首先设置设置通信端口号 SET /a UDP_START_PORT=20100，然后查看 Linux 虚拟机的 IP 地址例如 192.168.31.155 然后在脚本 SET IS_BROADCAST=192.168.31.155 中进行更改设置。20100 是 CopterSim 对外提供通信的端口，20101 是 MavROS 对外提供的通信端口。

在 MavROS 的启动脚本中也进行相应的端口设置如下：

```
<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_systen" 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>
```

其中//:符号中间应该为 Linux 虚拟机或机载电脑中的 IP 地址，但由于 MavROS 就在其中启动默认为其对应的 IP 地址，但也可手动加上。端口 20101 为 MavROS 对外通信的端口，192.168.31.117 为平台 window 下的 IP 地址。

10.4.2. MavROS 硬件在环

在 RflySim 平台中使用 MavROS 进行硬件在环时，需要使用相应的硬件在环脚本，将飞控通过 USB 口使用 USB 数据线与电脑连接用于与 CopterSim 通信以及通过 TELEM1 使用 USB 转 TTL 线连接电脑用于电脑与飞控通信。然后在 MavROS 的启动脚本中也进行相应的端口设置如下：

```
<!--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 与飞控通信

10.4.3.1. 获取飞控状态

```
state_sub = rospy.Subscriber('/mavros/state', State, state_callback)
```

在 mavros 中，要获取飞行控制器（Flight Control Unit，FCU）的状态，你可以使用 mavros/state 话题来获取当前飞行状态信息。mavros/state 话题用于发布当前飞行控制器的状态信息。这个话题的消息类型是 mavros_msgs/State，包含了飞行模式、连接状态、解锁状态等信息。飞控状态监听者定义如上。

10.4.3.2. 获取飞机姿态

```
rospy.Subscriber('/mavros/imu/data', Imu, imu_callback)
```

在 mavros 中获取飞机的姿态信息，可以使用 mavros/imu/data 话题来获取。这些话题提供了飞机的姿态数据，包括角度和角速度等信息。该话题发布了飞机的惯性测量单元（IMU）数据，包括角度和角速度。可以使用 ROS 订阅这个话题以获取姿态信息。这个话题的消息类型是 sensor_msgs/msg/Imu 消息类型。飞机姿态监听者定义如上。

10.4.3.3. 获取飞控位姿

本地位姿：

```
local_pose_sub=rospy.Subscriber('/mavros/local_position/pose',PoseStamped,local_pose_callback)
```

在 mavros 中，要获取飞行控制器的位姿信息，你可以使用 mavros/local_position/pose 话题来获取当前飞行器的本地位置和姿态信息。话题用于发布当前飞行控制器的本地位姿信息，包括位置坐标（例如东北天坐标系中的位置）和姿态信息（例如，欧拉角或四元数表示的姿态）。这个话题的消息类型是 geometry_msgs/PoseStamped。

全局位姿：

```
global_pose_sub=rospy.Subscriber('/mavros/global_position/global',NavSatFix,global_pose_callback)
```

在 mavros 中，要获取飞行控制器的全局位姿信息（通常以经纬度坐标表示），你可以使用 mavros/global_position/global 话题来获取当前飞行器的全局位置信息。该话题用于发布当前飞行控制器的全局位置信息，包括经纬度坐标和海拔高度。这个话题的消息类型是 sensor_msgs/NavSatFix 消息类型。

10.4.3.4. 切换飞控飞行模式

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

在 mavros 中，要设置飞行控制器的飞行模式，你可以使用 mavros/set_mode 服务来进行设置。mavros/set_mode 服务用于请求更改飞行控制器的飞行模式。通过调用此服务，你可以请求将飞行器切换到不同的飞行模式，例如手动模式、自稳模式、定位模式等。可以使用 ROS 服务客户端来调用 mavros/set_mode 服务以请求更改飞行模式。你需要创建一个服务请求消息，通常使用 mavros_msgs/SetMode 消息类型，特别地，飞控需要发送目标指令后才能切换 offboard 的模式。详细示例参考官网 http://docs.px4.io/main/zh/ros/mavros_offboard_python.html

10.4.3.5. 解锁飞控

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

在 mavros 中，要解锁飞行控制器（FCU），通常使用的接口是服务名为 mavros/cmd/arming 服务，这个服务允许你控制飞行控制器的解锁（arming）和上锁（disarming）。解锁是指启用飞行控制器的电机，使其准备好飞行，而上锁则是指禁用电机，

使其停止工作。为了调用 mavros/cmd/arming 服务，你需要创建一个服务请求消息，通常使用 mavros_msgs/CommandBool 消息类型。通常真机飞行，安全起见我们使用遥控器发送解锁指令，仿真时才使用改服务类型去解锁飞控。关于解锁示例参考 PX4 官网 http://docs.px4.io/main/zh/ros/mavros_offboard_python.html

10.4.4. MavROS 对飞控控制方式

mavros 控制飞控的话题有很多，但只要学会使用其中两个话题就够用了，我们只需要通过发布这两个话题中的其一或者组合即可实现对飞机的控制，mavros 其他对飞机的控制话题都是这两个话题功能拆解使用，因此这里不再对每个分功能话题的描述。注意每个话题控制控制频率应该在 5hz 以上，同时如果需要多个话题组合控制效果，最好去理解飞控底层代码

话题名	说明
/mavros/setpoint_raw/local	可以通过这个话题实现位置控制、速度控制、加速度控制，或者组合控制（此话题有对整个控制功能的拆解话题，这里不再介绍，感兴趣的朋友可以去阅读官方文档）
/mavros/setpoint_raw/attitude	可以通过这个话题实现姿态控制（此话题有对整个控制功能的拆解话题，这里不再介绍，感兴趣的朋友可以去阅读官方文档）

10.4.4.1. 姿态环控制

```
attitude_pub=rospy.Publisher('/mavros/setpoint_raw/attitude',AttitudeTarget,queue_size=10)
```

mavros_msgs/AttitudeTarget 消息类型的内容说明：

```
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

变量名	说明	使用方法
type_mask	类型掩码，屏蔽是否使用想用的功能，它的值可以直接使用消息类型内不对静态常量值，值说明如下： 1：屏蔽 roll 的角速度 2： 屏蔽 pitch 的角速度 4： 屏蔽 yaw 的角速度 64：屏蔽油门量控制 128：屏蔽姿态量控制	给 typt_mask 可以数值直接赋值，数值（二进制，十六进制，十进制）组合的形式都可以，推荐使用常量名直接相加赋值，如定义了 AttitudeTarget 变量 att，那么推荐使用这种方式 att.type_mask = att. IGNORE_ROLL_RATE + att.IGNORE_PITCH_RATE + ... 使用这方式更直观，比起二进制或者十六进制更方便阅读使用，当然自己也可以对组合的值重新定义变量；
orientation	需要控制飞机姿态的值，这里用四元数表示。	att.orientation.x = ‘value’ ....
body_rate	控制姿态各个分量的角速度，其中 x,y,z 对应着 roll_rate, pitch_rate,yaw_rate	att.body_rate.x = ‘value’ ....
thrust	给定的油门量	值范围（0~1），通常油门值在 0.6 时，飞机保持悬停。

需要注意的是，给飞控控制指令的时候，往往都需要设置时间，一般使用（C++ 里 ros::Time::now(); Python 里 rospy.Time.now()) 即可，然后使用发布话题 /mavros/setpoint_raw/attitude 将设置好的姿态控制数据发布出去。

10.4.4.2. 加速度环控制

```
accel_pub=rospy.Publisher('/mavros/setpoint_raw/local',PositionTarget ,queue_size=10)
```

在 mavros 中，你可以使用 /mavros/setpoint_raw/local 话题来进行飞控的加速度环控制发布。这个话题允许你发送期望的 x,y,z 三轴上加速度指令给飞控，从而实现加速度控制。这个话题的消息类型是 PositionTarget 消息类型，详细内容如下：

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

变量名	说明	使用方法
coordinate_frame	选择基于什么坐标系控制，可选值如下 1：虽然静态变量名称为 FRAME_LOCAL_NED，实际上使用的却是全局 ENU 坐标系，这里和 RflySim3D 中的坐标系存在 90 度的偏航角，详细请查阅 RflySim 与 mavros 使用需要注意的问题 2：FRAME_LOCAL_OFFSET_NED 这个值 PX4 暂不支持 3：FRAME_BODY_NED 同样虽然这里表明 NED,实际上使用的是相对机体坐标系的 FLU； 4：FRAME_BODY_OFFSET_NED 这个值 px4 不支持 注意：ROS 版本 melodic 以前的版本坐标系定义又不一样，这	因为控制量基于的坐标系有且仅能使用其中一种，因此可以使用数字赋值，也可以使用下面的方式， tar. coordinate_frame = tar. FRAME_LOCAL_NED，速度控制或者加速度控制一般使用 body 坐标系即 tar. coordinate_frame = tar. FRAME_BODY_NED

	里不过多介绍	

type_mask	选择控制类型，可以使基于位置控制，基于速度控制，基于加速度控制，或者可以组合使用，组合使用时，效果是各个控制量的合并效果，感兴趣的读者可以自行阅读飞控底层源码（要想控制的很流畅，这是必要的）。
	可选值有：
	1：屏蔽位置控制坐标点 x 分量
	2：屏蔽位置控制坐标点 y 分量
	4：屏蔽位置控制坐标点 z 分量
	8：屏蔽速度控制 x 方向分量
	16：屏蔽速度控制 y 方向分量
	32：屏蔽速度控制 z 方向分量
	64：屏蔽加速度或者推力控制 x 方向分量
	128：屏蔽加速度或者推力控制 y 方向分量
	256：屏蔽加速度或者推力控制 z 方向分量
	512：选择使用推力控制
	1024：屏蔽偏航角(yaw)控制
	2048：屏蔽偏航角速度(yaw_rate)控制
		对 type_mask 的赋值一般使用数值形式赋值如（二进制，十六进制），不过这里推荐使用静态变量名赋值，这样更直观，推荐使用下面的赋值方式：
		如果是单一的控制方式：
		tar.type_mask = ~(0)； //全置 1，屏蔽所有
		//然后针对单一的控制方式放开屏蔽,如这里仅加速度控制
		tar.type_mask = ~tar. IGNORE_AFX & ~tar. IGNORE_AFY & ~tar. IGNORE_AFZ & ~ tar.FORCE; //别忘了最后要把 Force 屏蔽，
		如果是多种控制方式组合使用
		假设控制方式为加速度和速度的组合
		tar.type_mask = 0 // 全置 0，使能所有控制，当然 Force 除外，Froce 为 0 表示屏蔽；然后有针对性屏蔽位值控制；
		tar.type_mask = tar. IGNORE_PX | tar.IGNORE_PY | tar.IGNORE_PZ;

velocity	速度控制，其中 x,y,z 对应着基于 coordinate_frame 坐标系下的各个分量的值	
acceleration_or_force	加速度或者推力控制，其中 x,y,z 对应着基于 coordinate_frame 坐标系下的各个分量的值，当 type_mask 中 Force 标志位为 1，表示使用推力控制，为 0 表示使用加速度控制，并不是所有的飞控都支持 Force 控制	
yaw	偏航角控制量，只有 type_mask 中标志位 IIGNORE_YAW 为 0 才有效	
yaw_rate	偏航角速度控制量，只有 type_mask 中标志位 IIGNORE_YAW_RATE 为 0 才有效	

同样的，除了控制量赋值，还需要把时间戳带上，一般使用（C++ 里 ros::Time::now(); Python 里 rospy.Time.now()）即可，然后使用发布话题/mavros/setpoint_raw/local 将设置好的加速度控制数据发布出去。

10.4.4.3. 速度环控制

```
vel_pub=rospy.Publisher('/mavros/setpoint_raw/local',PositionTarget ,queue_size=10)
```

速度控制和使用加速度，位置控制同一个话题控制，该话题组合了速度控制，加速度控制，位置控制方式，通过 type_mask 进行选择。下面把 PositionTarget 数据类型详细介绍一遍：

```
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
```

变量名	说明	使用方法
coordinate_frame	选择基于什么坐标系控制，可选值如下 1：虽然静态变量名称为 FRAME_LOCAL_NED，实际上使用的却是全局 ENU 坐标系，这里和 RflySim3D 中的坐标系存在 90 度的偏航角，详细请查阅 RflySim 与 mavros 使用需要注意的问题 7：FRAME_LOCAL_OFFSET_NED 这个值 PX4 暂不支持 8：FRAME_BODY_NED 同样虽然这里表明 NED,实际上使用的是相对机体坐标系的 FLU； 9：FRAME_BODY_OFFSET_NED 这个值 px4 不支持 注意：ROS 版本 melodic 以前的版本坐标系定义又不一样，这里不过多介绍	因为控制量基于的坐标系有且仅能使用其中一种，因此可以使用数字赋值，也可以使用下面的方式， tar. coordinate_frame = tar. FRAME_LOCAL_NED，速度控制或者加速度控制一般使用 body 坐标系即 tar. coordinate_frame = tar. FRAME_BODY_NED

type_mask	选择控制类型，可以使基于位置控制，基于速度控制，基于加速度控制，或者可以组合使用，组合使用时，效果是各个控制量的合并效果，感兴趣的读者可以自行阅读飞控底层源码（要想控制的很流畅，这是必要的）。
	可选值有：
	1：屏蔽位置控制坐标点 x 分量
	2：屏蔽位置控制坐标点 y 分量
	4：屏蔽位置控制坐标点 z 分量
	8：屏蔽速度控制 x 方向分量
	16：屏蔽速度控制 y 方向分量
	32：屏蔽速度控制 z 方向分量
	64：屏蔽加速度或者推力控制 x 方向分量
	128：屏蔽加速度或者推力控制 y 方向分量
	256：屏蔽加速度或者推力控制 z 方向分量
	512：选择使用推力控制
	1024：屏蔽偏航角(yaw)控制
	2048：屏蔽偏航角速度(yaw_rate)控
		对 type_mask 的赋值一般使用数值形式赋值如（二进制，十六进制），不过这里推荐使用静态变量名赋值，这样更直观，推荐使用下面的赋值方式：
		如果是单一的控制方式：
		tar.type_mask = ~(0)； //全置 1，屏蔽所有
		//然后针对单一的控制方式放开屏蔽，如这里仅速度控制
		tar.type_mask = ~tar. IGNORE_VX & ~tar. IGNORE_VX & ~tar. IGNORE_VX & ~tar.FORCE
		如果是多种控制方式组合使用
		假设控制方式为加速度和速度的组合
		tar.type_mask = 0 // 全置 0，使能所有控制；然和有针对性屏蔽位值控制；
		tar.type_mask = tar. IGNORE_PX | tar.IGNORE_PY | tar.IGNORE_PZ;

	制	
velocity	速度控制，其中 x,y,z 对应着基于 coordinate_frame 坐标系下的各个分量的值	
acceleration_or_force	加速度或者推力控制，其中 x,y,z 对应着基于 coordinate_frame 坐标系下的各个分量的值，当 type_mask 中 Force 标志位为 1，表示使用推力控制，为 0 表示使用加速度控制，并不是所有的飞控都支持 Force 控制	
yaw	偏航角控制量，只有 type_mask 中标志位 IIGNORE_YAW 为 0 才有效	
yaw_rate	偏航角速度控制量，只有 type_mask 中标志位 IIGNORE_YAW_RATE 为 0 才有效	

同样的，除了控制量赋值，还需要把时间戳带上，一般使用（C++ 里 ros::Time::now(); Python 里 rospy.Time.now()）即可，然后使用发布话题/mavros/setpoint_raw/local 将设置好的速度控制数据发布出去。

10.4.4.4. 位置环控制

```
Pose_pub=rospy.Publisher('/mavros/setpoint_position/local',PoseStamped,queue_size=10)
```

在 mavros 中，你可以使用 mavros/setpoint_position/local 话题来进行飞控的位置环控制发布。这个话题允许你发送期望的 NED 坐标系下的位置（x、y、z）指令给飞控，从而实现位置控制，实际控制中。这个话题的消息类型是 geometry_msgs/PoseStamped 消息类型。位置环控制发布者定义如上。

上述方式用的比较常见，因为简单，但是更多的推荐使用话题/mavros/setpoint_raw/local 去控制位置

```
Pose_pub=rospy.Publisher('/mavros/setpoint_raw/local ', PositionTarget,queue_size=10)
```

可以通过话题/mavros/setpoint_raw/local 位置控制，速度控制，加速度控制/推力控制，因此高效的控制方式需要多种组合控制，下面详细介绍数据类型 PositionTarget 以及位置控制方式

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

变量名	说明	使用方法
coordinate_frame	选择基于什么坐标系控制，可选值如下 1：虽然静态变量名称为 FRAME_LOCAL_NED，实际上使用的却是全局 ENU 坐标系，这里和 RflySim3D 中的坐标系存在 90 度的偏航角，详细请查阅 RflySim 与 mavros 使用需要注意的问题 7：FRAME_LOCAL_OFFSET_NED 这个值 PX4 暂不支持 8：FRAME_BODY_NED 同样虽然这里表明 NED,实际上使用的是相对机体坐标系的 FLU； 9：FRAME_BODY_OFFSET_NED 这个值 px4 不支持	因为控制量基于的坐标系有且仅能使用其中一种，因此可以使用数字赋值，也可以使用下面的方式， tar. coordinate_frame = tar. FRAME_LOCAL_NED，速度控制或者加速度控制一般使用 body 坐标系即 tar. coordinate_frame = tar. FRAME_BODY_NED

	注意：ROS 版本 melodic 以前的版本坐标系定义又不一样，这里不过多介绍	

type_mask	选择控制类型，可以使基于位置控制，基于速度控制，基于加速度控制，或者可以组合使用，组合使用时，效果是各个控制量的合并效果，感兴趣的读者可以自行阅读飞控底层源码（要想控制的很流畅，这是必要的）。
	可选值有：
	1：屏蔽位置控制坐标点 x 分量
	2：屏蔽位置控制坐标点 y 分量
	4：屏蔽位置控制坐标点 z 分量
	8：屏蔽速度控制 x 方向分量
	16：屏蔽速度控制 y 方向分量
	32：屏蔽速度控制 z 方向分量
	64：屏蔽加速度或者推力控制 x 方向分量
	128：屏蔽加速度或者推力控制 y 方向分量
	256：屏蔽加速度或者推力控制 z 方向分量
	512：选择使用推力控制
	1024：屏蔽偏航角(yaw)控制
	2048：屏蔽偏航角速度(yaw_rate)控制
		对 type_mask 的赋值一般使用数值形式赋值如（二进制，十六进制），不过这里推荐使用静态变量名赋值，这样更直观，推荐使用下面的赋值方式：
		如果是单一的控制方式：
		tar.type_mask = ~(0)； //全置 1，屏蔽所有
		//然后针对单一的控制方式放开屏蔽，如这里仅位置控制
		tar.type_mask = ~tar. IGNORE_PX & ~tar. IGNORE_PY & ~tar. IGNORE_PZ & ~tar.FORCE
		如果是多种控制方式组合使用
		假设控制方式为位置和速度的组合
		tar.type_mask = 0 // 全置 0，使能所有控制；然和有针对性屏蔽加速度控制；
		tar.type_mask = tar. IGNORE_AFX | tar.IGNORE_AFY | tar.IGNORE_AFZ;

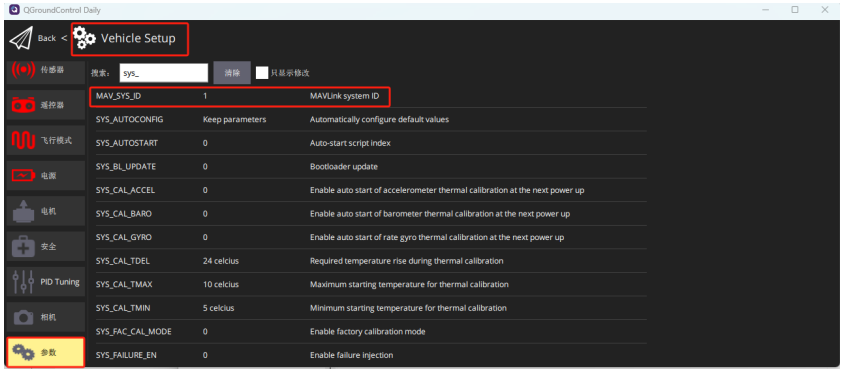
velocity	速度控制，其中 x,y,z 对应着基于 coordinate_frame 坐标系下的各个分量的值	
acceleration_or_force	加速度或者推力控制，其中 x,y,z 对应着基于 coordinate_frame 坐标系下的各个分量的值，当 type_mask 中 Force 标志位为 1，表示使用推力控制，为 0 表示使用加速度控制，并不是所有的飞控都支持 Force 控制	
yaw	偏航角控制量，只有 type_mask 中标志位 IIGNORE_YAW 为 0 才有效	
yaw_rate	偏航角速度控制量，只有 type_mask 中标志位 IIGNORE_YAW_RATE 为 0 才有效	

同样的，除了控制量赋值，还需要把时间戳带上，一般使用（C++ 里 ros::Time::now(); Python 里 rospy.Time.now()）即可，然后使用发布话题/mavros/setpoint_raw/local 将设置好的位置控制数据发布出去。

10.5. RflySim 多机 ROS 分布式部署

10.5.1. 飞控参数配置

将各个硬件飞控通过 USB 线与 NX 连接打开 QGroundControl 软件进行设置 ID 号。这是为了 CopterSimID 与飞控 ID 保持一致。设置如下：



10.5.2. MavROS 参数配置

对于每台机器上运行的 MAVROS 节点，需要配置以下参数：

```
<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>
```

其中参数 fcu_url 为飞控的连接方式，参数 gcs_url 为 QGC 所在主机的 IP 已默认，参数 tgt_system 为飞控的 ID 号，参数 tgt_component 指定 MAVLink 消息的目标组件 ID 为自动驾驶仪，参数 log_output 为日志输出方式配置，参数 fcu_protocol 用于指定与飞行控制器通信时使用的 MAVLink 协议版本，参数 namespace 指定该节点的命名空间。

10.5.3. TF 树搭建

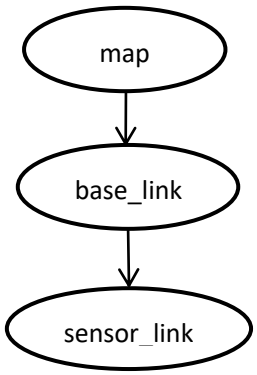
10.5.3.1. TF 树的简要说明

TF 树用来管理在 ROS 里面机器人关节（joint） 与之连接两个臂（link） 之间的坐标转换，关节可以是刚性的（fixed），也可以是可活动的（float），以机器臂为例，已知指尖的位置，要知道指尖到底座坐标系的位置，需要多个变换矩阵相乘，如果每个关节都需要知道指尖在当前关节连接的 link 坐标系下，都需要做相乘操作，太过繁琐。因此 ROS 推出 TF 树来管理这种复杂的转换关系，任何时刻，我们可以直接通过 link 获取想要的坐标系下的坐标，在此技术，ROS 推出了一系列的方便调试的工具，如 Rviz，tf_graph 等，那么到底有没有必要刻意去搭建一个 TF 树呢？ 这个看需求，很多算法里面不用 TF 树做坐标转换，有关坐标转换的运算，在算法内部完成，然后调试可视化使用一个局部的 tf 树 frame 就行，这种情况倒也不必去构建整个 TF 树。

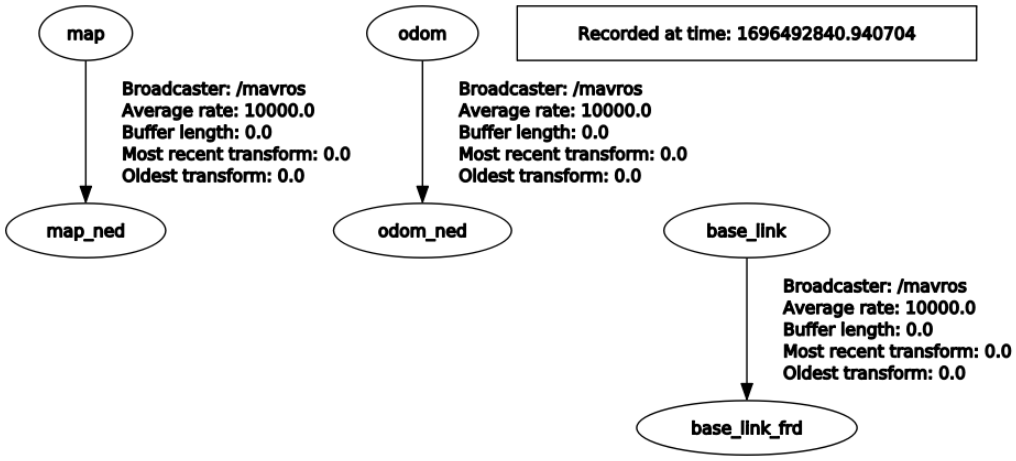
10.5.3.2. TF 树在 RflySim 以及 mavros 之间的应用

我们知道在 ROS 里面有这么三个基本的坐标系，map 坐标系，body 坐标系以及 sensor 坐标系，通常在 tf 树中对应 frame_id 分别为：map, base_link, (lidar_link, camera_link...)等等，往往一个复杂的系统会在这三者之间引进去别的坐标系，如：odom，base_link_foot 等。

RflySim 提供的传感器为了方便可视化，默认使用的 frame_id 都是 Map， 用户可以参考例程 RflySimAPIs\PythonVisionAPI\1-APIUsageDemos\18-ConfigROSTFAPIDemo, 自行配置各传感器数据的 frame_id，通常传感器坐标系需要转换到机体坐标系（base_link），而 base_link 需要转到世界坐标系（map）下才能做控制等算法处理，而 base_link 到 map 坐标系需要定位系统给出转换。因此最简单 TF 树如下:



我们运行 mavros 连接飞控，然后使用 rqt_tf(命令：rosrun rqt_graph rqt_graph)工具查看当前 tf 树的情况



此时的 TF 树都是零零碎碎的，我们可以通过外部定位源，将 base_link 连接到 map， 或者将 base_link 连接到 odom 再由 odom 连接到 map。构成一棵完整的 Tf 树。当然我们使用的一般不是 NED 坐标系，因此不用考虑 xxx_ned.

构建 tf 树有三种常用的方法：

1. 构建 Tf 树可以通过加载事先准备好的 urdf 文件构建，关于如何构建 URDF 文件参考 urdf/XML/link - ROS Wiki，然后通过下面方式加载这个文件，如 launch 文件中这样调用：

```
<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>
```

使用节点 joint_state_publisher 加载 urdf 文件内的 tf 数据，使用 robot_state_publisher 发布 tf 树，这种方法适合构建复杂系统的 tf 树，在 urdf 文件中很方便的构建所有 tf 数据结构。

2. 对简单 tf 树结构，当然仅仅针对那些静态的也就是坐标系之间是刚性连接的，仅需要在启动 launch 的文件加入这样一行内容就行，比如构建一个 lidar_link 到 base_link 的 tf 树结果，只需要这样：

```
<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：平移量(x,y,z), 四元数旋转 q（offset_x, offset_y, offset_z,q_x,q_y,q_z,q_w), frame_id,child_frame_id, 频率;

3. 对于非刚性连接的两个坐标系需要实时的通过代码发布对应的转换关系，如：

```
#include <ros/ros.h>
#include <tf/transform_broadcaster.h>   //加入 tf 变换的库头文件
#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;
};
```

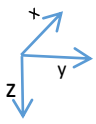
上述代码通过订阅来自飞控外部的视觉定位源，然后建立 map 到 base_link 的连接，因为载体会实时在移动，因此 base_link 与 map 之间不可能是刚性连接。

11. MavROS，RflySim 等坐标系介绍

11.1. RflySim3D 坐标系

11.1.1. RflySim3D Map 坐标系

RflySim3D 坐标系为北东地 FRD or NED (X **N**orth, Y **E**ast, Z **D**own)



CopterSim 界面的坐标为北东天坐标系(NEU)

11.1.2. 载体坐标系

只有确定了载体坐标系，才好确定传感器的安装位置以及角度。需要注意的是，Rflysim 3D 模型中的物体是可以缩放的，因此一般来讲传感器的安装位置根据需求也需要进行缩放，比如，假设模型的放大了 10 倍，在放大前的传感器在载体坐标系下为（1，0，0）在机头的位置，若想在放大后仍然要求传感器在机头位置，则需要更改传感器配置文件，设置（10，0，0）。

载体坐标系在 RflySim3D 里面也是 NED(坐标系)，因此传感器的安装位置以及姿态需要参考这个坐标系去配置

11.1.3. RflySim IMU 坐标系

RflySim 默认是输出的是 FRD (X **F**orward, Y **R**ight, Z **D**own)坐标系，但是为了与 mavros 统一，目前改成 FLU (X **F**orward, Y **L**eft, Z **U**p)坐标系

11.1.4. 传感器坐标系

相机坐标系为：(UWN)即 Z 轴指向北，X 轴指向天,Y 轴指向西边

激光雷达坐标系：FLU (X **F**orward, Y **L**eft, Z **U**p)

11.2. MavROS 坐标系

11.2.1. MavROS IMU 坐标系

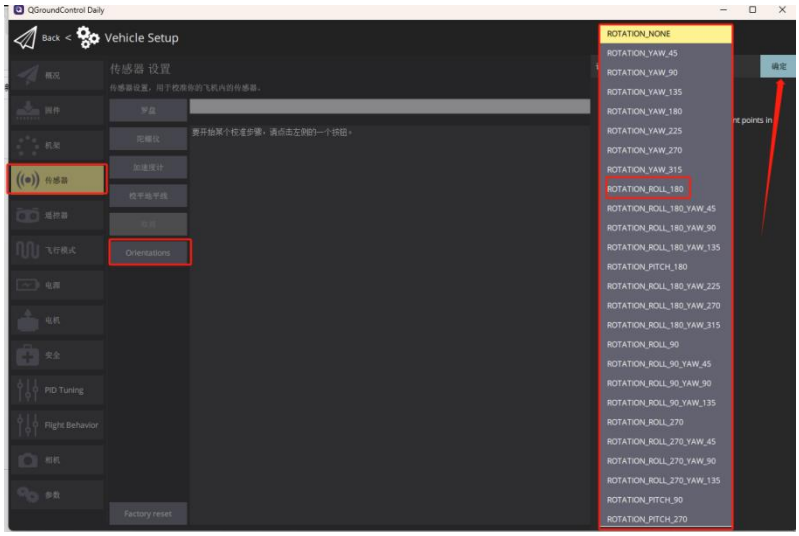
11.2.1.1. 软件在环仿真

mavros 输出的 IMU 坐标系默认是 FLU，注意：为了统一坐标系，新版的 VisionCaptureAPI 接口输出的 IMU 坐标系不再使用以前的 FRD (X **F**orward, Y **R**ight, Z **D**own)坐标系，与 MavROS 保持一致使用 FLU。

11.2.1.2. 硬件在环仿真

在使用 mavros 的 IMU 前先确定一下 IMU 坐标系，如果是北东地，那么把坐标系改成北东天，具体步骤如下：

1) 打开 QGC（QGroundControl） 并且连接上飞控;
2) 在工具栏选择 **齿轮** 图标 (机体设置)，然后在侧边栏选择 **传感器**
3) 选择 **Set Orientations 或者 Orientations** 按钮，如下图，根据 IMU 输出的坐标系，选择旋转至北东地的坐标系选项;



然后点击确定，记得需要重新给飞控上电启动，才能生效

11.2.2. MavROS map 坐标系

11.2.2.1. RflySim 与 mavros 使用需要注意的问题

在使用 RflySim 仿真结合 mavros 需要注意的问题，ros 中使用的 map 坐标系也是 ENU 坐标系，但是在基于 mavros 开发的过程中会发现，RflySim3D 中的坐标系是 NED，除了 Z 轴不一样外，两者还存在 90 度的 yaw 角，因此当我们通过/mavros/setpoint_position/local 或者通过/mavros/setpoint_raw/local 发送坐标位置进行指点起飞，在起飞时飞机会向 xy 平面顺时针旋转九十度，所以如果我们想直接使用 RflySim3D 中的某点坐标通过 mavros 发送时，需要将 x,y 互换，x 取负值，同时 z 取反，如果是通过姿态控制也需要注意这个差别。

这里还有一个细节需要注意，假设我们的飞机在一条长廊里飞行，按照习惯，我们希望机头方向指向长廊的另一端，而 RflySim3D 中的 x 方向与长廊墙壁平行,x 方向直行长廊的另一端。我们通过 mavros 控制将飞机从(0，0，0)点控制到长廊的另一头(100,0,0)，我们让飞机起飞，并通过话题 /mavros/setpoint_position/local 或者 /mavros/setpoint_raw/local 发送了一个起飞目标点（0,0,2），起飞后，你会发现此时的飞机已经面向左侧的墙壁了。这并不是我们想要的效果，因此我们需要在发送目标标点的同时给一个 90 度 yaw 偏航（position:0,0,2; orientation:(0.707,0,0,0.707)）,每次发送目标点都需要加上一个偏航的角度。当然这样使用这种方式指点飞行，没有什么问题，但是如果我们控制姿态，这种方法就不能用了，这时候，我们需要将启动 RflySim 的 bat 脚本初始化角度给一个 90 度的值“SET /a ORIGIN_YAW=90”，通过上述例子的描述，我们再使用 mavros 在 RflySim 里面仿真时，需要注意以下这几点：确定控制方式(位姿控制，速度控制，姿态控制)，机头在场景中的方向（如上述例子中，目标点不给偏航角，飞机就要“面壁思过”飞行也不是不可以），如果场景中的长廊方向即是 RflySim3D 的 y 方向，那，飞机起飞后，机头方向刚刚就是长廊方向，因此需要根据需求，灵活的设置。特别地，在应用于 SLAM 场景时，应将启动 RflySim 的 bat 脚本中的初始化角度设置成 SET /a ORIGIN_YAW=90，这是因为如果没有设置该值，在起飞旋转时，因为角速度过大，可能会造成 SLAM 算法初始化较大的误差。