

---

# API 说明文件检索大纲

## 目录

1.	RflySim 平台控制硬件接口 .....	1
1.1.	遥控器控制 .....	1
1.1.1.	真实遥控器控制 .....	1
1.1.2.	QGC+遥控器 USB 控制 .....	1
1.1.3.	QGC 虚拟遥控器控制 .....	3
1.2.	地面站控制 .....	5
1.2.1.	参数配置 .....	5
1.2.2.	按键命令（起飞、降落、返航）(Vehicle Setup) .....	6
1.2.3.	航线规划 (Plan) .....	12
1.2.4.	数据分析 (Analyze Tools) .....	15
1.2.5.	地面站设置 (Application settings) .....	18
1.3.	控制设备与通信介质 .....	21
1.3.1.	无线数传 (PC 通过数传连 PX4 并进行控制) .....	21
1.3.2.	有线串口模块 (NX 通过有线串口连 PX4 并进行控制) .....	21
1.3.3.	WIFI 模块 (带有机载板卡进行消息转发) .....	22
2.	RflySim 平台控制模式接口 .....	22
2.1.	常规飞行控制模式 .....	22
2.1.1.	起飞模式： .....	22
2.1.2.	降落模式： .....	23
2.1.3.	定点/悬停（盘旋）模式： .....	23
2.1.4.	任务模式： .....	24
2.1.5.	定高模式： .....	24
2.1.6.	自稳/手动控制（姿态角控制模式）模式： .....	24
2.1.7.	特技（角速度控制）模式： .....	25
2.1.8.	返航模式： .....	25
2.2.	外部控制模式 .....	25
2.2.1.	控制消息 .....	26
2.2.2.	控制接口 .....	29
2.2.3.	典型组合模式 .....	30
3.	控制模型 .....	31
3.1.	高精度模型+PX4 控制器组成的软/硬件在环仿真模型（依靠 CopterSim,	

---

从模型组复制) .....	31
3.2.    高精度模型+Simulink 控制器 组成的高精度综合模型（依靠 CopterSim）	
33	
3.2.1.    综合模型控制协议.....	33
3.2.2.    旋翼综合模型控制接口 .....	34
3.2.3.    固定翼综合模型控制接口 .....	36
3.3.    基于质点模型的简化综合模型（依靠 Python 旋翼，从集群 API 复制）	37
4.    RflySim 控制协议（仅企业定制版支持 Redis） .....	37
4.1.    总体介绍.....	37
4.2.    数据协议.....	39
4.2.1.    outHILStateData .....	39
4.2.2.    SOut2Simulator.....	39
4.2.3.    inHILCMDData .....	40
4.2.4.    outhILStateShort .....	40
4.2.5.    inOffboardShortData (最简控制协议，CopterSim 处于 UDP/Redis Sim ple 模式时支持) .....	41
4.3.    通信端口 .....	42
4.3.1.    UDP14540 系列+TCP4560 系列（与 PX4 通信，软件在环仿真时 PX4 默认端口）	42
4.3.2.    UDP16540 系列（与 PX4 通信，软件在环仿真时 RflySim 私有端口）	
42	
4.3.3.    串口（与 PX4 通信，硬件在环仿真端口） .....	42
4.3.4.    UDP20100 系列（Python/Simulink 获取状态信息并下发控制指令）	.4 2
4.3.5.    UDP30100 系列（获取 True 状态信息或者通过 inSIL 下发控制指令）	
43	
4.3.6.    UDP40100 系列（获取用户自定义消息） .....	43
4.3.7.    TCP6379 (Redis 端口) .....	43
4.4.    RflySim UDP 协议 .....	44
4.4.1.    CopterSim UDP_Simple .....	44
4.4.2.    CopterSim UDP_Full .....	45
4.4.3.    CopterSim Redis_Simple/Full.....	45
4.4.4.    Simulink 控制模式 Full/Simple/UltraSimple .....	46
4.4.5.    Python 控制模式（完整支持 CopterSim 的模式） .....	46
5.    MAVLink 协议 .....	46

---

5.1.	MAVLink 简介.....	46
5.1.1.	MAVLink 包格式.....	46
5.1.2.	MAVLink 数据解析.....	47
5.2.	常用的 MAVLink 消息.....	48
5.2.1.	HEARTBEAT (心跳包) .....	48
5.2.2.	ATTITUDE (姿态-欧拉角) .....	49
5.2.3.	ATTITUDE_QUATERNION (姿态-四元数) .....	50
5.2.4.	LOCAL_POSITION_NED (NED 位置) .....	51
5.2.5.	GLOBAL_POSITION_INT (Global 位置) .....	51
5.2.6.	ACTUATOR_OUTPUT_STATUS (电机原始输出) .....	52
5.2.7.	ATTITUDE_TARGET (当前期望姿态) .....	52
5.2.8.	POSITION_TARGET_LOCAL_NED (当前期望 NED 位置) .....	53
5.2.9.	POSITION_TARGET_GLOBAL_INT (当前期望 Global 位置) .....	54
5.2.10.	HOME_POSITION (Home 点位置) .....	55
5.2.11.	HIL_ACTUATOR_CONTROLS (PX4 到 Sim 的控制输出) .....	56
5.2.12.	HIL_SENSOR (Sim 到 PX4 的传感器信息) .....	57
5.2.13.	HIL_GPS (Sim 到 PX4 的 GPS 信息) .....	58
5.3.	CopterSim MAVLink_Simple.....	67
5.4.	CopterSim MAVLink_Full.....	67
6.	控制接口 (原版, PX4MavCtrlV4.py) .....	67
6.1.	Simulink 的 UDP 控制接口 .....	68
6.1.1.	UDP Send (UDP 发送字节流模块) .....	68
6.1.2.	Receice UDP (UDP 接收字节流模块) .....	68
6.1.3.	UDP_SIL_State_Receiver (接收仿真位置、速度、姿态等信息模块)	
68		
6.1.4.	UDP_True_State_Receiver (接收真实位置、速度、姿态等信息模块)	
69		
6.1.5.	位置控制 (位置消息打包成字节流) .....	70
6.1.6.	速度控制 (速度消息打包成字节流) .....	70
6.1.7.	模拟遥控器 PWM 控制.....	71
6.2.	Python 的 UDP 控制接口 .....	71
6.2.1.	PX4MavCtrlr:__init__()	71
6.2.2.	InitMavLoop()	72
6.2.3.	endMavLoop()	72
6.2.4.	initOffboard()	72

---

6.2.5.	initOffboard2()	73
6.2.6.	InitTrueDataLoop()	73
6.2.7.	EndTrueDataLoop()	73
6.2.8.	访问 PX4MavCtrler 成员变量读取状态	73
6.2.9.	SendVelNED()	74
6.2.10.	SendVelNEDNoYaw()	74
6.2.11.	SendVelFRD()	75
6.2.12.	SendVelNoYaw()	75
6.2.13.	SendPosNED()	75
6.2.14.	SendVelYawAlt()	75
6.2.15.	SendPosGlobal()	75
6.2.16.	SendPosNEDNoYaw()	76
6.2.17.	SendPosFRD()	76
6.2.18.	SendPosFRDNoYaw()	76
6.2.19.	SendPosNEDExt()	76
6.2.20.	sendPX4UorbRflyCtrl()	76
6.2.21.	SendAccPX4()	77
6.2.22.	endOffboard()	77
6.2.23.	stopRun()	77
6.3.	Simulink 的 MAVLink 控制接口（串口连接）	77
6.3.1.	mavlink_msg_sender	77
6.3.2.	mavlink_msg_receiver	78
6.3.3.	MavLink Serial Input&Output	78
6.4.	Python 的 MAVLink 控制接口（基于 pymavlink）	79
6.4.1.	SendMavArm()	79
6.4.2.	SendMavCmdLong()	79
6.4.3.	sendMavOffboardCmd()	80
6.4.4.	initRCSSendLoop()	80
6.4.5.	SendRCPwms()	80
6.4.6.	endRCSSendLoop()	80
6.4.7.	SendSetMode()	80
6.4.8.	SendAttPX4()	81
6.4.9.	enFixedWRWTO()	81
6.4.10.	SendCruiseSpeed()	81
6.4.11.	SendCopterSpeed()	81

---

6.4.12.	SendGroundSpeed()	82
6.4.13.	SendCruiseRadius()	82
6.4.14.	sendTakeoffMode()	82
6.4.15.	sendMavTakeOff()	82
6.4.16.	sendMavTakeOffLocal()	82
6.4.17.	sendMavTakeOffGPS()	83
6.4.18.	sendMavLand()	83
6.4.19.	sendMavLandGPS()	83
6.4.20.	sendMavSetParam()	83
6.4.21.	SendHILCtrlMsg()	83
6.4.22.	SendHILCtrlMsg1()	84
6.5.	基于 ROS 的 MAVLink 控制接口（基于 mavros， 复制下视觉组的）	84
7.	rflysim 标准库版控制接口（新版， 支持 Redis）	84
7.1.	ctrl	84
7.1.1.	代码结构	84
7.1.2.	offboard.py	84
7.1.3.	topics.py	88
7.2.	用户 Api	89
7.2.1.	ctrl	89
7.2.2.	test	95
7.3.	test_ctrl.py	96

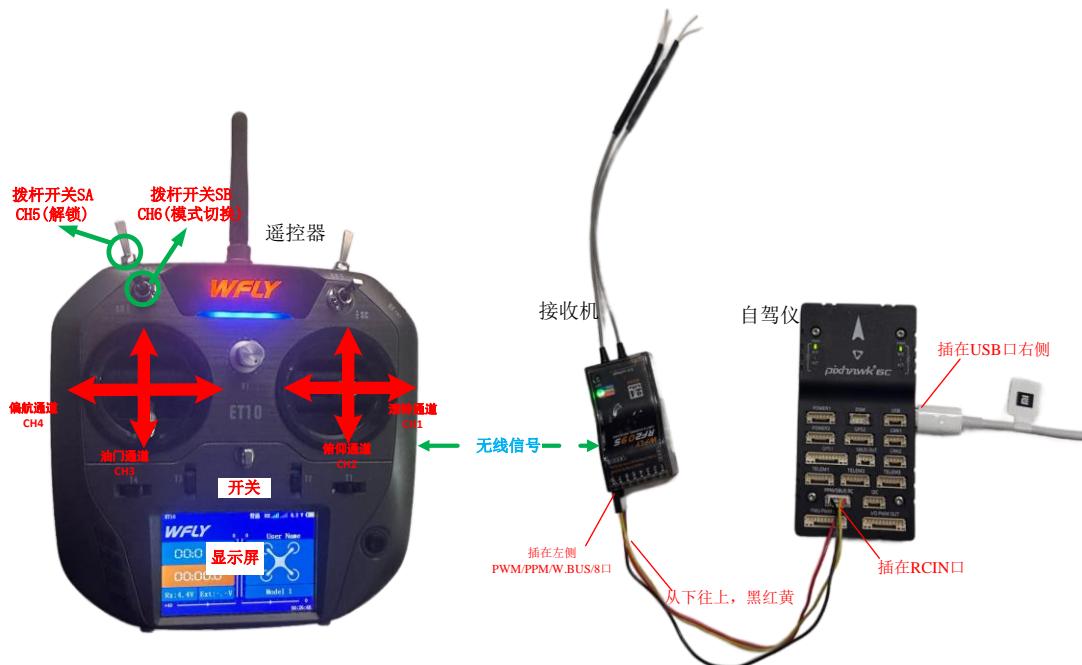
# 1. RflySim 平台控制硬件接口

## 1.1. 遥控器控制

### 1.1.1. 真实遥控器控制

本平台使用的遥控器推荐使用“美国手”的操纵方式，即左侧摇杆对应的油门与偏航控制量，而右侧摇杆对应滚转与俯仰。遥控器中滚转、俯仰、油门和偏航分别对应了接收机的 CH1~CH4 通道，左右上侧拨杆对应了 CH5/CH6 号通道，用于触发飞行模态切换。

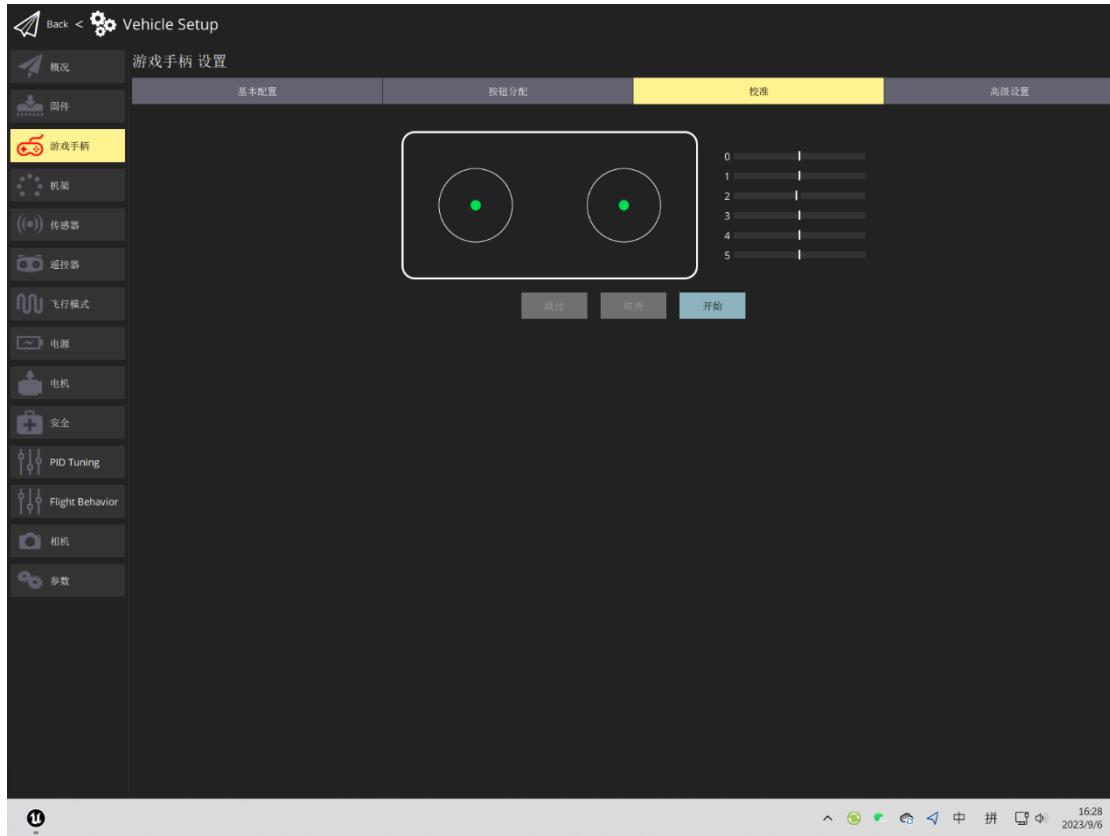
油门杆（CH3 通道）从最下端和最上端分别对应了 PWM 信号从 1100 到 1900 附近波动（通道不同或遥控器不同都会存在区别，因此需要校准）；滚转（CH1 通道）和偏航（CH4 通道）摇杆从最左端到最右端对应 PWM 信号从 1100 到 1900；俯仰（CH2 通道）摇杆从最下端到最上端对应 PWM 信号从 1900 到 1100；CH5 为二段开关，两个档位对应 PWM 信号为 1100、1900。CH6 为三段开关，三个档位对应 PWM 信号为 1100、1500 和 1900。更多遥控器相关配置请见：\*\\PX4PSP\\RflySimAPIs\\2.RflySimUsage\\1.BasicExps\\e11\_RC-Config\\Readme.pdf



### 1.1.2. QGC+遥控器 USB 控制

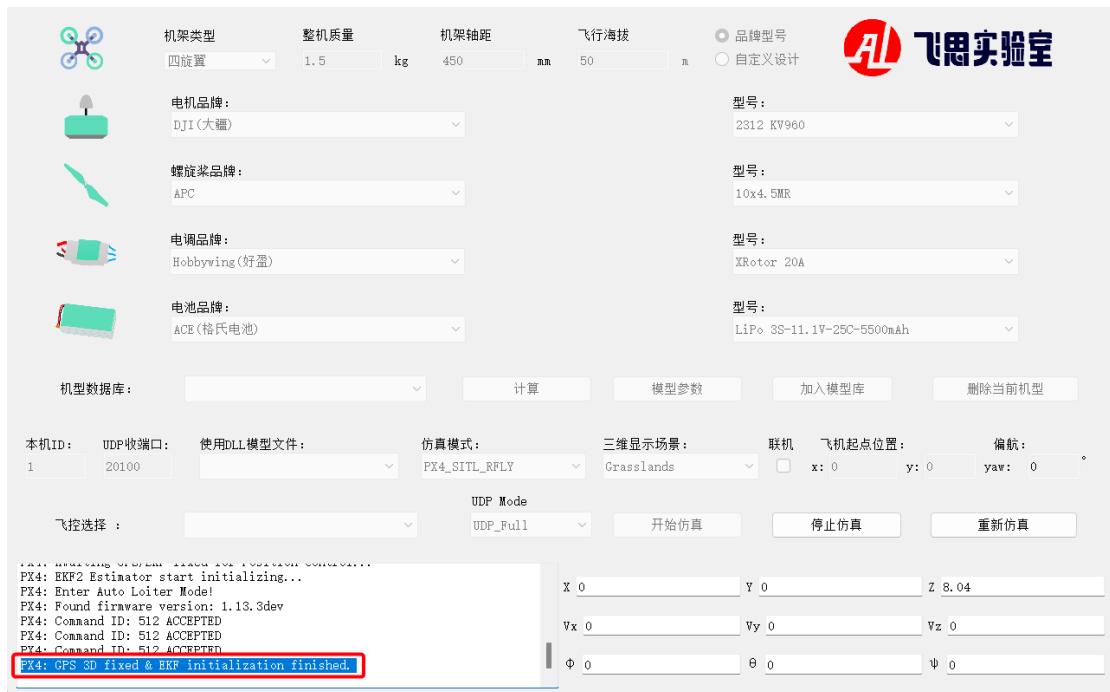
遥控器一般是通过接收机+飞控再与电脑进行链接后，即可再仿真电脑上进行相关实验，但是支持游戏手柄的遥控器，可将通过 USB 线将仿真电脑与遥控器进行链接，如：福斯 FS-i6s 即可通过 USB 线直连仿真电脑的 QGC，在进行首次仿真之前，需对遥控器进行校准，通过 USB 线链接仿真电脑，打开 QGC 软件在左侧游戏手柄中，点击“校准”，按照图

示进行校准即可。

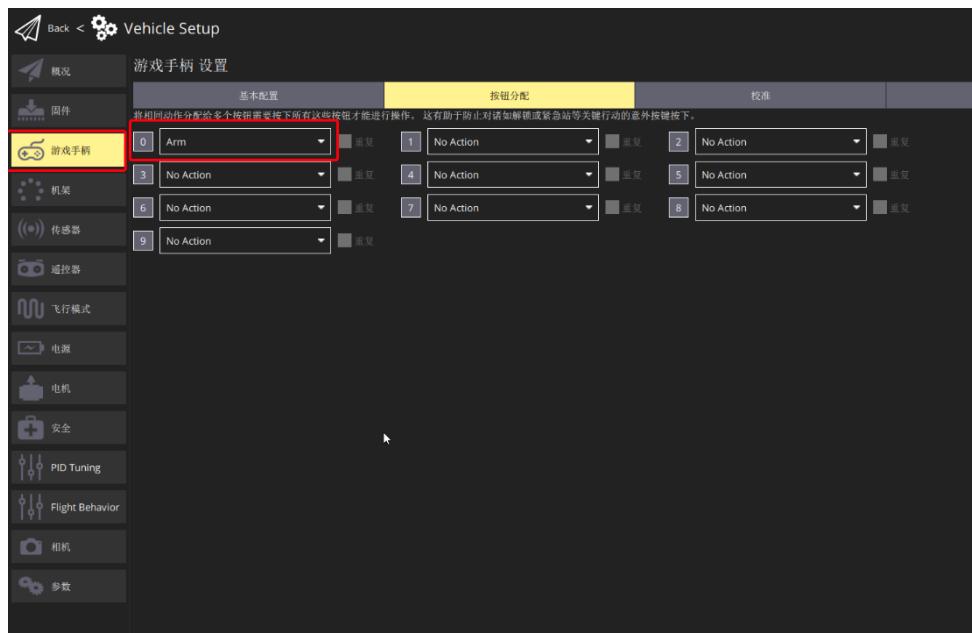


具体操作步骤如下：

1、双击运行 SITLRun 文件，启动一键软件在环仿真脚本，在弹出的 CMD 对话框中输入“1”，按下 Enter 键后，等待 RflySim 平台启动 CopterSim、RflySim3D 及 QGC 软件，等待 CopterSim 消息框显示：“PX4: GPS 3D fixed & EKF initialization finished”。



2、通过 USB 线链接电脑，打开游戏手柄的按钮分配，进行如下设置：



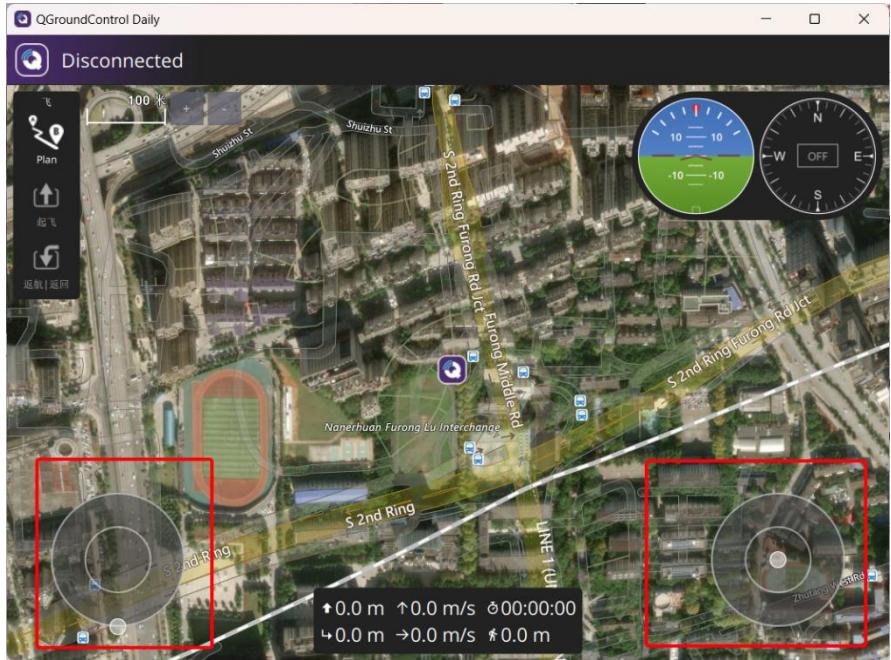
3、即可通过 SWB/CH5 通道进行解锁，在 RflySim3D 中看到飞机起飞。



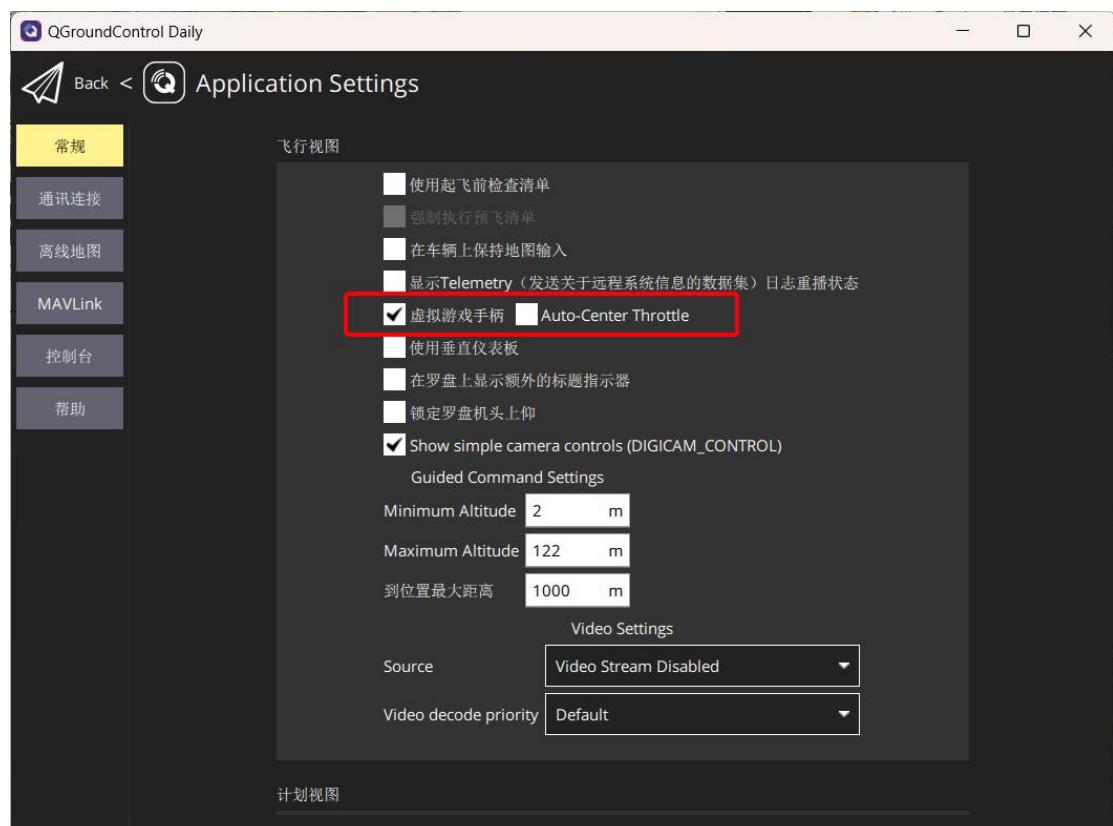
注：设置参数 COM\_RC\_IN\_MODE=1 - Joystick/No RC Checks。

### 1.1.3. QGC 虚拟遥控器控制

QGroundControl 允许您使用屏幕上的虚拟拇指控制车辆。它们在飞行视图中显示如下：



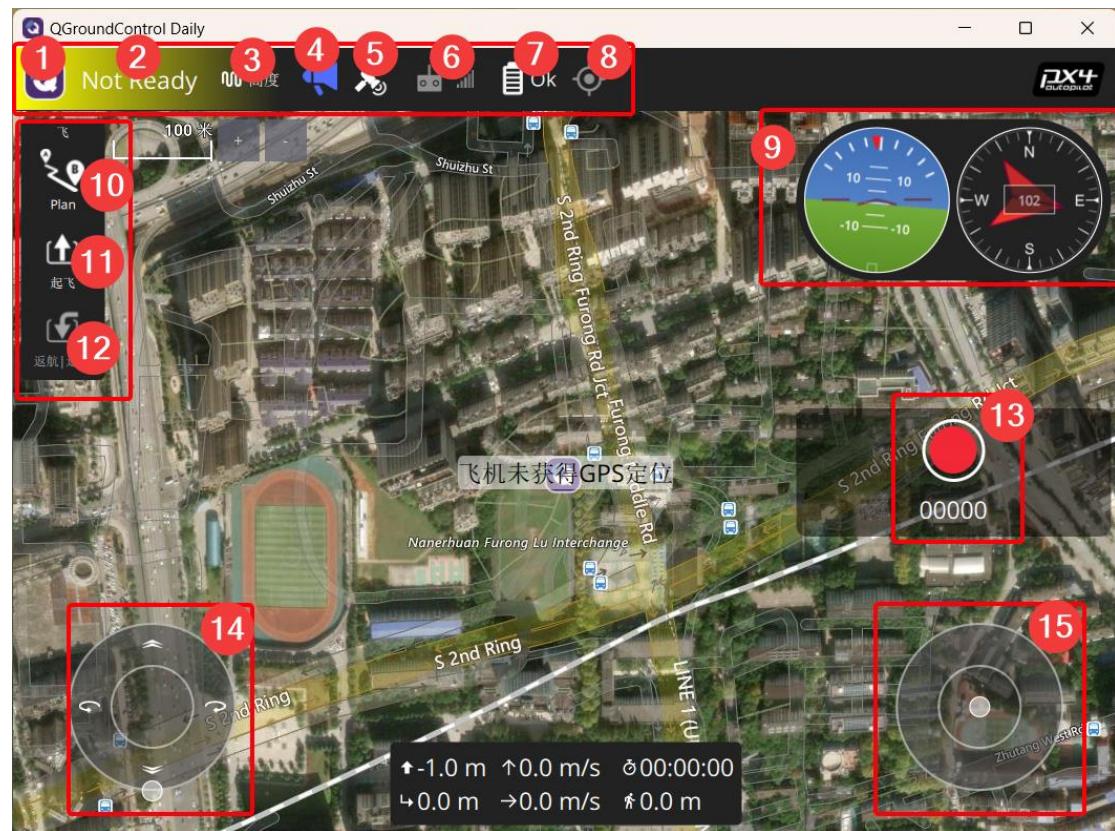
虚拟摇杆控制的响应不如使用 RC 发射机（因为信息是通过 MAVLink 发送的）。另一种选择是使用 USB 游戏杆/游戏板。使能虚拟摇杆：1.从顶部工具栏中选择 Q 图标->Application Settings，然后从侧栏中选择常规。2.选中虚拟游戏手柄复选框。



## 1.2. 地面站控制

### 1.2.1. 参数配置

QGroundControl的整体界面如下图所示，界面中各按钮的相关解释如下所示：



- ① 开始按钮：该按钮可以弹出快捷菜单，可进入载具初始化设置、分析工具使用以及相关的软件属性设置。
- ② 载具状态显示：一般从此处可快速查看载具的整体状态。
- ③ 控制模式选择：该按钮可以切换不同的控制模式，如：手动、自稳、特技等等控制模式。
- ④ 通知：此处可查看载具运行时的信息，如：警告信息、错误信息等。
- ⑤ GPS 状态：显示当前载具所能搜到的卫星数量。
- ⑥ 手柄链接状态显示。
- ⑦ 电池电量显示。
- ⑧ ROI 区域识别。
- ⑨ IMU 状态实时仪表盘。
- ⑩ 航线规划。
- ⑪ 起飞按钮。
- ⑫ 返航按钮。
- ⑬ 录制按钮：可录制 QGC 界面视频。

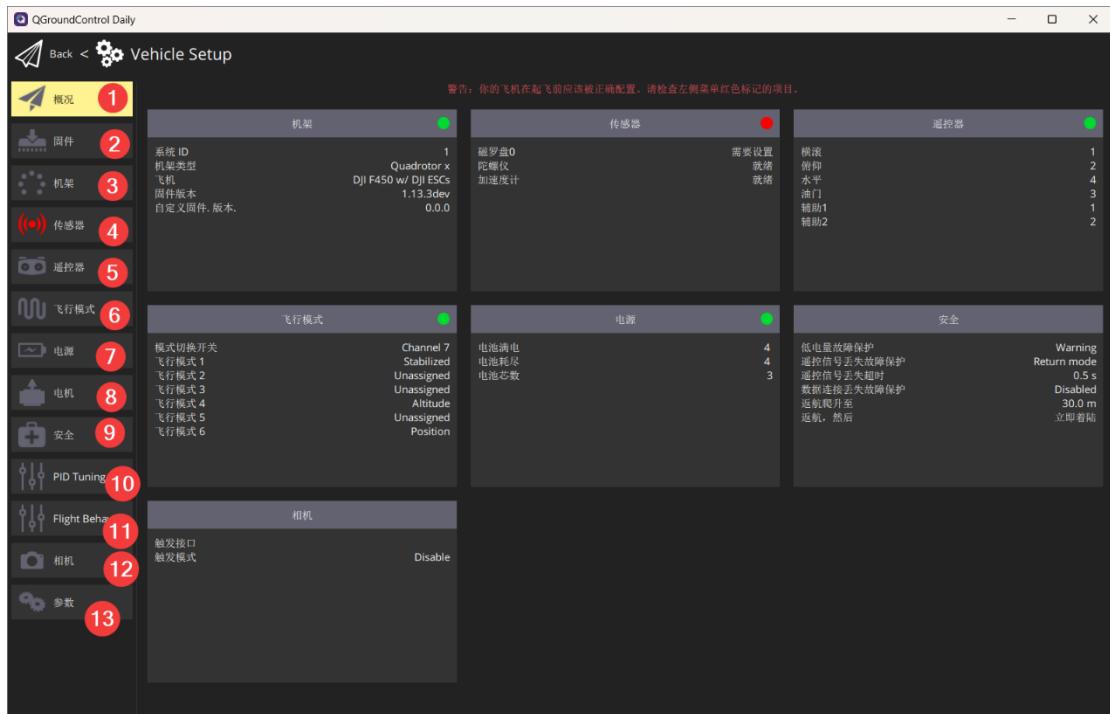
⑯ 虚拟手柄 CH3/CH4 通道。

⑰ 虚拟手柄 CH1/CH2 通道。

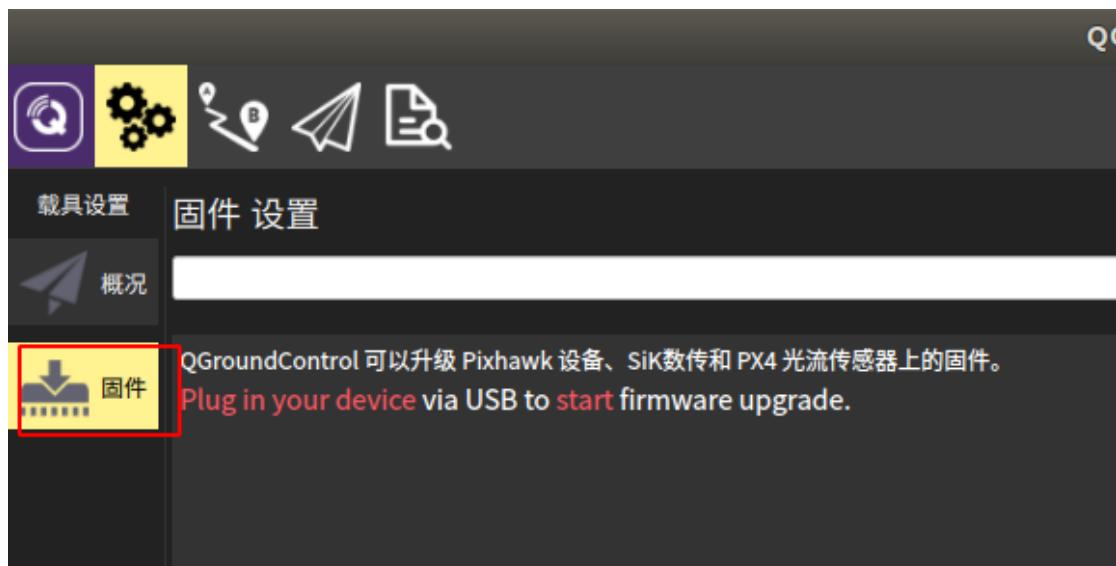
## 1.2.2. 按键命令（起飞、降落、返航）(Vehicle Setup)

QGC 地面站初始界面可以看到的相关按键命令如下

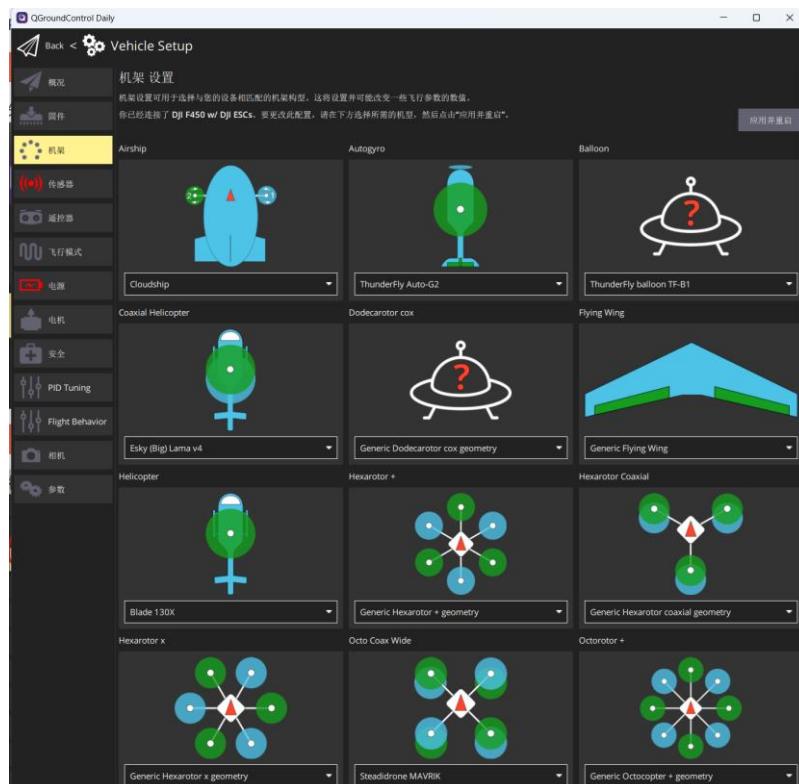
<https://docs.qgroundcontrol.com/master/en/SetupView/SetupView.html>



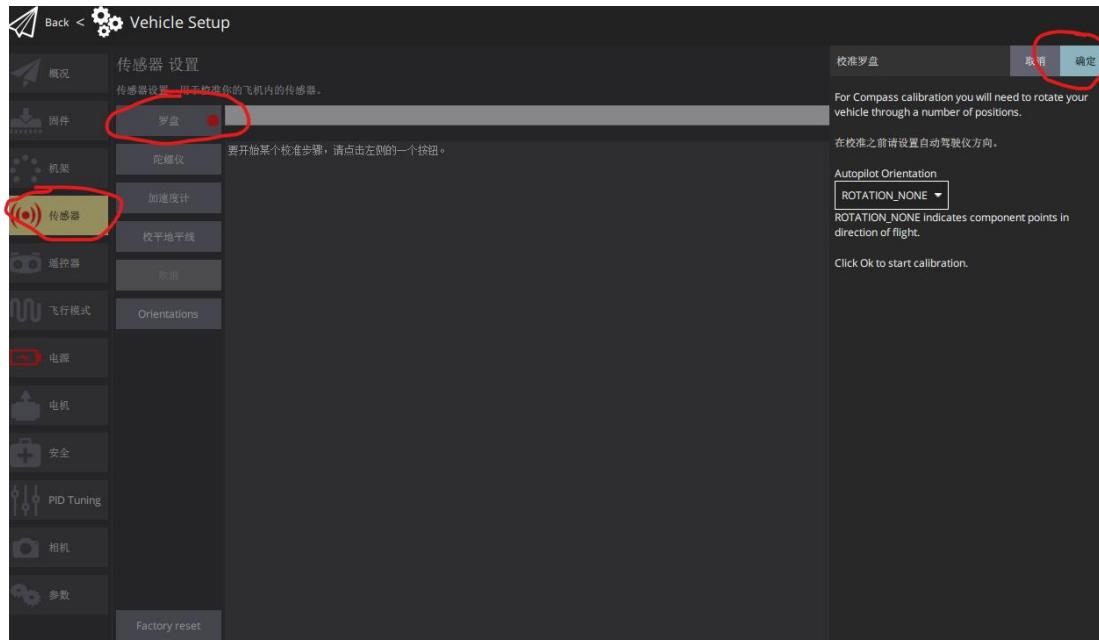
- ① 载具概况：显示当前所连接的载具整体状态情况，如：机架、传感器、遥控器、飞行模式等等。
- ② 固件：先不连接飞控，点击如下页面，然后用 USB 将飞连接电脑，注意飞控不要用电池或其他 USB 以外的设备供电。



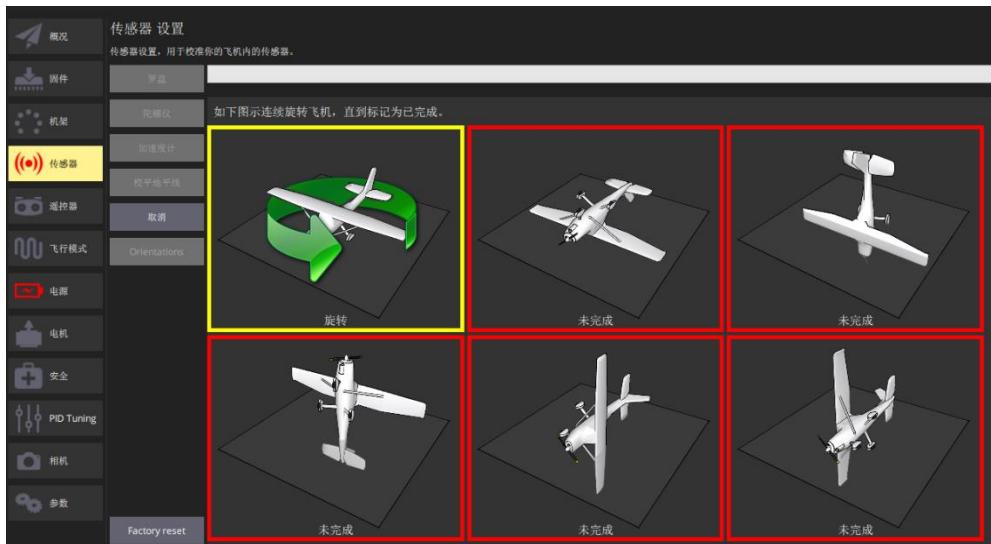
- ③ 机架：将飞控连接至地面站，将机架设置为自己想设置的机型，设置完后请右上方“应用并重启”才可生效。



- ④ 传感器：该传感器中主要包含 IMU 中涉及到的传感器，校准时，一般先校准罗盘，步骤如下：

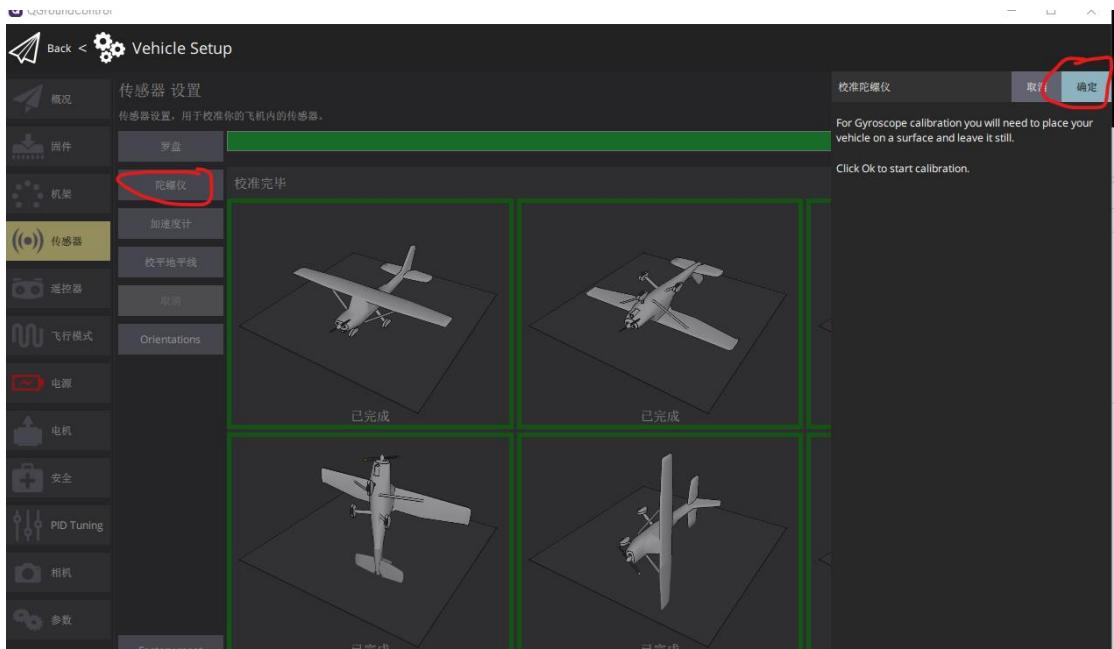


将无人机置于红色所示的任何方向，并保持静止。出现提示后（方向图像变为黄色），沿任意/两个方向绕指定轴旋转车辆。当前方向校准完成后，屏幕上的相关图像将变为绿色。



对所有方向重复校准过程。在所有方向校准完毕后，QGroundControl 将显示 Calibration complete (校准完成) (所有方向图像将显示为绿色，进度条将完全填满)。然后可以继续下一个传感器。

**校准陀螺仪：**单击陀螺仪传感器按钮，将无人机水平放在地面上，保持静止。单击“确定”开始校准。顶部的条形图满代表校准成功。

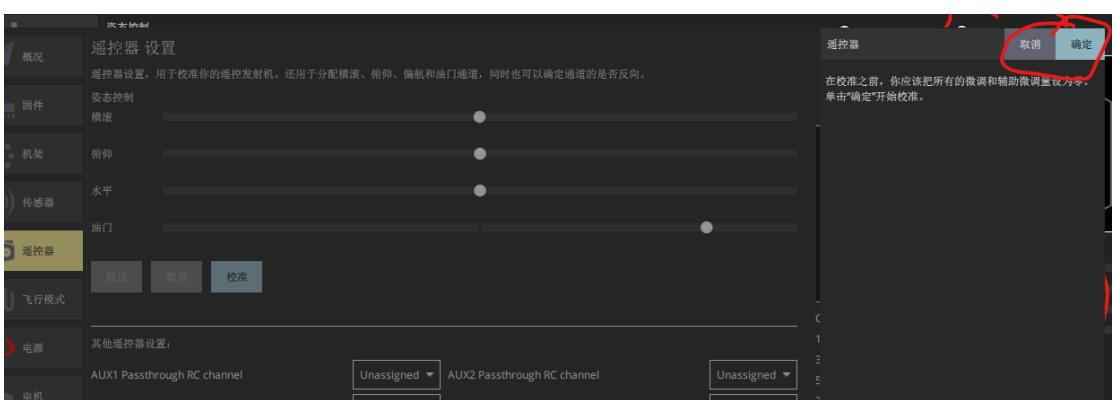


## ⑤ 遥控器

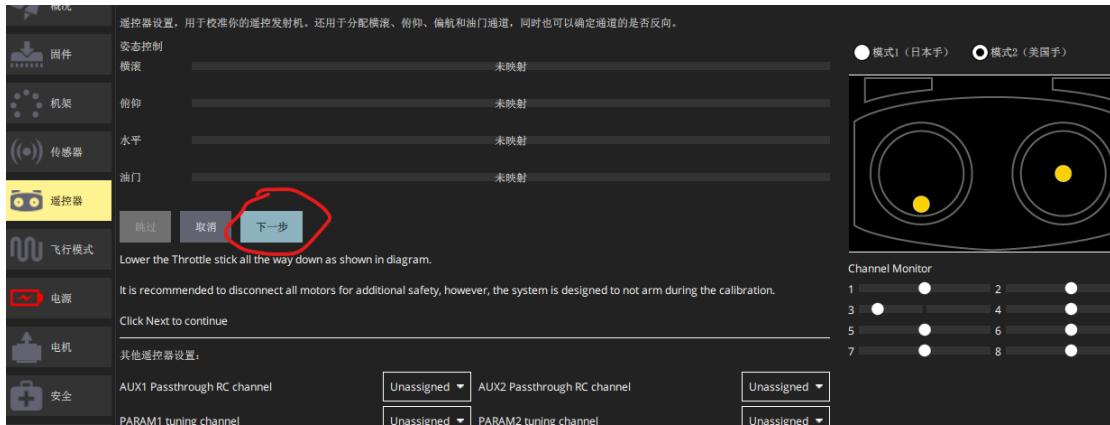
打开遥控器，切换到遥控器页面，检查右下角是否能识别到通道，如果能识别到通道，就可以进行校准，选择右上角的操作方式，然后点击校准



然后点击“确定”



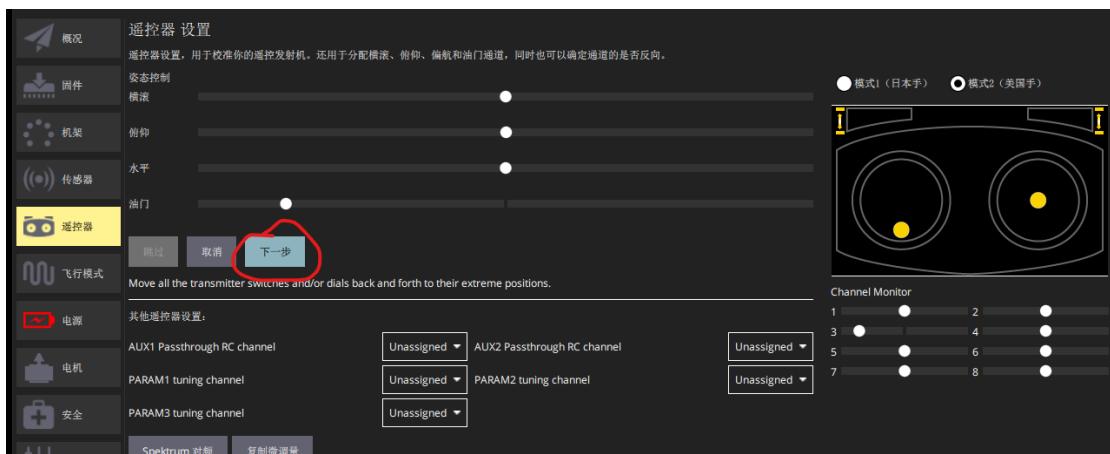
再点击“下一步”



将遥控器摇杆移动到下图中指示的位置。

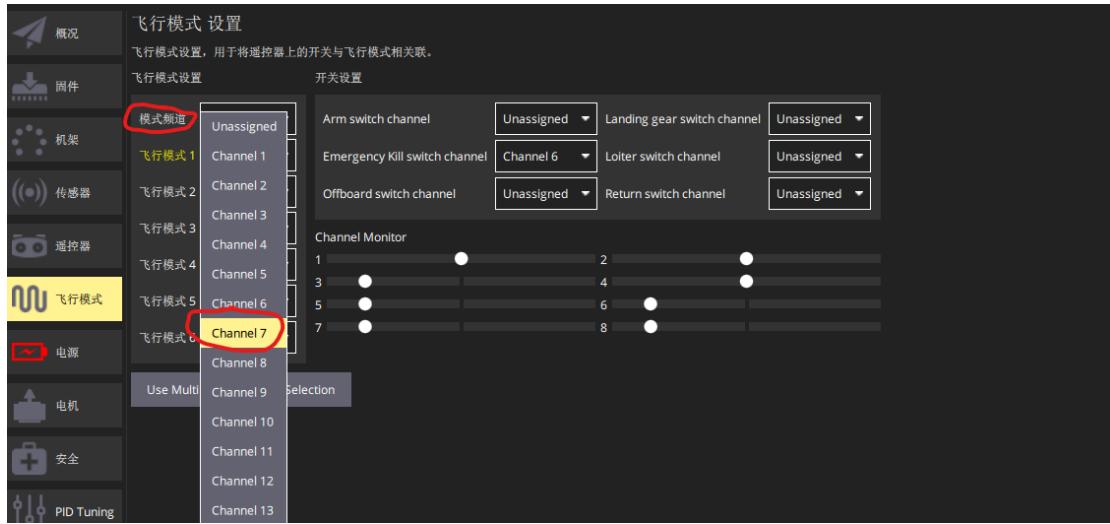


当杆就位时，地面站会提示下一个需要拨的位置，拨完所有位置后，按两次“下一步”保存设置。

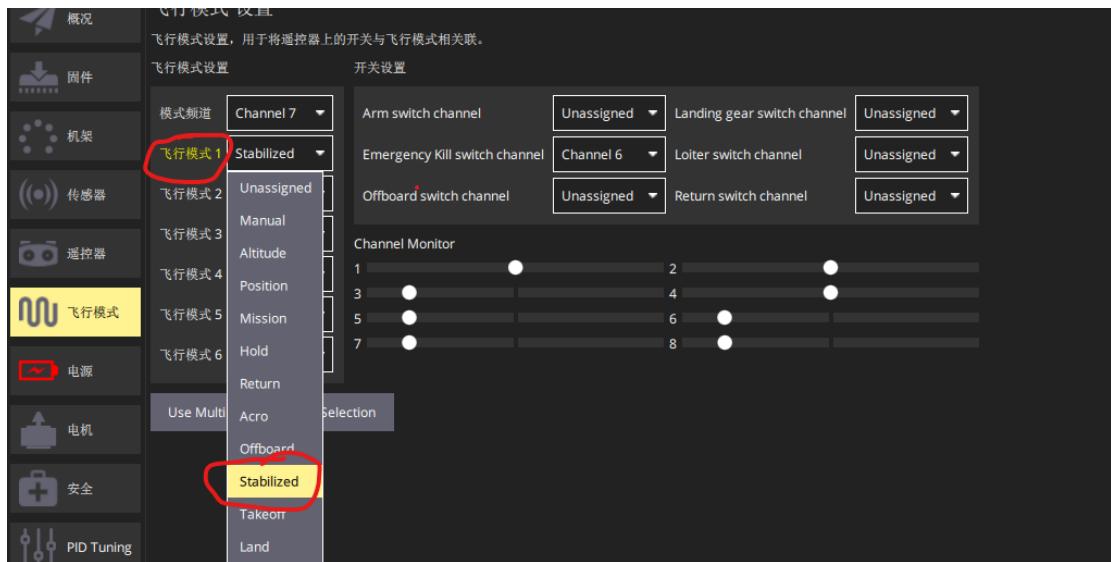


## ⑥ 飞行模式切换开关

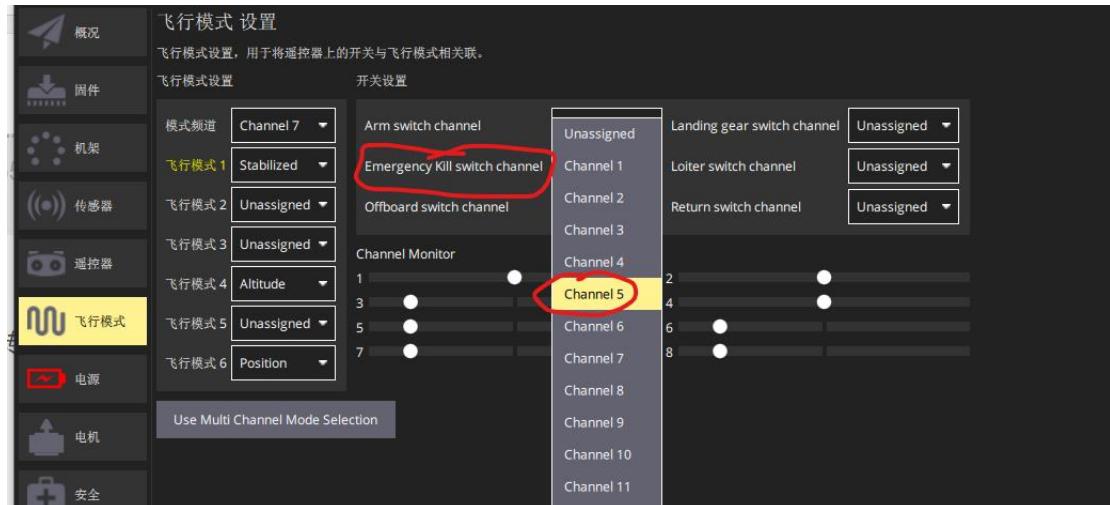
点击“模式频道”右侧的复选框，设置相应的遥控器拨码开关通道。



然后分别设置三档对应的飞行模式。

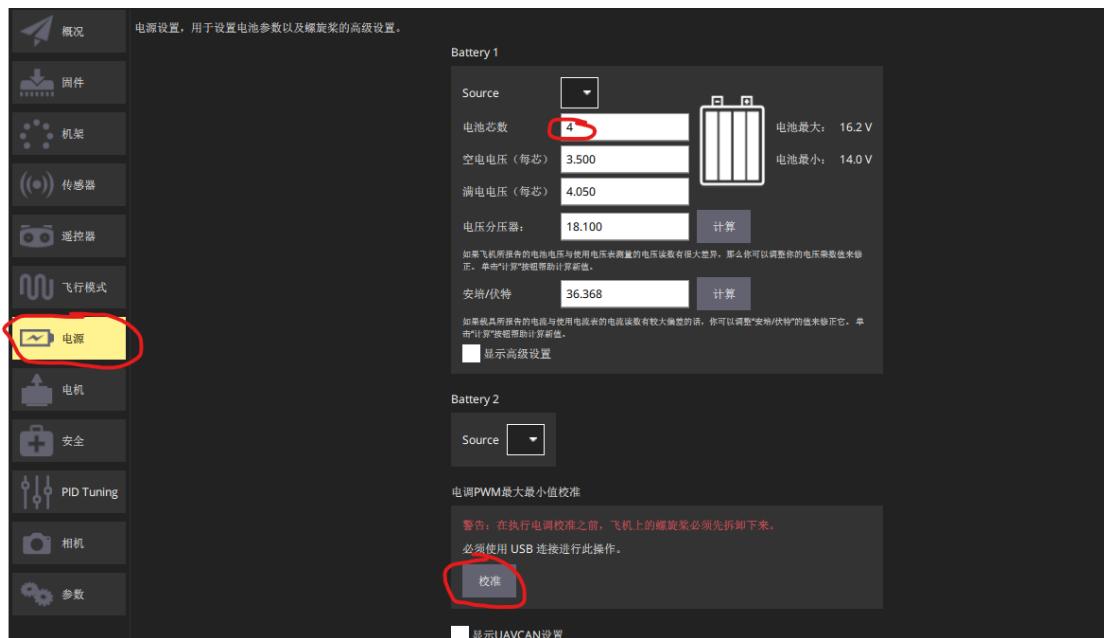


其他的开关通道在飞行模式右侧，如下，需要设置哪个，就把这个开关右侧的遥控器通道进行设置即可，我这里设置了一个刹车（Kill switch），通道为遥控器的第五个通道。刹车的作用是使电机直接停转，可根据需要进行设置



## ⑦ 电源

校准电调时，用 USB 将飞控连接到地面站，不接电池，不装桨叶，电调的信号线接到飞控上。切换到“电源”页面，输入电池芯数并回车，点击“校准”，然后插上电池即可校准。



⑧ 电机：显示电机的 PWM

⑨ 安全：该菜单下可对载具的低电量故障保护触发器、物体探测、遥控器信号丢失故障进行相关设置。

⑩ PID 整定：可对载具的 PID 控制参数进行整定。

⑪ Flight Behavior。

⑫ 相机设置

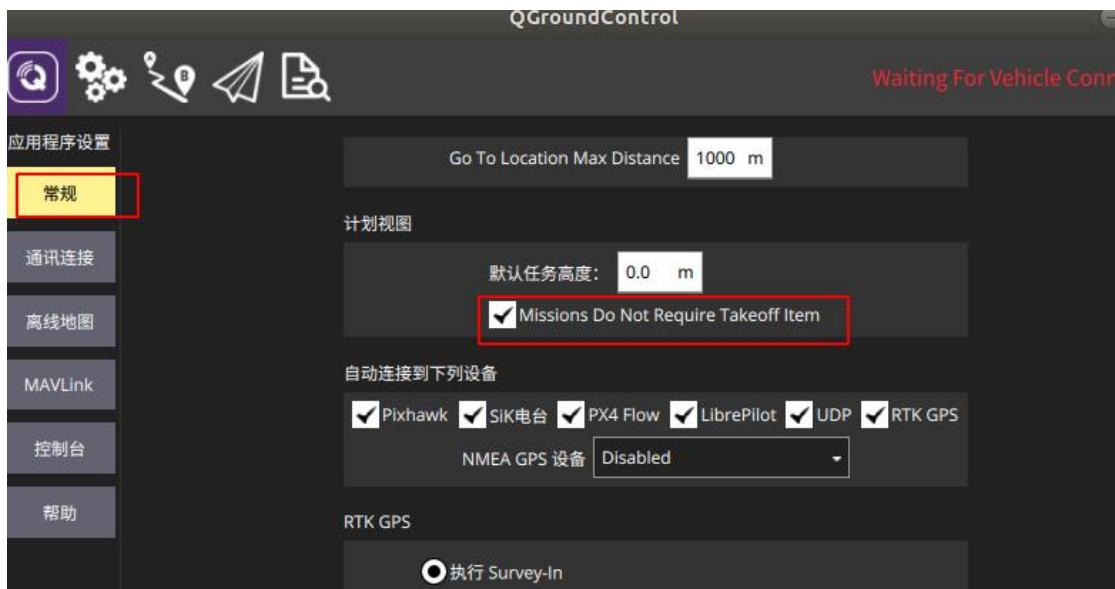
⑬ 参数：此处可修改 PX4 软件中定义的任意参数，修改完成后需重启才可生效。

## 1.2.3. 航线规划（Plan）

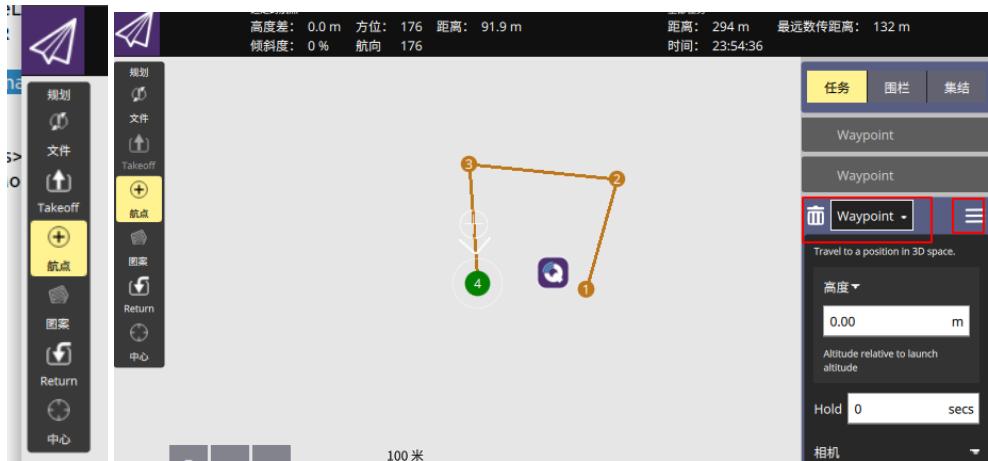
航路规划分支持手动打点功能，点击文件->空白，默认必须先设置 takeoff 点，才能设置其他航点



如果想不设置 takeoff 点直接设置航点（例如无人船等载具），可以在地面站常规设置中勾选下图选项：



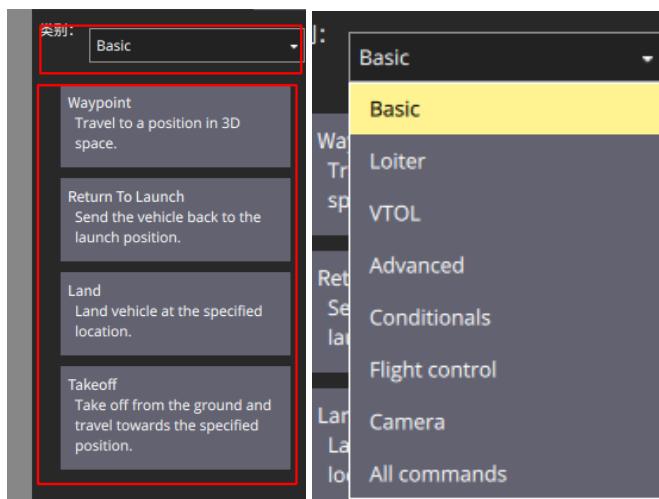
然后就可以不用设置起飞点直接设置航点了。点击“takeoff”后，会在无人机当前位置生成起飞点，然后再点击“航点”按钮，然后在地图上点击即可设置航点位置。设置完的航点也可以鼠标左键选中后进行拖动改变其位置。这里点击起飞点后，继续设置了三个航点，一共四个点，如下图，地图右侧有每个行点的设置页面。



点击相应航点左侧的小垃圾桶图标可以删除航点。

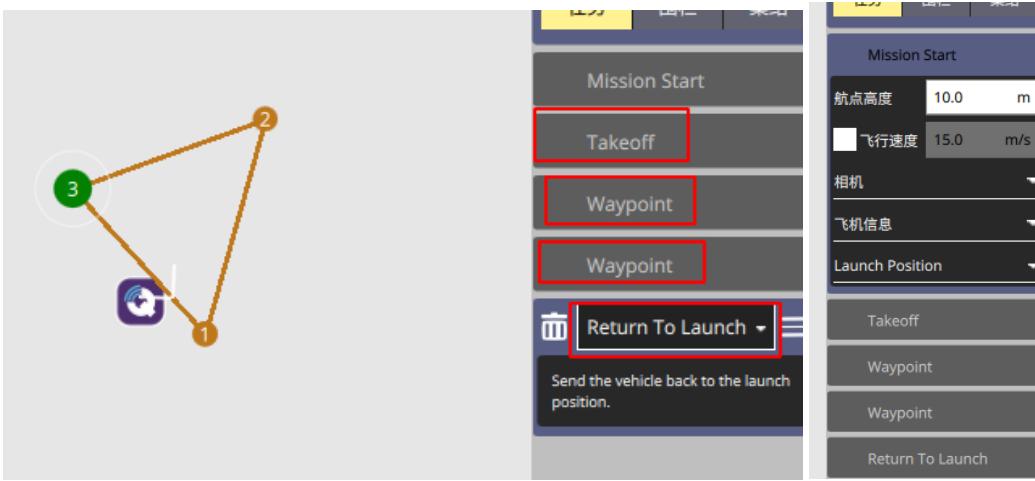


点击小垃圾桶图标右侧的 Waypoint 可以设置点的类型，默认是 Basic 类别，下面有 Waypoint(航点)、Return To Launch(返回起飞点)、Land(降落)、Takeoff(起飞)四种，分别对应到达该航点后执行的动作。上面四种点对于普通四旋翼的使用来说已经够了，但是如果需要执行一些高级一些的动作（如垂起模式切换），需要在下面的其他类别中去找。

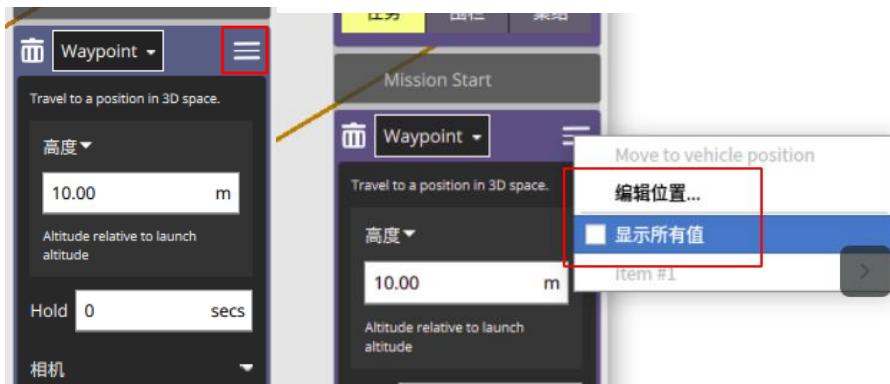


这里将第一个点设置为起飞点、第二和第三个点设置为航点，第四个点设置为返航点，这样的话无人机会起飞并先飞到第二个点，再飞到第三个点，在到达第三个点后返航到起

飞点。在 Mission Start 那一栏可以设置航点高度和飞行速度等值，这一栏的设置会对所有任务点都有效。



在每个任务点里面也可以设置高度和停留时间等值，但这里的设置只对该任务点有效，在每个任务点的右侧有一个三道杠的图标，可以点击该图标设置该任务点的更多信息



“编辑位置”可以对位置进行详细编辑。“显示所有值”可以对任务点的所有参数进行编辑

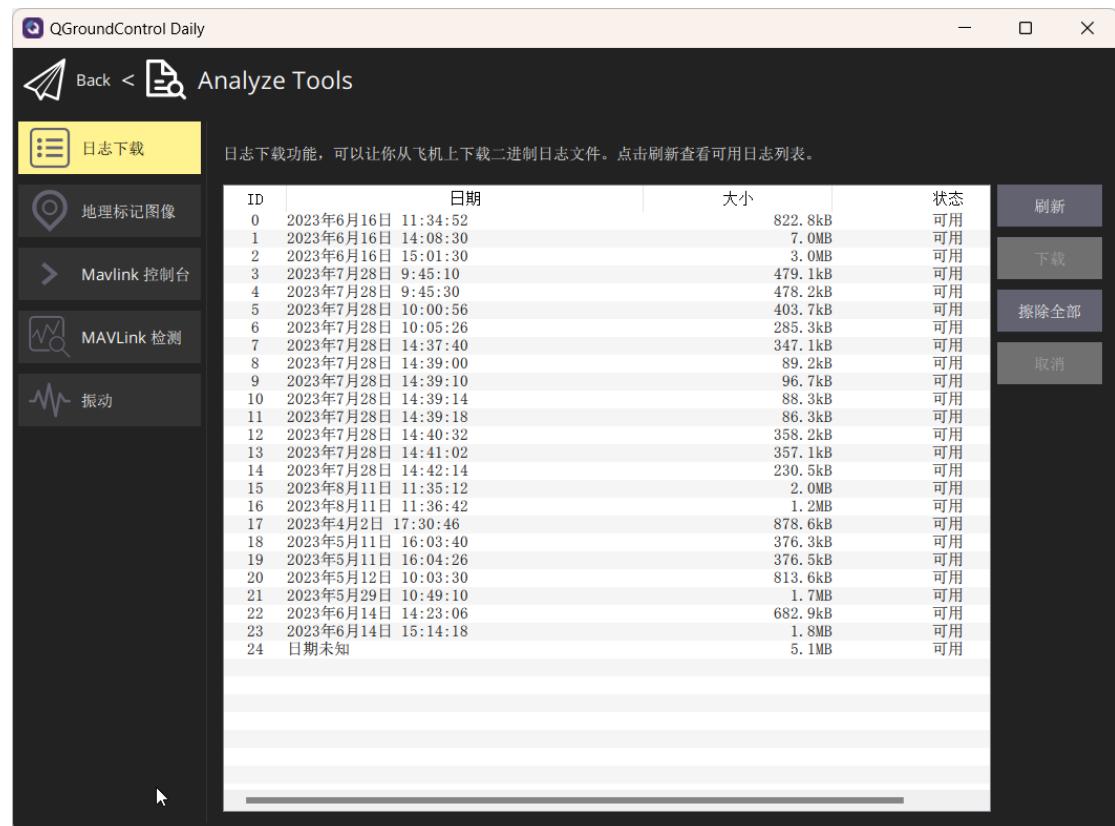


## 1.2.4. 数据分析 (Analyze Tools)

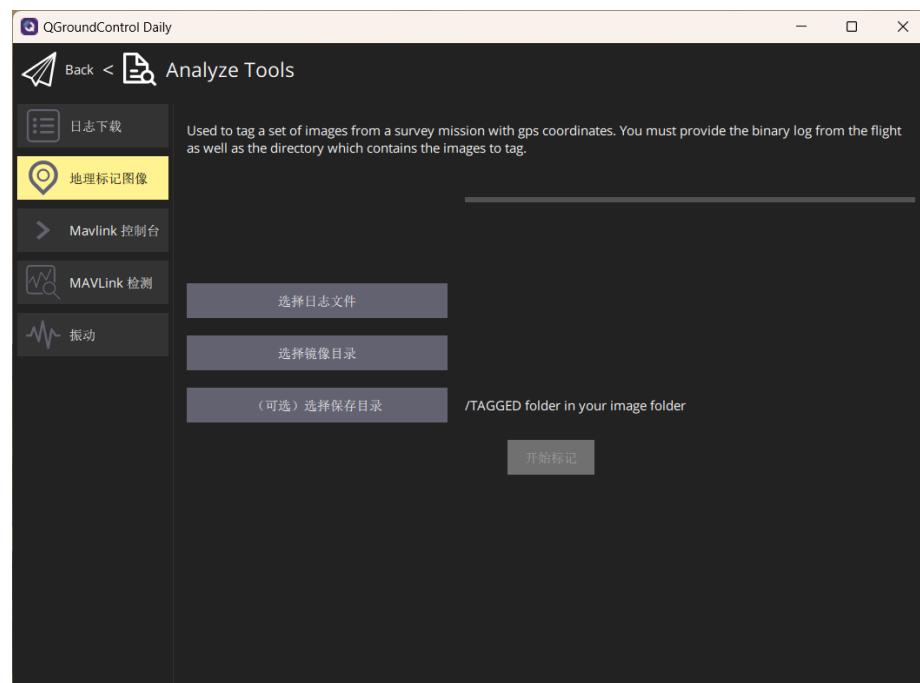
QGC 提供丰富的数据分析工具，主要有日志下载、地理标记图像、MAVlink 控制台、

MAVLink 检测、振动。

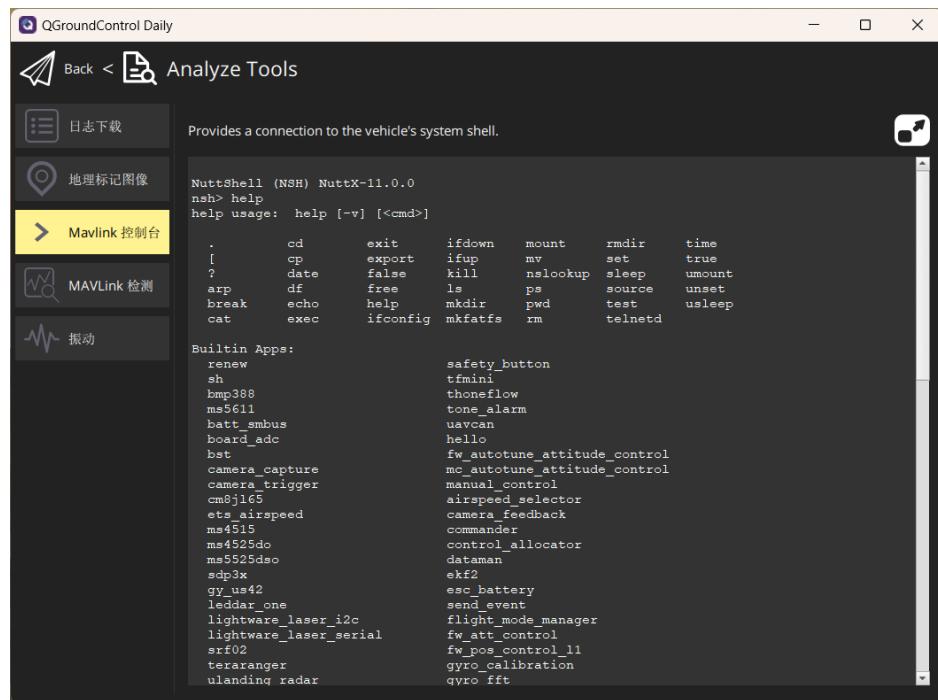
**日志下载:** 可以在链接飞控的情况下, 再在当前飞控内存卡内存储的日志信息, 选择任意一条即可下载到.ulg 格式的文件, 该文件可通过网站: [https://docs.px4.io/main/zh/log/file\\_log\\_analysis.html](https://docs.px4.io/main/zh/log/file_log_analysis.html) 进行分析日志



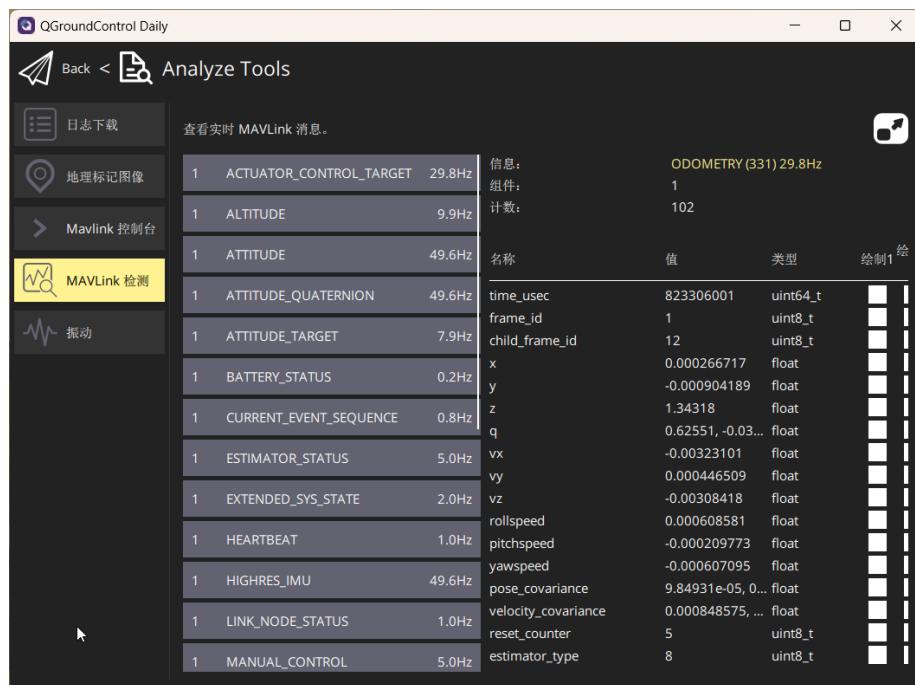
**地理标记图像:** 用于用 gps 标记一组调查任务的图像, 但必须提供航点的二进制日志以及包含要标记的图像的目录。



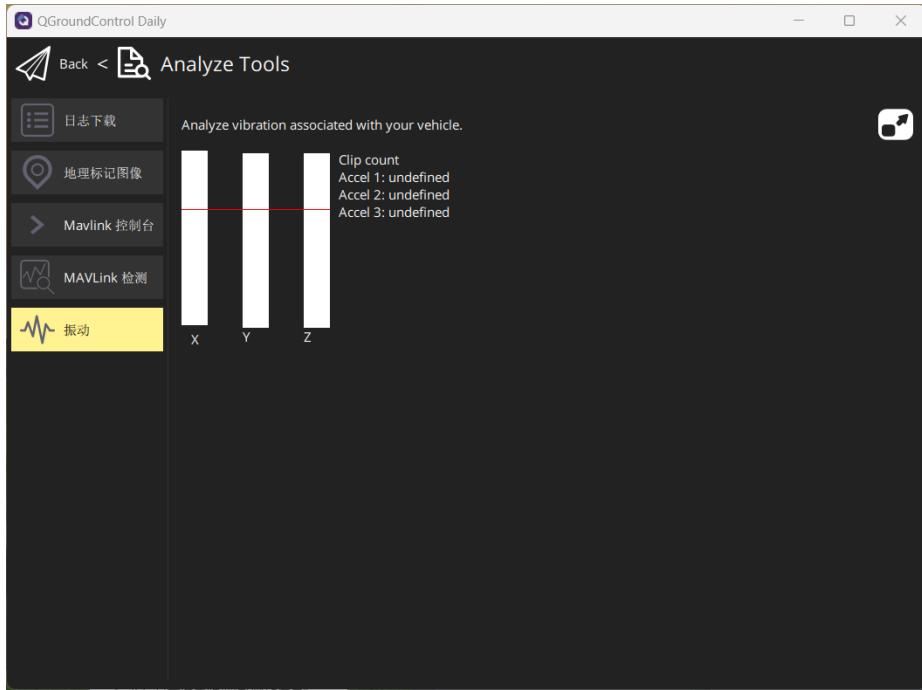
**MAVLink 控制台：**它提供一个可与载具上的飞控运行系统 Nuttx 的 Shell 进行数据通信链接。



MAVLink 检测:

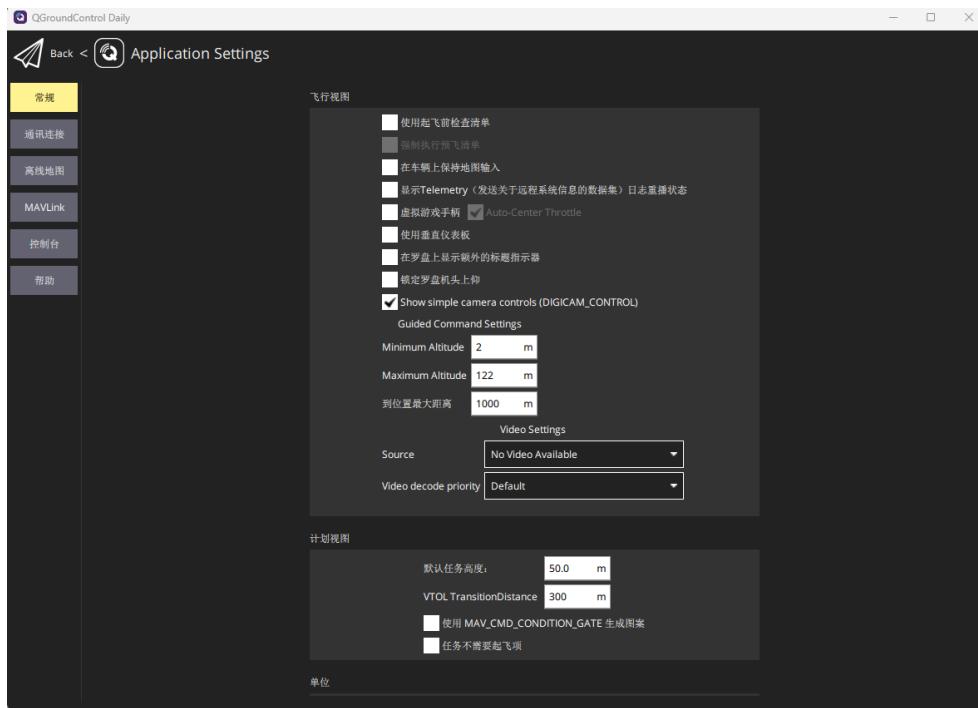


**振动：**分析与车辆相关的振动

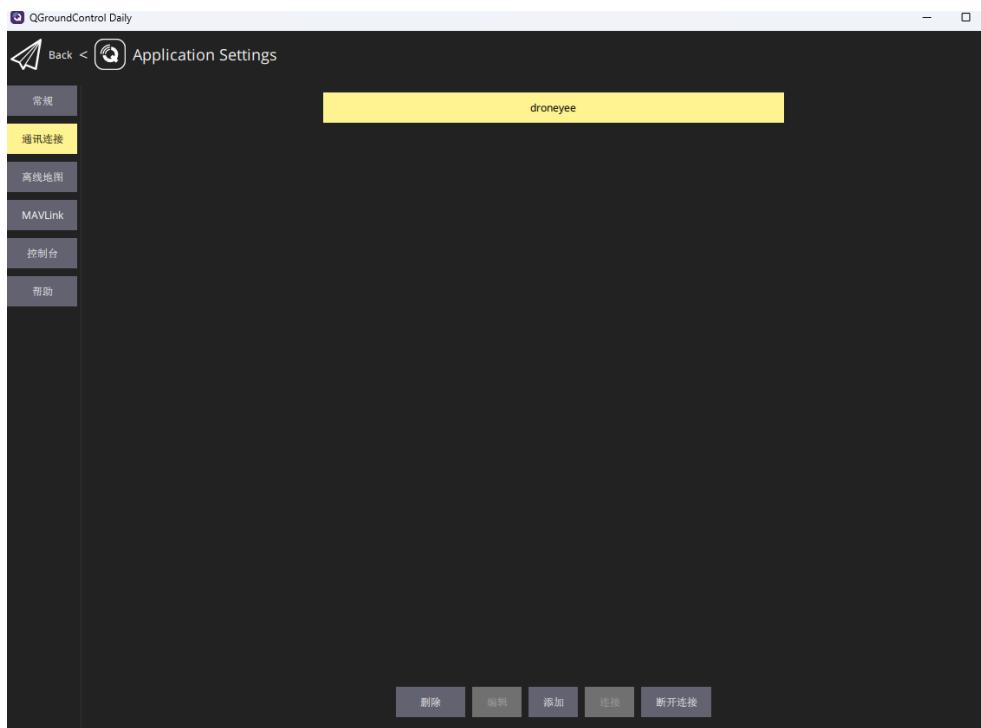


## 1.2.5. 地面站设置 (Application settings)

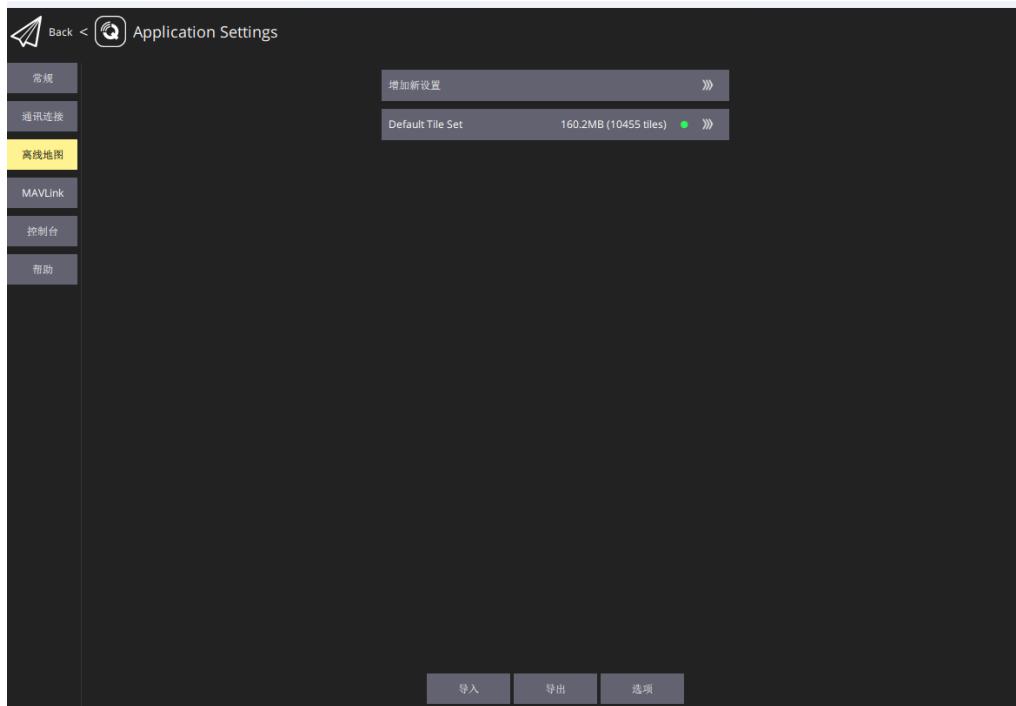
**常规选项:** 主要为应用程序的设置，这些用于指定：显示单元、自动连接设备、视频显示和存储、RTK GPS 等。



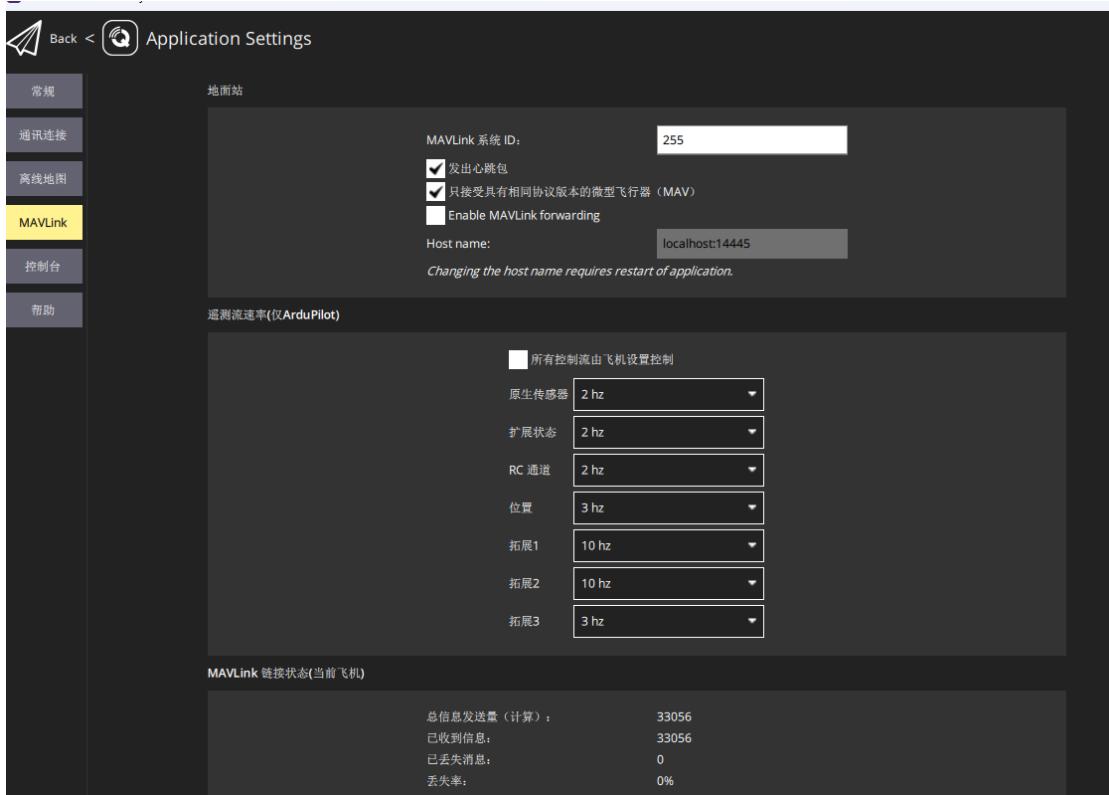
通信连接:允许您手动创建通信连接



**离线地图:** 允许您缓存地图，以便在没有互联网连接时使用



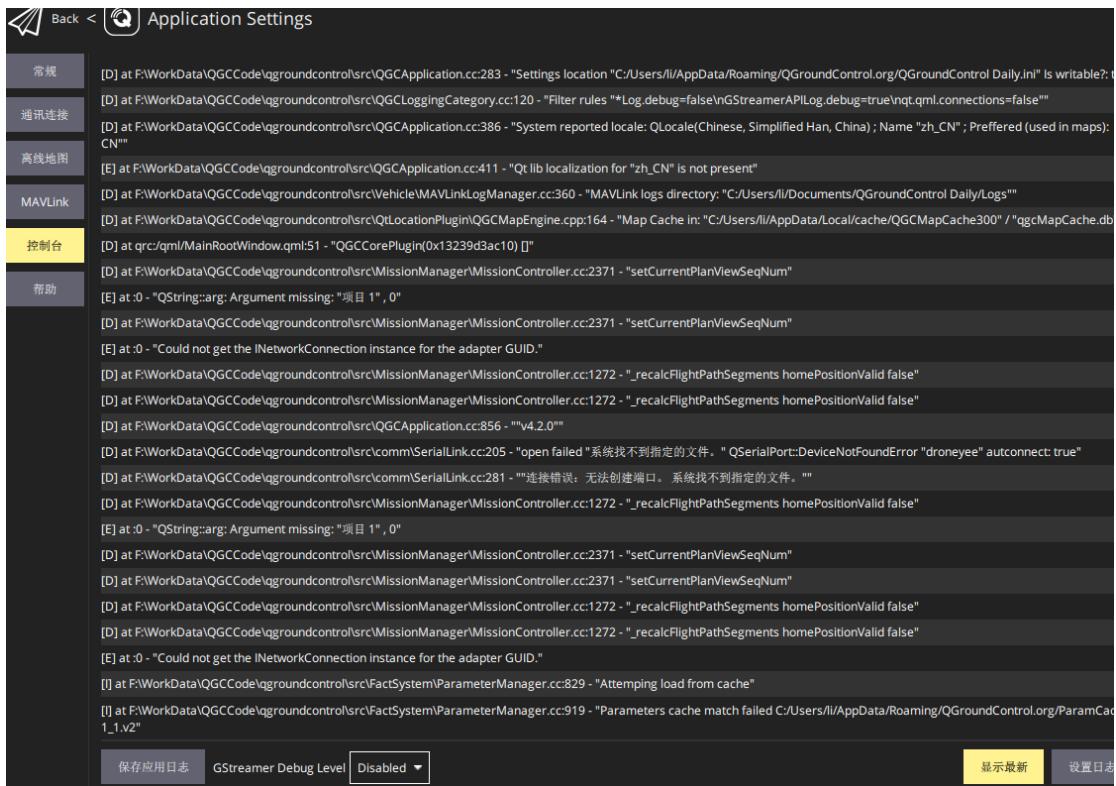
**MAVlink:** 允许您配置选项并查看特定于 MAVLink 通信的消息。这包括为 QGC 设置 MAVlink 系统 ID 和查看连接质量。



**控制台：**用于捕获应用程序日志以帮助应用程序问题

#### 常用的日志记录选项

选项	描述
LinkManagerLog,MultiVehicleManagerLog	调试连接问题。
LinkManagerVerboseLog	调试未检测到串行端口。可用串行端口的连续输出非常嘈杂。
FirmwareUpgradeLog	调试固件闪存问题。
ParameterManagerLog	调试参数加载问题。
ParameterManagerDebugCacheFailureLog	调试参数缓存 crc 未命中。
PlanManagerLog,MissionManagerLogGeoFenceManagerLogRallyPointManagerLog	调试计划上传/下载问题。
RadioComponentControllerLog	调试无线电校准问题。



## 1.3. 控制设备与通信介质

### 1.3.1. 无线数传 (PC 通过数传连 PX4 并进行控制)

无线数传 (选配) 可以用于建立 QGroundControl 地面站与 PX4 飞控之间的无线 MAVLink 连接。本节包含两个主题：已经支持的无线数传 和 在 PX4 系统中集成新的数传。

<http://docs.px4.io/main/en/telemetry/index.html> 中列举了 PX4 已支持的无线数传系统。包括使用 SiK Radio 固件的数传 和 3DR WiFi 无线数传。

### 1.3.2. 有线串口模块 (NX 通过有线串口连 PX4 并进行控制)

端口预配置：以下功能通常映射到所有单板上相同的特定串行端口，因此默认映射：

- MAVLink 被映射到拉 1 端口, 端口的波特率为 57600(对于遥测模块)。
- GPS 1 (GPS 驱动程序)被映射到具有自动波特率的 GPS 1 端口(使用此设置，GPS 将自动检测波特率-除了 Trimble MB-Two, 它需要 115200 波特率)。
- 在具有以太网端口的 Pixhawk 设备上，使用 MAV\_2\_CONFIG 将 MAVLink 映射到以太网端口。
- 其他端口默认没有指定功能(禁用)。

配置端口：

- 重新启动车辆以使附加配置参数可见。
- 将所选端口的波特率设置为所需值。

- 
- 配置模块特定参数(即 MAVLink 流和数据速率配置)。

GPS/Compass > Secondary GPS 部分提供了一个如何在 QGroundControl 中配置端口的实际示例(它展示了如何使用 GPS\_2\_CONFIG 在 TELE2 端口上运行辅助 GPS)

### 1.3.3. WIFI 模块（带有机载板卡进行消息转发）

WiFi 遥测技术使车辆上的 WiFi 无线电与 GCS 之间的 MAVLink 通信成为可能。WiFi 通常比普通的遥测无线电提供更短的距离，但支持更高的数据速率，并且更容易支持 FPV/视频馈送。通常车辆只需要一个无线电单元(假设地面站已经有 WiFi)。

PX4 支持通过 UDP 和 Wifi 遥测。它将心跳广播到端口 14550(255.255.255.255)，直到它从地面控制站接收到第一个心跳，此时它只会向这个地面控制站发送数据。

兼容的 WiFi 数传模块有：

- ESP8266 WiFi 模块
- ESP32 WiFi Module
- 3DR Telemetry Wifi (Discontinued)

## 2. RflySim 平台控制模式接口

### 2.1. 常规飞行控制模式

#### 2.1.1. 起飞模式：

起飞飞行模式使车辆起飞到指定的高度，等待进一步的输入。该模式需要一个良好的位置估计（如，从 GPS 中获取）。使用此模式前必须先解锁。这种模式是自动的，不需要用户干预来控制载具。遥控开关可以在任何无人机上更改飞行模式。在多旋翼中移动遥控器摇杆（或 VTOL 在多旋翼模式下）默认情况下会将无人机切换到位置模式，除非是处理电池失效保护。如果起飞时出现问题，故障检测器 将自动停止引擎。

多旋翼 (MC) 上升到 MIS\_TAKEOFF\_ALT 中定义的高度并保持位置。遥控器摇杆移动会把无人机切换到位置模式（默认）。起飞受以下参数影响：、

参数	描述
MIS_TAKEOFF_ALT	起飞期间的目标高度 (默认值: 2.5 米)
MPC_TKO_SPEED	上升速度 (默认值: 1.5 m/s)
COM_RC_OVERRIDE	控制多旋翼 (或者多旋翼模式下的 VTOL) 的摇杆移动量来切换到 <b>位置模式</b> 。可以分别为自动模式和 offboard 模式启用此功能，默认情况下在自动模式下启用此功能。
COM_RC_STICK_OV	导致发射机切换到 <b>位置模式</b> 的摇杆移动量 (如果 COM_RC_OVERRIDE 已启用)。

更多机型起飞模式解释请见：[http://docs.px4.io/v1.13/zh/flight\\_modes/takeoff.html](http://docs.px4.io/v1.13/zh/flight_modes/takeoff.html)

## 2.1.2. 降落模式:

陆地飞行模式使车辆降落在该模式被使用的位置，降落后，无人机将会在一小段时间后上锁（默认情况下）。该模式需要有效的位置估计，除非由于失效保护进入该模式，这种情况下仅需要高度估计（通常飞控内置一个气压计）。这种模式是自动的，不需要用户干预来控制载具。遥控器开关可以用于更改任何无人机的飞行模式。在多旋翼中移动遥控器摇杆（或 VTOL 在多旋翼模式下）默认情况下会将无人机切换到位置模式，除非是处理电池失效保护。

无人机将降落在模式所指定的位置。无人机以 MPC\_LAND\_SPEED 指定的速度下降，降落后会上锁（默认）。遥控器摇杆移动会把无人机切换到 位置模式（默认）。着陆受以下参数影响：

参数	描述
MPC_LAND_SPEED	着陆过程中的下降速率。鉴于地面情况未知，这个值应该设得相当小。
COM_DISARM_LAND	降落后自动上锁的超时时间，以秒为单位。如果设定为 -1，无人机将不会在着陆时上锁。
COM_RC_OVERRIDE	控制多旋翼（或者多旋翼模式下的 VTOL）的摇杆移动是否将控制权交给位置模式下的飞手。可以分别为自动模式和 offboard 模式启用此功能，默认情况下在自动模式下启用此功能。
COM_RC_STICK_OV	导致发射机切换到 位置模式 的摇杆移动量（如果 COM_RC_OVERRIDE 已启用）。

更多机型该模式解释请见：[http://docs.px4.io/v1.13/zh/flight\\_modes/land.html](http://docs.px4.io/v1.13/zh/flight_modes/land.html)

## 2.1.3. 定点/悬停（盘旋）模式:

定点/悬停（盘旋）模式(又名“等待” / “徘徊” )使载具停止并保持其当前的 GPS 位置和高度(多旋翼将悬停在 GPS 位置，而固定翼将绕其旋转)。该模式可用于暂停任务或帮助您在紧急情况下重新控制车辆。它通常通过预编程开关激活。多旋翼无人机悬停在当前位置和高度。遥控器摇杆移动会将无人机切换到 位置模式（默认）。可以使用以下参数配置此动作。

参数	描述
MIS_LTRMIN_ALT	留待模式的最小高度 (如果模式在较低的高度进行, 则飞行器将上升到此高度)。
COM_RC_OVERRIDE	控制多旋翼 (或者多旋翼模式下的 VOTL) 的摇杆移动量来切换到 <b>位置模式</b> 。可以分别为自动模式和 offboard 模式启用此功能, 默认情况下在自动模式下启用此功能。
COM_RC_STICK_OV	导致发射机切换到 <b>位置模式</b> 的摇杆移动量 (如果 COM_RC_OVERRIDE 已启用)。

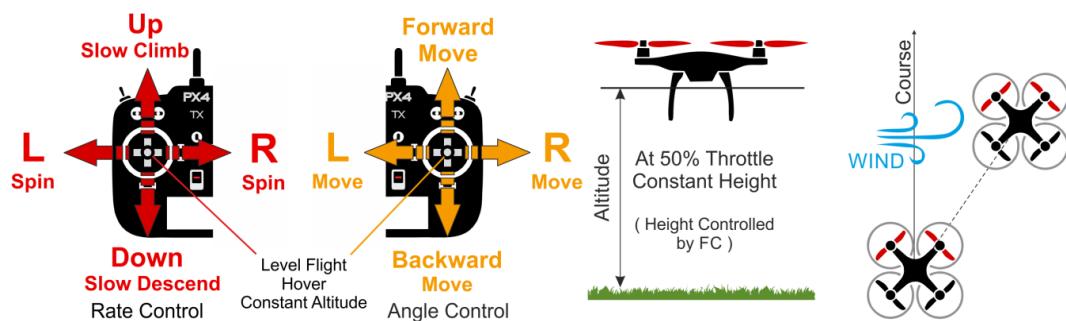
更多机型该模式解释请见: [http://docs.px4.io/v1.13/zh/flight\\_modes/hold.html](http://docs.px4.io/v1.13/zh/flight_modes/hold.html)

## 2.1.4. 任务模式:

任务模式使飞行器执行已上传到飞行控制器的预定义的自主任务(飞行计划)。通常使用地面站(GCS)应用程序如 QGroundControl(打开新窗口)(QGC)创建和上传任务。通常在地面控制站 (例如, [QGroundControl \(opens new window\)](#)) 中创建任务并在发射之前上载。它们也可以由开发者 API 创建, 和/或在飞行中上传。单个任务命令的处理方式适合于每个飞行器的飞行特性(例如, 直升机的盘旋和固定翼的盘旋)。 VTOL 飞机在固定翼模式下遵循固定翼的行为和参数, 在多旋翼模式下遵循旋翼机的行为和参数。更多解释请见: [http://docs.px4.io/v1.13/zh/flight\\_modes/mission.html](http://docs.px4.io/v1.13/zh/flight_modes/mission.html)

## 2.1.5. 定高模式:

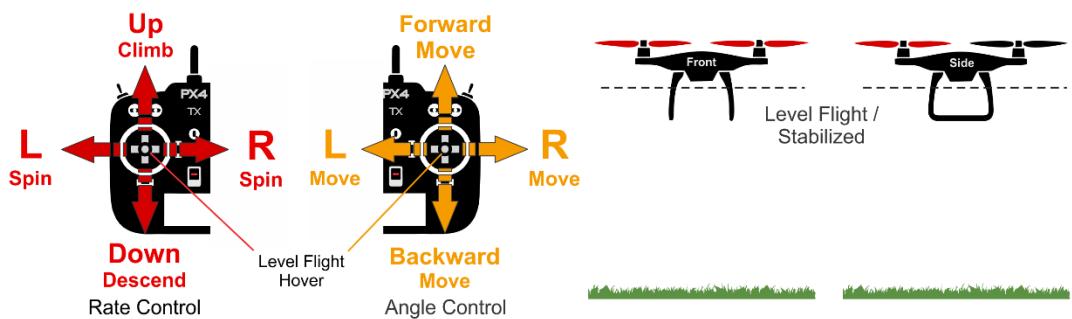
高空模式是一种相对容易飞行的 RC 模式, 其中滚转杆和俯仰杆控制飞行器在左右和前后方向(相对于飞行器的“前方”)的运动, 偏航杆控制在水平面上的旋转速率, 油门控制上升-下降速度。当操纵杆释放/居中时, 车辆将水平并保持当前高度。下图直观的展示了该模式 (以美国手的遥控器举例)。更多解释请见: [http://docs.px4.io/v1.13/zh/flight\\_modes/altitude\\_mc.html](http://docs.px4.io/v1.13/zh/flight_modes/altitude_mc.html)



## 2.1.6. 自稳/手动控制 (姿态角控制模式) 模式:

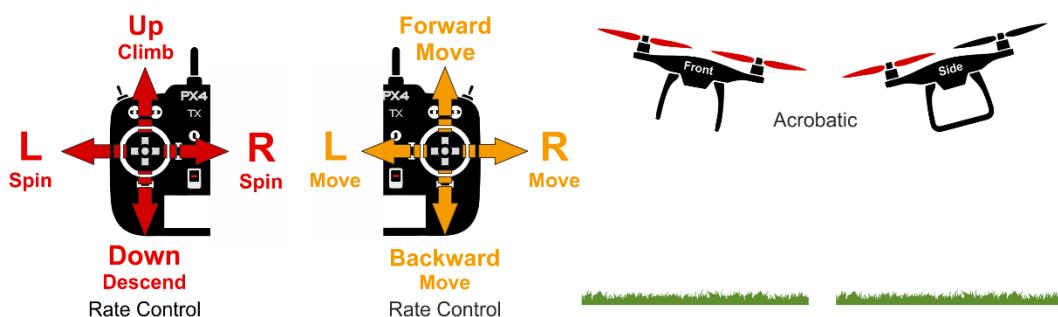
手动/稳定模式稳定多直升机时, RC 控制棒集中。要手动使机体移动/飞, 您可以移动摇杆使其偏离居中位置。如果您设置手动或稳定模式, 则启用多旋翼模式。在手动控制下,

横滚杆和俯仰杆控制车辆围绕各自轴的角度(姿态)，偏航杆控制水平面上的旋转速率，节流阀控制高度/速度。一旦释放摇杆，它们将会返回中心死区。一旦横滚和俯仰摇杆居中，多旋翼无人机将平稳并停止运动。然后机体将悬停在适当的位置/保持高度 - 前提是平衡得当，油门设置适当（在下方查看），并且没有施加任何外力（例如风）。飞行器将朝着任何风的方向漂移，您必须控制油门以保持高度。更多解释请见：[http://docs.px4.io/v1.13/zh/flight\\_modes/manual\\_stabilized\\_mc.html](http://docs.px4.io/v1.13/zh/flight_modes/manual_stabilized_mc.html)



## 2.1.7. 特技（角速度控制）模式：

该模式是 RC 模式，用于执行杂技动作，如翻转，翻滚和循环。滚动、俯仰和偏航杆控制围绕相应轴的旋转角速率，并且油门直接传递到输出混合器。当操纵杆居中时，飞机将停止旋转，但保持其当前朝向（在其侧面，倒置或任何其他方向）并根据当前动量移动。更多解释请参考：[http://docs.px4.io/v1.13/zh/flight\\_modes/acro\\_mc.html](http://docs.px4.io/v1.13/zh/flight_modes/acro_mc.html)



## 2.1.8. 返航模式：

该模式用于将车辆飞行到安全的无障碍路径上，到达安全的目的地，在那里它可以等待(悬停或盘旋)或着陆。PX4 提供了几种机制来选择安全的返航路径，返航目的地和着陆，包括使用其实位置，集结（“安全”）点，任务路径和任务着陆顺序。

更多解释请见：[http://docs.px4.io/v1.13/zh/flight\\_modes/return.html](http://docs.px4.io/v1.13/zh/flight_modes/return.html)

## 2.2. 外部控制模式

(参考 [https://docs.px4.io/main/en/flight\\_modes/offboard.html](https://docs.px4.io/main/en/flight_modes/offboard.html))

---

PX4 能够通过有线或者 wifi 被独立的辅助计算机控制，伙伴计算机通常通过 MAVLink API 进行通信，这种控制模式称为外部控制模式或者 Offboard 控制。载具执行辅助计算机通过 MAVLink 设定的位置、速度、姿态指令。Offboard 模式主要用于做空中机动，至于起飞、降落、返航则选用相应专有的模式更为合适。

Offboard 控制常作为上层算法开发的基本接口，如视觉避障、集群编队需要使用 Offboard 的位置和速度接口；如特技表演，则需使用更底层的姿态接口。除此之外，Offboard 消息也支持控制云台相机、武器装备等，只需将 target\_component 的值设为相应组件的 ID 即可。

Offboard 控制有一些关键参数，这些参数可以通过 QGC 查看或者修改。下表列举了几个关键的参数，用户可能在使用 Offboard 控制时遇到信号丢失的情况，而且信号丢失后进入了特定的模式，这些行为受下列参数的控制。

Offboard 控制关键参数表

COM_OF_LOSS_T	信号丢失超过该时间将触发 offboard lost failsafe。
COM_OBL_ACT	无 RC 信号下，offboard lost failsafe 触发后切换到的模式。0: Land, 1: Hold, 2: Return
COM_OBL_RC_ACT	有 RC 信号下，offboard lost failsafe 触发后切换到的模式。0: Position, 1: Altitude, 2: Manual, 3: Return , 4: Land
COM_RC_OVERRIDE	控制 stick movement 是否触发切换到位置模式。默认未使能。
COM_RC_STICK_OVERRIDE	stick movement 触发切换到位置模式的数量。(COM_RC_OVERRIDE 使能)

注意：

- 期望状态的设置必须大于 2Hz，否则无法激活该模式或者载具会自动退出该模式。
- 该模式需要位置或姿态信息。
- RC（遥控器）控制只允许切换模式。
- 载具必须在解锁后才能使用该模式。
- 不是所有 MAVLink 支持的坐标系和域值都被支持。

## 2.2.1. 控制消息

主要使用了三条 MAVLINK 消息

SET\_POSITION\_TARGET\_LOCAL\_NED ([https://mavlink.io/en/messages/common.html#SET\\_POSITION\\_TARGET\\_LOCAL\\_NED](https://mavlink.io/en/messages/common.html#SET_POSITION_TARGET_LOCAL_NED))

SET\_POSITION\_TARGET\_GLOBAL\_INT (<https://mavlink.io/en/messages/common.html>)

---

```
ml#SET_POSITION_TARGET_GLOBAL_INT)
    SET_ATTITUDE_TARGET (https://mavlink.io/en/messages/common.html#SET\_ATTITUDE\_TARGET)
```

MAVLINK 与 Offboard 控制相关的消息包含三个：**SET\_POSITION\_TARGET\_LOCAL\_NED**、**SET\_POSITION\_TARGET\_GLOBAL\_INT**、**SET\_ATTITUDE\_TARGET**，而第一个接口又是这三个消息中使用最多的。前两个都是位置控制的消息，包含期望位置、期望速度、期望加速度等信息。这两个位置控制消息的主要差别在于在发送 GPS 坐标时，由于单精度浮点数不能满足精度要求而需要将 GPS 坐标转换成 int型，这就使得这两个位置消息在数据格式上有所不同。

如下表所示为位置控制消息的具体字段详细信息，总共包含 16 个字段，填充了颜色的字段是两个位置控制消息有区别的地方。下表消息中 `target_system` 是目标系统 ID，一般连接建立起来之后就能自动获得。在常规的飞行控制中，`target_component` 使用默认值即可。

最为常用的坐标系有两种，一种是 `LOCAL_NED`，在该坐标系下位置、速度、加速度/力都位于 NED 坐标系下。另外一种是 `GLOBAL_INT`，在该坐标系位置将用纬度、经度、高度表示，而且纬度和经度被乘以  $10^7$  并转换为整数进行传输，而速度、加速度/力仍然表示在 NED 坐标系中。如果用户想用其它更特殊的坐标系，则可参考官方文档 <https://mavlink.io/en/messages/common.html> 去设置。

可以看到，偏航角和偏航角速率也在位置控制的消息当中。这是因为 Offboard 控制消息是从用户任务的角度设计的，在做集群编队或者视觉任务时需要转向，所以就把这两个信息也加入到了位置控制消息里面。

SET_POSITION_TARGET_LOCAL_NED/SET_POSITION_TARGET_GLOBAL_INT				
序号	名称	类型	单位	描述
1	<code>time_boot_ms</code>	<code>uint32_t</code>	ms	系统时间
2	<code>target_system</code>	<code>uint8_t</code>		系统 ID，每一个飞控可以认为有唯一的 ID
3	<code>target_component</code>	<code>uint8_t</code>		组件 ID，飞行控制时默认 ID 是 0，如果是控制云台相机或者武器装备则可以使用其它 ID
4	<code>coordinate_frame</code>	<code>uint8_t</code>		最常用的坐标系 MAV_FRAME_LOCAL_NED = 1 最常用坐标系 MAV_FRAME_GLOBAL_INT = 5
5	<code>type_mask</code>	<code>uint16_t</code>		位标志表明控制信息应该被忽略
6	x	float	m	NED 坐标系下 x 位置

	lat_int	int32_t	degE7	WGS84 坐标系纬度乘以 $10^7$
8	y	float	m	NED 坐标系下 y 位置
7	lon_int	int32_t	degE7	WGS84 坐标系经度乘以 $10^7$
8	z	float	m	NED 坐标系高度
	alt			海拔高度
9	vx	float	m/s	NED 坐标系 x 方向速度
10	vy	float	m/s	NED 坐标系 y 方向速度
11	vz	float	m/s	NED 坐标系 z 方向速度
12	afx	float	m/s <sup>2</sup>	NED 坐标系 x 方向加速度或力
13	afy	float	m/s <sup>2</sup>	NED 坐标系 y 方向加速度或力
14	afz	float	m/s <sup>2</sup>	NED 坐标系 z 方向加速度或力
15	yaw	float	rad	偏航角
16	yaw_rate	float	rad/s	偏航角速率

注：填充为蓝色的表示 SET\_POSITION\_TARGET\_LOCAL\_NED 独有的，填充为绿色的是 SET\_POSITION\_TARGET\_GLOBAL\_INT 独有的。

在进行特技飞行等高机动控制时，需要做更底层的控制，SET\_ATTITUDE\_TARGET 消息可以支持这个功能。前面 4 个字段与上面的消息相同，不同的是该消息支持指定四元数的姿态、角速率、油门值。

可以看出，姿态控制消息比位置控制消息更为简单。要想通过指定一系列的姿态值就控制住位置是不够的，还需要指定油门值。所以油门值就出现在了姿态控制消息中。

#### SET\_ATTITUDE\_TARGET

序号	名称	类型	单位	描述
1	time_boot_ms	uint32_t	ms	系统时间
2	target_system	uint8_t		系统 ID，每一个飞控可以认为有唯一的 ID
3	target_component	uint8_t		组件 ID，飞行控制时默认 ID 是 0，如果是控制云台相机或者武器装备则可以使用其它 ID
4	type_mask	uint16_t		位标志表明控制信息应该被忽略
5	q	float[4]		姿态四元数
6	body_roll_rate	float	rad/s	滚转角速率
7	body_pitch_rate	float	rad/s	俯仰角速率
8	body_yaw_rate	float	rad/s	偏航角速率

---

9	thrust	float		油门
10	thrust_body **	float[3]		暂时不用

## 2.2.2. 控制接口

位置接口

速度接口

姿态接口

加速度接口

力的接口

偏航角接口

偏航角速度接口

Offboard 控制支持位置、速度、姿态、加速度、力、角速率控制，并按照用户的使用场景封装在了两类消息中，一类是位置控制消息，另一类是姿态控制消息。无论通过 python 还是 matlab 去控制飞机，都是建立在这两类消息的基础之上。位置、速度等控制信息并不是必须要指定的，但是必须通过 type\_mask 标志位指定哪些信息可以忽略。type\_mask 是 uint16\_t 类型，每一个二进制位可以标识一种状态信息。

SET\_POSITION\_TARGET\_LOCAL\_NED 和 SET\_POSITION\_TARGET\_GLOBAL\_INT 虽然在数据链路中数据格式有所不同，但是它们都会被转换成 vehicle\_local\_position\_setpoint\_s 类型的 uORB 消息。由此可见，NED 坐标是 PX4 位置控制的基准坐标系，而其它坐标系只是为了方便用户描述特定的任务而提供的。

### 1) 位置类接口介绍

**位置接口**。包含 3 个数据，Offboard 位置作为期望位置输入控制器。位置接口因为坐标系的缘故，而比其它接口更为复杂。常用的坐标系包括 LOCAL\_NED 和 GLOBAL\_INT。当为 LOCAL\_NED 时，三个数据都为 float 型；当为 GLOBAL\_INT 时，纬度和经度是 int32\_t 类型，高度为 float 型。位置接口对应 3 个标志位，分别为第 0-2 位，例如当 type\_mask 的第 0 位为 0 时，表明有 x 位置。如果三个分量位置都指定，则第 0-2 位都为 0。

**速度接口**。速度接口属于位置控制类消息，包含 3 个数据。Offboard 指定的期望速度会叠加在位置控制器的输出上，也就是说真正的期望是位置控制器输出的期望速度与 Offboard 指定的期望速度之和。当位置控制器的输出为 0 时，真正的期望速度才是 Offboard 指定的期望速度。不论是 LOCAL\_NED 还是 GLOBAL\_INT，期望速度也是在 NED 坐标系的。速度接口对应的 type\_mask 为第 3-5 位，这些位可以指定也是独立的。

**加速度/力接口**。加速度/力接口属于位置控制类消息，包含 3 个数据。加速度/力会叠加到速度控制器的输出，只有速度控制器的输出为 0 时，期望的加速度才是 Offboard 指定

---

的值。Offboard 指定的加速度或者力当 type\_mask 第 9 位为 1 时表示力，否则表示加速度。而有没有加速度/力则是用第 6-8 位标识。加速度控制当前是不成熟的，虽然支持了该接口，但并不一定能实现较好的控制效果。LOCAL\_NED 和 GLOBAL\_INT 两种情形下，加速度和力都是描述在 NED 坐标系下。

**偏航角接口**。偏航角既可以属于位置控制类消息也可以属于姿态控制类消息，包含 1 个数据。因为位置控制类消息和姿态控制类消息是面向不同用户场景的，所以一般不会同时指定。由此可见，不论是哪种类型的任务，偏航角/偏航角速率是需要指定的。偏航角对应的 type\_mask 是第 10 位，当第 10 位为 0 时表示有偏航角。

**偏航角速率接口**。偏航角速率既可以属于位置控制类消息也可以属于姿态控制类消息，包含 1 个数据。偏航角速率对应的 type\_mask 是第 11 位，当第 11 位为 0 时表示有偏航角速率。

## 2) 姿态类接口介绍【新老接口都没有完整支持】

**角速率接口**。角速率接口属于姿态类接口，包含 3 个数据，分别为滚转、俯仰、偏航角速率。对应的 type\_mask 为第 0-2 位，如果第 0-2 位全部为 0，那么表示所有的偏航角速率都会指定。Offboard 指定的角速率会叠加到姿态控制器的输出之上，只有姿态控制器的输出为 0 时，真正的期望角速率才是 Offboard 指定的期望角速率。

**油门接口**。油门接口属于姿态类接口，包含 1 个数据。对应的 type\_mask 为第 6 位，当第 6 位为 0 时，表明指定油门。

**姿态接口**。姿态数据用四元数描述，总共包含 4 个数据。对应的 type\_mask 为第 7 位，当第 7 位为 0 时表明有姿态数据。姿态不支持独立指定姿态，要指定就必须指定 4 个数据。

### 2.2.3. 典型组合模式

PX4 的 Offboard 协议是非常灵活的，位置、速度、姿态等都可以自由组合。不过在实际的使用中，最为常用的模式往往不是自由组合而是有一些典型的模式。下面以旋翼机和固定翼为例介绍典型的组合模式。

**旋翼机典型组合模式**。如下表所示，列举了旋翼机典型的控制信息组合方式，这些方式可以通过设置上一节中的两类消息实现。RflySim 平台提供了下列模式的接口，用户可以直接调用。用户也可以通过 MAVLINK 自己设置这些消息，这样不仅可以实现下列模式，还可以实现更加高级的模式，如同时指定位置和速度。

此外，旋翼机支持载体坐标系下设定速度。让速度在载体坐标系，是通过 SET\_POSITION\_TARGET\_LOCAL\_NED 设置的。当 coordinate\_frame 设置值为 MAV\_FRAME\_BODY\_NED : 8 时，位置在 NED 坐标系，速度加速度在 BODY 坐标系下。

旋翼机典型组合模式

序号	描述
----	----

---

0	导航坐标系下速度模式[vx,vy,vz, yaw_rate]
1	机体坐标系下速度模式[vx,vy,vz, yaw_rate]
2	导航坐标系下位置模式[x,y,z, yaw]
3	机体坐标系下位置模式[x,y,z, yaw]
4	姿态油门控制指令[滚转、俯仰、偏航(弧度)、油门(0~1)]
5	加速度控制模式[ax,ay,az,yaw]
6	加速度控制模式[ax,ay,az,yaw_rate]

**固定翼的典型组合模式。**固定翼与旋翼机不同，并不能自由的控制飞行任意轨迹，也不支持独立的控制各个方向的速度。固定翼主要控制模式是[posx, posy, posz, speed]。水平位置的指定必须满足一定的约束，因为固定翼并不能停留在某一个特定的位置。固定翼一般不支持指定偏航角，因为固定翼总是让机头朝向前进的方向。

### 3. 控制模型

RflySim 平台提供高精度模型和质点模型两大类，高精度模型常用于高逼真的仿真，载具数量一般不多。而质点模型则可用于大规模集群。高精度模型又分为两类，一类是使用 PX4 的控制器这是最逼真的使用形式，另一类则是将控制器也集成到模型当中称为综合模型。综合模型的优点在于能更加稳定可靠的支持大规模高逼真集群仿真。

#### 3.1. 高精度模型+PX4 控制器 组成的软/硬件在环仿真模型（依靠 CopterSim，从模型组复制）

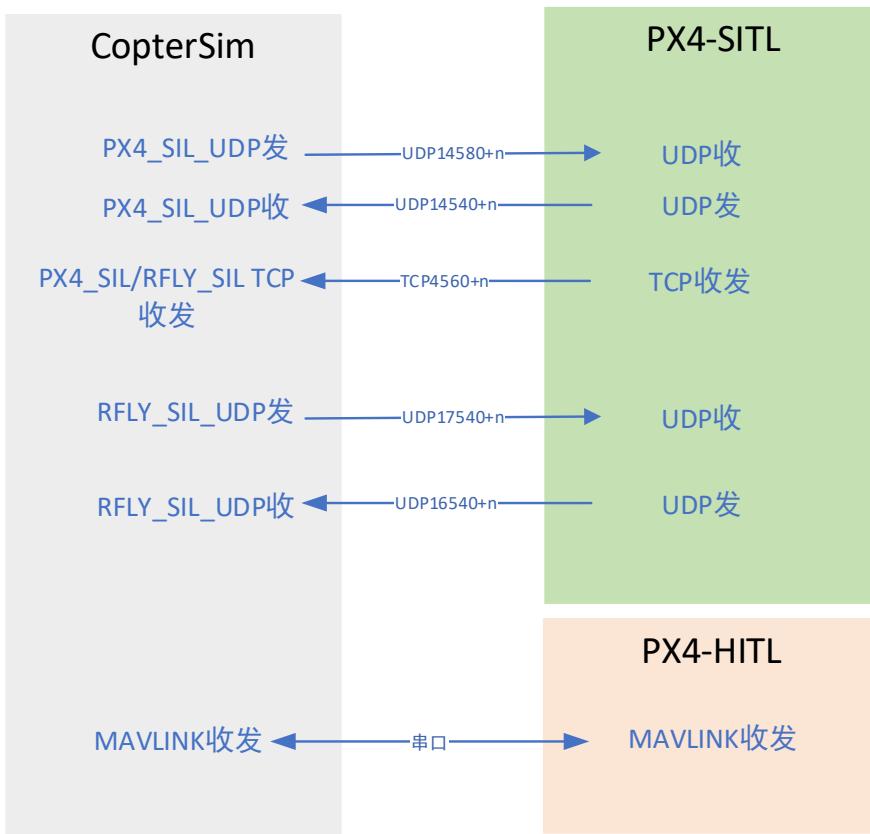
##### 3.1.1. CopterSim 与 PX4 通信端口

CopterSim 可以按如下的方式与 PX4 进行通信。无论是 UDP/TCP/串口，发送的数据包都遵循 MAVLink 协议。下图中主要可以归结为 3 种通信模式。

第一种是 PX4 自带的软件在环仿真模式，对应 CopterSim 的 **PX4\_SITL 模式**。对 CopterSim 而言，使用 UDP14580+n 端口发消息，使用 UDP14540+n 端口发消息或者使用 TCP4\_560+n 收发消息。在 PX4 仿真中，通常会根据具体的需求和性能要求选择使用 UDP 或 TCP 通信方式。例如，传输传感器数据时，可以使用 UDP 以确保数据的及时性，而在需要控制飞行器并确保指令的可靠传输时，可以使用 TCP。通常，UDP 更适合实时性要求高、可以容忍一些数据丢失的场景，而 TCP 更适合需要可靠性和完整性的数据传输场景。**PX4\_SITL** 模式存在的问题是能支持数量飞机不超过 40 架，否则将产生端口冲突。

第二种是 RflySim 平台提供的 **PX4\_SITL\_RFLY**。该模式下，CopterSim 使用 UDP1754\_0 发消息，使用 UDP16540 收消息。**PX4\_SITL\_RFLY** 相对 **PX4\_SITL** 模式的优点在于能够支持大规模集群，最大能支持到 1000 个载具而不至于产生端口冲突。

第三种是硬件在环仿真。在该模式下，CopterSim 将通过串口与 PX4 进行通信。



### 3.1.2. CopterSim 通过 TCP 发送给 PX4 的消息

CopterSim 通过 TCP 发送给 PX4 的消息包括 HIL\_GPS、HIL\_SENSOR、RC\_CHANNELS\_OVERRIDE。HIL\_GPS 是模拟的 GPS 数据，HIL\_SENSOR 模拟的是 IMU 数据，包括陀螺仪、加速度计、磁力计、气压计的数据。而 RC\_CHANNELS\_OVERRIDE 是对遥控器数据的模拟。

在真实飞行的过程中，遥控器的数据是射频信号，相对来说也是非常可靠的，所以模拟的时候也可以考虑采用 TCP 的方式。而传感器的数据则来自飞控板卡，所以采用 TCP 的方式模拟也是合适的。在实际中，一些外部的控制指令通过数传传输，相对来说不那么可靠，所以这些数据用 UDP 模拟也是合理的。

### 3.1.3. CopterSim 通过 UDP 发送给 PX4 的消息

通过 UDP 发送 PX4 的数据包括 MANUAL\_CONTROL、SET\_POSITION\_TARGET\_LOCAL\_NED、SET\_POSITION\_TARGET\_GLOBAL\_INT、PARAM\_SET、SET\_ATTITUDE\_TARGET、COMMAND\_LONG、HIL\_ACTUATOR\_CONTROLS、HEARTBEAT。并支持将 QGC 的数据转发给 CopterSim。

遥控器数据的操纵杆部分不仅会通过 TCP 发送，同时也会将 ch1-ch4 的数据转换成 MANUAL\_CONTROL 消息发送给 PX4。

---

## 3.2. 高精度模型+Simulink 控制器 组成的高精度综合模型（依靠 CopterSim）

在原有动力学模型的基础上实现控制器，构成综合模型。控制器使用 MATLAB Simulink 实现基本姿态控制、定点功能。控制器直接拿取模型的真实状态作为输入。综合模型协议最为关键的是约定输入输出接口。整体接口设计仅考虑完整模式，而简化模式在 CopterSim 中考虑。

### 3.2.1. 综合模型控制协议

OffBoard 控制。PX4 能够通过有线或者 wifi 被独立的辅助计算机控制，伙伴计算机通常通过 MAVLink API 进行通信。载具执行辅助计算机通过 MAVLink 设定的位置、速度、姿态指令。Offboard 模式主要用于做空中机动，至于起飞、降落、返航则选用相应专有的模式更为合适。

使用 inSIL 协议完成输入输出。

inSILInts 的第 0 个数字用于表征和修改状态，相应位为 1 时表明系统为相应状态。例如，第 1 位表示仿真模式，当接收到的 inSILInts[0] 第 1 位为 1 时表明系统进入仿真模式。

只有当 0:hasCMD 为 1 时才去设置一次状态，否则综合模型将沿用原有状态。原有状态可以来自设置外部设置值，也可以是内部状态自动转换。例如，接收到起飞命令后，首先切换到起飞模式，起飞完成后自动切换到定点模式。

inSILInts[0] Vehicle Command Bitmap

0:hasCMD	1: SIL	2: Armed	3:	4:	5:	6:	7:
有新命令	仿真	解锁					
8:Takeoff	9: Position	10: Land	11: Return	12:Lotier	13:Height	14:Home	15:
起飞	定点/waypoint	着陆	返航	盘旋（固定翼）	定高模式	水平位置控制	
16:OffboardPos	17:OffboardAtt	18:	19:	20:	21:	22:	23:
Offboard 位置控制系列	Offboard 姿态系列						
24:	25:	26:	27:	28:	29:	30:	31:

注：当使能位置控制时，水平位置和竖直位置同时使能

inSILInts[1] 的第 0-7 位为位置类标志，第 8-15 位为姿态类标志。

inSILInts[1] Offboard 控制标志

0:hasPo	1:hasVel	2:hasAcc	3:hasYaw	4:hasYawRa	5:	6:	7:
---------	----------	----------	----------	------------	----	----	----

s				te			
位置	速度	加速度	偏航角	偏航角速率			
8:hasAtt	9:hasRollRate	10:hasPitchRate	11: hasThrust	12:	13:	14:	15:
姿态	滚转角速率	俯仰角速率	油门				
16:NED	17:Global	18:	19:	20:	21:	22:	23:
位置和速度 NE D	位置和速度 Global						
24:	25:	26:	27:	28:	29:	30:	31:
整数类型纬度	整数类型精度						

inSILInts[6]表示整数类型的纬度，inSILInts[7]表示整数类型的经度。

备注：当仅作位置控制时对应的值为 1，当仅作速度控制时对应的值为 2，当仅作偏航角控制时对应的值为 8，当仅作角速率控制时对应的值为 16。如果需多种控制结合，则将单独控制时的值相加。

inSILFloats 协议。inSILFloats 用于存放实际的数据，其具体含义可根据 inSILInts 的设定发生一些变化。如前 3 个表示位置，但具体是那个坐标系下的则由 inSILInts 控制。

inSILFloats[0-2]=pos;

inSILFloats[3-5]=vel; // 速度|遥控器俯仰、滚转、偏航信号|inSILFloats[3]可以用作速率

inSILFloats[6-8]=acc;

inSILFloats[9-11]=att; // 姿态控制使用欧拉角，对于用户来说更加直观。

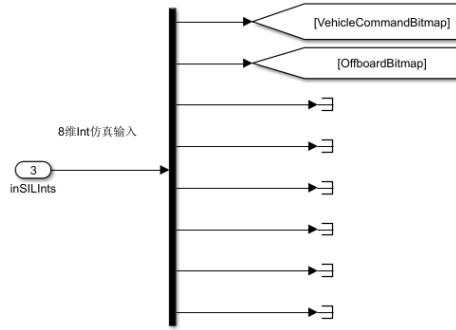
inSILFloats[12-14]=attRate;

inSILFloats[15]=thrust; // 油门

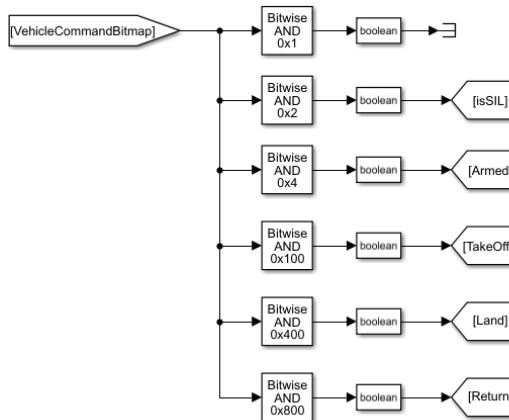
### 3.2.2. 旋翼综合模型控制接口

对于综合模型来说，其输出与普通高精度模型相似，而差别主要在于输入，所以下文讲详细介绍输入。

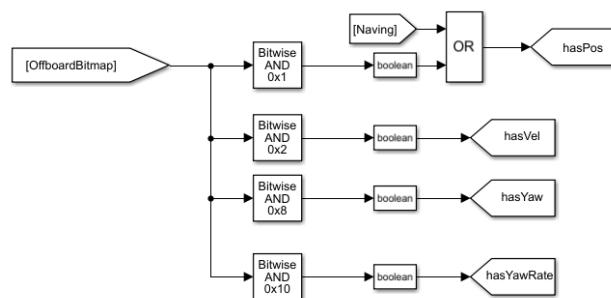
旋翼机协议解析是将 CopterSim 收到的网络包解析成 3.2.1 节的指令。inSILInts 是一个 8 维的输入，当前仅使用第 0 个数指令和第 1 个数 Offboard 模式。后续第 6 个数和第 7 个数将作为 Global 坐标系下的纬度和经度的整数表示。下图中，将 inSILInts 向量，分解为了 8 个单独的数。



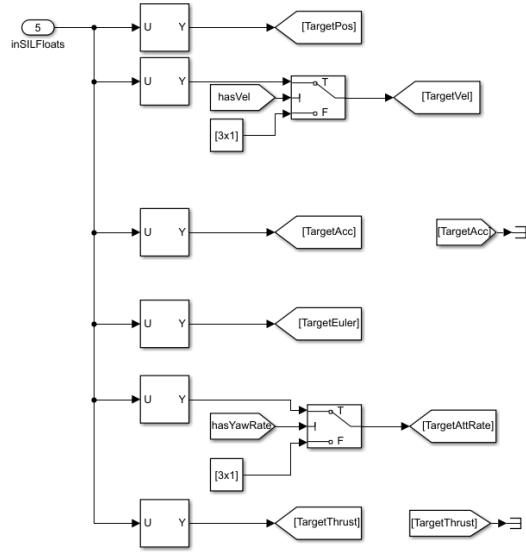
指令的每一位都有相应的含义，所以还需进一步对每一位进行解析。如下图使用 Bitwise 模块对位进行解析，第 0 位暂时没有使用。第 1 位标识是否是仿真模式，当该位为 1 时表示使用仿真模式，当该位为 0 时表示硬件在环模式，只有仿真模式时综合模型中的控制器才生效。第 2 位表示是否解锁，其它位还包括起飞、降落、返航功能，具体含义参见 3.2.2 节。



如下是标识有哪些 offboard 控制信息的标志。3.2.1 节的协议支持完整的 PX4 offboard 控制，下面支持了当前项目中最为紧需的，包括位置、速度、偏航、偏航角速率。



进一步，如下图所示解析 offboard 发送的实际位置、速度值等。在上图中是为了标识相应的值存不存在，而下图则是具体的数值。当前仅使用位置、速度、偏航、偏航角速率。在下图的解析中，使用了 hasVel 标志和 hasYawRate 标志。当这两个标志为假时，表明没有速度和偏航角速率的输入，这时将切换为遥控器模式。在遥控器模式下，1500 代表期望速度或期望偏航角速率为 0。



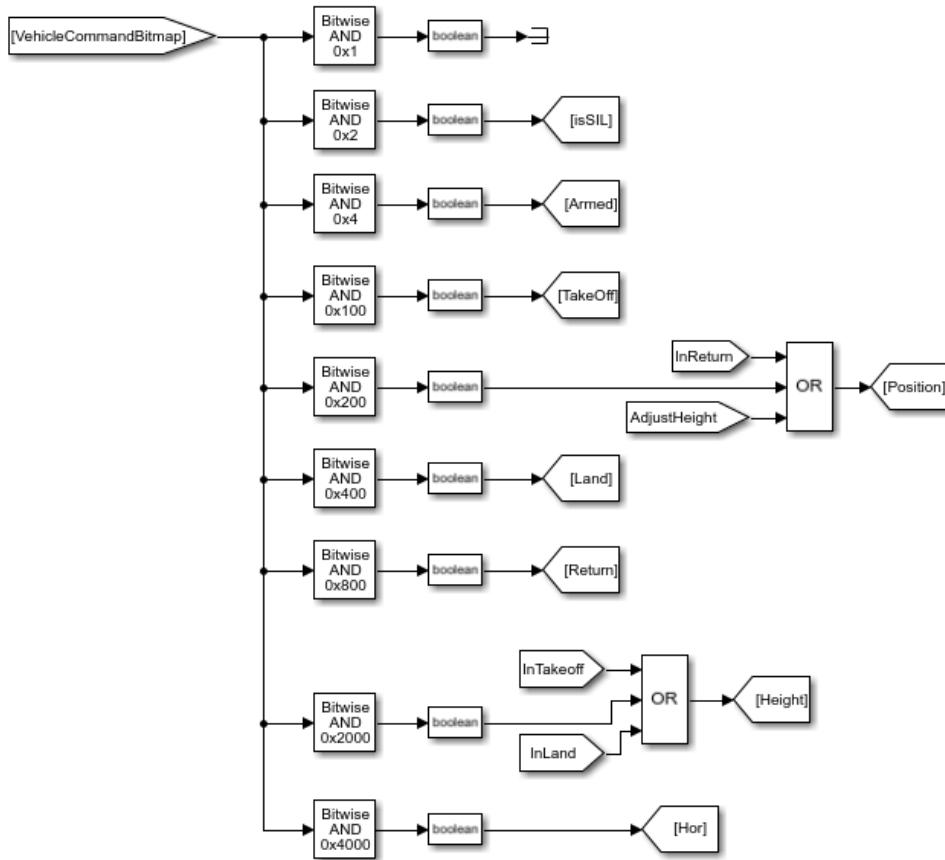
### 3.2.3. 固定翼综合模型控制接口

固定翼和旋翼相比遵循相互兼容的协议，但是固定翼的模式与旋翼有所不同，因为固定翼并不能像四旋翼那样悬停。如下图所示，是固定翼综合模型内的协议解析。增加了 Position、Height、Hor，而且起飞、返航、降落的行为也与旋翼机有所不同。

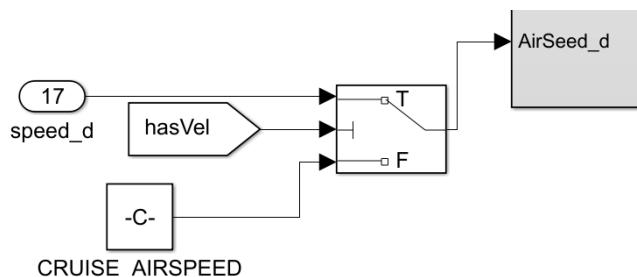
起飞，控制高度和速度。当固定翼收到起飞命令时以固定的俯仰角，默认  $15^\circ$  进行起飞。用户可以设置起飞高度，当到达起飞高度时，模型内部会进行判断，退出起飞模式。当起飞成功后，如果没有收到进一步的操作指令，将继续向前飞行，不会进行油门、俯仰、滚转的调整。

Position，同时控制水平位置和高度。用户可以通过 inSILInts 直接指定模式为位置控制，当返航或者降落时系统也可以自动触发位置模式。在位置模式下，如果设定了相应的位置，先飞到相应的位置。到达指定位置之后，如果没有指定下一个位置，将自动盘旋。在返航模式时，本质上就是水平位置回到出发点并且支持指定返航高度，所以这个功能可以使用 Position 模式进行实现。而在降落时，首先会调整飞机的高度，图中 AdjustHeight 就是描述的这一过程。调整飞机的高度是通过盘旋的方式完成的，在盘旋时水平位置也会发生变化，所以也采用位置模式进行控制。

Height 模式是控制高度和空速，而不对水平位置进行控制，即飞机只会朝前飞。在起飞时采用 Height 模式，而在降落的末尾阶段（飞机高度已经达到相应高度）也将进入 Height 模式。Hor 模式是单独控制水平位置。这个模式当前的模型支持，但一般较少使用。



与旋翼机不同，固定翼并不能自由的控制各个方向的速度。但是支持设置水平方向的速率。在完整的协议中，期望的速度有 3 个分量，在固定翼中仅使用第一个分量作为水平方向的速率。



### 3.3. 基于质点模型的简化综合模型（依靠 Python 旋翼，从集群 API 复制）

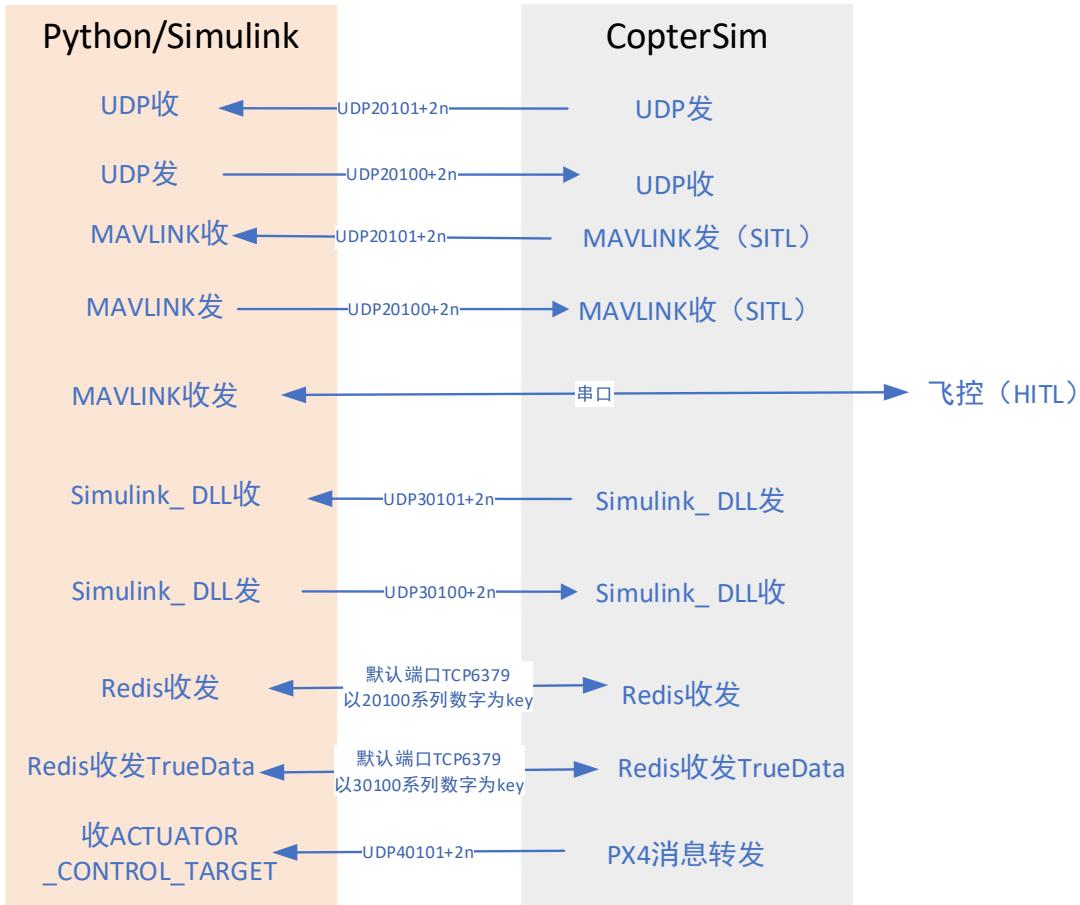
## 4. RflySim 控制协议（仅企业定制版支持 Redis）

### 4.1. 总体介绍

外部控制常通过 Python 或 Simulink 获取载具的状态，并根据载具状态发送控制指令。CopterSim 是整个平台的通信中心。CopterSim 支持不同的仿真模式，在不同仿真模式下整个仿真系统的组件构成有较大差异。在同一仿真模式下，还有不同的通信模式，包含 UDP/

MAVLink/Redis 等，不同通信模式能支持的仿真数量，状态信息具体程度，稳定性等有所不同。

如下图是 CopterSim 与 Python/Simulink 交互的整体框图。从通信模式来说，支持三种主模式 UDP/MAVLink/Redis，其中 Redis 仅在企业定制版支持。此外，还支持 FULL/Simpl e 两种子模式，这两种模式数据包的大小有差异。RflySim 支持传统的 MAVLink 消息格式，MAVLink 消息可以通过 UDP 转发也可以直接通过串口透传给飞控。而 Simulink\_DLL 模式是综合模型的模式，在该模式下，控制器集成在 CopterSim 中。而 Redis 模式下，



#### 4.1.1. CopterSim 支持的仿真模式

仿真模式主要指软件在环、硬件在环等，不同的仿真模式涉及的组件往往有差异。如软件在环相对于硬件在环不需要飞控硬件，而综合模型相对于软件在环则不需要跑一个 linux 子系统。

CopterSim 完整版支持 4 种仿真模式， $HITL = 0$ 、 $SITL = 1$ 、 $SITL\_RFLY = 2$ 、 $Simulink\_DLL = 3$ 。 $HITL$  表示硬件在环，需要连接飞控硬件才能将这种模式跑起来。 $SITL$  表示软件在环， $SITL\_RFLY$  也是软件在环，但相对于  $SITL$  而言优化了端口配置，不易发生端口冲突。 $Simulink\_DLL$  是专用于综合模型仿真，即控制器也做在模型里面，这样更加适合于大规模的集群仿真。

## 4.1.2. CopterSim 支持的连接模式

CopterSim 的连接模式是指 CopterSim 与其它组件的通信方式，当前支持的模式有 8 种，常规的模式有 6 种。UDP\_Full = 0、UDP\_Simple = 1、MAVLink\_Full = 2、MAVLink\_Simple = 3、MAVLink\_NoSend = 4、MAVLink\_NoGPS = 5、Redis\_Full = 6、Redis\_Simple = 7。

UDP、MAVLINK、Redis 是三种基本的通信方式，而按数据包是精简的还是完整的又分为两种子模式 Full 和 Simple。由此组合出了 6 种常规的模式。大规模集群，推荐使用 Redis\_Simple。

## 4.2. 数据协议

平台定义了一些控制相关的基本数据结构，这些数据结构与具体的通信模式无关，即不论使用 UDP、MAVLink 还是 redis，数据的解析都遵循这些约定的格式。

### 4.2.1. outHILStateData

outHILStateData 结构体是 CopterSim 将 PX4 估计的数据进行转发，包含时间戳、飞机 ID、位置、速度等信息。在 PX4 中通过 EKF2 会将传感器原始数据转换成对载具的状态估计，CopterSim 将从 PX4 收到的数据通过下面的数据结构转发给上层应用。

```
struct outHILStateData{ // mavlink data forward from Pixhawk
    uint32_t time_boot_ms; //Timestamp of the message
    uint32_t copterID; //Copter ID start from 1
    int32_t GpsPos[3]; //Estimated GPS position, lat&long: deg*1e7, alt: m*1e3 and up is positive
    int32_t GpsVel[3]; //Estimated GPS velocity, NED, m/s*1e2->cm/s
    int32_t gpsHome[3]; //Home GPS position, lat&long: deg*1e7, alt: m*1e3 and up is positive
    int32_t relative_alt; //alt: m*1e3 and up is positive
    int32_t hdg; //Course angle, NED,deg*1000, 0~360
    int32_t satellites_visible; //GPS Raw data, sum of satellite
    int32_t fix_type; //GPS Raw data, Fixed type, 3 for fixed (good precision)
    int32_t resrvInit; //Int, reserve for the future use
    float AngEular[3]; //Estimated Euler angle, unit: rad/s
    float localPos[3]; //Estimated locoal position, NED, unit: m
    float localVel[3]; //Estimated locoal velocity, NED, unit: m/s
    float pos_horiz_accuracy; //GPS horizontal accuracy, unit: m
    float pos_vert_accuracy; //GPS vertical accuracy, unit: m
    float resrvFloat; //float,reserve for the future use
}
```

### 4.2.2. SOut2Simulator

SOut2Simulator 的定义与 Simulink 里面新加的 MavVehicleInfo 结构体完全一致，可以直接对新的 simulink 模型的 MavVehicle3DInfo 输出口数据进行打包。这些状态是来自模型的真实数据。

```
struct SOut2Simulator {
```

---

```

int copterID; //Vehicle ID
int vehicleType; //Vehicle type
double runnedTime; //Current Time stamp (s)
float Vele[3]; //NED vehicle velocity in earth frame (m/s)
float PosE[3]; //NED vehicle position in earth frame (m)
float AngEuler[3]; //Vehicle Euler angle roll pitch yaw (rad) in x y z
float AngQuatern[4]; //Vehicle attitude in Quaternion
float MotorRPMS[8]; //Motor rotation speed (RPM)
float AccB[3]; //Vehicle acceleration in body frame x y z (m/s/s)
float RateB[3]; //Vehicle angular speed in body frame x y z (rad/s)
double PosGPS[3]; //vehicle longitude, latitude and altitude (degree,degree,m)
}

```

#### 4.2.3. inHILCMDData

inHILCMDData 是对系统进行底层控制的关键数据结构。copterID 的指定可以当同一个 Simulink 程序可以控制多机，而 mode 和 flag 的指定让用户可以发送更多上层的指令，如模式切换、上锁等。ctrls 则是控制量，通常是 PWM 波的脉宽。

```

struct inHILCMDData{
    uint32_t time_boot_ms;
    uint32_t copterID;
    uint32_t modes;
    uint32_t flags;
    float ctrls[16];
};

```

#### 4.2.4. outHILStateShort

```

struct outHILStateShort {
    int checksum; //校验位 1234567890
    int32_t gpsHome[3];
    float AngEular[3];
    float localPos[3];
    float localVel[3];
}

```

---

## 4.2.5. inOffboardShortData (最简控制协议, CopterSim 处于 UDP/Redis Simple 模式时支持)

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

checksum--校验位 1234567890, 固定值;

ctrlMode--模式选择, 具体模式参见下表;

controls[4]-- 四位控制量。

ctrlMode 支持的控制模式

模式标号	描述
0	导航坐标系下速度模式[vx,vy,vz, yaw_rate]
1	机体坐标系下速度模式[vx,vy,vz, yaw_rate]
2	导航坐标系下位置模式[x,y,z, yaw]
3	机体坐标系下位置模式[x,y,z, yaw]
4	姿态油门控制指令[滚转、俯仰、偏航(弧度)、油门(0~1)], 可自动解锁, 可自动进入 OffBoard 模式
5	姿态油门增量控制指令[滚转、俯仰、偏航、油门增量]--可自动解锁, 可自动进入 OffBoard 模式
6	加速度控制模式[ax,ay,az,yaw]
7	加速度控制模式[ax,ay,az,yaw_rate]
8	加速度控制模式[ax,ay,az,yaw_rate]
9	解锁所示模式[解锁,-,-,-]
10	表示设置固定翼飞机的速度和盘旋半径, [speed, radius, -, -]
11	表示 Mavlink 起飞命令, 自动解锁, 导航坐标系位置[x, y, z, -]
12	表示 Mavlink 起飞命令, 自动解锁, GPS 坐标系位置[纬度, 经度, 高度, -]
13	速度高度航向命令, 自动解锁并进入 offBoard 模式, GPS 坐标系位置[速度, 高度, 航向, -]
14	Global 坐标系下位置模式, GPS 坐标系位置[lat_int, lon_int, alt_float, yaw_flo at]
30	用于 VTOL 模式切换, 表示飞行模态切换指令, 对应 MAV_CMD_DO_VTO L_TRANSITION 的 mavlink 命令, controls[0]表示 State 位。定义见链接 ( <a href="https://mavlink.io/en/messages/common.html#MAV_VTOL_STATE">https://mavlink.io/en/messages/common.html#MAV_VTOL_STATE</a> )。 controls[1]表示 Immediate 位 (1: Force immediate 前置切换, 0: normal transition 普通切换.)。

---

## 4.3. 通信端口

### 4.3.1. UDP14540 系列+TCP4560 系列（与 PX4 通信，软件在环仿真时 PX4 默认端口）

UDP14540 系列端口是 PX4 默认情况下用于软件在环仿真的端口，UDP 端口用于发送重要性低一些的数据，TCP 端口用于发送重要性更高一些的数据。更加详细的描述参见 3.1.1 节。

### 4.3.2. UDP16540 系列（与 PX4 通信，软件在环仿真时 RflySim 私有端口）

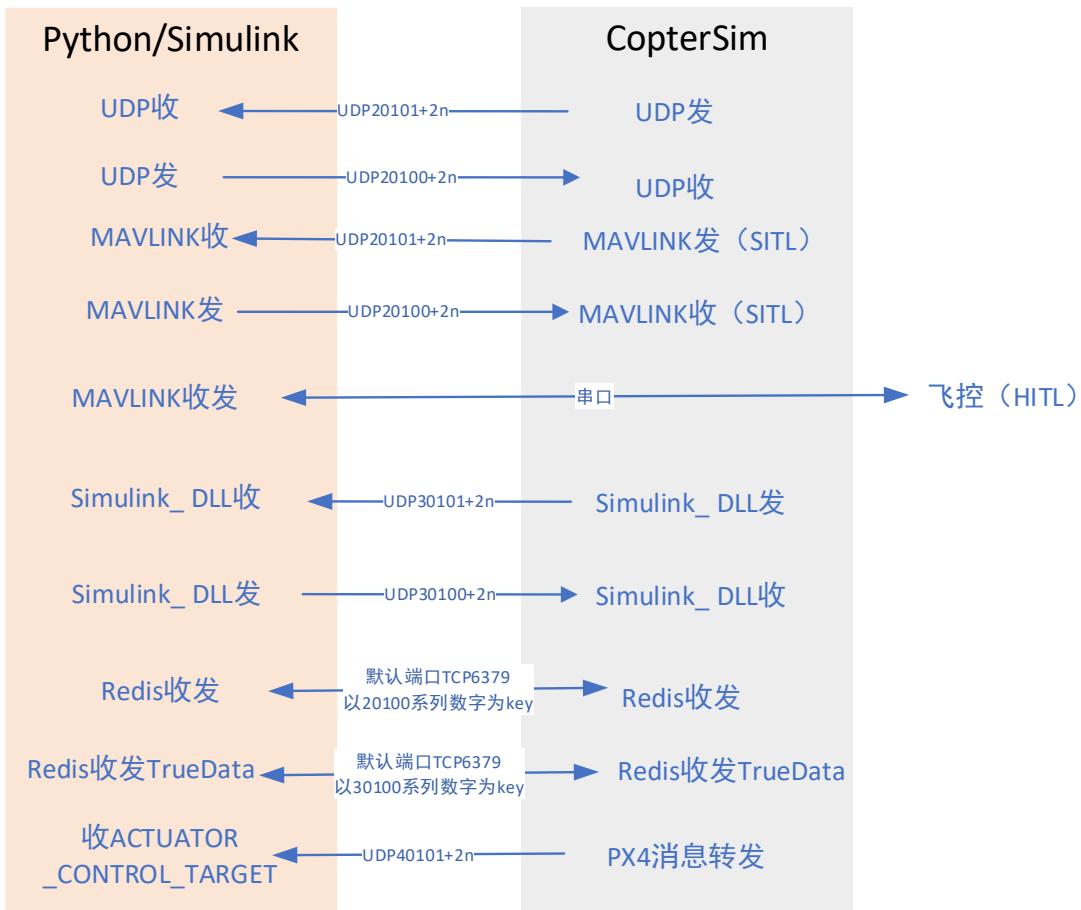
UDP16540 系列端口是 RflySim 平台自定义的系列端口，相对于 PX4 默认端口，16540 系列端口能支持更多数量的载具/更加详细的描述参见 3.1.1 节。

### 4.3.3. 串口（与 PX4 通信，硬件在环仿真端口）

通过串口传输 MAVLink 消息包，通常支持设置波特率等参数，当波特率较小时（小于 115200），消息的发送频率会降低，甚至部分消息直接不会发送。

### 4.3.4. UDP20100 系列（Python/Simulink 获取状态信息并下发控制指令）

如下图，是 Python/Simulink 程序与 CopterSim 进行交互的端口信息，在企业定制版中可支持 Redis。20100 系列端口是指 python/Simulink 利用 UDP20101+2n 发送给 CopterSim，其中 n 是飞机的 ID-1（假设 ID 从 1 开始编号），利用 UDP20100+2n 从 CopterSim 接收消息。Python/Simulink 发送给 CopterSim 的消息包括指令、Offboard 位置、速度、加速度控制消息等。而 Python/Simulink 从 CopterSim 接收消息主要是当前飞机的位置、速度等信息，利用这些信息可以做闭环控制。



Python从30101+2n读数据时，实际上读的是发给UE的数据。

### 4.3.5. UDP30100 系列（获取 True 状态信息或者通过 inSIL 下发控制指令）

UDP30100 系列端口用于获取 True 状态信息或者通过 inSIL 下发控制指令。True 信息是指载具真实的位置、速度等信息，这在仿真过程中是可以直接获取的。而通过 inSIL 发送控制指令，当前用于控制综合模型。（当前故障注入也使用此端口，所以控制综合模型和故障注入不能同时使用）。

### 4.3.6. UDP40100 系列（获取用户自定义消息）

UDP40100 端口用于获取用户自定义消息。当前主要用于获取 ACTUATOR\_CONTROL\_TARGET 这个消息，这个消息包含 PWM 波的脉宽。

### 4.3.7. TCP6379 (Redis 端口)

这是 Redis 通信的端口，当支持多机时，Redis 用 Key 来区分不同的飞机。

---

## 4.4. RflySim UDP 协议

平台可以通过 UDP 发送控制指令，UDP 协议是对 MAVLink 协议的简化，用户可以通过该协议使用常规的控制接口。

UDP 模式分为 UDP\_Full、UDP\_Simple。UDP\_Full 和 UDP\_Simple 最大的区别在于数据的简化程度，UDP\_Simple 的数据包相对 UDP\_Full 更小。对于 CopterSim 而言，使用 20100+2n 端口收消息，使用 20101+2n 端口发消息，其中 n=0, 1, 2…。

### 4.4.1. CopterSim UDP\_Simple

UDP\_Simple 模式适用于集群开发，为完成基本的位置、速度控制提供了最简实现。在 UDP\_Simple 模式下，接收的控制指令通过 inOffboardShortData 描述。包含校验位 checksum、坐标模式选择 ctrlMode 和 4 个 float 控制量 controls[4]。ctrlMode 可以有多种协议（要改 RflyUdpFast.cpp 和 CopterSim），ctrlMode 这里如果设置成<0（小于 0），表示是空命令，模块不会向外发布消息。ctrlMode 支持的具体协议如下表所示。

在 UDP\_Simple 模式下，一旦收到了 inOffboardShortData 消息，那么 Copter 会自动给 PX4 发送解锁和进入 Offboard 模式的消息。这样就简化了用户控制飞机的流程。

```
struct inOffboardShortData{  
    int checksum; // 校验位 1234567890  
    int ctrlMode; // 模式选择  
    float controls[4]; // 四位控制量  
}
```

上述的控制模式，用于给 PX4 发送控制指令。通常控制算法在计算出控制指令时，需要依据载具当前的位置、速度信息。UDP\_Simple 模式提供了获取这些信息的数据结构。

如下所示是 UDP\_Simple 模式接收 CopterSim 的简化数据。具体而言 checksum 需要确保为 1234567890，来验证数据正确性。gpsHome 飞机起飞点的经纬高数据。AngEular 飞机的欧拉角，俯仰滚转偏航，单位为度。localPos 飞机相对起飞点的相对坐标，北东地坐标系，单位米。localVel 飞机的速度，北东地，单位 m/s。gpsHome 是 int 型，先得到度（乘 1e-7）度（乘 1e-7）米（乘 1e-3）的格式，再用 Simulink 的模块，结合统一的 GPS 原点，可以得到本飞机相对统一 GPS 原点的偏移位置，再结合 localPos，可以得到飞机相对统一原点的实时位置。

---

```

struct outHILStateShort{
    int checksum; //校验位 1234567890
    int32_t gpsHome[3];
    float AngEular[3];
    float localPos[3];
    float localVel[3];
}

```

由上的描述可以看出，**UDP\_Simple** 模式下只提供了最简单的指令和获取最简单的信息。

为了让用户完整的支持 2.2.2 节的 Offboard 控制协议，设计了 **UDP\_Full** 模式。**UDP\_Full** 模式下，Offboard 控制消息是完整的 MAVLINK 协议，UDP 只是起到了转发的作用。不过，**UDP\_Full** 模式下，用户收到的消息，相对于 MAVLINK 而言仍然大大简化。

#### 4.4.2. CopterSim **UDP\_Full**

**UDP\_Full** 模式下，收到 Offboard 消息同样也会自动解锁并进入 Offboard 模式。所以 **UDP\_Full** 模式下，用户也不需要手动发送指令来解锁和进入 Offboard 模式。**UDP\_Full** 模式下收到的数据如 4.1.1 所示，用 **outHILStateData** 表示。包含 GPS 坐标系下的位置、速度，NED 坐标系下位置、速度等。在实际使用的时候，并不需要将 **outHILStateData** 里面的所有数据度解析出来，只需将感兴趣的数据提取即可。**outHILStateData** 是传感器数据经过 PX4 EKF2 处理后的结果。

对于 CopterSim 而言要传输 UDP 的数据，还需对数据包进行进一步封装。如下是 **netDataShortShort** 的数据结构，最大支持 112 个数据。由此可见 **netDataShortShort**，是完整支持了 **outHILStateData** 数据的传输。

```

typedef struct _netDataShortShort {
    TargetType tg;
    int len;
    char payload[112];
} netDataShortShort;

```

用户还可以获取真实的数据 **SOut2Simulator**，该类型数据既可以在 **UDP\_Simple** 模式下读取也可以在 **UDP\_Full** 模式下读取，因为这类数据使用的是额外的端口。CopterSim 相应的收端口为 30100，发端口为 30101 端口。**SOut2Simulator** 是来自模型的真实状态信息，没有添加噪声，这也是只有在仿真过程中才能获取的数据。

类似的，**SOut2Simulator** 数据通过 **netDataShort** 结构进行传输，也就是说 **SOut2Simulator** 所有数据的总长度是 192 个字节。

```

typedef struct _netDataShort {
    int tg;
    int len;
    char payload[192];
} netDataShort;

```

#### 4.4.3. CopterSim **Redis\_Simple/Full**

#### 4.4.4. Simulink 控制模式 Full/Simple/UltraSimple

#### 4.4.5. Python 控制模式（完整支持 CopterSim 的模式）

## 5. MAVLink 协议

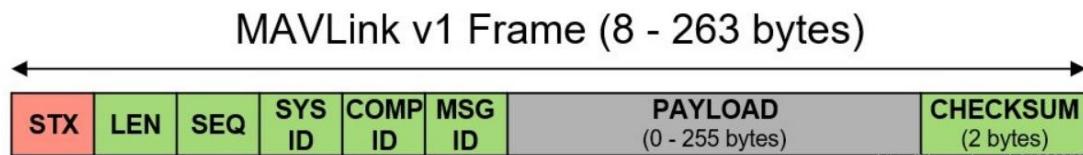
### 5.1. MAVLink 简介

MAVLink (Micro Air Vehicle Link) 是一种用于小型无人载具的通信协议，于 2009 年首次发布。该协议广泛应用于地面站 (Ground Control Station, GCS) 与无人载具 (Unmanned vehicles) 之间的通信，同时也应用在载具上机载计算机与 Pixhawk 之间的内部通信中，协议以消息库的形式定义了参数传输的规则。MAVLink 协议支持无人固定翼飞行器、无人旋翼飞行器、无人车辆等多种载具。官方使用文档网站：<https://mavlink.io/en/messages/common.html>。

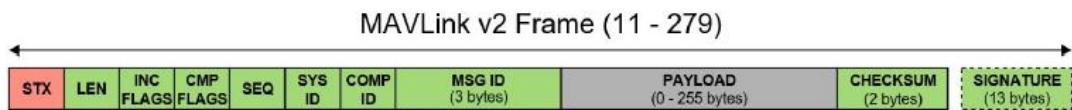
#### 5.1.1. MAVLink 包格式

MAVLink 分为 MAVLink 1 和 MAVLink 2。MAVLink 2 相对于 MAVLink 1 提供了更多的功能和安全性，尤其是在需要更复杂通信和更高安全性的应用中。但需要注意的是，MAVLink 2 可能需要更多的计算资源和带宽来支持可变长度的数据包和消息签名，因此在资源受限的系统中可能需要权衡考虑。

MAVLink 1 的数据包格式如下：



MAVLink 2 的数据包如下：



---

### 5.1.2. MAVLink 数据解析

所有字节流存入 buffer，依次读取 buffer 中的字节数据，遇到 STX 标志位（MAVLink v1 的标志位是 0xFE，v2 的标志位为 0xFD）开始识别一条消息直到消息尾部，如果消息校验正确则将消息发送给处理程序。给定一定长度的字节流 buffer，长度为 length，通过下列脚本解析，每解析出一个 mavlink 数据包就执行 onMavLinkMessage 函数

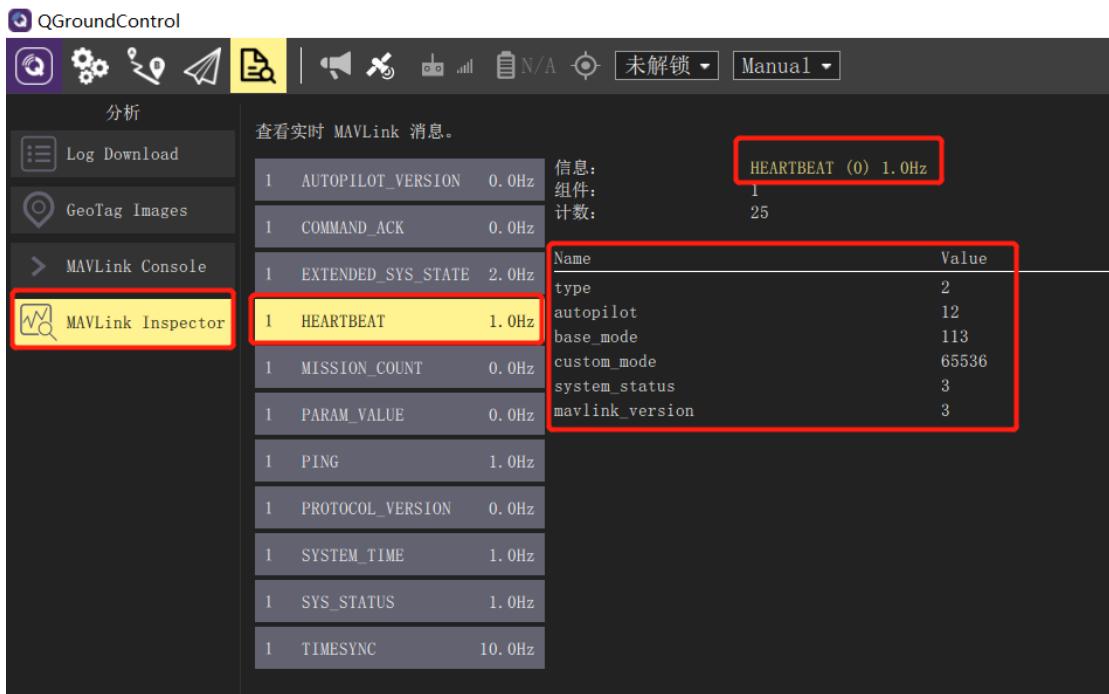
```
for(int i = 0 ; i < length ; ++i){  
    msgReceived = mavlink_parse_char(MAVLINK_COMM_1, (uint8_t)buffer[i], &message, &status);  
    if(msgReceived){  
        emit onMavLinkMessage(message);  
    }  
}
```

其中：void onMavLinkMessage(mavlink\_message\_t message); 是得到一个 MAVLink 消息包后的处理函数，需要根据这个消息的 ID 来识别当前包的用途（心跳包，GPS 位置，姿态等），并提取出感兴趣的数据。

解析函数实现如下，根据 message.msgid 跳到对应的\_decode 函数，解码出数据

```
void onMavLinkMessage(mavlink_message_t message){  
    switch (message.msgid){  
        case MAVLINK_MSG_ID_GLOBAL_POSITION_INT:{  
            mavlink_global_position_int_t gp;  
            mavlink_msg_global_position_int_decode(&message, &gp);  
            outHilData.time_boot_ms = m_LastReceiveMavMsg;  
            outHilData.GpsPos[0]=gp.lat;  
            outHilData.GpsPos[1]=gp.lon;  
            outHilData.GpsPos[2]=gp.alt;  
            outHilData.relative_alt = gp.relative_alt;  
            outHilData.GpsVel[0]=gp.vx;  
            outHilData.GpsVel[1]=gp.vy;  
            outHilData.GpsVel[2]=gp.vz;  
            outHilData.hdg = gp.hdg;  
            break;  
        }  
    }  
}
```

在 QGC 的 MAVLink Inspector 页面中可以浏览 Pixhawk 发送的所有 MAVLink 包，查看各个包的频率以及具体数值。



[wxme] start ->

可以通过修改 MavlinkConsolePage.qml 文件，改变界面布局显示。通过 QGCMAVLink Message 处理相关信息

->end[wxme]

## 5.2. 常用的 MAVLink 消息

SET\_ATTITUDE\_TARGET、SET\_POSITION\_TARGET\_LOCAL\_NED、SET\_POSITION\_TARGET\_GLOBAL\_INT 是三个最常用的外部控制消息，在 2.2.1 节已经进行了详细的介绍，本节不再赘述。

### 5.2.1. HEARTBEAT（心跳包）

HEARTBEAT 是心跳包。心跳消息显示系统或组件存在并响应。类型和自动驾驶字段（以及消息组件 ID）允许接收系统适当地处理来自此系统的进一步消息（例如，根据自动驾驶配置用户界面）。该微服务的文档位于 <https://mavlink.io/en/services/heartbeat.html>。

名称	数据类型	描述
type	uint8_t	车辆或组件类型。对于飞行控制器组件，是指车辆类型（四旋翼、直升机等）。对于其他组件，是指组件类型（例如相机、云台等）。应优先使用此信息来识别组件类型，而不是使用组件 ID。
autopilot	uint8_t	飞控类型，对于 PX4 就是 MAV_AUTOPILOT_PX4，是固定的值。
base_mode	uint8_t	MAVLink 基础系统模式位图，包含系统解锁、手动输入、HIL、自稳、自主等模式。
custom_mode	uint32_t	飞控定义的模式，如 PX4 定义了准备、起飞、盘旋、降

---

		落等。
system_status	uint8_t	系统状态，系统启动、校正中、非正常飞行模式(failsafe)等
mavlink_version	uint8_t	MAVLink 版本

信息:	HEARTBEAT (0) 1.0Hz	
组件:	1	
计数:	20	
<hr/>		
名称	值	类型
type	2	uint8_t
autopilot	12	uint8_t
base_mode	29	uint8_t
custom_mode	50593792	uint32_t
system_status	3	uint8_t
mavlink_version	3	uint8_t

QGC 中显示的 HEARTBEAT 消息

### 5.2.2. ATTITUDE (姿态-欧拉角)

ATTITUDE 是姿态欧拉角。载体坐标系下的姿态（右手坐标系，Z 轴向下，Y 轴向右，X 轴向前，ZYX 顺序，本体坐标系）。

名称	数据类型	单位	描述
time_boot_ms	uint32_t	ms	时间戳，从系统 boot 后开始计算
roll	float	rad	滚转角 $[-\pi, \pi]$
pitch	float	rad	俯仰角 $[-\pi, \pi]$
yaw	float	rad	偏航角 $[-\pi, \pi]$
rollspeed	float	rad/s	滚转角速率
pitchspeed	float	rad/s	俯仰角速率
yawspeed	float	rad/s	偏航角速率

信息:	ATTITUDE (30) 50.0Hz	
组件:	1	
计数:	188952	
<hr/>		
名称	值	类型
time_boot_ms	3826195	uint32_t
roll	0.000219161	float
pitch	0.000231194	float
yaw	-0.00396733	float
rollspeed	-0.00499866	float
pitchspeed	-0.00123594	float
yawspeed	-0.00152956	float

QGC 中显示的 ATTITUDE 消息

### 5.2.3. ATTITUDE\_QUATERNION (姿态-四元数)

ATTITUDE\_QUATERNION 是姿态四元数。在载体坐标系下（右手坐标系，Z 轴向下，X 轴向前，Y 轴向右）表示的姿态，以四元数形式表示。四元数的顺序为 w、x、y、z，零旋转将表示为(1 0 0 0)。

名称	数据类型	单位	描述
time_boot_ms	uint32_t	ms	时间戳，从系统 boot 后开始计算
q1	float		w, 实部，旋转角度一半的余弦值的平方根
q2	float		x, (x,y,z) 表示旋转轴的方向
q3	float		y
q4	float		z
rollspeed	float	rad/s	滚转角速率
pitchspeed	float	rad/s	俯仰角速率
yawspeed	float	rad/s	偏航角速率

信息:	ATTITUDE_QUATERNION (31) 50.8Hz	
组件:	1	
计数:	235066	
<hr/>		
名称	值	类型
time_boot_ms	4748520	uint32_t
q1	0.999993	float
q2	0.000174674	float
q3	0.00039012	float
q4	-0.00365487	float
rollspeed	-0.00240021	float
pitchspeed	0.00325524	float
yawspeed	0.00538497	float
repr_offset_q	0, 0, 0, 0	float

QGC 中显示的 QUATERNION

## 5.2.4. LOCAL\_POSITION\_NED (NED 位置)

LOCAL\_POSITION\_NED 是 NED 坐标系下的位置、速度。已经滤波后的本地位置（例如，融合了计算机视觉和加速度计数据）。坐标系是右手坐标系，Z 轴向下（航空坐标系，NED / 北-东-下约定）。

名称	数据类型	单位	描述
time_boot_ms	uint32_t	ms	时间戳，从系统 boot 后开始计算
x	float	m	x 位置
y	float	m	y 位置
z	float	m	z 位置
vx	float	m/s	x 方向速度
vy	float	m/s	y 方向速度
vz	float	m/s	z 方向速度

信息:	LOCAL_POSITION_NED (32) 50.6Hz	
组件:	1	
计数:	4168	
名称	值	类型
time_boot_ms	10826655	uint32_t
x	-0.0780307	float
y	0.0861752	float
z	-10.0277	float
vx	0.0193858	float
vy	-0.00996945	float
vz	0.0131545	float

QGC 中显示的 LOCAL\_POSITION\_NED

## 5.2.5. GLOBAL\_POSITION\_INT (Global 位置)

GLOBAL\_POSITION\_INT。滤波后的全球位置（例如，融合了 GPS 和加速度计数据）。位置是在 GPS 坐标系中表示的（右手坐标系，Z 轴向上）。它被设计为经过缩放的整数消息，因为浮点数的分辨率不足以满足需求。速度仍然在 NED 坐标系下。

名称	数据类型	单位	描述
time_boot_ms	uint32_t	ms	时间戳，从系统 boot 后开始计算
lat	int32_t	degE7	纬度
lon	int32_t	degE7	经度
alt	int32_t	mm	海拔高度
relative_alt	int32_t	mm	距离地面的高度，向上为正
vx	int16_t	cm/s	x 方向速度，向北为正
vy	int16_t	cm/s	y 方向速度，向东为正
vz	int16_t	cm/s	z 方向速度，向下为正

---

hdg	int16_t	cdeg	载具偏航角，范围从 0.0 度到 359.99 度，如果未知，可设置为：UINT16_MAX。cdeg 是厘米，1 度等于 100 厘米。
-----	---------	------	---

信息：	GLOBAL_POSITION_INT (33) 50.0Hz	
组件：	1	
计数：	6664	
<b>名称</b>		
名称	值	类型
time_boot_ms	10876590	uint32_t
lat	401540299	int32_t
lon	1162593684	int32_t
alt	68184	int32_t
relative_alt	10016	int32_t
vx	2	int16_t
vy	0	int16_t
vz	0	int16_t
hdg	35992	uint16_t

QGC 中显示的 GLOBAL\_POSITION\_INT

### 5.2.6. ACTUATOR\_OUTPUT\_STATUS (电机原始输出)

ACTUATOR\_OUTPUT\_STATUS。伺服输出的原始值（例如，在 Pixhawk 上，来自主要（MAIN）和辅助（AUX）端口）。此消息取代了 SERVO\_OUTPUT\_RAW。

名称	数据类型	单位	描述
time_uesc	uint64_t	us	时间戳，从系统 boot 后开始计算
active	uint32_t		
actuator	float[32]		伺服/电机输出数组的值。零值表示未使用的通道。

### 5.2.7. ATTITUDE\_TARGET (当前期望姿态)

ATTITUDE\_TARGET。报告飞行器当前由自动驾驶仪指定的期望姿态。如果飞行器是通过发送 SET\_ATTITUDE\_TARGET 消息来控制的，这应该与发送的命令相匹配。

名称	数据类型	单位	描述
time_boot_ms	uint32_t	ms	时间戳，从系统 boot 后开始计算
type_mask	uint8_t		期望姿态位图，表明哪些期望姿态信息应该被忽略
q	float[4]		期望四元数 (w, x, y, z 序，零旋转对应 1, 0, 0, 0)
body_roll_rate	float	rad/s	期望滚转角速率
body_pitch_rate	float	rad/s	期望俯仰角速率
body_yaw_rate	float	rad/s	期望偏航角速率
thrust	float		期望油门

---

信息:	ATTITUDE_TARGET (83) 50.0Hz	
组件:	1	
计数:	171589	
<hr/>		
名称	值	类型
time_boot_ms	14175425	uint32_t
type_mask	0	uint8_t
q	0.999947, -0.000991172, ...	float
body_roll_rate	0.00458304	float
body_pitch_rate	-0.00314479	float
body_yaw_rate	-0.000370033	float
thrust	0.609561	float

QGC 中显示的 ATTITUDE\_TARGET

## 5.2.8. POSITION\_TARGET\_LOCAL\_NED (当前期望 NED 位置)

POSITION\_TARGET\_LOCAL\_NED。报告由自动驾驶仪指定的当前期望飞行器位置、速度和加速度。如果飞行器是通过发送 SET\_POSITION\_TARGET\_LOCAL\_NED 消息来进行这种方式的控制，那么这些报告应该与发送的命令相匹配。在不设置 SET\_POSITION\_TARGET\_LOCAL\_NED 时，POSITION\_TARGET\_LOCAL\_NED 也是有值的，因为 PX4 自带的 Navigator 可以发布 POSITION\_TARGET\_LOCAL\_NED。

名称	类型	单位	描述
time_boot_ms	uint32_t	ms	系统时间
coordinate_frame	uint8_t		最常用的坐标系 MAV_FRAME_LOCAL_NED = 1
type_mask	uint16_t		位标志表明控制信息应该被忽略
x	float	m	NED 坐标系下 x 位置
y	float	m	NED 坐标系下 y 位置
z	float	m	NED 坐标系高度
vx	float	m/s	NED 坐标系 x 方向速度
vy	float	m/s	NED 坐标系 y 方向速度
vz	float	m/s	NED 坐标系 z 方向速度
afx	float	m/s <sup>2</sup>	NED 坐标系 x 方向加速度或力
afy	float	m/s <sup>2</sup>	NED 坐标系 y 方向加速度或力
afz	float	m/s <sup>2</sup>	NED 坐标系 z 方向加速度或力
yaw	float	rad	偏航角
yaw_rate	float	rad/s	偏航角速率

信息:	POSITION_TARGET_LOCAL_NED (85) 50.0Hz	
组件:	1	
计数:	3472779	
名称	值	类型
time_boot_ms	80199785	uint32_t
coordinate_frame	1	uint8_t
type_mask	0	uint16_t
x	-0.0597772	float
y	0.0779866	float
z	-10.013	float
vx	0.00549753	float
vy	-0.00251879	float
vz	-0.0141983	float
afx	-0.00935648	float
afy	-0.000793941	float
afz	-0.0157687	float
yaw	0.488225	float
yaw_rate	0	float

QGC 中显示的 POSITION\_TARGET\_LOCAL\_NED

## 5.2.9. POSITION\_TARGET\_GLOBAL\_INT (当前期望 Global Position)

POSITION\_TARGET\_GLOBAL\_INT。报告由自动驾驶仪指定的当前期望飞行器位置、速度和加速度。如果飞行器是通过发送 SET\_POSITION\_TARGET\_GLOBAL\_INT 消息来进行这种方式的控制，那么这些报告应该与发送的命令相匹配。

名称	类型	单位	描述
time_boot_ms	uint32_t	ms	系统时间
coordinate_frame	uint8_t		最常用坐标系 MAV_FRAME_GLOBAL_INT = 5
type_mask	uint16_t		位标志表明控制信息应该被忽略
lat_int	int32_t	degE7	WGS84 下 x 位置
lon_int	int32_t	degE7	WGS84 下 y 位置
alt	float	m	海拔高度或者相对高度，取决于坐标系
vx	float	m/s	NED 坐标系 x 方向速度
vy	float	m/s	NED 坐标系 y 方向速度
vz	float	m/s	NED 坐标系 z 方向速度
afx	float	m/s <sup>2</sup>	NED 坐标系 x 方向加速度或力
afy	float	m/s <sup>2</sup>	NED 坐标系 y 方向加速度或力
afz	float	m/s <sup>2</sup>	NED 坐标系 z 方向加速度或力
yaw	float	rad	偏航角
yaw_rate	float	rad/s	偏航角速率

信息:	POSITION_TARGET_GLOBAL_INT (87) 1.0Hz	
组件:	1	
计数:	70006	
<hr/>		
名称	值	类型
time_boot_ms	80748895	uint32_t
coordinate_frame	5	uint8_t
type_mask	0	uint16_t
lat_int	401540300	int32_t
lon_int	1162593686	int32_t
alt	68.1721	float
vx	-0.0104241	float
vy	-0.0224574	float
vz	0.0129328	float
afx	-0.0241467	float
afy	0.0063342	float
afz	0.00536636	float
yaw	0.495925	float
yaw_rate	0	float

QGC 中显示的 POSITION\_TARGET\_GLOBAL\_INT

### 5.2.10. HOME\_POSITION (Home 点位置)

HOME\_POSITION 包含家庭位置信息。家庭位置是系统将返回并降落的默认位置。这个位置必须在起飞期间由系统自动设置，也可以使用 MAV\_CMD\_DO\_SET\_HOME 命令明确设置。全球位置和本地位置分别在各自的坐标系中编码位置，而 q 参数编码了地表的方向。在正常情况下，它描述了航向和地形坡度，飞行器可以利用这些信息来调整着陆途中的飞行。着陆途中的 3D 矢量描述了系统在正常飞行模式下应该飞向的点，然后沿着该矢量执行着陆序列。注意：可以通过发送 MAV\_CMD\_REQUEST\_MESSAGE 并设置 param1=24 2（或废弃的 MAV\_CMD\_GET\_HOME\_POSITION 命令）来请求此消息。

名称	类型	单位	描述
latitude	int32_t	degE7	WGS84 下 x 位置
longitude	int32_t	degE7	WGS84 下 y 位置
altitude	int32_t	mm	海拔高度，向上为正
x	float	m	NED 坐标系位置 x
y	float	m	NED 坐标系位置 y
z	float	m	NED 坐标系位置 z
q	float[4]		四元数表示起飞位置的世界到地表法线和航向变换。用于指示地面的航向和坡度。如果无法提供准确的航向和地表坡度的四元数，则所有字段都应设置为 NaN。
approach_x	float	m	接近矢量结束点的本地 X 位置。多旋翼飞行器应该根据它们的起飞路径来设置这个位置。在草地上降落的固定翼飞行器应该

---

			以与多旋翼飞行器相同的方式设置它。而在跑道上降落的固定翼飞行器应该将其设置为与起飞相反的方向，假设起飞是从跑道的阈值/触地区域开始的。
approach_y	float	m	同上
approach_z	float	m	同上
time_usec	uint64_t	us	时间戳（UNIX 纪元时间或自系统启动以来的时间）。接收端可以通过检查数字的大小来推断时间戳的格式（自 1970 年 1 月 1 日或自系统启动以来）。

信息:	HOME_POSITION (242) 0.8Hz	
组件:	1	
计数:	35392	
名称	值	类型
latitude	401540300	int32_t
longitude	1162593685	int32_t
altitude	58168	int32_t
x	-0.0566117	float
y	0.0734819	float
z	-0.0134557	float
q	0.9999992, 0, 0, -0.003974...	float
approach_x	0	float
approach_y	0	float
approach_z	0	float
time_usec	79743935000	uint64_t

QGC 中显示的 HOME\_POSITION

## 5.2.11. HIL\_ACTUATOR\_CONTROLS (PX4 到 Sim 的控制输出)

HIL\_ACTUATOR\_CONTROLS。从自动驾驶仪发送到模拟器。用于硬件在环路（Hardware in the Loop, HIL）控制输出的消息（用于替代 HIL\_CONTROLS）。

名称	数据类型	单位	描述
time_uecs	uint64_t	us	时间戳（UNIX 纪元时间或自系统启动以来的时间）。接收端可以通过检查数字的大小来推断时间戳的格式（自 1970 年 1 月 1 日或自系统启动以来）。
controls	float[16]		控制输出范围在-1 到 1 之间。通道分配取决于模拟硬件的配置。
mode	uint8_t		系统的状态，包括介绍状态等。
uint8_t	uint64_t		这是一个标志位字段，其中的 1 表示使用锁步模拟。

信息:	HIL_ACTUATOR_CONTROLS (93) 9.2Hz	
组件:	1	
计数:	685595	
名称	值	类型
time_usec	1777495338503	uint64_t
controls	0.609, 0.607, 0.607, 0.608...	float
mode	129	uint8_t
flags	1	uint64_t

QGC 中显示的 HIL\_ACTUATOR\_CONTROLS

### 5.2.12. HIL\_SENSOR (Sim 到 PX4 的传感器信息)

HIL\_SENSOR。IMU (惯性测量单元) 读数以国际单位制 (SI 单位) 在 NED (北-东-下) 机体坐标系中表示。这意味着加速度和角速度等测量值以米/秒<sup>2</sup>和弧度/秒为单位, 相对于飞行器的前、右和下方向。

名称	类型	单位	描述
time_usec	uint64_t	us	时间戳 (UNIX 纪元时间或自系统启动以来的时间)。接收端可以通过检查数字的大小来推断时间戳的格式 (自 1970 年 1 月 1 日或自系统启动以来)。
xacc	float	m/s <sup>2</sup>	x 方向加速度
yacc	float	m/s <sup>2</sup>	y 方向加速度
zacc	float	m/s <sup>2</sup>	z 方向加速度
xgyro	float	rad/s	x 方向角速度
ygyro	float	rad/s	y 方向角速度
zgyro	float	rad/s	z 方向角速度
xmag	float	gauss	x 方向磁场强度
ymag	float	gauss	y 方向磁场强度
zmag	float	gauss	z 方向磁场强度
abs_pressure	float	hPa	绝对气压
diff_pressure	float	hPa	差分气压
pressure_alt	float		气压高度
temperature	float		气温
fields_updated	uint32_t		这是一个位图, 用于表示自上次消息以来已经更新的字段。位图中的每一位通常代表一个字段, 当相应的字段已经更新时, 对应的位被设置为 1, 否则为 0。这种位图的使用可帮助在通信中有效地传输和识别哪些字段已经发生了变化, 而无需传输

---

			整个消息的所有数据。
id	uint8_t		传感器 ID (从零开始索引)。用于处理多个传感器输入时，可以使用此 ID 来区分不同的传感器。这有助于系统识别和区分不同来源的数据，特别是在多传感器环境中。每个传感器通常都会分配一个唯一的 ID，以便进行标识和数据处理。

注：仿真模型传给 PX4 的数据不能在 QGC 中显示。

### 5.2.13. HIL\_GPS (Sim 到 PX4 的 GPS 信息)

HIL\_GPS。全球位置，由全球定位系统 (GPS) 返回。这不是系统的全局位置估计，而是一个原始传感器值。要获取系统的全局位置估计，请查看 GLOBAL\_POSITION\_INT 消息。这个消息包含了直接从 GPS 接收到的原始位置信息，而不是经过处理和估算的全局位置。

名称	类型	单位	描述
time_usec	uint64_t	us	时间戳 (UNIX 纪元时间或自系统启动以来的时间)。接收端可以通过检查数字的大小来推断时间戳的格式 (自 1970 年 1 月 1 日或自系统启动以来)。
fix_type	uint8_t		0-1: no fix, 2: 2D fix, 3: 3D fix。某些应用程序将不使用此字段的值，除非至少为 2，因此请始终正确填写修正值。这个字段通常用于指示 GPS 接收器的定位质量，0 表示无法定位，2 表示二维定位，3 表示三维定位。在某些应用中，只有当 GPS 具有至少二维修正时，才会使用位置信息，因此在报告 GPS 定位信息时，确保正确填写修正字段对于数据的准确性非常重要。
lat	int32_t	degE7	WGS84 纬度
lon	int32_t	degE7	WGS84 经度
alt	int32_t	mm	海拔高度，向上为正。
eph	uint16_t		GPS 水平位置精度因子 (HDOP，无单位 * 100)。如果未知，可以设置为 UINT16_MAX。HDOP 是一个表示 GPS 定位精度的因子，它通常是一个无单位的小数，乘以 100 以获得一个整数值。较低的 HDOP 值表示较高的定位精度，而较高的 HDOP 值表示较低的定位精度。如果不知道 GPS 的水平位置精度因子，可以将该字段设置

---

			为 <code>UINT16_MAX</code> , 以指示未知值。
<code>epv</code>	<code>uint16_t</code>		GPS 垂直位置精度因子 (VDOP, 无单位 * 100)。如果未知, 可以设置为 <code>UINT16_MAX</code> 。VDOP 是一个表示 GPS 定位精度的因子, 它通常是一个无单位的小数, 乘以 100 以获得一个整数值。较低的 VDOP 值表示较高的垂直定位精度, 而较高的 VDOP 值表示较低的垂直定位精度。如果不知道 GPS 的垂直位置精度因子, 可以将该字段设置为 <code>UINT16_MAX</code> , 以指示未知值。
<code>vel</code>	<code>uint16_t</code>	<code>cm/s</code>	GPS 地面速度。如果未知, 可以将其设置为 <code>UINT16_MAX</code> , 表示未知值。GPS 地面速度表示 GPS 接收器所测得的设备在地面上移动的速度。这个值通常以米每秒 ( <code>m/s</code> ) 为单位, 表示设备的速度。如果无法获取地面速度的准确值, 可以将字段设置为 <code>UINT16_MAX</code> , 以表示未知值。
<code>vn</code>	<code>int16_t</code>	<code>cm/s</code>	GPS 速度在地球固定的 NED 坐标系中的北向分量。
<code>ve</code>	<code>int16_t</code>	<code>cm/s</code>	GPS 速度在地球固定的 NED 坐标系中的东向分量。
<code>vd</code>	<code>int16_t</code>	<code>cm/s</code>	GPS 速度在地球固定的 NED 坐标系中的向下分量。
<code>cog</code>	<code>uint16_t</code>	<code>cdeg</code>	地面航向 (不是航向, 而是移动方向), 范围从 0.0 度到 359.99 度。如果未知, 可以将其设置为 <code>UINT16_MAX</code> , 表示未知值。地面航向是指设备相对于地面的运动方向, 而不是设备的朝向。通常以度为单位表示, 表示设备相对于正北方向的方向。如果无法获取地面航向的准确值, 可以将字段设置为 <code>UINT16_MAX</code> , 以表示未知值。
<code>satellites_visible</code>	<code>uint8_t</code>		可见卫星数量
<code>id</code>	<code>uint8_t</code>		GPS ID 号
<code>yaw</code>	<code>uint16_t</code>	<code>cdeg</code>	飞行器相对于地球的北方的偏航角, 其中零表示不可用, 使用 36000 表示正北方。这个角度表示飞行器的朝向相对于地球的真北方向, 通常以百分之一度 ( <code>centi degrees</code> ) 表示。如果无法获得偏航角的准

---

			确值，可以将其设置为零，表示不可用，或者设置为 36000，表示正北方向。这个值用于表示飞行器相对于地球北极的方向。
--	--	--	--

注：仿真模型传给 PX4 的数据不能在 QGC 中显示。

## 5.3. 微服务

MAVLink 系统采用的高级别协议称为微服务，使用微服务以便更好地进行互操作。例如，QGroundControl、ArduPilot 和 PX4 自动驾驶仪都共享一个通用命令协议，用于发送需要确认的点对点消息。

微服务用于交换多种类型的数据，包括：参数、任务、轨迹、图像、其他文件。如果数据远大于单个消息的容量，服务将定义如何拆分和重新组装数据，以及如何确保重新传输任何丢失的数据。其他服务提供命令确认和/或错误报告。

大多数服务采用客户端-服务器模式，即 GCS（客户端）发起请求，车辆（服务器）响应数据。MAVLink 定义了以下的微服：心跳/连接协议、任务协议、参数协议、扩展参数协议、命令协议、手动控制（操纵杆）协议、相机协议、相机定义、万向协议 v2、Arm 授权协议、图像传输协议、文件传输协议 (FTP)、登陆目标协议、Ping 协议、路径规划协议（轨迹接口）、电池协议、地形协议、隧道协议、开放无人机 ID 协议、高延迟协议、组件元数据协议 (WIP)、有效负载协议、交通管理 (UTM/ADS-B)、事件接口 (WIP)、时间同步等协议，具体可见 <https://mavlink.io/en/services/>。

本节主要介绍几个在 QGC 二次开发过程中，会涉及到的协议

### 5.3.1 心跳/连接协议

心跳协议用于通告 MAVLink 网络上系统（QGC 或者其他操作软件）的存在，及其系统和组件 ID、车辆类型、飞行堆栈、组件类型和飞行模式。

心跳协议可以实现以下功能：1、发现连接到网络的系统并推断它们何时断开连接。如果定期收到组件的 HEARTBEAT 消息，则认为该组件已连接到网络；如果未收到预期消息，则认为该组件已断开连接。2、根据组件类型和其他属性（例如，根据车辆类型布局 GCS 界面），适当处理来自组件的其他消息。3、将消息路由到不同接口上的系统。

必须定义心跳协议的广播速率 HEARTBEAT，以及定义系统在“丢失”多少消息被视为超时/与网络断开连接。例如在 RF 遥测链路上，组件通常以 1 Hz 的频率发布其心跳，如果未收到四条或五条消息，则认为另一个系统已断开连接。

如果组件没有检测到另一个系统，则它可以选择不在通道（除通道之外 HEARTBEAT）上发送或广播信息，并且在接收心跳时它将继续向系统发送消息。因此，系统必须：(1) 即使没有命令远程系统，也会广播心跳；(2) 当它们处于故障状态时不要广播心跳（即不要从不知道组件其余部分状态的单独线程发布心跳）。

连接 QGroundControl 的具体代码可以在 MultiVehicleManager.cc 中找到（参见 参考资料

---

```
void MultiVehicleManager::_vehicleHeartbeatInfo)。
```

相关协议详细介绍可见 <https://mavlink.io/en/services/heartbeat.html>

### 5.3.2 任务协议

任务子协议实现 GCS 或开发人员 API 与无人机/组件交换任务（飞行计划）、地理围栏和安全点信息。

该协议涵盖：(1) 上传、下载、清除任务、设置/获取当前任务项编号、当前任务项发生变化时收到通知等操作。(2) 用于交换任务物品的消息类型和枚举。(3) 大多数系统通用的任务项（“MAVLink 命令”）。该协议支持重新请求尚未到达的消息，从而允许通过有损链路可靠地传输任务。

该协议主要包括三种类型的“任务”：飞行计划、地理围栏和集结/安全点。

飞行计划任务类型命令格式：1、用于导航/移动的 NAV 命令（MAV\_CMD\_NAV\_\*）（例如 MAV\_CMD\_NAV\_WAYPOINT、MAV\_CMD\_NAV\_LAND）2、DO 命令（MAV\_CMD\_DO\_\*）用于立即执行操作，例如更改速度或激活伺服系统（例如 MAV\_CMD\_DO\_CHANGE\_SPEED）。3、CONDITION 命令（MAV\_CMD\_CONDITION\_\*）用于根据条件更改任务的执行 - 例如，在执行下一个命令（MAV\_CMD\_CONDITION\_DELAY）之前暂停任务一段时间。

地理围栏任务类型命令格式：前缀为 MAV\_CMD\_NAV\_FENCE\_（例如 MAV\_CMD\_NAV\_FENCE\_RETURN\_POINT）。

集结点任务类型命令格式：只有一个集结点 MAV\_CMD：MAV\_CMD\_NAV\_RALLY\_POINT。

任务项目（）在 MISSION\_ITEM\_INTMAV\_CMD 消息中传输/编码。该消息包括用于识别特定任务项（命令 ID）的字段以及最多 7 个特定于命令的可选参数。

可以在 QGroundControl 的 src/MissionManager/PlanManager.cc 中看到具体实现

相关协议详细介绍可见 <https://mavlink.io/en/services/command.html>

### 5.3.3 参数协议

参数协议用于在 MAVLink 组件之间交换配置设置。每个参数都表示为键/值对。键通常是人类可读的参数名称（最多 16 个字符）和一个值 - 可以是多种类型之一。

键/值对定义有以下原则：

- 1、可读的名称很小但很有用（它可以对参数名称进行编码，用户可以从中推断出参数的用途）。
- 2、可以“开箱即用”支持实现该协议的未知自动驾驶仪。
- 3、GCS 不必提前知道远程系统上存在哪些参数（尽管实际上 GCS 可以通过附加参数元数据（例如最大值和最小值、默认值等）提供更好的用户体验）。

---

4、添加参数只需要对带参数的系统进行更改即可。加载参数的 GCS 和 MAVLink 通信库不需要任何更改。

可以在 QGroundControl 的 src/FactSystem/ParameterManager.cc 中看到具体实现

相关协议详细介绍可见 <https://mavlink.io/en/services/parameter.html>

### 5.3.4 命令协议

MAVLink 命令协议可保证 MAVLink 命令的传送。命令一般 MAV\_CMD 的值，定义最多 7 个参数的值。这些参数和命令 id 被编码为 COMMAND\_INT 或 COMMAND\_LONG 以便发送。

该协议通过期望来自命令的匹配确认 (COMMAND\_ACK) 来指示命令到达和结果来提供可靠的传送。如果未收到确认，则必须自动重新发送命令。

COMMAND\_INT 用于特定命令的飞行堆栈支持，在发送包含位置或导航信息的命令时使用。这是因为它允许为位置和高度值指定坐标系，否则可能会“未指定”。此外，纬度/经度可以以更高的精度在 COMMAND\_INT 参数 5 和 6 中以缩放整数形式发送（比在中以浮点值发送时更精确 COMMAND\_LONG）。

COMMAND\_LONG 用于发送 MAV\_CMD 在参数 5 和 6 中发送浮点属性的命令，因为如果在 COMMAND\_INT 中发送，这些值将被截断为整数。

如果飞行堆栈支持，则可以在任一消息中使用非位置命令或在参数 5 和 6 中指定整数的命令。

COMMAND\_INT 飞行堆栈可以支持任一消息或/或两者中的命令 COMMAND\_LONG，尽管会损失精度、舍入误差和/或未定义的参考系。但是，鼓励他们仅支持中的位置命令 COMMAND\_INT 以及在 中的参数 5 和 6 中具有浮点值的命令 COMMAND\_LONG。飞行堆栈可以根据需要使用 COMMAND\_ACK.result 或拒绝以“错误”消息类型发送 MAV\_RESULT\_COMMAND\_LONG\_ONLY 的命令。MAV\_RESULT\_COMMAND\_INT\_ONLY 仅支持特定消息类型中的特定命令的飞行堆栈可以更普遍地使用这些结果值来指示命令的正确消息类型

相关协议详细介绍可见 <https://mavlink.io/en/services/command.html>

### 5.3.5 手动控制协议（操纵杆）

手动控制协议允许使用“标准操纵杆”（或支持相同轴命名法的类似操纵杆的输入设备）来控制系统。

该协议仅通过消息来实现 MANUAL\_CONTROL。它定义了 target 要控制的系统、四个主轴 (x、y、z、r) 和两个扩展轴 (s、t) 的运动，以及两个 16 位字段来表示最多 32 个按钮 (buttons、buttons2) 的状态。可以禁用未使用的轴，并且必须使用该字段的位 0 和 1 显式启用扩展轴 enabled\_extensions。

该协议的目的相对简单和抽象，并提供了一种控制车辆主要运动的简单方法，以及可

---

以使用按钮触发的几个任意功能。

这使得 GCS 软件能够为多种类型的车辆提供简单的控制，并允许具有不寻常功能的新车辆类型在对 MAVLink 协议或现有地面控制站 (GCS) 软件进行最小（如果有）更改的情况下运行。

可以在 QGroundControl 的 `src/Joystick/Joystick.cc` 中看到具体实现

相关协议详细介绍可见 [https://mavlink.io/en/services/manual\\_control.html](https://mavlink.io/en/services/manual_control.html)

### 5.3.6 相机协议

相机协议用于配置相机有效负载并请求其状态。它支持照片拍摄、视频拍摄和流媒体。它还包括用于查询和配置板载摄像头存储的消息。

相机内置对 MAVLink 相机协议的支持，Workswell 相机：WIRIS Pro、WIRIS Pro SC、WIRIS Security、WIRIS Agro、GIS-320（来源）；PhaseOne 相机（来源）。

相机管理器为不直接实现 MAVLink 支持的相机提供 MAVLink 相机协议接口。这些通常在配套计算机上运行：MAVLink 相机管理器（积极维护）、无人机代码相机管理器。

相机组件应遵循心跳/连接协议并发送恒定的心跳流（通常为 1Hz）。每个相机必须使用不同的预定义相机组件 ID：MAV\_COMP\_ID\_CAMERA 到 MAV\_COMP\_ID\_CAMERA6。第一次从新相机检测到心跳时，GCS（或其他接收系统）应启动相机识别过程。

相关协议详细介绍可见 <https://mavlink.io/en/services/camera.html>

### 5.3.7 图像传输协议

图像传输协议使用 MAVLink 作为通信通道，将任何类型的图像（原始图像、Kinect 数据等）从一个 MAVLink 节点传输到另一个 MAVLink 节点。它基本上获取实时摄像机图像，将其分割成小块并通过 MAVLink 发送。

图像流组件使用两个 MAVLink 消息：握手消息 `DATA_TRANSMISSION_HANDSHAKE` 用于启动图像流并描述要发送的图像，数据容器消息 `ENCAPSULATED_DATA` 用于传输图像数据。

该通信由 QGroundControl 发起，并 `DATA_TRANSMISSION_HANDSHAKE` 请求启动流。消息指定（1）`type`: `mavlink.h` 中枚举 `MAVLINK_DATA_STREAM_TYPE` 中的任何类型；（2）`jpg_quality`: 所需的图像质量（对于 JPEG 等有损格式）（3）初始请求中的所有其他字段必须为零。

当目标 MAV 收到握手请求时，它会发回一个 `DATA_TRANSMISSION_HANDSHAKE`。此行为提供了对请求的确认以及有关即将流式传输的图像的信息：

- `type`: 要流式传输的图像类型（与请求的类型相同）
- `size`: 图像大小（以字节为单位）。
- `width`: 图像宽度（以像素为单位）。
- `height`: 图像高度（以像素为单位）。

- 
- `packetsENCAPSULATED_DATA`: 要发送的 MAVLink 数据包数量
  - `payload`: 每个数据包的有效负载大小 (通常为 252 字节)
  - `jpg_quality`: 图像质量 (与要求相同)

然后图像数据被分成块以适合 `ENCAPSULATED_DATA` 消息并通过 MAVLink 发送。每个数据包都包含一个序列号以及所属图像流的 ID。

图像流传输器定期发送新图像，无需进一步交互。每个新图像都带有一个新的 `DATA_TRANSMISSION_HANDSHAKEACK` 数据包 size，其中包含更新的图像 `packets` 和 `payload` 字段。在此 ACK 数据包之后，新图像作为一系列 `ENCAPSULATED_DATA` 数据包到达。

要停止图像流，GSC 必须发送一个新的 `DATA_TRANSMISSION_HANDSHAKE` 请求数据包，其值全部为 0。MAVLink 节点将通过发送回 `DATA_TRANSMISSION_HANDSHAKE` 也包含 0 值来确认这一点。

相关协议详细介绍可见 [https://mavlink.io/en/services/image\\_transmission.html](https://mavlink.io/en/services/image_transmission.html)

### 5.3.8 文件传输协议 (FTP)

文件传输协议 (FTP) 支持通过 MAVLink 进行文件传输。它支持常见的 FTP 操作，例如：读取、截断、写入、删除和创建文件、列出和删除目录。

该协议遵循客户端-服务器模式，其中所有命令均由 GCS (客户端) 发送，无人机 (服务器) 使用包含所请求信息的 ACK 或包含错误的 NAK 进行响应。GCS 在大多数命令后都会设置超时，并且如果被触发，可能会重新发送命令。如果收到具有相同序列号的请求，无人机必须重新发送响应。

所有消息 (命令、ACK、NAK) 都在 `FILE_TRANSFER_PROTOCOL` 消息内交换。该消息类型定义是最小的，具有用于指定目标网络、系统和组件以及“任意”可变长度有效负载的字段。

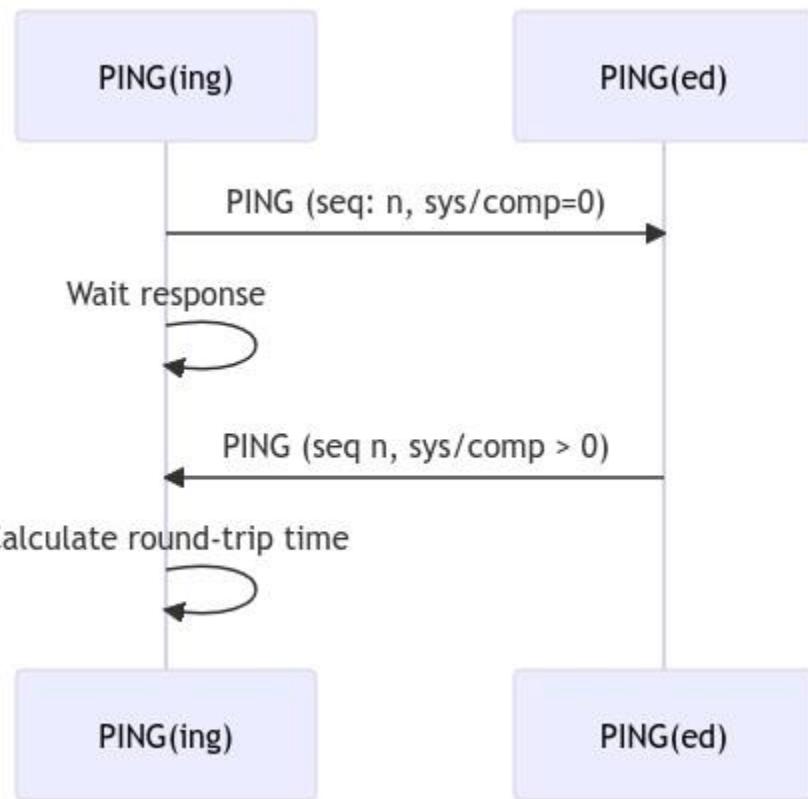
实现协议所需的不同命令和其他信息被编码在有效负载中 `FILE_TRANSFER_PROTOCOL`。本主题说明编码、打包格式、命令和错误以及发送命令的顺序以实现核心 FTP 功能。

可以在 QGroundControl 的 `src/uas/FileManager.cc` 和 `FileManager.h` 中看到具体实现

相关协议详细介绍可见 <https://mavlink.io/en/services/ftp.html>

### 5.3.9 PING 协议

PING 协议使系统能够测量任何连接上的系统延迟：串行端口、无线电调制解调器、UDP 等。简化的时序图如下：



PING 系统最初会使用以下内容填充 PING 消息

- `time_usec`: 当前系统时间戳。
- `seq`: 当前 PING 序列号 (`n, n+1, ...`)。这应该对发送的每条消息进行迭代 PING，并溢出回零。
- `target_system` 和 `target_component`: 0 (表示 PING 请求)。
- 消息标头自动包含发送方系统。

该消息可以由多个系统接收。所有 ping ed 系统都应使用另一条 PING 消息进行响应，

其中：

- 接收到的原始时间戳和序列号 PING 将在响应中发回。
- `target_system` 和 `target_component` 设置为来自传入 ping 消息标头的 ping 系统的 ID。

可以在 QGroundControl 的 `src/uas/FileManager.cc` 和 `FileManager.h` 中看到具体实现

相关协议详细介绍可见 <https://mavlink.io/en/services/ftp.html>

### 5.3.10 路径规划协议（轨迹接口）

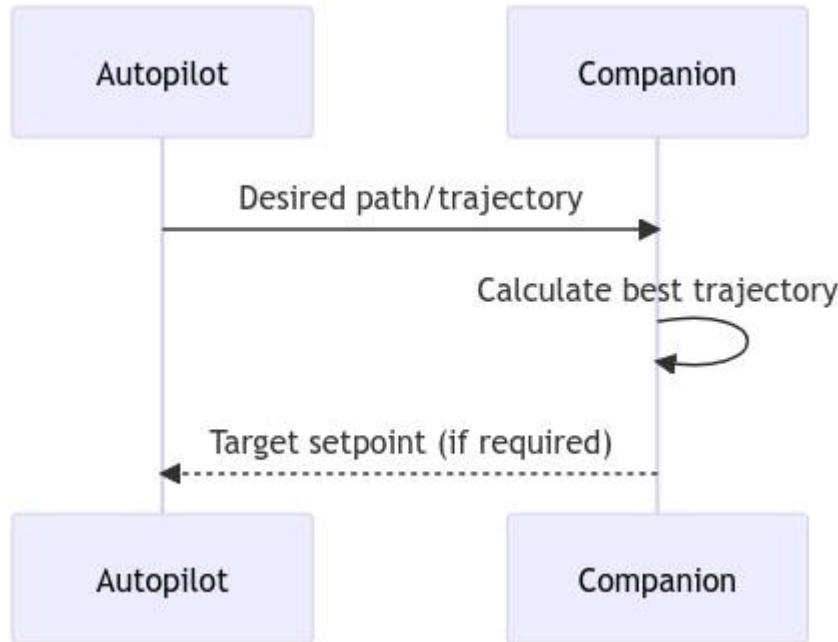
路径规划协议（也称为轨迹接口）是系统向另一个系统请求动态路径规划（即自动驾驶仪向配套计算机请求路径）的通用协议。

该协议主要适用于到达目的地的路径约束未知或可能动态变化的情况，但它也可用于任何其他路径管理活动。示例包括：执行预先计划的任务时避开障碍物、确定自形成/修复群的路径、将地理围栏管理卸载到配套计算机等。

需要路径规划的（自动驾驶仪）系统发送包含其当前位置和所需轨迹的消息。路径规

---

划系统（配套计算机）分析所需的路线，并发回带有新路径设定值的消息流。相关流程如图



相关协议详细介绍可见 <https://mavlink.io/en/services/trajectory.html>

### 5.3.11 电池协议

MAVLink 提供了许多用于提供电池信息的消息：通过 BATTERY\_STATUS 可以定期获取的电池状态信息。通过 SMART\_BATTERY\_INFO 可以获取电池部分信息例如设备名称。

针对系统中的每个电池单独发送消息（消息具有用于识别相应电池的实例 ID 字段）。

GCS 有责任提供适当的机制，允许用户评估具有多个电池的系统上的总体电池状态。

通过非 MAVLink 总线连接到飞行控制器的智能电池被视为飞行控制器组件的一部分。

具体来说，电池消息与自动驾驶系统和组件 ID 以及 MAV\_TYPE 车辆类型一起发出。

作为 MAVLink 网络上不同组件的智能电池必须：发出 HEARTBEAT = MAV\_TYPE\_BATTERY - 在 MAVLink 系统内具有唯一的组件 ID HEARTBEAT.type。默认情况下，前两个电池实例应使用 MAV\_COMP\_ID\_BATTERY 和 MAV\_COMP\_ID\_BATTERY2。后续实例可以使用任何备用/未使用的 ID。

相关协议详细介绍可见：<https://mavlink.io/en/services/battery.html>

### 5.3.12 事件接口（WIP）

事件接口是一种通用且灵活的机制，允许一个组件可靠地将偶发事件和状态更改通知 GCS（或任何其他组件）。例如，该界面可用于通知准备就绪、校准完成以及达到目标起飞高度。

该接口提供由飞行堆栈或其他组件共享的公共事件以及特定于实现的事件。MAVLink “常见”事件在 `mavlink/libevents/events/common.json` 中定义。

---

该接口提供以下主要功能：

- 具有重传功能的可靠交付
- 用于报告系统运行状况和布防检查的一致界面。
- 最大限度地减少自动驾驶仪侧的缓冲区要求。
- 最小化二进制消息长度
- 通用：与自动驾驶仪和 GCS 无关。
- 长期稳定、可扩展
- 允许将参数附加到事件。
- 可能的类型：uint8、int8、uint16、int16、uint32、int32、int64、uint64、float
- 枚举和位字段可以构建在这些类型之上
- 启用自动处理（例如来自包含事件的飞行日志）。
- 最大限度地减少嵌入式实现的自动生成代码量。
- 事件量平均 <1 Hz（可能会随着协议参数调整而变化，例如重传超时）。
- 事件可以有针对性或广播
- 任何组件都可以发送事件，包括摄像头、配套计算机、地面站等。
- 事件具有元数据，例如日志级别。它们还可以有详细的、更广泛的描述，可能带有 URL。
- 支持消息文本和消息翻译。

相关协议详细介绍可见 <https://mavlink.io/en/services/events.html>

### 5.3. CopterSim MAVLink\_Simple

MAVLink\_Simple 通过封装 MAVLink 协议的细节，使用户能够更轻松地建立通信链接、发送和接收消息，而无需深入了解 MAVLink 的底层细节，用户只需

### 5.4. CopterSim MAVLink\_Full

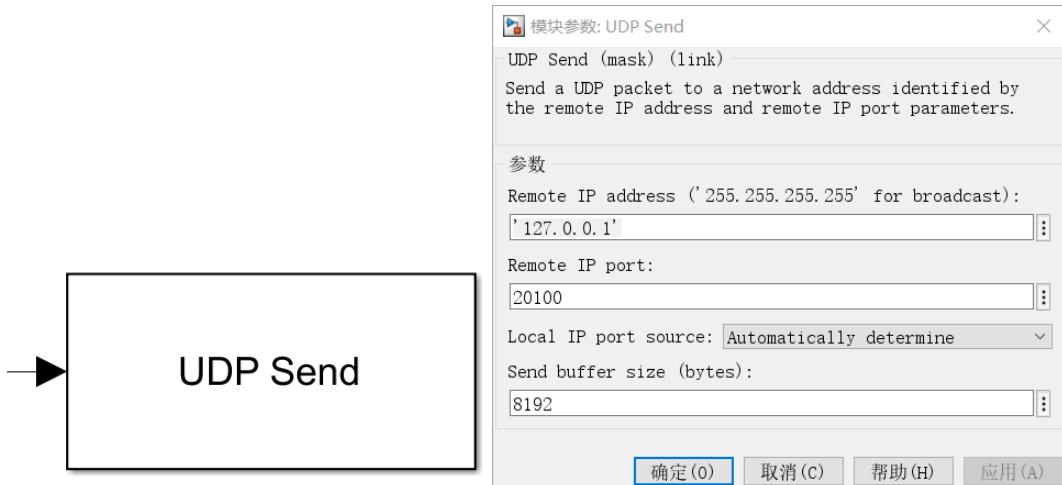
## 6. 控制接口（原版，PX4MavCtrlV4.py）

当前虽然 CopterSim 支持 UDP\_Simple 控制姿态，但是 PX4MavCtrlV4.py 的实现尚不支持。

## 6.1. Simulink 的 UDP 控制接口

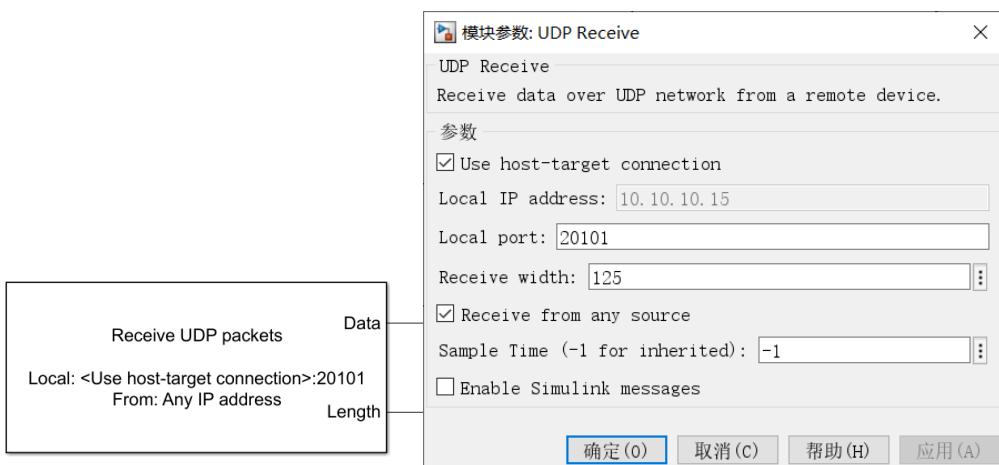
### 6.1.1. UDP Send (UDP 发送字节流模块)

UDP Send 是 Simulink 提供的字节流发送模块，其输入是字节流，并输出到网络中的特定端口。该模块有 3 个参数可配，网络 IP、网络端口、缓存大小。用户在将原始数据打包完成后，即可使用 UDP Send 模块将数据发送出去。



### 6.1.2. Receice UDP (UDP 接收字节流模块)

Simulink 提供 Receive UDP packets 模块接收 UDP 数据。CopterSim 发送的 UDP 数据可以使用该模块进行接收。Receive UDP packets 可以勾选本机作为主机，也可以不勾选时指定 IP。此外还需指定端口和接收数据的长度。

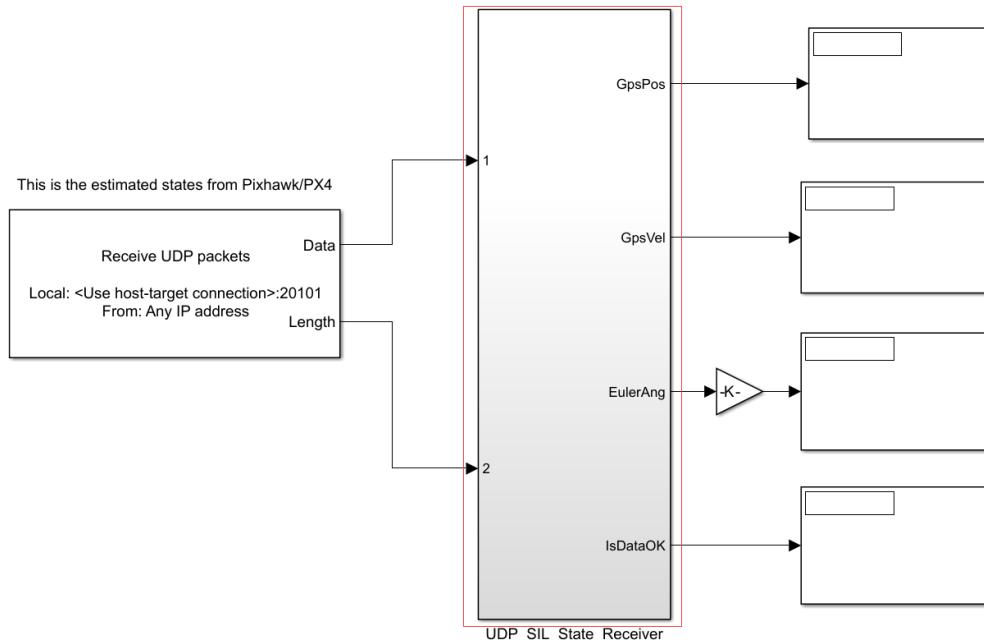


### 6.1.3. UDP\_SIL\_State\_Receiver (接收仿真位置、速度、姿态等信息模块)

UDP\_SIL\_State\_Receiver 用于接口 PX4 EKF2 估计的信息。平台中默认只解析了位置、速度、姿态信息，而实际来自 PX4 的消息数据格式为 **outhILStateData**，还包含相对高度、

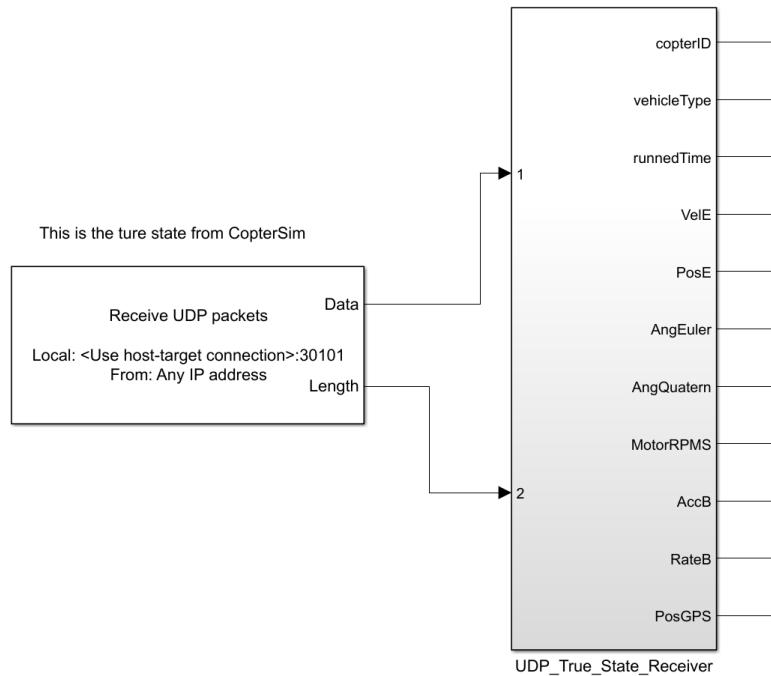
NED 位置、速度等。用户如果需要使用这些信息，则需修改 UDP\_SIL\_State\_Receiver 的内部实现，已将这些信息解析出来。

输出参数	GpsPos	GPS 位置, int32_t, lat&long: deg*1e7, alt: m*1e3
	GpsVel	GPS 速度, int32_t, NED, m/s*1e2->cm/s
	EulerAng	欧拉角, float
	IsDataOK	判断数据是否有效, bool, 有效时为 1
输出参数	data	字节流, unit8
	len	字节流长度, unit16



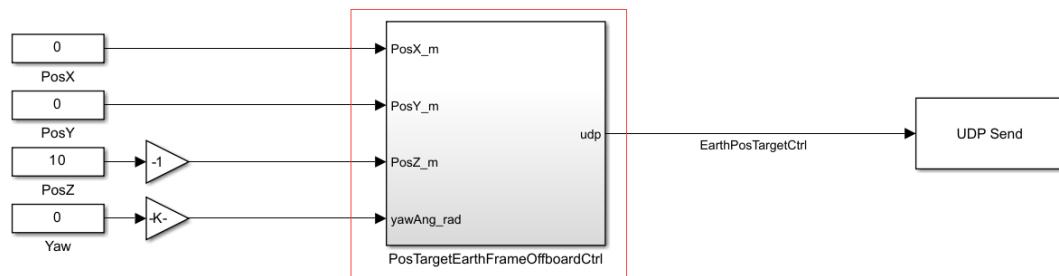
#### 6.1.4. UDP\_True\_State\_Receiver（接收真实位置、速度、姿态等信息模块）

UDP\_True\_State\_Receiver 用于解析 SOut2Simulator 数据。从 UDP 拿数据使用的模块与接收 SIL 数据相同，但是端口由 20101 修改为了 30101。类似的，用户可以修改 UDP\_True\_State\_Receiver 内部实现，以减少或者增加解析的具体数据。



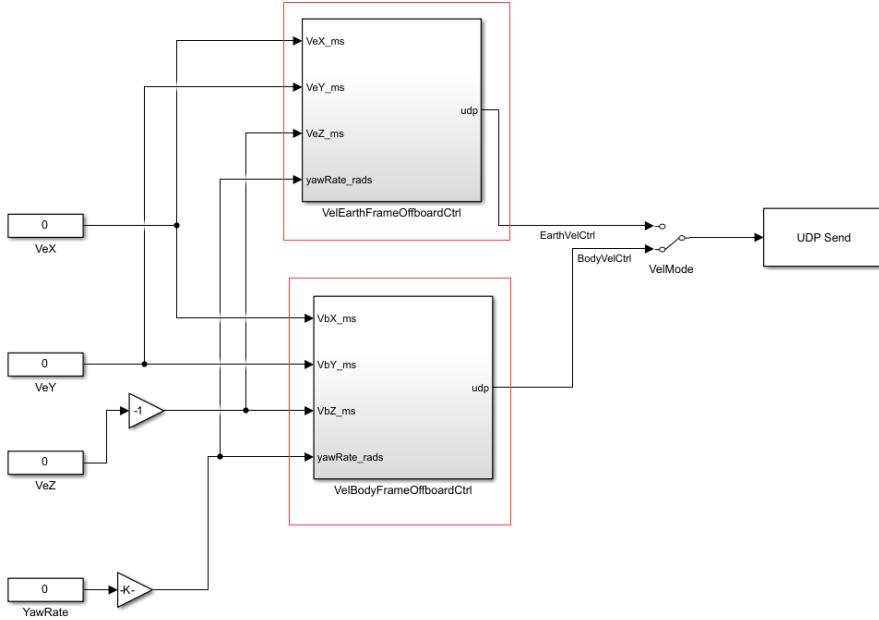
### 6.1.5. 位置控制（位置消息打包成字节流）

平台中提供了 PosTargetEarthFrameOffboardCtrl 模块用于将位置控制信息打包成字节流。用户可以通过该模块进行位置控制。在程序的运行过程中，可以修改左侧 PosX、PosY、PosZ 和 Yaw 的值。如果要进行更复杂的控制，则可将 PosX、PosY、PosZ 和 Yaw 替换为正弦信号或者自定义的函数等。



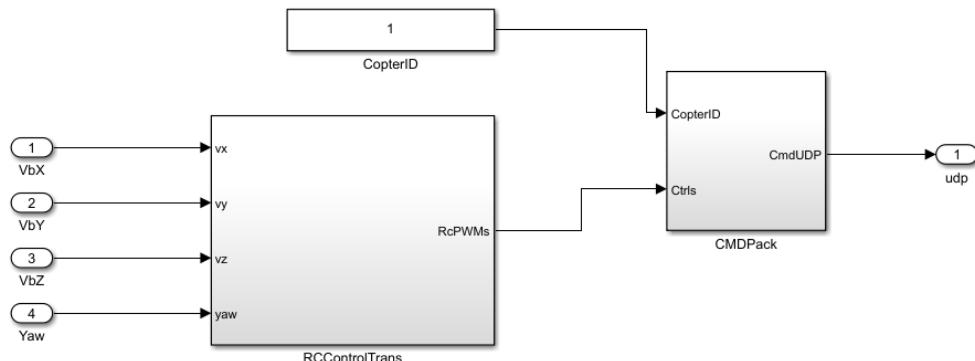
### 6.1.6. 速度控制（速度消息打包成字节流）

速度控制通常支持 NED 坐标系和 body 坐标系控制。VelEarthFrameOffboardCtrl 与 VelBodyFrameOffboardCtrl 内部指定的坐标系不一样。Simulink 速度控制使用的是平台简化的 UDP\_Simple 模式，即 [vx, vy, vz, yaw\_rate]。



### 6.1.7. 模拟遥控器 PWM 控制

模拟遥控器控制的关键是将速度信号转换成 PWM 波。该模块的输入仍然是速度，但是输出却是 PWM 波的脉宽。RCOverrideMavlink 模块，最终封装出来的数据需按 inHILCM DData 进行解析。



## 6.2. Python 的 UDP 控制接口

PX4MavController 是 python 控制的核心实现类，也是用户通过 python 获取载具状态并进行控制的核心类。

### 6.2.1. PX4MavController: `__init__(self, port=20100, ip='127.0.0.1')` (参数的初始化)

函数原型： `__init__(self, port=20100, ip='127.0.0.1')`

参数： `(self, port=20100, ip='127.0.0.1')`

---

返回值： PX4MavCtrler 类型的对象

PX4MavCtrler: `__init__()` 是类型的初始化函数，完成载具状态、连接参数的初始化。可以指定两个参数，分别是端口和 ip。`__init__()`会在初始化对象时自动调用。

	名称	描述	默认值
输出参数	port	端口	20100, number
	ip	UDP 通信 ip 地址	“127.0.0.1”, string
返回值	PX4MavCtrler	一个 PX4MavCtrler 对象	--

## 6.2.2. InitMavLoop() (初始化 CopterSim 到 MAVlink 的监听)

函数原型： `InitMavLoop(self, UDPMode=2)`

参数： `InitMavLoop(self, UDPMode=2)`， 初始化对象时的参数设置会影响该函数的行为。  
`UDPMODE` 用于指定通信的模式。

返回值： 无

PX4MavCtrler: `InitMavLoop()` 函数用于建立接收连接和发送连接，几乎每个例程都要使用该接口。该函数支持指定参数 `UDPMODE`, 0 -`UDP_Full`、1-`UDP_Simple`、2- `MAVLink_Full`、3- `MAVLink_Simple`、4- `MAVLink_NoSend`。`InitMavLoop()`会根据模式的不同，初始化不同连接对象。然后，会初始化两个线程分别用于获取状态信息和发送控制指令。接收线程会调用 `getMavMsg()`，该线程在 `InitMavLoop()`就被启动。发送线程会调用 `OffboardSendMode()`，该线程只有进入 `Offboard` 模式时才会调用。

## 6.2.3. endMavLoop() (停止 Mavlink 的监听)

函数原型： `endMavLoop (self)`

参数： 无

返回值： 无

PX4MavCtrler: `endMavLoop()` 用于停止监听 `Mavlink` 消息和运行。它调用了 [stopRun\(\)](#) 方法来停止从 20100 端口或串口接收消息。

## 6.2.4. initOffboard() (发送 offboard 给 PX4)

函数原型： `initOffboard(self)`

参数： 无

返回值： 无

PX4MavCtrler: `initOffboard()` 用于将载具的模式切换为 `Offboard` 模式，进而用户可以发送 `Offboard` 位置、速度、加速度、偏航角、偏航角速率等控制信息。该函数不仅会让载具

---

进入 Offboard 模式，而且会让载具解锁。当 initOffboard()运行后，Offboard 消息将通过 OffboardSendMode()，不断被发送。initOffboard()会将速度、偏航角速率设置为 0 并发送。

### 6.2.5. initOffboard2()（发送 offboard 给 PX4）

函数原型：initOffboard2(self)

参数：无

返回值：无

PX4MavCtrlr: initOffboard2()同样用于进入 Offboard 模式。与 initOffboard()的差别在于，initOffboard2()不会将控制信息重置为 0。initOffboard2()适用于空中切换为 Offboard 模式，且不期望将控制信息都设为 0 的情形。

### 6.2.6. InitTrueDataLoop()（初始化 UDP True 的监听）

函数原型：InitTrueDataLoop(self)

参数：无

返回值：无

PX4MavCtrlr: InitTrueDataLoop()是初始化 UDP True 数据监听循环。它首先绑定 UDP 套接字到指定的端口 (self.port+1+10000)，然后将 stopFlagTrueData 设置为 False，表示不停止监听循环。接着创建一个线程 tTrue，目标函数为 getTrueDataMsg，然后启动线程 tTrue。接着绑定另一个 UDP 套接字到另一个端口 (self.port+1+20000)，然后将 stopFlagPX4Data 设置为 False，表示不停止监听循环。再创建一个线程 tPX4，目标函数为 getPX4DataMsg，然后启动线程 tPX4。

### 6.2.7. EndTrueDataLoop()（结束 UDP True 的监听）

函数原型：EndTrueDataLoop(self)

参数：无

返回值：无

EndTrueDataLoop 是结束 True 数据模式。它首先将 stopFlagTrueData 设置为 True，表示停止 True 数据的监听循环。接着将 hasTrueDataRec 设置为 False，表示没有接收到 True 数据。最后关闭 UDP 套接字 udp\_socketTrue。最后将 stopFlagPX4Data 设置为 True，表示停止 PX4 数据的监听循环。然后等待 tPX4 线程结束。最后关闭 UDP 套接字 udp\_socketPX4

### 6.2.8. 访问 PX4MavCtrlr 成员变量读取状态

PX4MavCtrlr 这个类有以下成员变量

变量	注释	变量	注释
uavTimeStamp	飞行器时间戳	trueTimeStamp	真实时间戳

uavAngEular	估计的欧拉角	trueAngEular	真实的欧拉角
uavAngRate	估计的角速度	trueAngRate	真实的角速度
uavPosNED	估计的本地位置 (N ED 坐标系)	truePosNED	真实的位置 (NED 坐标系)
uavVelNED	估计的本地速度	trueVelNED	真实的速度
uavPosGPS	估计的 GPS 位置 (NED 坐标系)	uavPosGPSHome	估计的 GPS 起始位 置 (NED 坐标系)
uavGlobalPos	估计的全局位置 (转换为 UE4 地图 坐标系)	trueAngQuatern	真实的四元数
trueMotorRPMS	真实的电机转速	trueAccB	真实的加速度
truePosGPS	真实的 GPS 位置	trueSimulinkData	真实的 Simulink 数 据
useCustGPSOri	是否使用自定义 GP S 方向	trueGpsUeCenter	真实的 GPS UE 中 心位置
GpsOriOffset	GPS 方向偏移量	uavThrust	估计的推力
pos	位置	vel	速度
acc	加速度	yaw	偏航角
yawrate	偏航角速率		

### 6.2.9. SendVelNED() (给 PX4 发送最大速度)

函数原型：SendVelNED(self,vx=0,vy=0,vz=0,yawrate=0)

参数：(self,vx=0,vy=0,vz=0,yawrate=0)， self 表明 PX4MavCtrler 共享对象会影响该函数的行为。每个参数的默认值都为 0。

返回值：无

发送 NED 坐标系下速度控制信息，带有偏航角速率。该函数包含四个输入参数，分别为 vx=0, vy=0, vz=0, yawrate=0。每个参数的默认值都是 0，即在 SendVelNED() 不带任何参数的情况下该函数将发送期望速度 0 期望偏航角速率 0。

### 6.2.10. SendVelNEDNoYaw() (无偏航情况下发送最大速 度)

函数原型：SendVelNEDNoYaw(self,vx,vy,vz)

参数：(self,vx,vy,vz)， self 表明 PX4MavCtrler 共享对象会影响该函数的行为。参数没有默认值，vx,vy,vz 都必须指定。

返回值：无

发送 NED 坐标系下的速度控制信息，不带偏航角速率。该函数有 3 个参数，vx,vy,vz。这 3 个参数没有默认值。值得说明的是，不带偏航角速率和偏航角速率为 0 是两码事。

---

## 6.2.11. SendVelFRD() (在 FRD 框架下发送最大速度)

函数原型：SendVelFRD(self,vx=0,vy=0,vz=0,yawrate=0)。

参数：(self,vx=0,vy=0,vz=0,yawrate=0)， self 表明 PX4MavCtrler 共享对象会影响该函数的行为。每个参数的默认值都为 0。

返回值：无

该函数是指定载体坐标系下速度期望值，单位 m/s。在不带任何参数的情况下将发送期望速度 0 期望偏航角速率 0。

## 6.2.12. SendVelNoYaw() (不受横滚下在 FRD 框架下发送最大速度)

函数原型：SendVelNoYaw(self,vx,vy,vz)

参数：(self,vx,vy,vz)， self 表明 PX4MavCtrler 共享对象会影响该函数的行为。参数没有默认值，vx,vy,vz 都必须指定。

返回值：无

发送 FRD 载体坐标系下的速度控制信息，不带偏航角速率。该函数有 3 个参数，vx,vy,vz。这 3 个参数没有默认值。值得说明的是，不带偏航角速率和偏航角速率为 0 是两码事。

## 6.2.13. SendPosNED() (发送坐标位置给 PX4)

函数原型：SendPosNED (self,x=0,y=0,z=0,yaw=0)

参数：x、y、z 分别表示 NED 坐标系下位置，yaw 表示偏航角，默认值都为 0。

返回值：无

发送 NED 坐标系下位置和偏航角。

## 6.2.14. SendVelYawAlt() (发送姿态给 PX4)

函数原型：SendVelYawAlt(self,vel=10,yaw=6.28,alt=-100)

参数：vel 表示水平方向的速率，yaw 表示偏航角，alt 表示高度。

返回值：无

发送 NED 坐标系下的高度、偏航角和水平方向速率。

## 6.2.15. SendPosGlobal() (发送目标位置给 PX4)

函数原型：SendPosGlobal(self,lat=0,lon=0,alt=0,yawValue=0,yawType=0)

参数：lat&lon 表示纬度、经度，单位°；alt 表示高度，向下为正，单位 m。yawValue 的具体含义由 yawType 决定。yawType 0，不指定 yaw；yawType 1，偏航角控制；yawType 2，偏航角速率。

---

返回值：无

发送全局位置，该函数的输入纬度、经度单位是°，高度单位是 m。但数据发送时会自动转换为 MAV\_FRAME\_GLOBAL\_INT 形式。

## 6.2.16. `SendPosNEDNoYaw()`（发送无偏航控制的目标位置）

函数原型：`SendPosNEDNoYaw (self,x=0,y=0,z=0)`

参数：x、y、z 分别表示 NED 坐标系下位置。

返回值：无

控制 NED 坐标系下位置，不指定偏航角。

## 6.2.17. `SendPosFRD()`（发送 FRD 下的位置给 PX4）

该函数的原意是控制 FRD 载体坐标系下的位置，但实际 PX4 不支持这种格式，所以该函数并不会像预期的那样运行。

## 6.2.18. `SendPosFRDNoYaw()`（发送无偏航控制的 FRD 下的位置给 PX4）

该函数的原意是控制 FRD 载体坐标系下的位置，但实际 PX4 不支持这种格式，所以该函数并不会像预期的那样运行。

## 6.2.19. `SendPosNEDExt()`（发送目标位置给固定翼）

函数原型：`SendPosNEDExt(self,x=0,y=0,z=0,mode=3,isNED=True)`

参数：x、y、z 表示位置，mode 用于控制固定翼的飞行模式。mode 0，滑行模式；mode 1，起飞模式；mode 2，着陆模式；mode 3，盘旋模式；mode 4，零油门、滚转和俯仰。

返回值：无

该函数仅用于固定翼，支持多种模式。

## 6.2.20. `sendPX4UorbRflyCtrl()`（发送数据给 CopterSim）

函数原型：`sendPX4UorbRflyCtrl(self,data=[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],modes=1,flags=1)`

参数：data 是 16 维的控制量，默认全为 0；modes 是模式，flags 是标志。

返回值：无

该函数实现发送 inHILCMDData 数据给 CopterSim。

## 6.2.21. SendAccPX4()（向 PX4 发送加速度信息）

函数原型：SendAccPX4(self,afx=0,afy=0,afz=0,yawValue=0,yawType=0,frameType=0)

参数：afx、afy、afz 表示加速度，yawValue 表示偏航角或角速度。

yawType 0: 没有 yaw; yawType 1: 偏航角控制; yawType 2: 偏航角速率控制。

frameType 0: NED 坐标系; frameType 1: FRD 载体坐标系。

返回值：无

该函数用于加速度控制，该接口可以在 **UDP\_Full** 和 **MAVLINK** 模式下支持。CopterSim 已经支持 **UDP\_Simple** 模式下支持控制加速度，但是当前 **PX4MavCtrlV4.py** 的 python 接口尚不支持。

## 6.2.22. endOffboard()（向 PX4 发送 out offboard 模式）

函数原型：endOffboard(self)

参数：无

返回值：无

将 **isInOffboard** 设置为 False，并将 PX4 的相应标志恢复为进入 Offboard 之前的状态。

## 6.2.23. stopRun()（停止监听 mavlink 消息）

函数原型：stopRun(self)

参数：无

返回值：无

**stopRun** 的作用是停止 mavlink 的监听循环。它首先检查是否已经解锁 (**isArmed**)，如果已解锁，则发送一个解锁命令给飞控。然后设置 **stopFlag** 为 True，以停止监听循环。接着等待 0.5 秒，然后等待 t1 线程结束。如果当前处于 Offboard 模式，则调用 **endOffboard** 方法来结束 Offboard 模式。如果 **UDPMODE** 大于 1.5，则关闭与飞控的连接。否则，如果不是串口连接，则关闭 UDP 套接字。

## 6.3. Simulink 的 MAVLink 控制接口（串口连接）

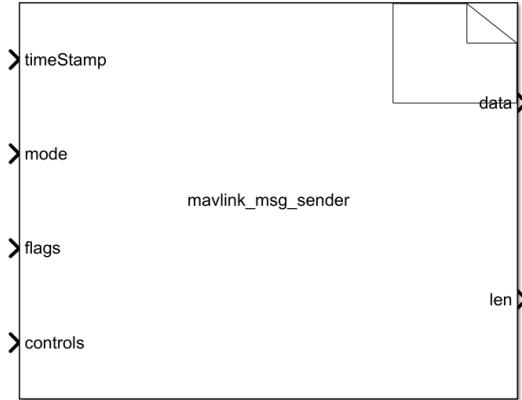
### 6.3.1. **mavlink\_msg\_sender**

在 Simulink 中提供如下接口，用于将数据打包成 MAVLink 格式。**mavlink\_msg\_sender** 是一个 S 函数，提供的默认接口是用于发送控制消息。用户如果只是发送常规的控制指令，直接使用该模块打包数据即可。如果有一些特殊的需求，则可按照 Simulink S 函数编写方法修改输入输出数据格式。【**7.RflySimExtCtrl\1.BasicExps\extAPIUsage**】

输入参数	timeStamp	时间戳, unit32
	mode	模式, unit8
	flags	标志, uint32

---

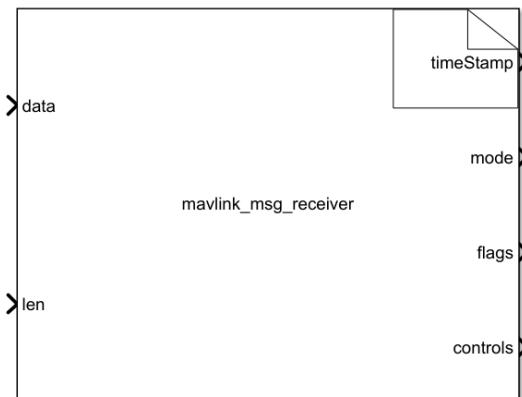
	controls	16 维控制信号, single
输出参数	data	字节流, unit8
	len	字节流长度, unit16



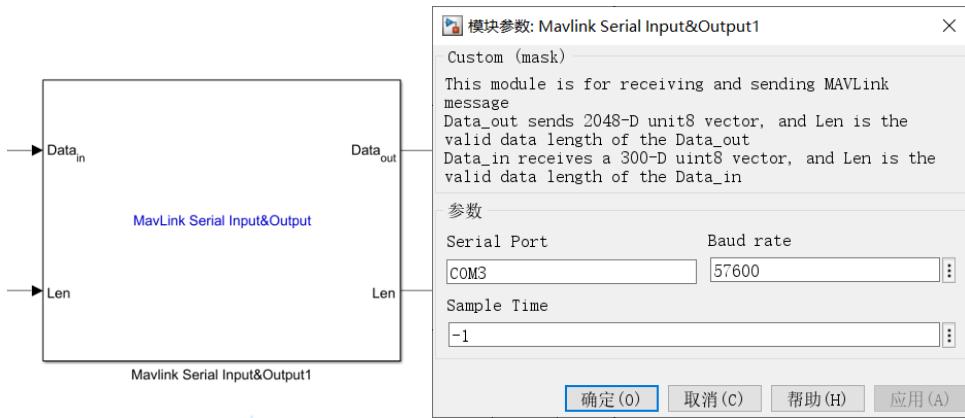
### 6.3.2. `mavlink_msg_receiver`

`mavlink_msg_receiver` 是 `mavlink_msg_sender` 的逆向过程，用于解析字节流数据。`mavlink_msg_receiver` 的数据格式需要与 `mavlink_msg_sender` 严格对应。【7.RflySimExtCtrl\1.BasicExps\c0\_ExtAPIUsage】

	timeStamp	时间戳, unit32
	mode	模式, unit8
	flags	标志, uint32
	controls	16 维控制信号, single
输出参数		
输入参数	data	字节流, unit8
	len	字节流长度, unit16



### 6.3.3. MavLink Serial Input&Output



## 6.4. Python 的 MAVLink 控制接口（基于 pymavlink）

### 6.4.1. SendMavArm () (向 PX4 发出解除武装命令)

函数原型：SendMavArm(self, isArm=0)

参数：isArm 表示是否解锁无人机，1 表示解锁，2 表示上锁

返回值：无。

SendMavArm 是向 PX4 发送命令来解锁或上锁无人机。

### 6.4.2. SendMavCmdLong() (向 PX4 发送命令长)

函数原型：SendMavCmdLong(self, command, param1=0, param2=0, param3=0, param4=0, param5=0, param6=0, param7=0)

参数：command 表示要发送的命令。

param1：保持时间（固定翼忽略，旋翼停留航点时间）

param2：接受半径（如果处在这个半径的球体内，则该航路点算到达）

param3：通过半径 0 表示通过 WP，如果半径 > 0 则通过 WP。顺时针轨道为正值，逆时针轨道为负值允许轨迹控制。

Param4：偏航航路点所需偏航角(旋翼)。使用当前系统偏航航向模式(例如，偏航到下一个航路点，偏航到家等)。

Param5：纬度

Param6：经度

Param7：高度（单位米）

返回值：无。

SendMavCmdLong 是向 PX4 发送 Mavlink 的命令长消息。并根据当前的连接方式和飞行模式，选择不同的发送方式。如果是串口连接或者真实飞行模式，函数会使用 the\_connection 对象的 mav.command\_long\_send 方法发送命令。如果是 UDP 连接模式，函数会使用 mav0 对象的 command\_long\_encode 方法发生命令。

---

### 6.4.3. sendMavOffboardCmd() (发送离板指令给 PX4)

函数原型：sendMavOffboardCmd(self,type\_mask,coordinate\_frame, x, y, z, vx, vy, vz, afx, afy, afz, yaw, yaw\_rate)

参数：type\_mask 表示控制模式的位掩码； coordinate\_frame 表示坐标系的类型；

x, y, z 表示目标位置的坐标； vx、vy、vz 表示目标速度的分量； afx、afy、afz 表示目标加速度的分量； yaw 表示目标航向角； yaw\_rate 表示目标航向角速率；

返回值：无。

sendMavOffboardCmd 是向 PX4 发送发送 Offboard 命令。

### 6.4.4. initRCSendLoop() (初始化遥控器)

函数原型：initRCSendLoop(self, Hz=30)

参数：Hz 表示遥控发送的频率，默认为 30Hz。

返回值：无。

initRCSendLoop 初始化遥控，并发送循环

### 6.4.5. SendRCPwms() (更新遥控器的 PWM 值)

函数原型：SendRCPwms(self, Pwms)

参数：Pwms 是一个包含 PWM 值的列表

返回值：无。

SendRCPwms 是将给定的 PWM 值列表发送到遥控器。

### 6.4.6. endRCSendLoop() (停止遥控发送循环)

函数原型：endRCSendLoop(self)

参数：无

返回值：无。

endRCSendLoop 是结束遥控器发送循环。

### 6.4.7. SendSetMode() (发送 mavlink 命令，切换飞行模式)

函数原型：SendSetMode(self,mainmode,cusmode=0)

参数：mainmode 表示主要飞行模式； cusmode 表示自定义模式，可选参数，默认值为 0。

返回值：无。

SendSetMode 是向 PX4 发送 MAVLink 命令，用于改变飞行模式。

---

#### 6.4.8. SendAttPX4() (向 PX4 发送速度控制信号)

函数原型：SendAttPX4(self,att=[0,0,0,0],thrust=0.5,CtrlFlag=0,AltFlg=0)

参数：att 是表示姿态，thrust 表示油门。

CtrlFlag 0, 欧拉角, 单位度;

CtrlFlag 1, 欧拉角, 单位 rad;

CtrlFlag 2, 欧拉角, 四元数;

CtrlFlag 3, 欧拉角, 角速率 rad/s;

CtrlFlag 4, 欧拉角, 角速率°/s。

AltFlg 0, 油门值缩放到 0~1。

AltFlg>0, 油门值就是期望的高度。

返回值：无

该函数用于控制载具的姿态，该接口只有在 MAVLink 模式下支持。CopterSim 可以在 UDP 模式下进行硬件在环仿真，在这种情况下，Python 脚本可以绕过 CopterSim 直接 PX4 发送 MAVLink 消息。CopterSim 已经支持 UDP\_Simple 模式下支持控制姿态，但是当前 PX4MavCtrlV4.py 的 python 接口尚不支持。

#### 6.4.9. enFixedWRWTO() (命令飞机在跑道上起飞)

函数原型：enFixedWRWTO(self)

参数：无

返回值：无。

enFixedWRWTO 是发送命令以启用飞机在跑道上起飞.

#### 6.4.10. SendCruiseSpeed() (发送命令改变飞机巡航速度)

函数原型：SendCruiseSpeed(self,Speed=0)

参数：Speed 表示巡航速度的值，默认为 0。

返回值：无。

SendCruiseSpeed 是发送命令以改变飞机的巡航速度（单位：m/s）。

#### 6.4.11. SendCopterSpeed() (设置多旋翼最大飞行速度)

函数原型：SendCopterSpeed(self,Speed=0)

参数：Speed 表示最大速度的值，默认为 0。

返回值：无。

SendCopterSpeed 是发送命令设置多旋翼飞行器飞行的最大速度。

---

#### 6.4.12. **SendGroundSpeed()** (设置飞机的地面速度)

函数原型：SendGroundSpeed(self,Speed=0)

参数： Speed 表示地速的值， 默认为 0

返回值：无。

SendGroundSpeed 是发送命令以改变飞机的地面速度（单位：m/s）。

#### 6.4.13. **SendCruiseRadius()** (设置飞机的巡航半径)

函数原型：SendCruiseRadius(self,rad=0)

参数： rad 表示巡航半径的值， 默认为 0。

返回值：无。

SendCruiseRadius 是发送命令以改变飞机的巡航半径（单位：米）。

#### 6.4.14. **sendTakeoffMode()** (起飞命令)

函数原型：sendTakeoffMode(self,alt=0)

参数： alt 表示起飞高度， 默认为 0。

返回值：无。

sendTakeoffMode 是发送命令使飞机起飞。

#### 6.4.15. **sendMavTakeOff()** (命令飞机起飞到预定位置)

函数原型：sendMavTakeOff(self,xM=0,yM=0,zM=0,YawRad=0,PitchRad=0)

参数： xM、 yM、 zM 表示目标位置的本地坐标； YawRad 表示目标航向角（弧度）；

PitchRad 表示目标俯仰角（弧度）；

返回值：无。

sendMavTakeOff 发送命令使飞机起飞到指定的本地位置（单位：米）。

#### 6.4.16. **sendMavTakeOffLocal()** (命令飞机飞行到所的需 本地位置)

函数原型：sendMavTakeOffLocal(self,xM=0,yM=0,zM=0,YawRad=0,PitchRad=0,AscendRate=2)

参数： xM、 yM、 zM 表示目标位置的本地坐标； YawRad 表示目标航向角（弧度）；

PitchRad 表示目标俯仰角（弧度）； AscendRate 表示上升速率（单位：m/s）

返回值：无。

sendMavTakeOffLocal 发送命令使飞机起飞到指定的本地位置（单位：米），只是相对于 sendMavTakeOff 函数多了一个传入的参数 AscendRate 上升率

---

## 6.4.17. `sendMavTakeOffGPS()` (命令飞机飞行到所需的全球位置)

函数原型： `sendMavTakeOffGPS(self,lat,lon,alt,yawDeg=0,pitchDeg=15)`

参数： lat、lon、alt 表示目标位置的全局坐标； yawDeg 表示目标航向角（度）； pitchDeg 表示目标俯仰角（度）；

返回值： 无。

`sendMavTakeOffGPS` 是发送命令使飞机起飞到指定的全球位置（单位：度）

## 6.4.18. `sendMavLand()` (降落到指定位置)

函数原型： `sendMavLand(self,xM,yM,zM)`

参数： xM、yM、zM 表示目标位置的本地坐标

返回值： 无。

`sendMavLand` 发送命令使飞机降落到指定的本地位置（单位：米）。

## 6.4.19. `sendMavLandGPS()` (降落到指定的全局位置)

函数原型： `sendMavLandGPS(self,lat,lon,alt)`

参数： lat、lon、alt 表示目标位置的全局坐标。

返回值： 无。

`sendMavLandGPS` 发送命令使飞机降落到指定的全局位置（单位：度）。

## 6.4.20. `sendMavSetParam()` (发送命令给 PX4 以更改期望参数)

函数原型： `sendMavSetParam(self,param_id, param_value, param_type)`

参数： param\_id 表示要改变的参数的 ID； param\_value 表示要设置的参数值；

param\_type 表示参数的类型；

返回值： 无。

`sendMavSetParam` 向 PX4 发送命令，用于改变指定的参数值的 ID、数值大小、和类型。

## 6.4.21. `SendHILCtrlMsg()` (PX4 发送给 ComSim)

函数原型： `SendHILCtrlMsg(self,ctrls)`

参数： ctrls 是一个列表，表示要发送的控制信号。

返回值： 无。

`SendHILCtrlMsg` 是向 PX4 发送 hil\_actuator\_controls 命令，用于控制飞机的姿态和动作。

如果是串口连接或者真实飞行模式，函数会使用 the\_connection 对象的 mav.hil\_actuator\_con

---

trols\_send 方法发送命令。如果是 UDP 连接模式，函数会使用 mav0 对象的 hil\_actuator\_controls\_encode 方法发送命令。

### 6.4.22. SendHILCtrlMsg1() (发送调试指令)

函数原型：SendHILCtrlMsg1(self)

参数：无

返回值：无。

SendHILCtrlMsg1 是向 PX4 发送 debug\_vect 命令，用于发送调试向量信息。如果是串口连接或者真实飞行模式，函数会使用 the\_connection 对象的 mav.debug\_vect\_send 方法发送命令。如果是 UDP 连接模式，函数会使用 mav0 对象的 debug\_vect\_encode 方法发送命令。

## 6.5. 基于 ROS 的 MAVLink 控制接口（基于 mavros，复制下视觉组的）

## 7. rflysim 标准库版控制接口（新版，支持 Redis）

### 7.1. ctrl

ctrl 模块是外部控制模块，支持通过 UDP\_Full、UDP\_Simple、MAVLINK\_Full、MAVLINK\_Simple、Redis\_Full、Redis\_Simple 等模式，获取无人机的位置、速度、姿态信息，并对无人机的位置、速度、航向进行控制。

#### 7.1.1. 代码结构

ctrl 模块的代码放于 Python38/Lib/site-packages/rflysim/ctrl 目录下。包含 \_\_init\_\_.py、api.py、offboard.py、topics.py 四个文件。\_\_init\_\_.py 和 api.py 主要是提供用户接口，将在下一节“用户 api”中详细介绍。

#### 7.1.2. offboard.py

offboard.py 是外部控制的核心源码，支持通过 UDP\_Full、UDP\_Simple、MAVLINK\_Full、MAVLINK\_Simple、Redis\_Full、Redis\_Simple 等模式，获取无人机的位置、速度、姿态信息，并对无人机的位置、速度、航向进行控制。offboard.py 面向高级开发者用户，能支持的功能多，但接口调用起来相对复杂。

offboard.py 类型表

类型	功能
SimMode	仿真模式，包含 HITL、SITL、SITL_RFLY、Simu link_DLL 等模式。
CtMode	是指连接的模式，包含 UDP/MAVLINK/Redis 等主模式。根据数据包的大小，又分为 Simple 和 Full 模式。为了进一步减少数据开销，还有 MAVLink_NoSend 和 MAVLink_NoGPS 两种模式。
Coordinate	目前仅支持 3 种坐标系发送控制信息，LOCAL_NED、GLOBAL_INT、NED_BODY。其对应的编码参考 MAVLINK 协议。
PX4MainMode	PX4 支持的主模式，手动、定高、定点等。
PX4SubMode	PX4 支持的子模式，起飞、任务、着陆、跟随等。
RedisKey	Redis 模式下通过键值区分不同的飞机。
PX4CmdLong	PX4 长命令，包含 7 个数据。
FIFO	用于 MAVLINK 数据读写。
RflySimCP	综合模型控制协议。
OffboardSimpleMode	简化控制协议，该协议中一个 int 型数据标识模式，剩余 4 个 float 型数据是实际的数据。由于 float 和 int 都是 4 字节，所以 float 数据也可以换成 int 数据。
OffboardPosType	标明 Offboard 消息中有哪些信息，完整兼容 PX4 Offboard 位置消息，该消息包含位置、速度、加速度、油门、偏航角、偏航角速率等。
OffboardPosSend	用于构建用于发送的数据包。
VehicleStatus	用于表示飞机的状态，包含 ID、位置、速度、姿态等信息。
EarthFrame	提供地理坐标系(lla)、地心地固系(ecef)、站心坐标系(NED、ENU)之间的转换。

---

PID	提供 PID 控制器，单通道。
Offboard	最核心的类，下面详细介绍

Offboard 类是 Ctrl 甚至整个 RflySim 最为核心的类。从功能上讲，主要分为两类功能，接收数据和发送数据。编码时，接收数据的函数用 receive 开头，而发送数据的函数都用 send 开头。

如下图所示，是接收消息相关的函数。接收消息的入口函数是 receive\_msg\_loop()，该函数是一个 while 大循环，会在 Offboard 类初始化的时候通过一个单独的线程启动。receive\_msg\_loop() 的作用是根据连接模式的不同调用不同的接收函数和数据解析函数。因为 redis 模式和 UDP 模式极为相似，很多代码类似，就把 redis 和 udp 封装在一起实现。简而言之，当模式为 UDP/Redis 时，receive\_msg\_loop() 将调用 receive\_udp\_redis\_msg()，否则调用 receive\_mav\_msg()。然后调用分别的 update 函数解析数据，解析出来的数据都会放在 vehicle\_status 中。所以高级开发者用户如果要访问无人机的状态信息，只需读取 `self.vehicle_status` 即可。

由于在 UDP/Redis 模式下区分控制指令的 Simple 和 Full 模式，所以 receive\_udp\_redis\_msg() 还会进一步区分 Simple 和 Full 模式。虽然 MAVLINK 也有 Simple 和 Full 模式的差别，但 MAVLINK 的 Simple 和 Full 模式不体现在控制指令上，所以对于外部控制来说无需区分 Simple 和 Full。值得说明的是，使用 MAVLINK 模式时，如果是软件在环那么其实际的通信方式也是 UDP。但与 UDP 模式的差异在于，MAVLINK 模式会对数据包进行编码。而在 UDP 模式时，MAVLINK 编码过程在 CopterSim 中进行。

```
def receive_mav_msg(self):...
def update_vehicle_status_by_mav(self):...
def receive_full_udp_redis_msg(self):...
def receive_simple_udp_redis_msg(self):...
def receive_udp_redis_msg(self):...
def update_vehicle_status_by_udp_redis(self):...
def receive_msg_loop(self):...
```

---

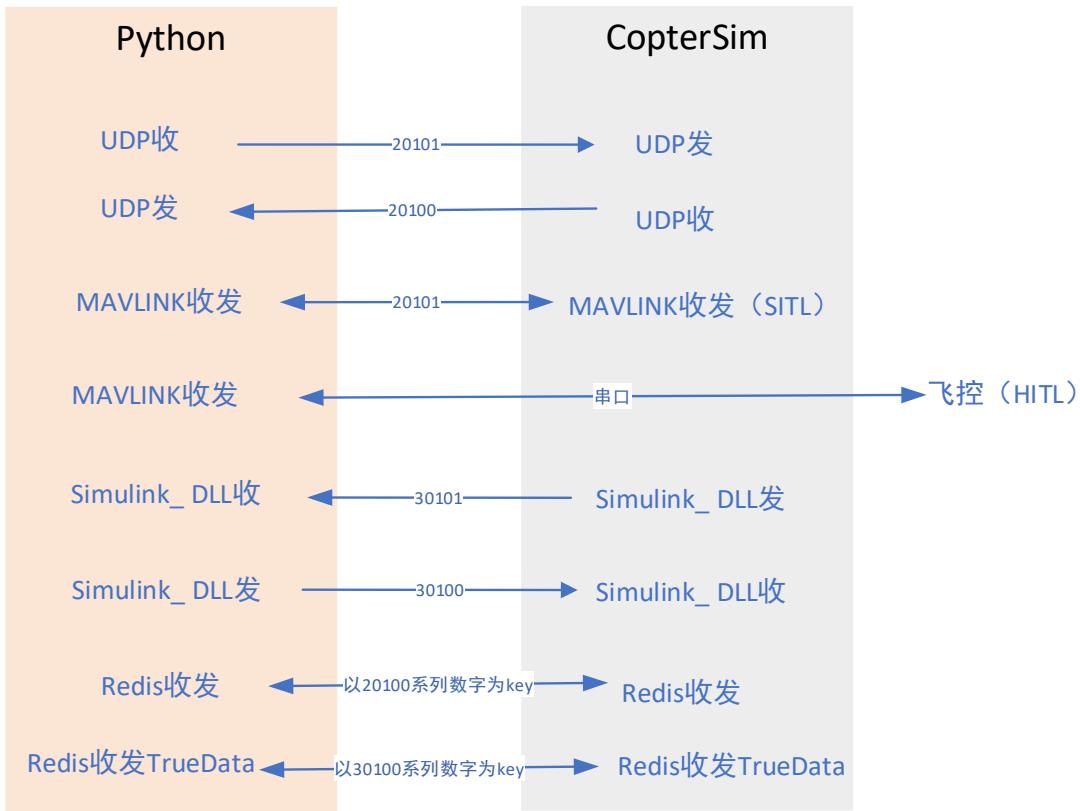
需要补充说明的是当使用 Simulink\_DLL 模式进行仿真的时候，数据通过 30100 系列端口进行发送，而且数据格式也与其它模式下的 UDP/Redis 有差异，所以在 receive\_full\_udp\_redis\_msg() 中会单独针对 Simulink\_DLL 模式进行处理。

如下图所示，是发送消息的接口。类似的，发送消息的入口函数是 send\_msg\_loop()，该函数也是一个 while 大循环，会在 Offboard 类初始化的时候通过一个单独的线程启动。由于 Simulink\_DLL 模式指令都放在 inSIL 中，与 (HI TL、SITL) 差别较大，所以 send\_msg\_loop() 会单独判断 Simulink\_DLL 模式并调用 send\_simulink\_dll()。发送消息也区分 MAVLINK/UDP\_Redis，在 send\_msg() 中分别调用 send\_position\_target\_mav() 和 send\_position\_target\_udp\_redis()。在 UDP 和 Redis 模式下，CopterSim 在接收到 Offboard 位置信息时会自动解锁并进入 Offboard 模式。除了 send\_cmd\_udp() 以外，其它几个 send 函数都只支持 MAVLINK 模式。

```
def send_position_target_mav(self):...
def send_position_target_udp_redis(self):...
def send_cmd_udp(self):...
def send_param_mav(self, param_id, param_value, param_type):...
def send_cmd_mav(self, cmd_long):...
def send_flight_mode(self, main_mode, sub_mode=0):...
def send_msg(self):...
def send_simulink_dll(self):
    self.send_sil_int_float(self.in_sil_ints, self.in_sil_floats)
def send_msg_loop(self, time_interval_s=0.01):...
def send_redis_data(self, key, buf):...
def send_sil_int_float(self, in_sil_ints, in_sil_floats):...
```

上文有提到接收消息和发送消息都会单独开一个线程。即控制一个飞机会有 3 个线程，即主线程、发送线程、接收线程。当有  $n$  个飞机时，线程总数为  $3n$  个。在 offboard.py 中通过 init\_connection\_loop() 初始化，该函数支持以模式作为参数，也就是说发送和接收消息进程的初始化会因连接模式 UDP/MAVLINK/Redis (Full\Simple) 的不同而不同。对于非 Simulink\_DLL 模式，使用 20100 系列端口进行通信。如下图所示，对于 CopterSim 来说，所用 20100 端口收消息，使用 20101 端口发消息。当增加飞机数量时，端口值逐步递增，如增加第 2 架飞机，其端口为 20102、20103。对于 Simulink\_DLL 模式，CopterSim 使

用 30100 端口接收信息，使用 30101 端口发送信息。类似的，当有多架飞机时，端口也从 30100 开始递增。对于 Redis 模式，使用 pub/sub 的方式进行通信，每一个飞机的收发都有一个 topic。沿用 UDP 通信一些编号的方式，将原来的端口作为 key，即 topic 的名称来进行 Redis 通信。



### 7.1.3. topics.py

topics.py 对 MAVLINK 消息包进行了封装，只包含了必要的姿态和位置信息，具体参见如下表。

pythonSDK 封装的 MAVLINK 消息

消息名称	描述
HeartBeat	心跳包，包含主模式、子模式、系统状态等信息。
Attitude	姿态，包含时间、滚转、俯仰、偏航及相应的角速度。
AttitudeTarget	期望姿态，包含时间、期望滚转、俯仰及相应的角速度
LocalPositionNED	位置，NED 坐标系下位置和速度。
PositionTargetLoca	期望位置，NED 坐标系下期望位置和速度。

1NED	
HomePosition	home 点位置，纬度、经度、高度等。
GlobalPositionInt	全局位置，经纬度表示成 INT 型，即在原来的基础上乘以 $10^7$ 转换为 INT，这样可以避免浮点数损失有效位数。

## 7.2. 用户 Api

### 7.2.1. ctrl

`__init__.py` 用于描述 `ctrl` 模块中可被外部访问的类。包括 `offboard.py` 中的“EarthFrame”大地坐标系转换类，“PID”控制器类，“Ctrl”外部控制接口类，“CtMode”连接模式类，“SimMode”仿真模式类，“Coordinate”坐标系类。

`Ctrl` 类是对 `Offboard` 类的封装，提供更加易用的用户接口。从实现上来说，`Ctrl` 是 `Offboard` 的子类。通过 `super` 调用将 `Ctrl` 类的参数传递给其父类 `Offboard`。如下表，是构建 `Ctrl` 实例的方法，默认情况下使用接口的默认参数即可。当用户设定自定义端口时，要小心端口冲突。

创建外部控制实例接口

<pre>Ctrl(port=20100,       ip='127.0.0.1',       redis_host='12       7.0.0.1', redis_port=637       9,       redis_pass=None,       connect_to_h       w=False,       sim_mode=       SimMode.SITL_RFLY)</pre>	<p><code>Ctrl</code> 构造函数用于创建外部控制实例，包含 7 个参数</p> <ul style="list-style-type: none"> <li><code>port</code>——基准端口，一般使用默认值即可，如果此处修改了端口值，一方面要确保 <code>CopterSim</code> 同步修改，另一方面避免与其它端口冲突。</li> <li><code>ip</code>——使用 UDP 模式进行通信时的 ip</li> <li><code>redis_host</code>——使用 Redis 进行通信时 server 端的 ip。</li> <li><code>redis_port</code>——Redis 的端口，当使用 Redis 模式且修改了 Redis 默认端口，这里要配套。</li> <li><code>redis_pass</code>——Redis 的密码</li> <li><code>connect_to_hw</code>——是否连接到硬件的标志，当做硬件在环和实飞时，该项为 <code>True</code>。</li> <li><code>sim_mode</code>——包含 HITH、SITL、SITL_RFLY、Simulink_DLL 等模式。</li> </ul> <p>示例：</p> <ol style="list-style-type: none"> <li>所有参数均使用默认参数  <code>ctrl = rflysim.Ctrl()</code></li> <li>进行多机综合模型仿真  <code>ctrl = rflysim.Ctrl(port=20100 + i * 2, sim</code></li> </ol>
--	--

---

_mode=SimMode.Simulink_DLL)
-----------------------------

初始化数据输入输出接口 `init_loop()`。该函数在一般的使用过程中都必须调用，而且参数的指定往往也是需要的。例如开发者在做集群仿真时会选择性能开销较小的 `UDP_Simple` 或者 `Redis_Simple`，而在做故障注入时会选择 `MAVLINK_FULL` 模式。

#### 接口 初始化收发数据线程

<code>init_loop(     ct_mode=CtMode.MAVLink_F     ull, offboard=False)</code>	用于初始化数据收发接口，启动相应线程，调用 <code>Ctrl</code> 类控制飞机总需要调用该接口。 参数： <code>ct_mode</code> ——表示连接模式 <code>offboard</code> ——是否进行 <code>offboard</code> 控制 示例： <code>ctrl.init_loop(self.ct_mode, offboard=True)</code>
---	--

起飞、返航、着陆指令接口，这些指令仅支持综合模型。对于综合模型的起飞、返航、着陆仅需调用下面的函数即可，下列的函数都支持指定 NED 坐标系下的高度。

#### 起飞、返航、着落指令接口

<code>takeoff(height=0)</code>	调用该函数可以让综合模型起飞，支持指定高度，该高度是 NED 坐标。旋翼机起飞后将悬停，固定翼起飞后将大致保持高度 <b>继续前飞</b> 。
<code>return_home(height=0)</code>	调用该函数可以控制综合模型进行返航，即水平位置返回到 <code>home</code> 点，支持设置返航高度(NED)。旋翼到达相应点后将悬停，固定翼将 <b>盘旋</b> 。
<code>land(height=0)</code>	调用该函数可以控制综合模型着陆，支持设置着陆高度(NED)。固定翼着落，在真实场景一般有跑道，而且地面支持力也是一个复杂的过程，目前仅能降落到相应的高度，因地面信息未知，可能出现钻入地下的情况。

---

期望位置发送接口。支持 NED 坐标系和 global 坐标系下发送位置。用户编程时，调用 send\_pos\_global() 传入的 pos 是浮点型的，后台程序会将该值乘以  $10^7$  并转换为整数发送给 CopterSim 或者直接通过串口发送出去。

#### 位置发送接口

send_pos_ned(pos, yaw)	发送 NED 坐标系下期望位置和偏航角
send_pos_global(pos, yaw)	发送 Global 坐标系下位置，接口支持浮点数设置，但发送时会自动转换为 INT 进行发送

期望速度发送接口。支持 NED 坐标系下和载体坐标系下期望速度和偏航角速率的设置。

send_vel_ned(vel, yaw_rate)	发送 NED 坐标系下期望速度和偏航角速率
send_vel_body(vel, yaw_rate)	发送载体坐标系下期望速度和偏航角速率

姿态发送接口。该接口目前仅支持固定翼综合模型。

send_att(att, thrust)	发送欧拉角和油门
-----------------------	----------

获取无人机的状态信息。获取无人机状态信息的接口都以 get 开头，主要是获取位置姿态等信息。在初始化消息收发接口后，下面的状态就会被更新。值得注意的是，在使用 Simple 模式进行通信时，Global 坐标系下的位置是没有的。

#### 获取无人机状态信息接口

get_pos_ned()	获取 NED 坐标系下的位置
get_pos_global()	获取 Global 坐标系下的位置
get_home_pos()	获取 home 点位置，Global 坐标
get_vel_ned()	获取 NED 坐标系速度
get_euler()	获取欧拉角

---

get_vehicle_id()	获取载具 ID
get_time_s()	获取仿真时间或飞控运行时间

SimMode 类。用户可以直接访问 SimMode 类的成员，以控制仿真模式。具体的仿真模式如下表所示，用户在进行仿真时需要设置正确的仿真模式。

SimMode 类接口

HITL	硬件在环
SITL	软件在环
SITL_RFLY	软件在环，进行了端口设置优化，支持开 100 架以上飞机
Simulink_DLL	综合模型仿真 示例： <code>rflysim.Ctrl(port=20100 + i * 2, sim_mode=</code> <b>SimMode.Simulink_DLL)</b>

CtMode 类。用户可以直接访问 CtMode 类的成员，以配置连接的模式。当前支持的主要模式包括 UDP\_Full、UDP\_Simple、MAVLink\_Full、MAVLink\_Simple、Redis\_Full、Redis\_Simple。MAVLink\_NoSend、MAVLink\_NoGPS 没有完整支持。

如果用户做单机实验，且需要获取较底层的数据，建议使用 MAVLINK\_Full 模式。如果做集群实验可以选择 UDP\_Simple/Redis\_Simple，Redis 仅企业定制版支持，Redis 相对 UDP 而言更加稳定、可靠、高效。用户如果在熟悉平台的基本例程，一般可以选择 UDP\_Full。使用举例：下面的 self.ct\_mode 可以是表格中的任意模式。`self.ctrl.init_loop(CtMode.UDP_Full, offboard=True)`

CtMode 类接口

UDP_Full	UDP 模式，完整控制指令
UDP_Simple	UDP 模式，简化控制指令
MAVLink_Full	MAVLink 模式，完整 MAVlink 协议
MAVLink_Simple	MAVLink 模式，简化 MAVlink 协议
MAVLink_NoSend	CopterSim 不转发所有 MAVLINK 消息
MAVLink_NoGPS	CopterSim 不转发 GPS 消息

---

Redis_Full	Redis 模式，通过 Redis 收发数据，与 UDP_Full 有相同的数据格式
Redis_Simple	Redis 模式，通过 Redis 收发数据，与 UDP_Simple 有相同的数据格式

Coordinate 类。支持局部 NED、Global 位置、载体坐标系下速度加速度等，具体协议与 MAVLINK 保持一致。

Coordinate 类接口

LOCAL_NED	NED 坐标系
GLOBAL_INT	经纬度乘以 $10^7$ 转换成 INT 传输
NED_BODY	位置在 NED 坐标系，速度加速度在 BODY 坐标系下

EarthFrame 类。EarthFrame 提供地理坐标系、地心地固系、站心坐标系之间的转换。地理坐标系(Latitude, Longitude, Altitude, ll) , 又称 global 坐标系：纬经高, 经纬度单位为°，高度单位为 m。地心地固系(Earth-Centered, Earth-Fixed, ecef)：原点(0, 0, 0)为地球质心, z 轴与地轴平行指向北极点, x 轴指向本初子午线与赤道的交点, y 轴垂直于 xoz 平面(即东经 90 度与赤道的交点)构成右手坐标系。站心坐标系：以测站为原点的坐标系。一般两种：东北天(East, North, Up, ENU), 北东地(North, East, Down, NED)PX4 选用 NED 坐标系，无人机一般取起点为测站，即原点。EarthFrame 可以实现上述 3 中坐标系之间的转换，在与站心坐标系进行转换时，需要知道站心的 global 位置，即纬经高。

EarthFrame 接口

ll2ecef(pos_global)	纬经高转换为以地球质心为原点的 x、y、z
enu2ecef(pos_global, pos_global_home)	东北天坐标转换为以地球质心为原点的 x、y、z
ecef2enu(pos_global, pos_global_home)	以地球质心为原点的 x、y、z 转换为东北天
ll2enu(pos_global, pos_global_home)	纬经高转换为东北天坐标
ll2ned(pos_global, pos_global_home)	纬经高转换为北东地坐标
ecef2lla(pos_global)	以地球质心为原点的 x、y、z 转换为经纬高
enu2lla(pos_enu, pos_global_home)	东北天转换为经纬高

ned211a(pos_ned, pos_global_home)	北东地转换为经纬高
-----------------------------------	-----------

PID 类。PID 类提供一个简单的控制器，在做 Offboard 控制时，可以使用 PID 控制器由期望位置计算得到期望速度。

PID(p=0.0, i=0.0, d=0.0)	PID 构造函数，可以指定 PID 参数，默认所有参数为 0
pid(err)	传入参数为状态误差，输出 PID 结果
reset()	重置积分

OffboardSimpleMode 类。该类是 RflySim 平台独有的精简模式，整个控制命令仅占用 20 字节的空间。该模式仅适用于高级开发者用户，做上层控制的用户无需关注该类。

OffboardSimpleMode 类接口

Vel_Yaw_Rate_NED	导航坐标系下速度模式[vx, vy, vz, yaw_rate]
Vel_Yaw_Rate_BODY	机体坐标系下速度模式[vx, vy, vz, yaw_rate]
Pos_Yaw_NED	导航坐标系下位置模式[x, y, z, yaw]
Pos_Yaw_BODY	机体坐标系下位置模式[x, y, z, yaw]
Att_Throttle	姿态油门控制指令[滚转、俯仰、偏航(弧度)、油门(0~1)]--可自动解锁，可自动进入 OffBoard 模式
Att_Throttle_Add	姿态油门增量控制指令[滚转、俯仰、偏航、油门增量]--可自动解锁，可自动进入 OffBoard 模式
Acc	加速度控制模式[ax, ay, az, --]
Acc_Yaw	加速度控制模式[ax, ay, az, yaw]
Acc_Yaw_Rate	加速度控制模式[ax, ay, az, yaw_rate]
Arm	解锁所示模式[解锁, -, -, -]
Speed_Radius_FW	表示设置固定翼飞机的速度和盘旋半径, [speed, radius, -, -]
Takeoff_NED	表示 Mavlink 起飞命令, 自动解锁, 导航坐标系位置[x, y, z, -]
Takeoff_Global	表示 Mavlink 起飞命令, 自动解锁, GPS 坐标系位置[纬度, 经度, 高度, -]
Speed_Height_Dir_Global	速度高度航向命令, 自动解锁并进入 offBoard 模式, GPS 坐标系位置[速度, 高度, 航向, -]
Pos_Yaw_Global_Int	Global 坐标系下位置模式, GPS 坐标系位置[lat_int, lon_int, alt_float, yaw_float]
VTOL_Switch	用于 VTOL 模式切换

RflySimCP 类。该类是综合模型的控制协议，仅适用于高级开发者用户。该类是 3.2.2 节协议的代码表示，当前并没有使用下表的所有标志，部分标志

---

是预留给后面更复杂的功能使用。

#### RflySimCP 类接口

ILen	inSILInts 的长度
ICmd	对应 inSILInts[0], 表示指令对应的下标
IOffboard	对应 inSILInts[1], 表示 Offboard 模式对应下标
ILat	对应 inSILInts[6], 支持全局坐标时存放纬度
ILon	对应 inSILInts[7], 支持全局坐标时存放经度
CmdEn	inSILInts[0]第 0 位使能, 代表有指令, 预留
CmdSIL	inSILInts[0]第 1 位使能, 代表有进入仿真模式
CmdArmed	inSILInts[0]第 2 位使能, 代表解锁
CmdTakeoff	inSILInts[0]第 8 位使能, 代表起飞
CmdPosition	inSILInts[0]第 9 位使能, 代表定点
CmdLand	inSILInts[0]第 10 位使能, 代表着陆
CmdReturn	inSILInts[0]第 11 位使能, 代表返航
CmdOffboard_Pos	inSILInts[0]第 16 位使能, 代表有 Offboard 位置, 预留
CmdOffboard_Att	inSILInts[0]第 17 位使能, 代表有 Offboard 姿态, 预留
CmdBase	CmdEn+CmdSIL, 发送命令时需要带上该头
HasPos	inSILInts[1]第 0 位使能, 代表有位置
HasVel	inSILInts[1]第 1 位使能, 代表有速度
HasAcc	inSILInts[1]第 2 位使能, 代表有加速度
HasYaw	inSILInts[1]第 3 位使能, 代表有偏航角
HasYawRate	inSILInts[1]第 4 位使能, 代表有偏航角速率
HasAtt	inSILInts[1]第 8 位使能, 代表有姿态
HasRollRate	inSILInts[1]第 9 位使能, 代表有滚转角速率
HasPitchRate	inSILInts[1]第 10 位使能, 代表有俯仰角速率
HasThrust	inSILInts[1]第 11 位使能, 代表有油门
NED	inSILInts[1]第 16 位使能, 代表在 NED 坐标系下, 预留
Global	inSILInts[1]第 17 位使能, 代表在 Global 坐标系下, 预留
FLen	inSILFloats 长度
FPos	位置起始下标
FVel	速度起始下标
FAcc	加速度起始坐标
FAtt	姿态起始坐标
FAttRate	角速率起始坐标
FThrust	油门坐标

### 7.2.2. test

test 文件夹用于存放用例。这些用例一方面用于看护 pythonSDK 的功能, 另一方面教用户如何使用平台 api 进行二次开发。

---

### 7.3. test\_ctrl.py

该文件主要对外部控制的接口进行测试。下面介绍每个测试用例的关键逻辑和使用方法。

TestTakeoff 类。该类是使用 rflysim pythonSDK 最简单的例程，发送 NED 坐标系下的位置实现起飞。该类的构造函数支持配置连接模式，包括 UDP/MAVLink/Redis (Full+Simple) 总共 6 中模式。TestTakeoff 的构造函数定义了一个 Ctrl() 实例 self.ctrl，run() 函数调用 self.ctrl.init\_loop() 初始化了数据收发线程，再调用 self.ctrl.send\_pos\_ned() 发送期望的位置实现起飞。等待到达起飞高度，程序退出。

TestTakeoffVel 类。通过速度控制使得无人机起飞并到达指定高度。为了更加准确的通过速度控制到达指定位置，用例中使用了 PID 控制器，PID 控制器的输入是高度误差，而输出是高度方向的期望速度。在该用例中，通过 loop 函数实现了期望速度的生成，并通过 self.ctrl.send\_vel\_ned() 实现发送。

TestTracking 类。该类使用位置控制实现了“8 字”形追踪，该例程展现了如何通过 self.ctrl.send\_pos\_ned() 实现更为复杂的轨迹控制。其核心函数是 tracking\_8\_shape()，该函数用角度表示无人机在“8 字”上的位置，相当于在极坐标系下进行控制。当无人机到达一个位置时，相应的极坐标角度增加一个间隔。但是控制接口只支持笛卡尔坐标，所以将极坐标转换为笛卡尔坐标后发送给飞机。

TestGetPos 类。获取不同坐标系下的当前位置和 home 点。该类型同样支持指定 6 种模式，但是在 simple 模式下没有 global 位置，故 simple 模式下 global 位置为 0。

TestPosFrame 类。对比 NED 坐标系和 Global 坐标系下位置控制。在大范围仿真时，往往需要使用 Global 位置设定目标点。TestPosFrame 是一个综合使用 NED 坐标系和 Global 坐标系的例程。run() 函数中首先在 NED 坐标系下实现起飞，并等待到达指定高度。到达指定高度后，水平位置 x+10，高度+5。通过 EarthFrame 实例，将 NED 坐标系下的位置转换为 Global 位置。然后，调用 send\_pos\_global() 发送给飞机。最后使用 NED 坐标系下位置判断是否到达指定位置，如果能够到达不仅证明 send\_pos\_global() 成功发送，而且证明 EarthFrame 的

---

转换也是正确的。

TestVelFrame 类。对比 NED 坐标系下速度指令和 body 坐标系下速度指令的不同。该类可以视为 TestTakeoffVel 的一个扩展。关键修改点在于，程序会根据坐标系指定的不同选择 send\_vel\_ned 还是 send\_vel\_body。在没有做偏航运动时，载体坐标和 NED 坐标系是重合的，所以需要做一个偏航运动才能看出在两种坐标系下发送相同的数据带来的不一样的现象。具体而言，当在 NED 的 x 方向指定一个速度时将向北飞行，而在 body 坐标系下指定 x 方向速度将超机头方向前进。

TestSynModel 类。该类是对综合模型的一个系统的测试用例，支持指定飞机数量和模式。每一架飞机的控制会单独使用一个线程，每个线程需要传入飞机实例和飞机的初始位置。传入的飞机初始位置和 CopterSim 的初始位置需要保持一致，避免起飞时就出现水平方向的运动。例程中通过 init\_poses() 实现了初始位置的计算。loop() 是每一架飞机执行的动作，执行这些特定的动作是为了尽可能多的覆盖综合模型的接口。

TestFWSynModel 类。这是测试固定翼综合模型的类，支持指定飞机数量、模式，是否使能姿态控制。该测试用例在 att=True 时，将用直接采用姿态模式控制起飞、平飞、转弯、盘旋等。否则将使用 takeoff()、send\_pos\_ned()、return\_home()、land() 等上层函数进行轨迹的控制。

## 8. QGC 二次开发

主要介绍基于 Windows 环境下，使用 Qt Creator 5.15.2 + Visual Studio 2019 工具，对 QGroundControl 进行二次开发，实现部分定制功能。包括 QGC 二次开发环境准备及搭建、模块介绍、新增功能案例介绍。

### 8.1. 开发环境准备及搭建

在抓取 QGC 源码前，需要配置好 GitHub，确保本地能从 GitHub 拉取 code。

参考 QGC 官网 <https://github.com/mavlink/qgroundcontrol> 抓取 code，包括子模块，通过 gitcode 抓取完成后。需要参考 .gitmodules 文件，确认子模块是否抓取成功，例如检查 qgroundcontrol 目录下是否存在 ./src/GPS/Drivers/src 源码。如果不存在，请单独抓取相关源码，放在对应目录，其他子目录使用相同方法处理。如果不抓取相关子目录源码，将导致编译失败。另外请注意 Master 分支和 Tag 分支对应的子模块不共用，使用不同分支 Code，需要抓取对应的子模块。

使用指令无法抓取源码（GitHub 配置失败），可以使用在 Code->Local->Download ZIP

---

下载压缩包，参考`.gitmodules`文件，确认子模块是否抓取成功。建议抓取最新 Tag 源码，相对 Master 版本更稳定。确定源码及子模块下载完成后，下载 IDE 工具。

参考 QGC 官网 <https://github.com/mavlink/qgroundcontrol>，分别安装 Visual Studio 2019 和 Qt 5.15.2，注意需要先安装 Visual Studio 2019，再安装 Qt。安装 Qt 5.15.2 时，编译工具选项 MVSC\_2019 64-bit，编译工具安装完成后，并确定能正常使用后。打开 Qt Creator 程序，通过 Qt Creator->文件->打开文件或项目->选择 qgroundcontrol 目录下的 `qgroundcontrol.pro` 完成项目的加载。如果编译失败，请检查 Qt 构建套件是否为 Qt5.15.2+MVSC\_2019 64-bit，如果环境变量中有配置多个 Qt 版本，请将 Qt5.15.2 配置，放在其他版本前面。

## 8.2. 模块介绍

QGC 旨在提供可跨多个操作系统平台以及多种设备尺寸和样式运行的单代码库。

QGC 用户界面是使用 Qt QML 实现的。QML 提供硬件加速，这是平板电脑或手机等低功耗设备的一项关键功能。QML 还提供了一些功能，使我们能够更容易地创建单个用户界面，该界面可以适应不同的屏幕尺寸和分辨率。

QGC UI 的目标更多的是平板电脑+触摸风格的 UI，而不是基于桌面鼠标的 UI。这使得单个 UI 更容易创建，因为平板电脑风格的 UI 也往往可以在台式机/笔记本电脑上正常工作。

本章主要解释 QGC 的工作原理，

### 8.2.1 通讯流程

QGC 与设备之间通讯主要通过 MavLink 相关协议进行，关于 MavLink 详细介绍见本章第五节。QGC 主要通过 LinkManager、MAVLinkProtocol、MissionManager 相关类处理 MavLink 消息。

LinkManager 始终打开 UDP 端口等待设备发送的心跳包，检测到设备（Pixhawk、SiK Radio、PX4 Flow）与计算机建立 UDP 连接后，会创建新的 SerialLink。然后来自 Link 的传入字节被发送到 MAVLinkProtocol 类的处理函数，将字节转换为 MAVLink 消息。再对消息进行分析后，如果消息是 HEARTBEAT，就会让 MultiVehicleManager 类就会收到通知，并根据 HEARTBEAT 消息中的信息创建新的车辆对象，车辆对象实例化与车辆匹配的插件。与车辆对象关联的 ParameterLoader PARAM\_REQUEST\_LIST 向连接的设备发送一个消息，以使用参数协议加载参数。参数加载完成后，与车辆对象关联的 MissionManager 使用任务协议从连接的设备请求任务项目，参数加载完成后，VehicleComponents 将在“设置”视图中显示其 UI

### 8.2.2 插件架构

尽管 MAVLink 规范定义了与车辆通信的标准通信协议。该规范的许多方面都需要固件开发人员来解释。因此，在许多情况下，为了完成相同的任务，与运行一种固件的车辆的

---

通信与与运行不同固件的车辆的通信略有不同。此外，每个固件可以实现 MAVLink 命令集的不同子集。

另一个主要问题是 MAVLink 规范不涵盖车辆配置或通用参数集。因此，与车辆设置相关的所有代码最终都是特定于固件的。此外，任何必须引用特定参数的代码也是特定于固件的。

考虑到固件实现之间的所有这些差异，创建一个可以支持每个地面站应用程序的单个地面站应用程序可能非常棘手，同时又不会使代码库退化为大量基于车辆所使用的固件的 if/then/else 语句。

QGC 使用插件架构将固件特定代码与所有固件通用的代码隔离开来，以允许超出标准 QGC 所能提供的进一步自定义。目前主要通过 FirmwarePlugin、FirmwarePlugin 偶个类，管理拓展的插件功能。

FirmwarePlugin 类通常用于为未标准化的 Mavlink 部分创建标准接口

AutoPilotPlugin 类用于拓展提供车辆设置的用户界面

QGCCorePlugin 用于通过标准接口公开 QGC 应用程序本身与车辆无关的功能。然后，自定义构建使用它来调整 QGC 功能集以满足他们的需求。

### 8.2.3 重要类介绍

LinkManager, LinkInterface 类是处理与设备的特定类型的通信管道。例如串行端口或通过 WiFi 的 UDP。所有链接的基类是 LinkInterface。每个链接都在其自己的线程上运行，并向 MAVLinkProtocol 发送字节。该 LinkManager 对象跟踪系统中所有打开的链接。LinkManager 还通过串行和 UDP 链路管理自动连接。

MAVLinkProtocol 类是从链接获取传入字节并将其转换为 MAVLink 消息。MAVLink HEARTBEAT 消息被路由到 MultiVehicleManager。所有 MAVLink 消息都会路由到与该链接关联的车辆。

MultiVehicleManager 类只在系统内创建一个对象。当它在之前未见过的链接上收到 HEARTBEAT 时，它会创建一个 Vehicle 对象。MultiVehicleManager 还跟踪系统中的所有车辆，并处理从一辆活动车辆到另一辆的切换，并正确处理被移除的车辆。

Vehicle 类是实现了与物理设备通信的主要接口，注意：还有一个与每个 Vehicle 关联的 UAS 对象，该对象是一个已弃用的类，并且正在慢慢被淘汰，所有功能都转移到 Vehicle 类。此处不应添加新代码。

FirmwarePlugin,类是固件插件的基类。固件插件包含固件特定代码，因此车辆对象在支持 UI 的单一标准接口方面是干净的。

FirmwarePluginManager 是一个工厂类，它基于车辆的 MAV\_AUTOPILOT/MAV\_TYPE 组合创建 FirmwarePlugin 实例。

---

## 8.2.4 用户界面主要组件

QGC 中 UI 设计的主要模式是用 QML 编写的 UI 页面，很多时候与用 C++ 编写的自定义“控制器”进行通信。这遵循 MVC 设计模式的某种修改变体。

QML 代码通过以下几种机制绑定到与系统相关的信息：(1) 自定义控制器 (2) 使用 QGroundControl 提供对活动车辆等事物的访问的全局对象 (3) 使用 FactSystem 提供对参数的访问，在某些情况下还提供对自定义事实的访问。注意：由于 QGC 中使用的 QML 的复杂性以及它依赖于与 C++ 对象的通信来驱动 ui，因此不可能使用 Qt 提供的 QML Designer 来编辑 QML。

QGC 没有针对不同屏幕尺寸和/或外形尺寸的不同编码 UI。一般来说，它使用 QML 布局功能来重排一组 QML UI 代码以适应不同的外形尺寸。在某些情况下，它会在小屏幕尺寸上提供较少的细节以使其适合。但这是一个简单的可见性模式。使用 FactSystem 系统，用于管理系统内所有单独的数据。然后将该数据模型连接到控件。QGC UI 是从一组基本的可重用控件和 UI 元素开发的。这样，添加到可重用控件的任何新功能现在都可以在整个 UI 中使用。这些可重用控件还连接到 FactSystem Facts，然后 FactSystem Facts 自动提供适当的 UI。

QGC 有一套标准的字体和调色板，通过引用以下俩个模块进行使用。

```
import QGroundControl.Palette      1.0
import QGroundControl.ScreenTools  1.0
```

QGC 软件主题有俩种风格设计：浅色和深色。浅色调色板适合室外使用，深色调色板适合室内使用。通常，您不应该直接为 UI 指定颜色，而应该始终使用调色板中的一种颜色。如果您不遵循此规则，您创建的用户界面将无法从浅色/深色样式更改。

**QGCMAPalette** 用于在地图上绘制的颜色。由于地图样式不同，特别是卫星地图和街道地图，您需要使用不同的颜色才能清晰地绘制它们。卫星地图需要较浅的颜色才能看到，而街道地图需要较暗的颜色才能看到。该 **QGCMAPalette** 项目为此提供了一组颜色以及在地图上的浅色和深色之间切换的能力。

**ScreenTools** 项提供可用于指定字体大小的值。它还提供有关屏幕尺寸以及 QGC 是否在移动设备上运行的信息。

以下控件是标准 Qt QML 控件的 QGC 变体。它们提供与相应 Qt 控件相同的功能，只不过它们是使用 QGC 调色板绘制的。

- **QGCButton**
- **QGCCheckBox**
- **QGCColoredImage**
- **QGCCComboBox**
- **QGCFlickable**
- **QGCLabel**

- 
- QGCMovableItem
  - QGCRadioButton
  - QGCSlider
  - QGCTextField

以下定义控件是 QGC 独有的，用于创建标准 UI 元素。

- DropButton - 单击时会弹出选项面板的 RoundButton。示例是平面图中的“同步”按钮。
- ExclusiveGroupItem - 用作支持 QML ExclusiveGroup 概念的自定义控件的基础项。
- QGCView - 系统中所有顶级视图的基本控制。提供对 FactPanels 的支持并显示 QGCViewDialogs 和 QGCViewMessages。
- QGCViewDialog - 从 QGCView 右侧弹出的对话框。您可以指定对话框的接受/拒绝按钮以及对话框内容。示例用法是当您单击参数时，它会弹出值编辑器对话框。
- QGCViewMessage - QGCViewDialog 的简化版本，允许您指定按钮和简单的文本消息。
- QGCViewPanel - QGCView 内的主要视图内容。
- RoundButton - 使用图像作为其内部内容的圆形按钮控件。
- SetupPage - 所有设置车辆组件页面的基本控件。提供标题、描述和组件页面内容区域

## 8.2.5 事实系统

Fact System 提供了一组功能，可标准化并简化 QGC 用户界面的创建。主要由以下几个类实行：

Fact 类实现系统内的单个值。

FactMetaData 类，实现每个事实都有关联。它提供有关事实的详细信息，以驱动自动用户界面生成和验证

Fact Controls 是事实控件是连接到事实的 QML 用户界面控件，它为 FactMetaData 用户提供一个控件来修改/显示与事实关联的值。

FactGroup 类实现一组事实。它用于组织事实并管理用户定义的事实。

可以通过重写自定义固件插件类中 factGroups 的函数来添加用户定义的事实。FirmwarePlugin 这些函数返回事实组映射的名称，用于标识添加的事实组。可以通过扩展类来添加自定义事实组 FactGroup。可以 FactGroup 通过提供包含必要信息的 json 文件，使用适当的构造函数来定义 FactMetadatas。

adjustMetaData 通过重写类也可以更改现有事实的元数据 FirmwarePlugin。

可以使用或 factGroups 访问与车辆相关的事 实（包括属于车辆固件插件功能返回的事 实组的事 实）getFact("factName")getFact("factGroupName.factName")，有关更多信息，请参

阅 FirmwarePlugin.h 中的注释。

## 8.2.6 主要视图

AppSettings.qml 绘制了 Application Settings 页面，每个按钮加载一个单独的 QML 页面  
在 SetupView.qml 绘制了 Vehicle Setup 页面，固定按钮/页面集：摘要、固件，其余按钮/页面来自 AutoPilotPlugin VehicleComponent 列表。

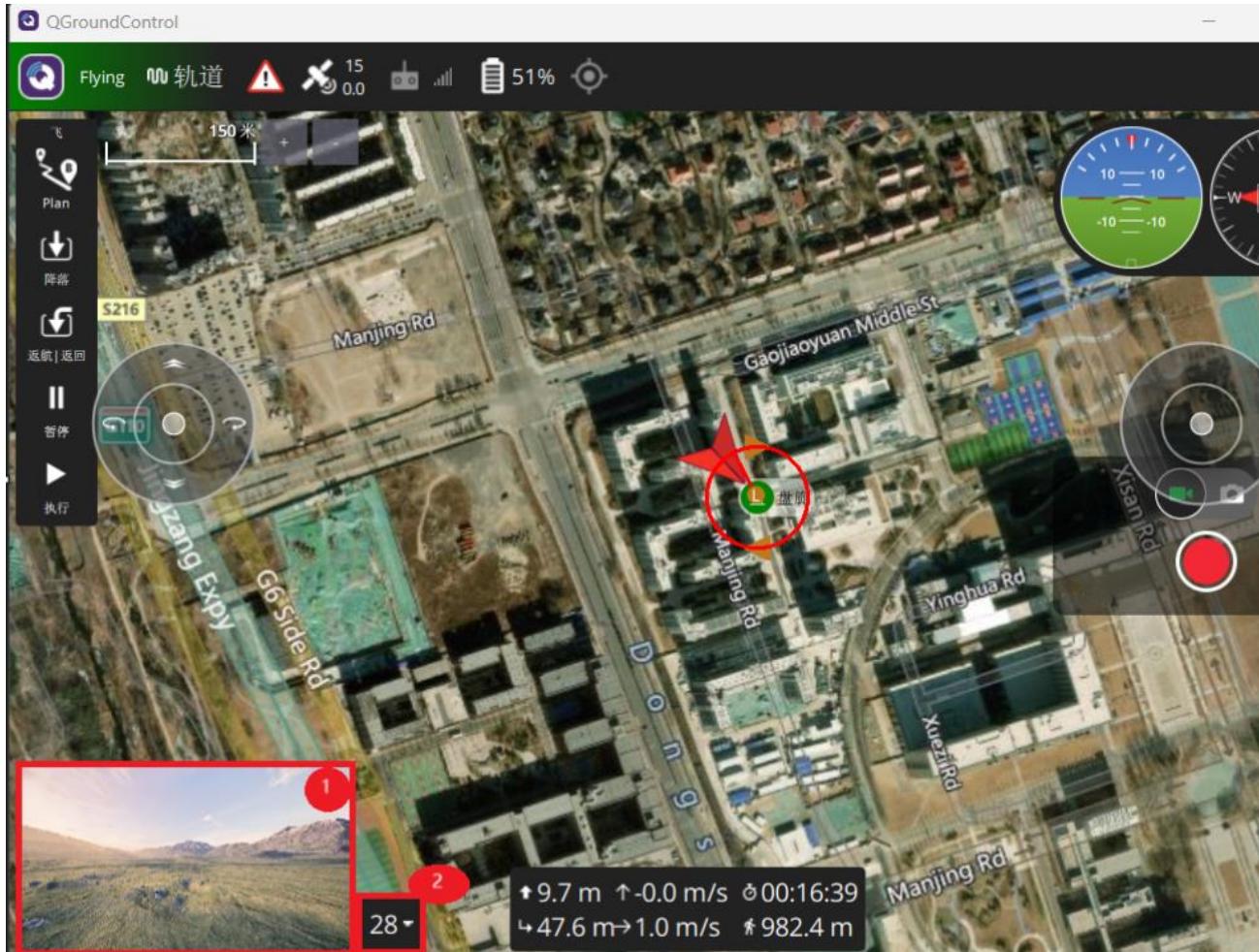
PlanView.qml 中主视觉 UI 是 FlightMap 控件 QML 与 MissionController (C++) 进行通信，MissionController 为视图提供任务项数据和方法，用于从 json 元数据层次结构动态编辑特定任务项命令。这个层次结构称为任务命令树。这样，添加新命令时只需创建 json 元数据。

FlightDisplayView.qml 中，QML 代码与 (C++) 通信 MissionController 以进行任务显示，仪器小部件与活动车辆对象通信，两个主要的内部视图是：FlightDisplayViewMap、FlightDisplayViewVideo

## 8.3. 新增功能案例

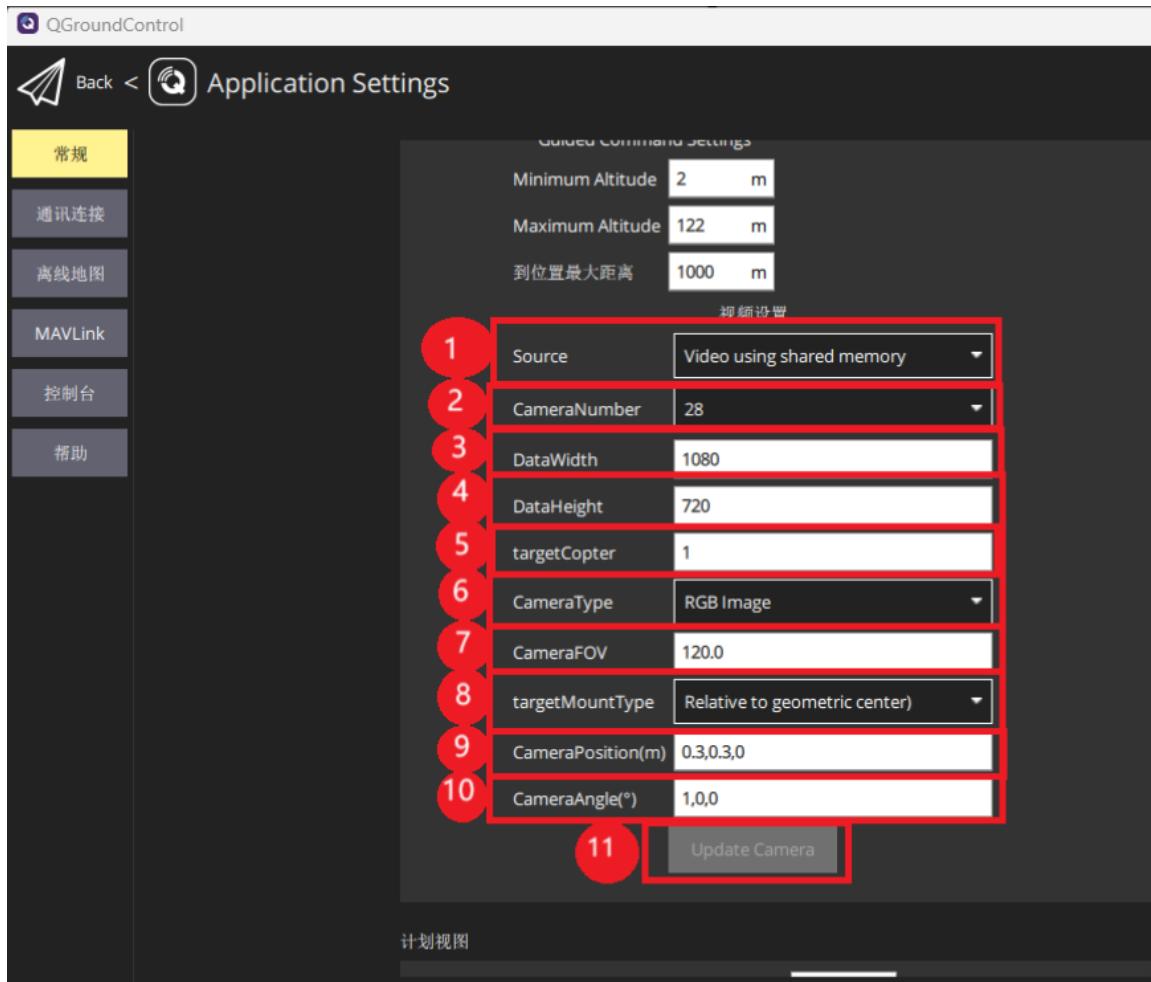
基于 RflySim 平台，增加从共享内存中读取图像显示在 QGC 中，以及创建和设置 RflySim3D 相机功能。

增加从共享内存中读取图像显示功能后的 UI 界面：



- ① 共享内存图像显示窗口，可以像 QGC 显示相机数据一样，对窗口进行缩放及切换。图像显示 UI 修改可见 FlyViewVideo.qml 中添加的 FlightDisplayViewSharedMemory.qml 文件，通过增加相关 QML 文件的 Object Name，确保在 QGCCorePlugin.cc 能获取到指定的 QML 对象，然后将共享内存数据绘制在 QML 上。
- ② RflySim3D 相机 ID 切换，通过下拉框选择对应相机 ID，加载的相机配置就会不一样。相机 ID 切换修改可见 FlyView.qml 中添加的 FactComboBox 组件，基于 QGC 的事实系统，添加 cameraID 函数创建对象，绑定 VideoManager 类中的 \_cameraIDChanged 参数，将 UI 值的修改通过 UDP 发送给 RflySim3D，实现相机 ID 的切换。

在 QGC 中增加设置 RflySim3D 中相机的 UI 界面：



① 视屏显示来源切换，通过下拉框选择对视屏显示使用共享内存数据，会将共享内存获取数据显示在视频显示界面。添加相关接口的修改见 GeneralSettings.qml 中 videoSource 相关修改。

② RflySim3D 相机 ID 切换，通过下拉框选择对应相机 ID。相机 ID 切换修改可见 GeneralSettings.qml 中添加 id 为 cameraNumberLabel 的组件，基于 QGC 的事实系统，添加 cameraID 函数创建对象，绑定 VideoManager 类中的 \_cameraIDChanged 函数，将 UI 值的修改通过 UDP 发送给 RflySim3D，实现相机 ID 的切换。

---

③、④RflySim3D 相机像素切换，修改相关值，可以设置获取图像的大相机 ID 切换修改可见 GeneralSettings.qml 中添加 id 为 pixel\* 的组件，基于 QGC 的事实系统，添加 dataWidth 函数创建对象，绑定 VideoManager 类中的 \_dataWidthChanged 函数，将 UI 值的修改通过 UDP 发送给 RflySim3D，实现相机参数的切换。

⑤相机绑定的飞机 ID 编号。

⑥相机图像类型设置，可将图像设置为 RGB 图像、深度图像、灰度图像等格式。

⑦相机视场角设置

⑧相机绑定类型设置

⑨相机绑定位置设置

⑩相机安装角度设置

⑪相机参数更新设置

以上设置相关 UI 添加在 GeneralSettings.qml 文件中，基于 QGC 的事实系统，在 VideoSettings.cc 中创建 Fact 对象，然后绑定 VideoManager 中的槽函数，C++ 相关类及可响应 QML 中 UI 修改的值，并将修改的值保存在 VisionSensorReq 结构提供。最后通过点击 Update Camera 按钮，调用 VideoManager::clickCreateBtn() 槽函数，将 VisionSensorReq 结构体通过 UDP 广播给 RflySim3D 程序，实现相机的更新