
API 说明文件检索大纲

| | |
|--|----|
| 1、 运动建模接口总体介绍 | 1 |
| 2、 环境配置 | 1 |
| 2.1 VISUAL STUDIO 安装 | 1 |
| 2.2 MATLAB 编译环境配置 | 1 |
| 2.3 SIMULINK 代码生成设置 | 2 |
| 2.4 LINUX 版环境配置 | 5 |
| 3、 基本操作流程 | 5 |
| 3.1 载具 SIMULINK 模型开发 | 5 |
| 3.2 模型编译生成 C/C++ 文件 | 6 |
| 3.3 DLL/SO 模型生成 | 7 |
| 3.4 PX4 混控规则介绍 | 8 |
| 3.5 BAT 脚本相关参数修改 | 17 |
| 3.5.1 硬件在环仿真脚本 | 17 |
| 3.5.2 软件在环仿真脚本 | 17 |
| 3.6 SITL 仿真 | 18 |
| 3.7 HITL 仿真 | 19 |
| 3.8 通过 QGC 或遥控器进行仿真测试 | 22 |
| 3.9 外部接口通信调试 | 22 |
| 4、 DLL 文件生成脚本-GENERATEMODELDLLFILE.P | 22 |
| 5、 DLL/SO 模型与通信接口 | 22 |
| 5.1 总体介绍 | 22 |
| 5.2 重要参数 | 23 |
| 5.2.1 ModelInit_PosE | 23 |
| 5.2.2 ModelInit_AngEuler | 24 |
| 5.2.3 ModelInit_Inputs | 24 |
| 5.2.4 ModelParam_uavtype | 25 |
| 5.2.5 ModelParam_GPSLatLong | 25 |
| 5.2.6 ModelParam_envAlitude | 26 |
| 5.3 数据协议 | 26 |
| 5.3.1 飞控仿真输入接口 | 26 |
| 5.3.1.1 inPWMs (电机控制量输入) | 26 |

| | |
|--|----|
| 5.3.1.2 inCopterData (飞控状态量输入) | 27 |
| 5.3.2 飞控仿真输出接口 | 27 |
| 5.3.2.1 MavHILSensor (传感器接集合) | 27 |
| 5.3.2.2 MavHILGPS (GPS 接口) | 27 |
| 5.3.3 仿真数据输出接口 | 28 |
| 5.3.3.1 MavVehile3Dinfo (真实仿真数据输出) | 28 |
| 5.3.3.2 outCopterData (自定义日志输出) | 28 |
| 5.3.3.4 ExtToUE4 (自定义显示数据输出) | 28 |
| 5.3.4 自动代码生成控制器通信接口 | 28 |
| 5.3.4.1 ExtToPX4 (自定义 uORB 数据输出) | 28 |
| 5.3.4.2 inCopterData (uORB 数据输入) | 28 |
| 5.3.5 碰撞数据接收接口—inFloatsCollision..... | 28 |
| 5.3.6 外部数据传入接口 | 28 |
| 5.3.6.1 inSILInts (整型数据输入) | 28 |
| 5.3.6.2 inSILFloats (浮点型数据输入) | 28 |
| 5.3.6.3 inFromUE (RflySim3D 数据输入) | 28 |
| 5.3.7 实时参数修改接口—FaultParamsAPI..... | 29 |
| 5.4 通信接口 | 29 |
| 5.4.1 20100++2 系列端口 | 29 |
| 5.4.2 20101++2 系列端口 | 29 |
| 5.4.3 30100++2 系列端口 | 29 |
| 5.4.4 30101++2 系列端口 | 29 |
| 5.4.5 TCP 端口 | 29 |
| 5.4.6 飞控 USB 串口 | 29 |
| 6、SIMULINK 建模模板介绍..... | 30 |
| 6.1 Motor Model 电机模块..... | 30 |
| 6.2 Force and Moment Model 力和力矩模块..... | 30 |
| 6.3 PhysicalCollisionModel 物理碰撞模块..... | 31 |
| 6.4 GroundSupportModel 地面支撑模块..... | 31 |
| 6.5 6DOF 刚体运动学模块 | 32 |
| 6.6 SensorOutput 传感器输出模块..... | 33 |
| 6.7 3DOutput 三维显示模块 | 33 |
| 6.8 Gazebo 模型模块..... | 34 |
| 6.8.1 ESC_ALL 模块 | 35 |
| 6.8.2 ESC 模块 | 36 |

| | |
|----------------------------|----|
| 6.8.3 Motor_ALL 模块 | 37 |
| 6.8.4 Motor 模块..... | 38 |
| 6.8.5 LiftDrag_ALL 模块..... | 38 |
| 6.8.6 LiftDrag 模块 | 39 |
| 7、RFLYSIM 已支持载具仿真操作介绍..... | 40 |
| 7.1 四旋翼模型..... | 40 |
| 7.2 六旋翼..... | 40 |
| 7.3 四轴八旋翼..... | 40 |
| 7.4 小型固定翼..... | 40 |
| 7.5 垂直起降无人机..... | 40 |
| 7.5.1 4+1 垂起..... | 40 |
| 7.5.2 四旋翼尾座式垂起..... | 40 |
| 7.6 无人车..... | 40 |
| 7.6.1 阿卡曼底盘无人车..... | 40 |
| 7.6.2 差动无人车 | 40 |
| 7.7 无人船..... | 40 |
| 7.8 直升机..... | 41 |
| 8、外部控制接口..... | 41 |
| 8.1 QGC..... | 41 |
| 8.2 Simulink 控制接口..... | 41 |
| 8.3 Python 控制接口..... | 41 |
| 9、参考资料..... | 43 |

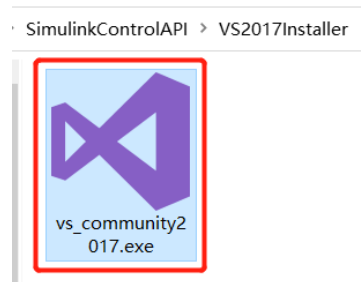
1、运动建模接口总体介绍

2、环境配置

2.1 Visual Studio 安装

RflySim 模型开发需要用到 Visual Studio 编译器，例如 MATLAB S-Function Builder 模块的使用、Simulink 自动生成 C/C++模型代码等。此处推荐安装 Visual Studio 2017，在线安装步骤（需联网）如下：

Step 1: 双击运行 “RflySimAPIs3.0\4.RflySimModel\1.BasicExps\VS2017Installer\vs_community2017.exe”；



Step 2: 选择 “使用 C++的桌面开发”，点击右下角 “安装”，等待安装完成即可。



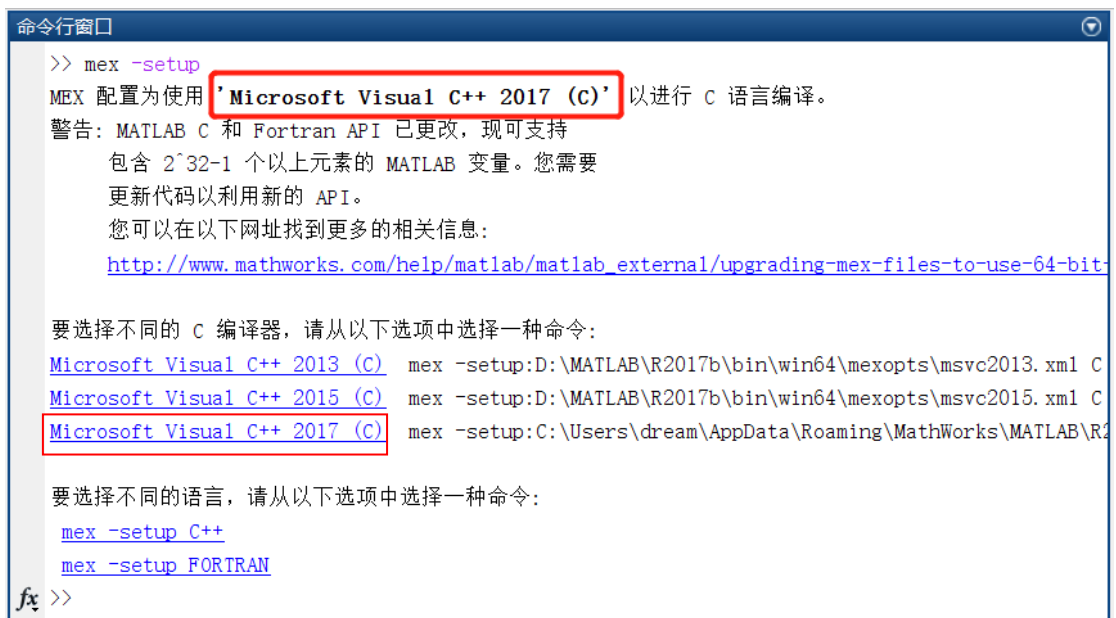
注意：1、高版本 MATLAB 也可安装 VS2019，但是 MATLAB 只能识别到低于自己版本的 Visual Studio，因此 MATLAB2017b 无法识别 VS 2019。

2、请不要更改 VS 默认安装目录（例如装到 D 盘），会导致 MATLAB 无法识别。

2.2 MATLAB 编译环境配置

在 MATLAB 的命令行窗口中输入指令 “mex -setup”，一般来说会自动识别并安装上 VS 2017 编译器，如右图所示显示 “MEX 配置使用 ‘Microsoft Visual C++ 2017’ 以进行编译”

说明安装正确。



```
>> mex -setup
MEX 配置为使用 'Microsoft Visual C++ 2017 (C)' 以进行 C 语言编译。
警告: MATLAB C 和 Fortran API 已更改, 现可支持
包含 2^32-1 个以上元素的 MATLAB 变量。您需要
更新代码以利用新的 API。
您可以在以下网址找到更多的相关信息:
http://www.mathworks.com/help/matlab/matlab\_external/upgrading-mex-files-to-use-64-bit

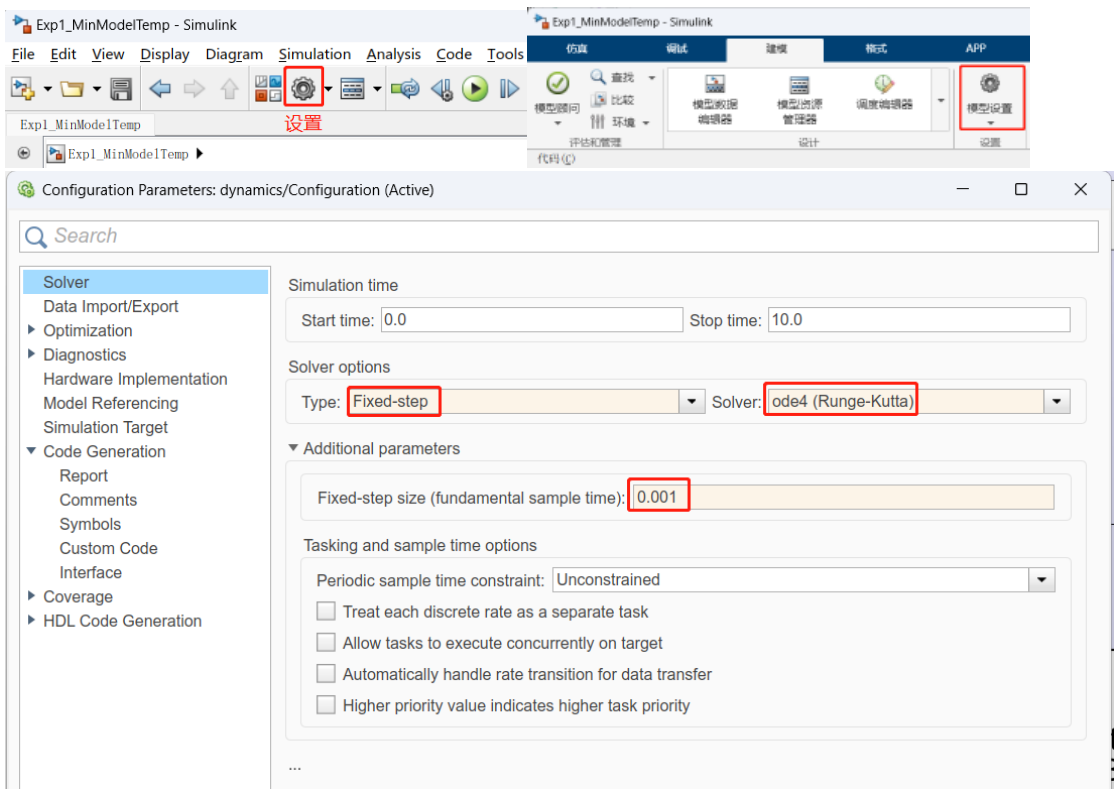
要选择不同的 C 编译器, 请从以下选项中选择一种命令:
Microsoft Visual C++ 2013 (C) mex -setup:D:\MATLAB\R2017b\bin\win64\mexopts\msvc2013.xml C
Microsoft Visual C++ 2015 (C) mex -setup:D:\MATLAB\R2017b\bin\win64\mexopts\msvc2015.xml C
Microsoft Visual C++ 2017 (C) mex -setup:C:\Users\dream\AppData\Roaming\MathWorks\MATLAB\R2
要选择不同的语言, 请从以下选项中选择一种命令:
mex -setup C++
mex -setup FORTRAN
fx >>
```

2.3 Simulink 代码生成设置

在使用 RflySim 平台进行载具模型的软硬件在环仿真前, 需要完成模型 Simulink 源程序的编译及 DLL 文件生成。

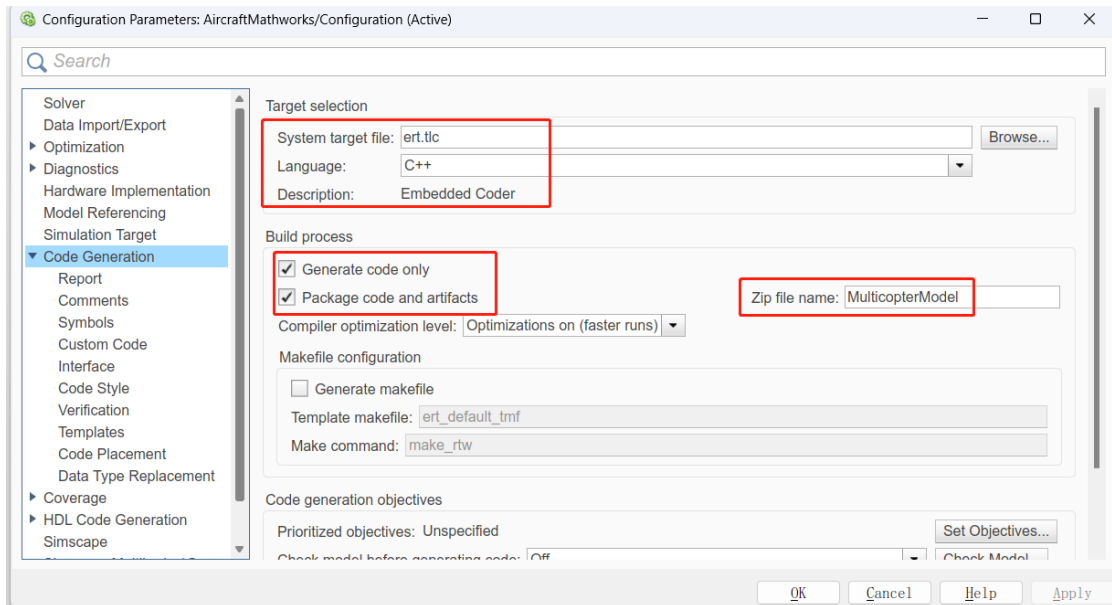
Simulink 模型编译设置步骤如下:

Step 1: 打开 Simulink “设置” 页面, 设置仿真为定步 (Fixed-step) 长, 四阶龙格库塔法 (ode4 Runge-Kutta) 求解器, 步长为 0.001s (也可以根据需求设置成其他)。

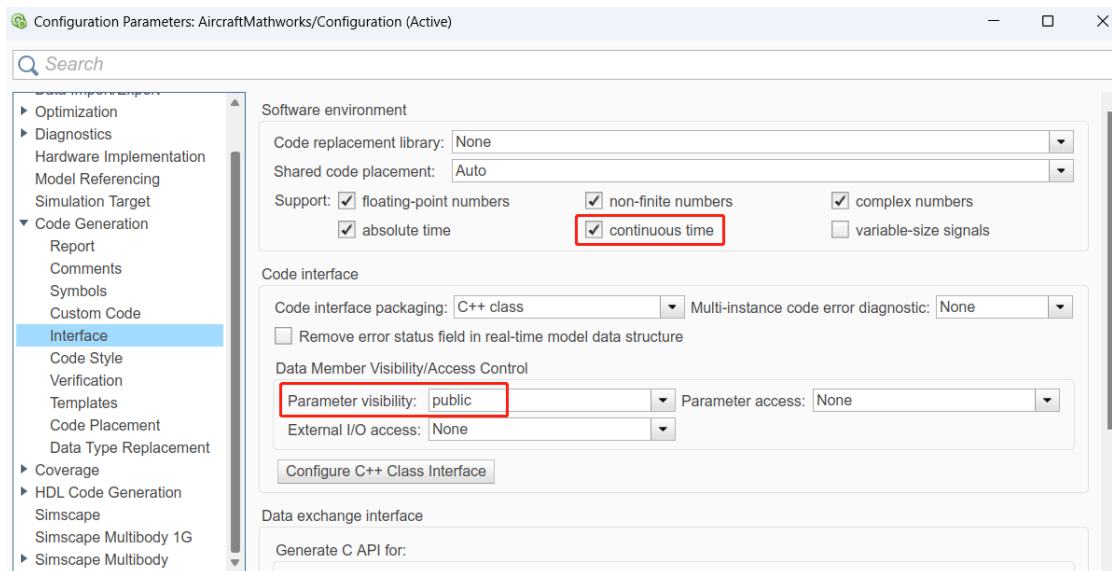


Step 2: 代码生成方式选择 ert.tlc, 可用于 windows, Linux 和各类嵌入式平台; 语言

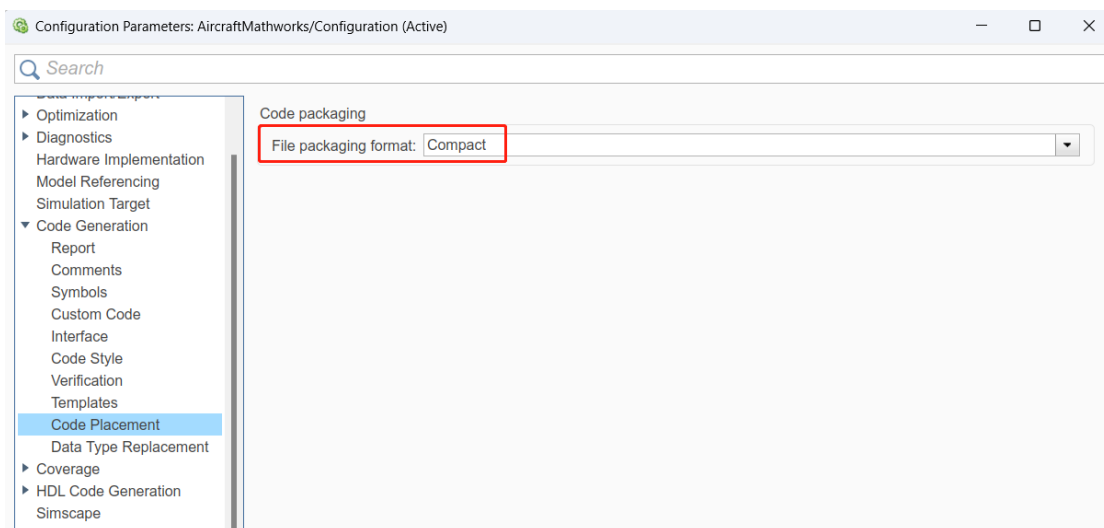
选择 C++，便于通过继承方式调用生成代码；编译过程选择“代码和工具打包”，Zip 文件名设置为“MulticopterModel”。



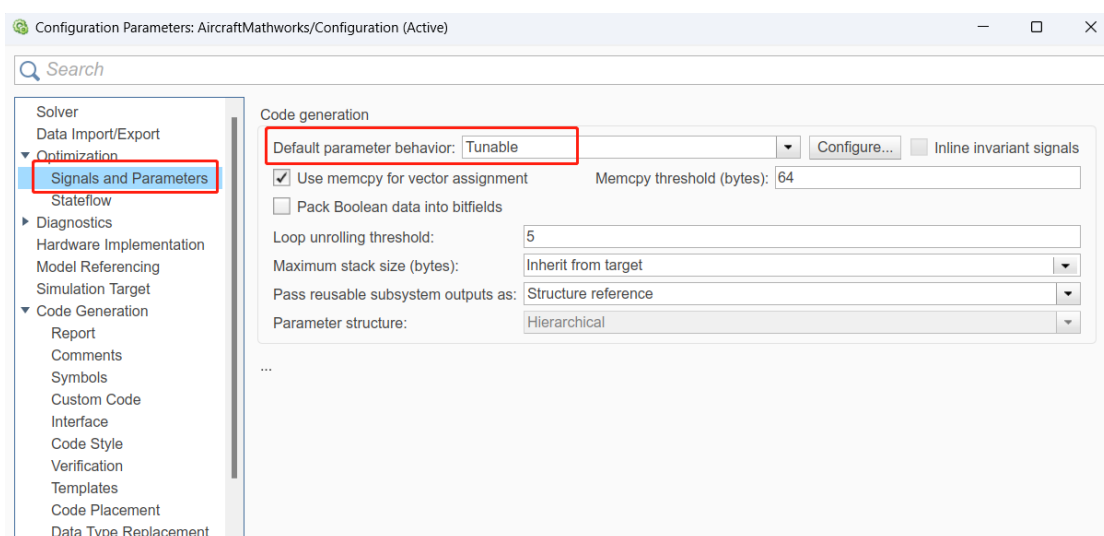
Step 3: 因为包含连续模块（积分模块）因此需要勾选 **continuous time**，不然编译报错。此外将参数可见性 **Parametervisibility** 设为 **public**，参数结构体为共有变量，便于访问。



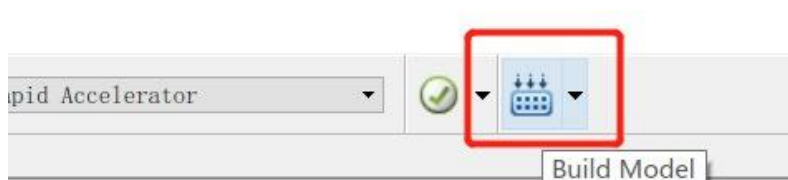
Step 4: 在 **Codeplacement** 页设置文件打包类型为 **compact**，尽量避免生成多余文件，使得代码的可读性最强。



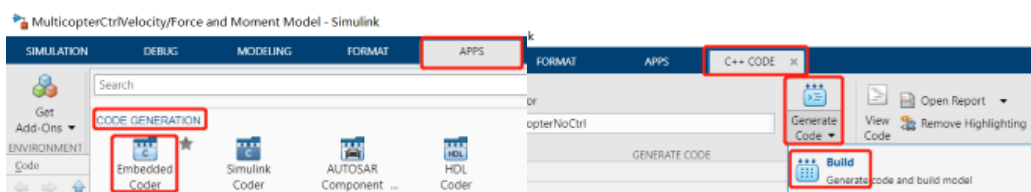
Step 5: 设置参数为 Tunable(可调), 使得我们可以运行时修改参数。注: inline 形式更省内存, 但是不便于访问参数, 不便于实现参数实时修改或者模型故障注入。



完成了载具 Simulink 模型编译设置后, 点击 Simulink 的编译按钮, 即可生成 C/C++ 代码, 方法如下: 对于 MATLAB 2019a 及之前版本, 工具栏样式见下图, 直接点击它的编译按钮 “Build” 即可。



对于 2019b 及之后的版本, 点击 APPS - CODE GENERATION - Embedded Coder 才能弹出代码生成工具栏, 在其中如下图所示点击 “C++CODE” - “Generate Code” - “Build” 按钮就能编译生成代码。



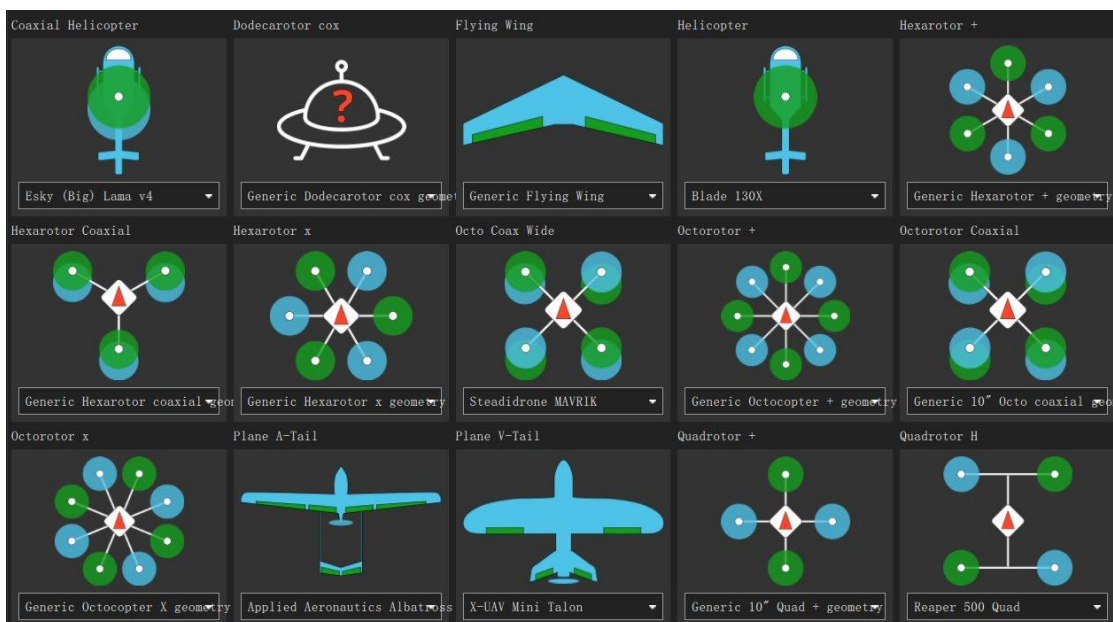
2.4 Linux 版环境配置

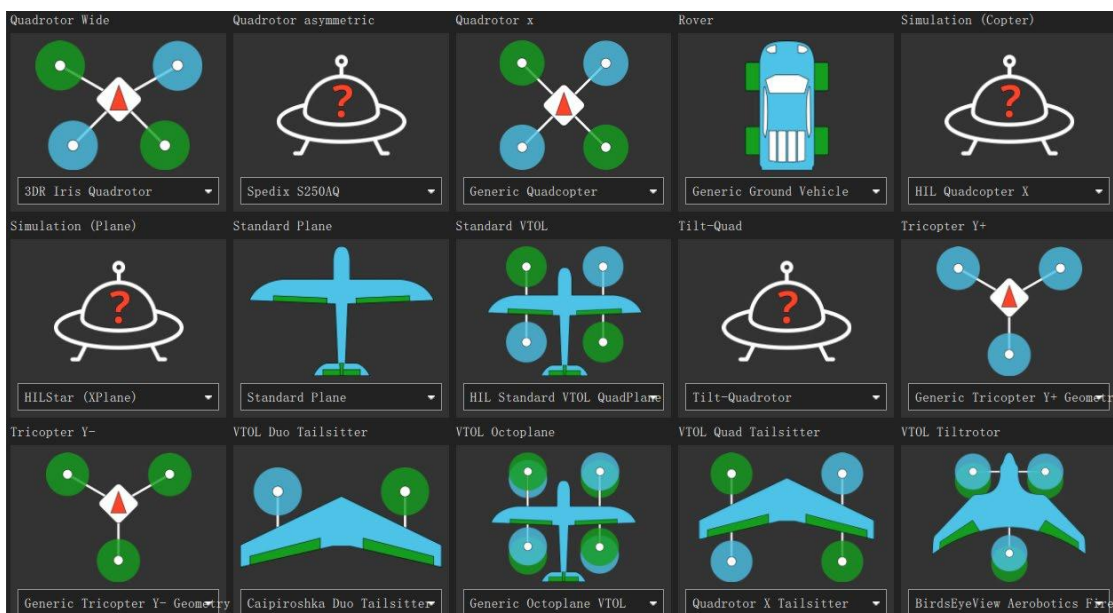
3、基本操作流程

3.1 载具 Simulink 模型开发

RflySim 载具模型基于 MATLAB/Simulink 进行开发，采用模块化的建模思路，将无人载具动力运动模型分为电机模块、力和力矩模块、环节模块、六自由度模块和传感器模块等。

RflySim 支持任意 PX4 可控机型的软硬件在环仿真，所有支持机型可从 QGroundControl 的 Airframe(机架)页面中查看，如下图所示。



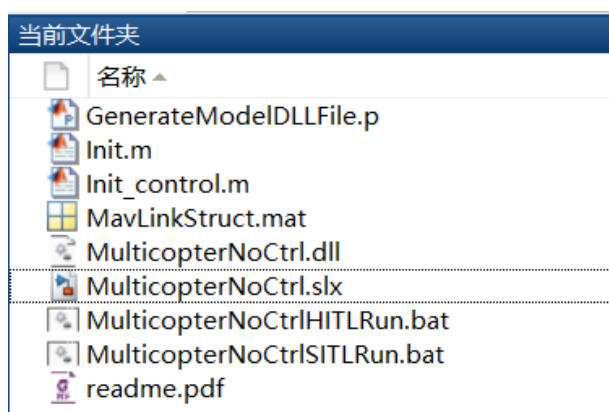


目前 RflySim 包含了旋翼、固定翼、无人车、无人船、标准垂直起降无人机、四旋翼尾座式垂起无人机和直升机，其他模型需要用户按照 RflySim 模型模板在 Simulink 中搭建。

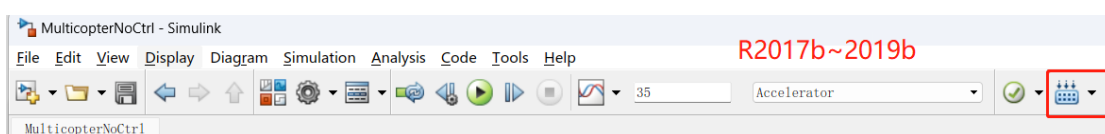
RflySim 模型模板见 RflySimAPIs3.0\4.RflySimModel\0.SourceCode\DLLModelTemp。评价模型开发完成的标准为：模型逻辑正确；运行不报错。

3.2 模型编译生成 C/C++文件

载具 Simulink 模型开发完成后，通过 MATLAB 打开模型源文件（例如，四旋翼为 MulticopterNoCtrl.slx）。



在 Simulink 上方菜单栏中，点击编译命令（MATLAB 2017b~2019b 版本可在菜单栏中直接点击编译，2022a 及以上版本操作流程为：APP-Embedded Coder-编译）。

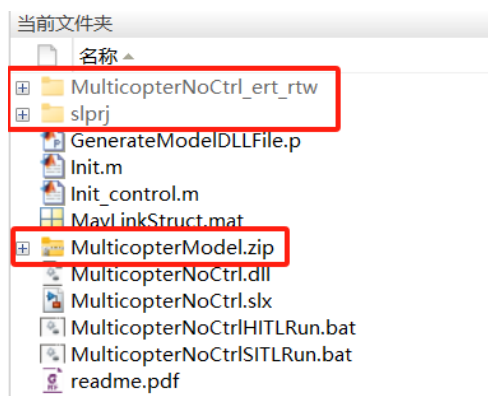




在 Simulink 的下方点击 View diagnostics 指令，即可弹出诊断对话框，可查看编译过程。
在诊断框中弹出 Build process completed successfully，即表示编译成功。



Simulink 模型编译完成后，会生成***_ert_rtw 文件夹及 MulticopterModel.zip 压缩包，表明模型编译成功。

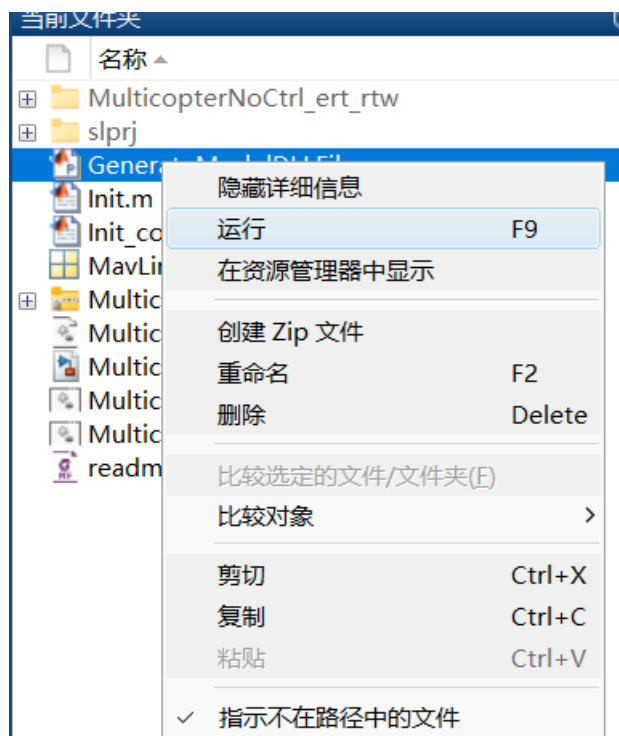


3.3 DLL/SO 模型生成

使用 RflySim 平台进行载具软硬件在环仿真时，需要将 DLL(windows 下)/SO (Linux 下) 模型导入到 CopterSim，形成运动仿真模型，因此，在 Simulink 模型编译完成后，需要将模型对应的 C++文件打包成 DLL/SO 模型。

Windows 系统下:

得到***_ert_rtw 文件夹和 MulticopterModel.zip 压缩包后, 运行 GenerateModelDLLFile.p 文件, 即可得到 DLL 模型。



下图所示, 即表明 DLL 模型生成成功。

```
命令行窗口
modeldllgen.cpp
    c1 modeldllgen.obj MulticopterNoCtrl.obj /link /DLL /out:MulticopterNoCtrl.dll
用于 x64 的 Microsoft (R) C/C++ 优化编译器 19.16.27051 版
版权所有 (C) Microsoft Corporation。保留所有权利。

Microsoft (R) Incremental Linker Version 14.16.27051.0
Copyright (C) Microsoft Corporation. All rights reserved.

/out:modeldllgen.exe
/DLL
/out:MulticopterNoCtrl.dll
modeldllgen.obj
MulticopterNoCtrl.obj
    正在创建库 MulticopterNoCtrl.lib 和对象 MulticopterNoCtrl.exp
    Compiling successfully the MulticopterNoCtrl.dll has been generated.
fx >>
```

Linux 系统下:

3.4 PX4 混控规则介绍

用户在使用 RflySim 平台进行载具软硬件在环仿真时, 需要确认机架类型, PX4 官网机架定义可查看[机架参考 | PX4 自动驾驶用户指南](#)。

下面以飞翼模型为例，说明如何确认官方飞翼机架类型以及混控文件：

Step 1:

在[机架参考 | PX4 自动驾驶用户指南](#)中找到飞翼，在下图中可看到，机架名为 **Generic Flying Wing**,**SYS_AUTOSTART=3000**。

Plane

Flying Wing



| Name | |
|---------------------|---|
| Generic Flying Wing | Maintainer: John Doe <john@example.com> SYS_AUTOSTART = 3000 |

Step 2:

打开“PX4PSP\Firmware\ROMFS\px4fmu_common\init.d\airframes”路径，可找到对应机架文件。
















| Windows (C:) > PX4PSP > Firmware > ROMFS > px4fmu_common > init.d > airframes | | | | |
|---|-----------------|--------|------|--|
| 名称 | 修改日期 | 类型 | 大小 | |
| 1002_standard_vtol.hil | 2021/9/29 10:56 | HIL 文件 | 2 KB | |
| 1100_rc_quad_x_sih.hil | 2021/9/29 10:56 | HIL 文件 | 1 KB | |
| 2100_standard_plane | 2021/9/29 10:54 | 文件 | 1 KB | |
| 2105_maja | 2021/9/29 10:54 | 文件 | 2 KB | |
| 2106_albatross | 2021/9/29 10:54 | 文件 | 2 KB | |
| 2200_mini_talon | 2021/9/29 10:54 | 文件 | 2 KB | |
| 2507_cloudship | 2021/9/29 10:54 | 文件 | 1 KB | |
| 3000_generic_wing | 2021/9/29 10:54 | 文件 | 1 KB | |
| 3030_io_camflyer | 2021/9/29 10:54 | 文件 | 2 KB | |
| 3031_phantom | 2021/9/29 10:56 | 文件 | 2 KB | |
| 3032_skywalker_x5 | 2021/9/29 10:54 | 文件 | 1 KB | |
| 3033_wingwing | 2021/9/29 10:56 | 文件 | 2 KB | |
| 3034_fx79 | 2021/9/29 10:56 | 文件 | 1 KB | |
| 3035_viper | 2021/9/29 10:54 | 文件 | 1 KB | |
| 3036_pigeon | 2021/9/29 10:56 | 文件 | 2 KB | |

Step 3:

用 Vs Code 打开机架文件 3000_generic_wing, 可看到文件中设置了混控文件。

```
C: > PX4PSP > Firmware > ROMFS > px4fmu_common > init.d > airframes > $ 3000_generic_wing
1  #!/bin/sh
2  #
3  # @name Generic Flying Wing
4  #
5  # @type Flying Wing
6  # @class Plane
7  #
8  # @output MAIN1 left aileron
9  # @output MAIN2 right aileron
10 # @output MAIN4 throttle
11 #
12 # @output AUX1 feed-through of RC AUX1 channel
13 # @output AUX2 feed-through of RC AUX2 channel
14 # @output AUX3 feed-through of RC AUX3 channel
15 #
16 # @maintainer
17 #
18 # @board bitcraze_crazyflie exclude
19 #
20
21 . ${R}etc/init.d/rc.fw_defaults
22
23 set MIXER fw_generic_wing
24
```

Windows (C:) > PX4PSP > Firmware > ROMFS > px4fmu_common > mixers

| 名称 | 修改日期 | 类型 | 大小 |
|---|-----------------|--------|------|
|  dodeca_bottom_cox.aux.mix | 2021/9/29 10:56 | MIX 文件 | 1 KB |
|  dodeca_top_cox.main.mix | 2021/9/29 10:56 | MIX 文件 | 1 KB |
|  firefly6.aux.mix | 2021/9/29 10:56 | MIX 文件 | 1 KB |
|  firefly6.main.mix | 2021/9/29 10:56 | MIX 文件 | 1 KB |
|  fw_generic_wing.main.mix | 2021/9/29 10:54 | MIX 文件 | 2 KB |
|  FX79.main.mix | 2021/9/29 10:56 | MIX 文件 | 2 KB |
|  generic_diff_rover.main.mix | 2021/9/29 10:54 | MIX 文件 | 1 KB |
|  hexa_+.main.mix | 2021/9/29 10:54 | MIX 文件 | 1 KB |
|  hexa_cox.main.mix | 2021/9/29 10:54 | MIX 文件 | 1 KB |
|  hexa_x.main.mix | 2021/9/29 10:54 | MIX 文件 | 1 KB |
|  IO_pass.main.mix | 2021/9/29 10:54 | MIX 文件 | 1 KB |
|  mount.aux.mix | 2021/9/29 10:54 | MIX 文件 | 1 KB |
|  mount_legs.aux.mix | 2021/9/29 10:56 | MIX 文件 | 1 KB |
|  octo_+.main.mix | 2021/9/29 10:54 | MIX 文件 | 1 KB |
|  octo_cox.main.mix | 2021/9/29 10:54 | MIX 文件 | 1 KB |

Step 4:

用 Vs Code 打开混控文件 fw_generic_wing.main.mix，内容如下：

```

C: > PX4PSP > Firmware > ROMFS > px4fmu_common > mixers > fw_generic_wing.main.mix
1  Generic wing mixer
2  =====
3
4  This file defines mixers suitable for controlling a delta wing aircraft.
5  The configuration assumes the elevon servos are connected to servo
6  outputs 0 and 1 and the motor speed control to output 3. Output 2 is
7  assumed to be unused.
8
9  Inputs to the mixer come from channel group 0 (vehicle attitude), channels 0
10 (roll), 1 (pitch) and 3 (thrust).
11
12 See the README for more information on the scaler format.
13
14 Elevon mixers
15 -----
16 Three scalars total (output, roll, pitch).
17
18 On the assumption that the two elevon servos are physically reversed, the pitch
19 input is inverted between the two servos.
20
21 The scaling factor for roll inputs is adjusted to implement differential travel
22 for the elevons.
23
24 M: 2
25 S: 0 0 -8000 -8000 0 -10000 10000
26 S: 0 1 6000 6000 0 -10000 10000
27
28 M: 2
29 S: 0 0 -8000 -8000 0 -10000 10000
30 S: 0 1 -6000 -6000 0 -10000 10000
31
32 Output 2
33 -----
34 This mixer is empty.
35
36 Z:
37
38 Motor speed mixer
39 -----
40 Two scalars total (output, thrust).
41
42 This mixer generates a full-range output (-1 to 1) from an input in the (0 - 1)
43 range. Inputs below zero are treated as zero.
44
45 M: 1
46 S: 0 3 0 20000 -10000 -10000 10000
47

```

至此，我们已清楚 PX4 飞翼的机架文件和混控文件，下面介绍 PX4 混控规则。

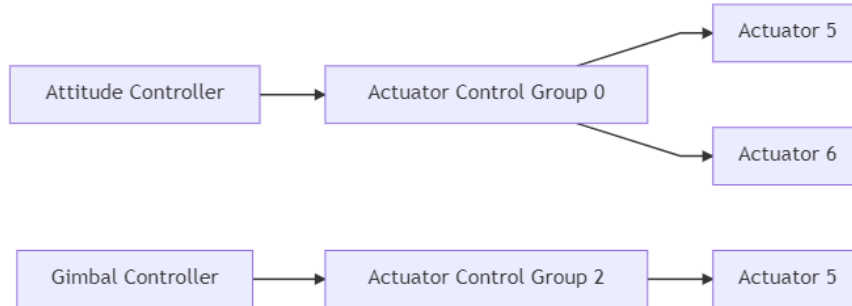
PX4 混控规则：

注：详细定义见[混控器和执行器 | PX4 自动驾驶用户指南](#)

PX4 架构保证了核心控制器中不需要针对机身布局做特别处理。混控指的是把输入指令（例如：遥控器打右转）分配到电机以及舵机的执行器（如电调或舵机 PWM）指令。对于固定翼的副翼控制而言，每个副翼由一个舵机控制，那么混控的意义就是控制其中一个副翼抬起而另一个副翼落下。同样的，对多旋翼而言，俯仰操作需要改变所有电机的转

速。将混控逻辑从实际姿态控制器中分离出来可以大大提高复用性。

特定的控制器发送一个特定的归一化的力或力矩指令（缩放至 $-1..+1$ ）给混控器，混控器则相应地去设置每个单独的执行器。控制量输出驱动程序（比如：UART, UAVCAN 或者 PWM）则将混控器的输出所放为执行器实际运行时的原生单位，例如输出一个值为 1300 的 PWM 指令。



PX4 的控制通道输出主要有 4 个控制组(Control Group)，分别为：

- **actuator_controls_0**: 飞控的主要控制通道，用于输出俯仰、滚转、偏航、油门等各个通道的控制量。具体定义如下：

Control Group #0 (Flight Control)

- 0: roll ($-1..1$)
- 1: pitch ($-1..1$)
- 2: yaw ($-1..1$)
- 3: throttle (0..1 normal range, $-1..1$ for variable pitch / thrust reversers)
- 4: flaps ($-1..1$)
- 5: spoilers ($-1..1$)
- 6: airbrakes ($-1..1$)
- 7: landing gear ($-1..1$)

- **actuator_controls_1**: 备用控制通道，在 VTOL 中用于输出固定翼模式的控制输出。具体定义如下：

Control Group #1 (Flight Control VTOL/Alternate)

- 0: roll ALT ($-1..1$)
- 1: pitch ALT ($-1..1$)
- 2: yaw ALT ($-1..1$)
- 3: throttle ALT (0..1 normal range, $-1..1$ for variable pitch / thrust reversers)
- 4: reserved / aux0
- 5: reserved / aux1
- 6: reserved / aux2
- 7: reserved / aux3

- **actuator_controls_2**: 云台控制通道。具体定义如下：

Control Group #2 (Gimbal)

- 0: gimbal roll
- 1: gimbal pitch
- 2: gimbal yaw
- 3: gimbal shutter
- 4: reserved
- 5: reserved
- 6: reserved
- 7: reserved (parachute, $-1..1$)

- **actuator_controls_3**: 遥控映射通道。具体定义如下：

Control Group #3 (Manual Passthrough)

- 0: RC roll
- 1: RC pitch
- 2: RC yaw
- 3: RC throttle
- 4: RC mode switch
- 5: RC aux1
- 6: RC aux2
- 7: RC aux3

PX4 混控器文件语法

Mixer 文件是定义一个或多个 Mixer 定义的文本文件:一个或多个输入与一个或多个输出之间的映射。主要有四种类型的定义: multirotor mixer, helicopter mixer, summing mixer, and null mixer。

- multirotor mixer: 定义输出 4、6 或 8 的+型或 x 型旋翼载具。
- helicopter mixer: 定义直升机斜盘伺服器和主电机 ESCs 的输出(尾桨是一个单独的混合器)。
- summing mixer: 将零或多个控制输入组合成单个执行器输出。输入被缩放, 混合函数在应用输出缩放器之前对结果求和。
- summing mixer: 产生一个输出为零的执行器输出(当不处于故障安全模式时)。

每个混控器产生的输出量取决于混控器的类型和配置。如: multirotor mixer 根据其机型可有 4、6 或 8 个输出, 而 summing mixer 或 summing mixer 只产生一个输出。可以在每个文件中指定多个混控器。输出顺序(将混合器分配给执行器)特定于读取混控器定义的设备, 对于 PWM, 输出顺序与声明顺序相匹配。

每个混控器文件最开始定义的语句为:

```
<tag>: <mixer arguments>
```

其中 tag 表示所选择的混控器类型, 如下:

```
R: Multirotor mixer  
H: Helicopter mixer  
M: Summing mixer  
Z: Null mixer
```

Summing Mixer—加法混控器

Summing Mixer 用于控制无人机执行器和伺服。它将零或多个控制输入组合成单个执行器输出。输入被缩放, 混合函数在应用输出缩放器之前对结果求和。最小执行器遍历时间限制也可以在输出标量中指定(回转率的逆)。简单的混合器定义如下:

```
M: <control count>
```

```
0: <-ve scale> <+ve scale> <offset> <lower limit> <upper limit> <traversal time>
```

若<control count>为零, 则总和有效为零, 混合器将输出一个固定值, 该值受<lower limit>和<upper limit>约束。

第二行用上述的标量参数定义输出标量。虽然计算作为浮点操作执行, 但存储在定义文件中的值按 10000 倍缩放; 即-0.5 的偏移量被编码为-5000。输出标度上的<traversal time>(可选)用于执行器, 若数值过大, 可能会损坏飞行器一如: 倾转旋翼垂直起降飞行器上的

倾斜执行器。可以用来限制执行器的变化速率(如果没有指定,则不应用速率限制)。例如:
`<traversal time>`值 20000 将限制执行器的变化率,使得从`<lower limit>`和`<upper limit>`至少需要 2 秒,反之亦然。

注 1: `<traversal time>`应该只在硬件需要时使用!

注 2: 不要对控制车辆姿态的致动器(如用于气动表面的伺服器)施加任何限制,因为这很容易导致控制器不稳定。

继续定义`<control count>`的输入及其缩放,形式为:

S: `<group>` `<index>` `<-ve scale>` `<+ve scale>` `<offset>` `<lower limit>` `<upper limit>`

注 3: `s`: 必须在 `0`: 的下面。

注 4: 任何具有油门输入的混控器输出(`S`: 中的`<group>=0` 和`<index>=3`)均无法在解锁或预解锁状态下工作。如: 有四个输入(滚转、俯仰、偏航和油门)的伺服器,即使有滚转/俯仰/偏航信号也不会在解锁状态下运动。

`<group>` 值标识标量器要读取的控制组, `<index>` 值表示该组中的偏移量。这些值特定于读取混控器定义的设备。当用于混合载具控制时,混控器组 0 为载具姿态控制组,而 0~3 通常分别为滚转、俯仰、偏航和推力。其余字段使用上面讨论的参数配置控制缩放器。当计算作为浮点运算执行时,存储在定义文件中的值按 10000 倍缩放;即-0.5 的偏移量被编码为-5000。下面解释一个典型的混合器文件示例。详细示例解析请见: https://docs.px4.io/v1.13/en/dev_airframes/adding_a_new_frame.html#mixer-file。

Null Mixer—空混控器

该混控器不消耗任何控制通道,生成一个值始终为零的单个执行器输出。通常,Null Mixer 用作混频器集合中的占位符,以实现执行器输出的特定模式。它也可以用来控制用于故障安全装置的输出值(正常使用时输出为 0;在故障安全期间,混控器被忽略,而使用故障安全值代替)。定义如下:

Z:

Multicopter Mixer—多旋翼混控器

Multicopter Mixer 将四个控制输入(滚转、俯仰、偏航、推力)组合成一组执行器输出,用于驱动电机速度控制器。定义如下:

R: `<geometry>` `<roll scale>` `<pitch scale>` `<yaw scale>` `<idle speed>`

支持的机型有:

- 4x - 四旋翼 x 型配置
- 4+ - 四旋翼+型配置
- 6x - 六旋翼 x 型配置
- 6+ - 六旋翼+型配置
- 8x - 八旋翼 x 型配置
- 8+ - 八旋翼+型配置

横滚、俯仰和偏航比例值决定了相对于推力控制的横滚、俯仰和偏航控制的比例。当计算作为浮点运算执行时,存储在定义文件中的值按 10000 倍缩放;如: 0.5 则被编码为 5

000。横滚、俯仰和偏航输入的范围从-1.0 到 1.0，而推力输入的范围从 0.0 到 1.0。每个执行器的输出范围为-1.0 至 1.0。

空转速度的范围从 0.0 到 1.0。空转速度是相对于电机的最大速度，它是当所有控制输入为零时，电机被命令旋转的速度。在执行器饱和的情况下，所有执行器的值被重新调整，使饱和和执行器限制为 1.0。

Helicopter Mixer—直升机混控器

Helicopter Mixer 将三个控制输入(滚转、俯仰、推力)组合成四个输出(旋转斜盘和主电机 ESC 设置)。直升机混合器的第一个输出是主马达的油门设置。随后的输出是旋转斜盘的伺服器。尾桨可以通过添加一个简单的混合器来控制。推力控制输入用于主电机设置以及斜盘的集体螺距。它使用了一个油门曲线和一个俯仰曲线，都由五个点组成。

注：油门和俯仰曲线将“推力”杆输入位置映射到油门值和俯仰值(分别)。这允许飞行特性调整为不同类型的飞行。

Helicopter Mixer 定义如下：

```
H: <number of swash-plate servos, either 3 or 4>
T: <throttle setting at thrust: 0%> <25%> <50%> <75%> <100%>
P: <collective pitch at thrust: 0%> <25%> <50%> <75%> <100%>
```

T:定义油门曲线的点。**P:**定义俯仰曲线的点。两条曲线都包含 0 到 10000 之间的 5 个点。对于简单的线性变化，曲线的五个值应该是 0、2500、5000、7500、10000。

每个旋转斜盘的伺服器(3 或 4)的定义如下：

```
S: <angle> <arm length> <scale> <offset> <lower limit> <upper limit>
```

<angle>以度为单位，0 度为机头方向。正角度是顺时针。**<arm length>**是标准化的长度，即 10000 等于 1。如果所有的伺服臂都是相同的长度，值应该都是 10000。较大的臂长会减少伺服偏转的量，而较短的臂长会增加伺服偏转。伺服输出按 <scale> / 10000 缩放。缩放后，**<offset>**被应用且它值应该在 -10000 和 +10000 之间。在全伺服范围内，**<lower limit>**和 **<upper limit>**应为 -10000 和 +10000。

尾桨可以通过加一个 Summing Mixer 来控制：

```
M: 1
S: 0 2 10000 10000 0 -10000 10000
```

通过将尾桨直接映射到偏航命令。这适用于两个伺服控制的尾翼，以及尾翼与专用电机。

130 叶片直升机混合器文件如下所示：

```
H: 3
T: 0 3000 6000 8000 10000
P: 500 1500 2500 3500 4500
# Swash plate servos:
S: 0 10000 10000 0 -8000 8000
S: 140 13054 10000 0 -8000 8000
S: 220 13054 10000 0 -8000 8000

# Tail servo:
M: 1
```

S: 0 2 10000 10000 0 -10000 10000

- 在 50% 推力时，油门曲线的斜率略陡，达到 6000(0.6)。
- 在 100% 推力的情况下，以较小的坡度达到 10000(1.0)。
- 俯仰曲线是线性的，但不会使用其整个范围。
- 在 0% 油门时，总距操纵杆设置已经在 500(0.05)。
- 在最大油门下，总距操纵杆仅为 4500(0.45)。
- 对这种类型的直升机使用更高的数值会使叶片失速。
- 这架直升机的旋转斜盘系统位于 0、140 和 220 度的角度。
- 伺服臂长不相等。
- 与第一个伺服系统相比，第二个和第三个伺服系统的手臂长度为 1.3054。
- 伺服器被限制在 -8000 和 8000，因为它们是机械约束。

VTOL Mixer—垂直起降无人机混控器

垂直起降系统使用多旋翼混合器作为多旋翼模式下的输出，总和混合器作为固定翼模式下的输出。垂直起降无人机的混控器系统既可以组合成一个单独的混控器，其中所有的执行器都连接到 IO 或 FMU 端口，也可以分成单独的混控器文件用于 IO 和 AUX。

3.5 bat 脚本相关参数修改

3.5.1 硬件在环仿真脚本

常规硬件在环仿真脚本，支持输入串口序列（英文逗号“,”分隔），来启动多机的硬件在环仿真。注：REM 打头的行是注释语句，不会被执行。其他的 bat 脚本语法规则可自行搜索学习。

注：本脚本的飞机位置是由脚本按矩形队列自动生成的，控制变量包括：

SET /a START_INDEX=1（初始飞机序号，本脚本生成的飞机的 CopterID，以此 START_INDEX 为初始值，依次递增 1）

SET /a TOTAL_COPTER=8（总飞机数量，仅在多机联机仿真才需要赋值，告诉本脚本实际的飞机总数，以此来确定矩形队列的边长）

SET UE4_MAP=Grasslands（设置地图名字）

SET /a ORIGIN_POS_X=0（矩形编队的原点 X 位置，单位米，只支持整数输入）

SET /a ORIGIN_POS_Y=0（矩形编队的原点 Y 位置，单位米，只支持整数输入）

SET /a ORIGIN_YAW=0（矩形编队的原点 yaw 角度，单位度，只支持整数输入）

SET /a VEHICLE_INTERVAL=2（矩形编队的飞机间隔，单位米，只支持整数输入）

SET /a UDP_START_PORT=20100（接收外部控制数据的 UDP 通信接口，与 CopterID 对应会自动加 2，这里通常不需要修改，仅在电脑端口被占用时才修改）

set DLLModel=0（用来设置导入到 CopterSim 进行硬件在环仿真的 DLL 模型名称，由 Simulink 模型编译后通过 GenerateModelDLLFile.p 生成的 DLL 模型的文件名决定，设置为 0 时，默认采用四旋翼 DLL 模型）

set SimMode=0（仿真模式，这里设置为 0 或 PX4_HITL，表示硬件在环仿真）

SET IS_BROADCAST=0（是否联机仿真，这里可输入目标 IP 地址序列）

SET UDPSIMMODE=0（UDP_START_PORT 端口接收的数据协议，UDP 模式传输的是平台私有结构体，支持 Simulink 控制；MAVLink 模式传输的是 MAVLink 协议，支持 Python 和 mavros 等控制模式）

3.5.2 软件在环仿真脚本

常规软件在环脚本，支持输入飞机数量，自动开启多机软件在环仿真。

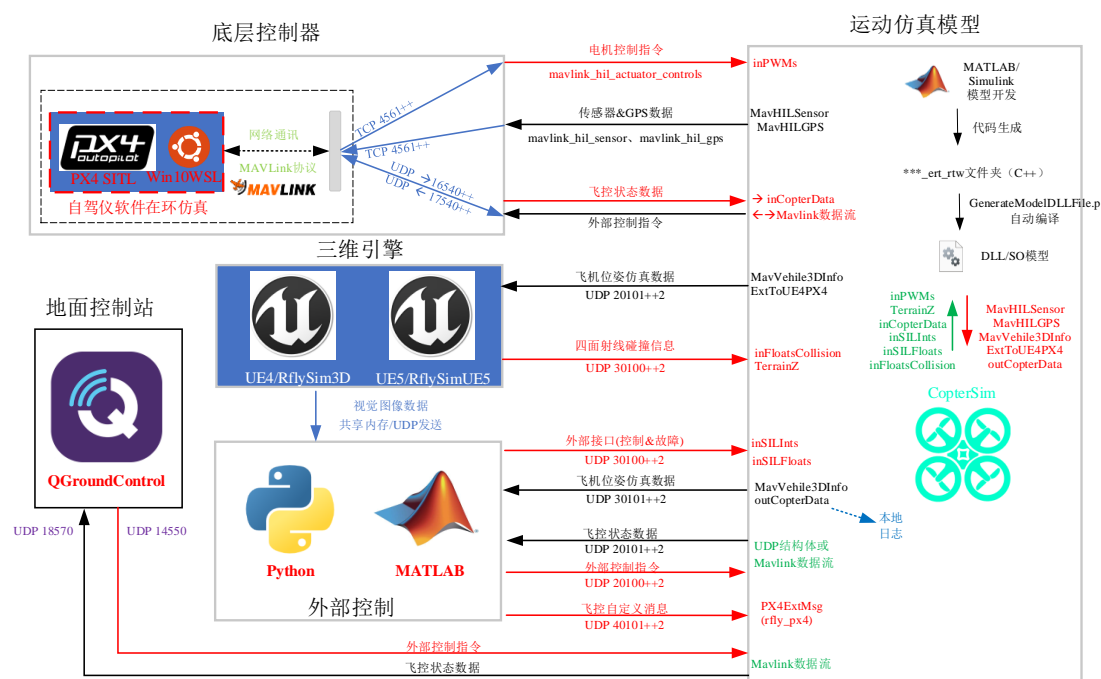
与 HITLRun.bat 相比，关键代码如下

set SimMode=2（这里设置为软件在环模式，对应 CopterSimUI 的值）

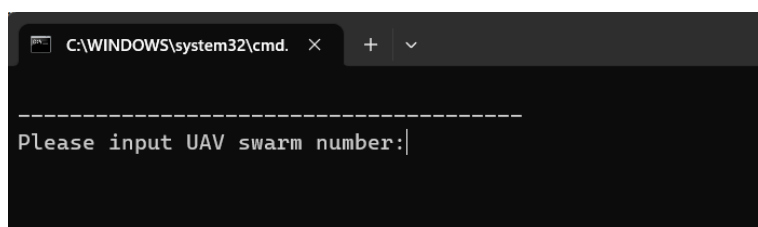
set PX4SITLFrame=iris（这里用来设置 PX4 仿真机架类型，设置为机架文件的非数字部分，例如，iris 对应的四旋翼，generic_wing 对应的飞翼，hexa_x 对应的六旋翼，standard_plane 对应的固定翼，机架类型确认方法见 3.4 PX4 混控规则介绍）

3.6 SITL 仿真

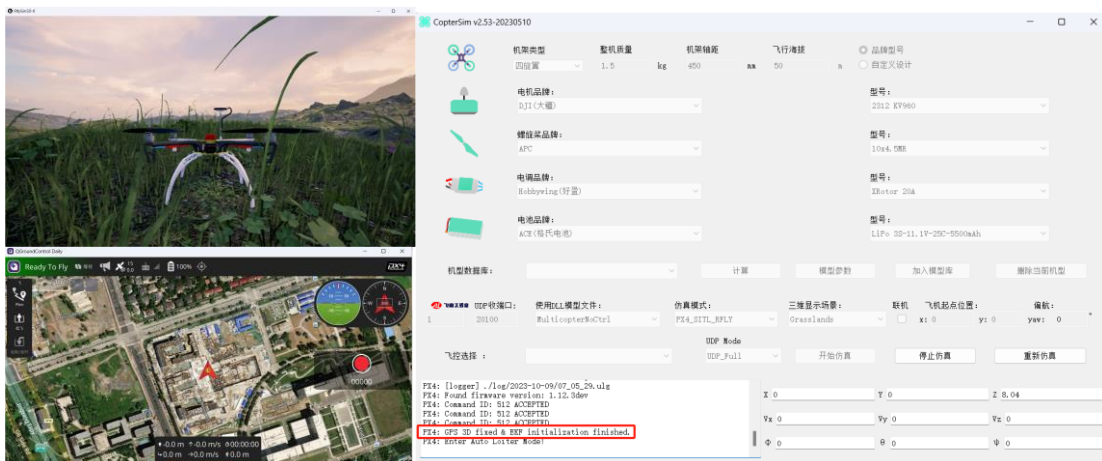
RflySim 软件在环仿真系统通过软件在环仿真脚本（如：SITL.bat）进行构建，脚本运行完成后，将会在计算机上启动 PX4 仿真环境、CopterSim、QGroundControl 以及三维引擎（RflySim3D/RflySimUE5），并且自动配置完通讯端口。RflySim 平台（包括 CopterSim、QGroundControl 和三维引擎）与 PX4 通讯通过 MAVLink 消息进行，SITL 仿真时通过 PX4 PSP\Firmware\src\modules\simulator\simulator_mavlink.cpp 来处理这些消息。



运行指定 SITLRun.bat 脚本后，会弹出提示框，在光标处输入数字，即可在三维场景中创建对应数量的无人载具，并完成初始化。单独 1 台电脑支持多机仿真，可同时仿真的无人载具数量由计算机性能和通信负载决定。

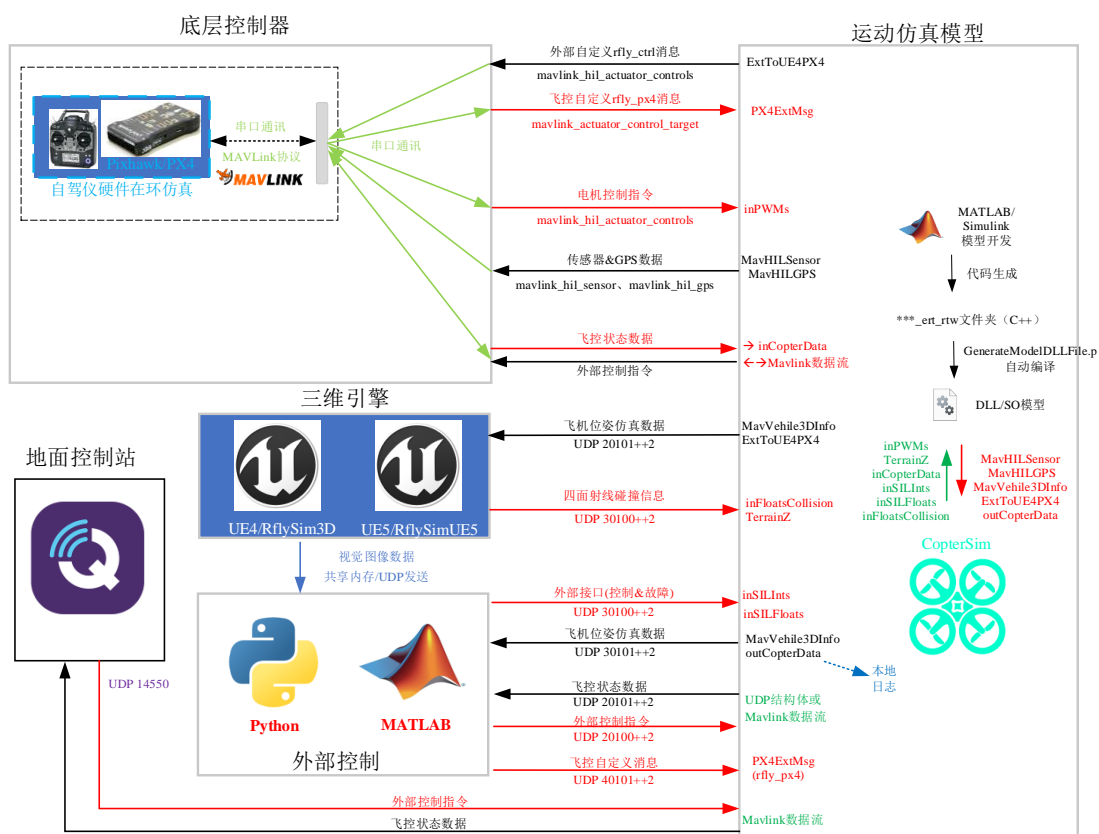


下面以四旋翼软件在环启动脚本进行说明，在提示框光标处输入 1 后回车，系统将启动 QGroundControl、RflySim3D 和 CopterSim 三种软件，在 RflySim3D 里能看到四旋翼模型，当 CopterSim 左下方消息提示栏中提示“GPS 3D fixed&EKF initialization finished”时，表明软件在换仿真初始化完成，可以开始仿真。



3.7 HITL 仿真

硬件在环仿真（HITL 或 HIL）是将 PX4 固件在真实的飞行控制器（即飞控）硬件上运行的仿真模式。硬件在环仿真时，通过 USB 将飞控连接至主机，进而可以通过串口方式实现 RflySim 平台和飞控的通讯。



下面以四旋翼模型为例，介绍硬件在环仿真开始前飞控的一般配置步骤：

注：例程路径为*:\PX4PSP\RflySimAPIs\4.RflySimModel\2.AdvExps\c2_MultiModelCtrl\












2.MultiModelCtrl

Step 1: 确定仿真使用的飞控类型以及固件版本，平台推荐使用 Pixhawk 6C 飞控，固件版本为 1.13.3。

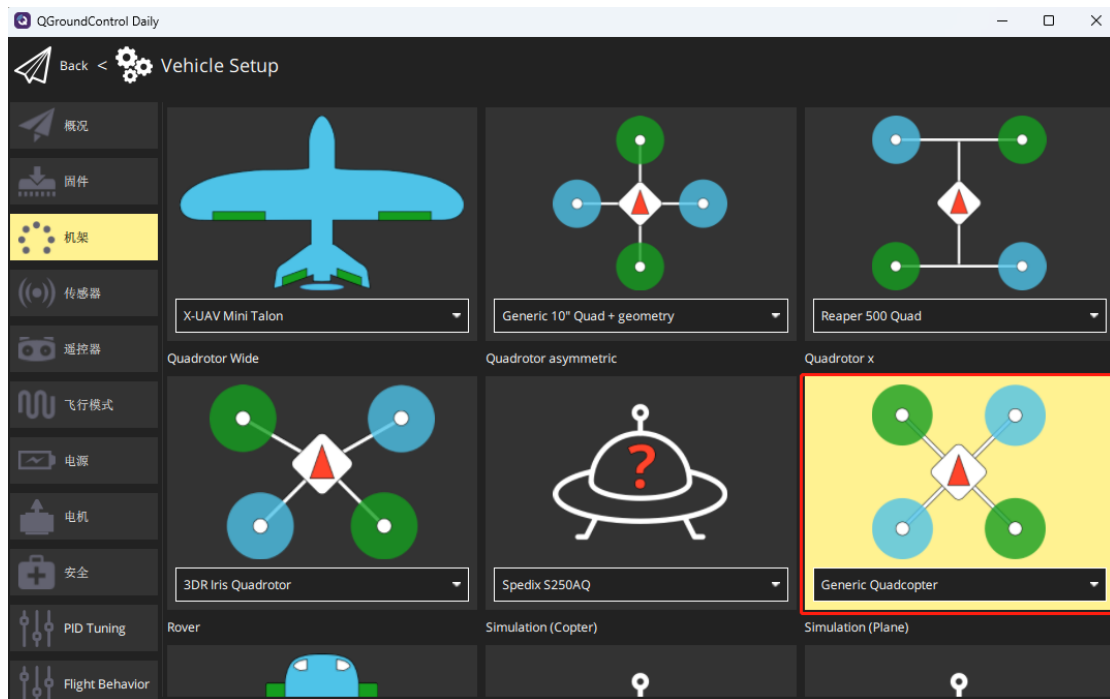
Step 2: 通过 USB-TypeC 线将飞控连接至电脑 USB 端口。



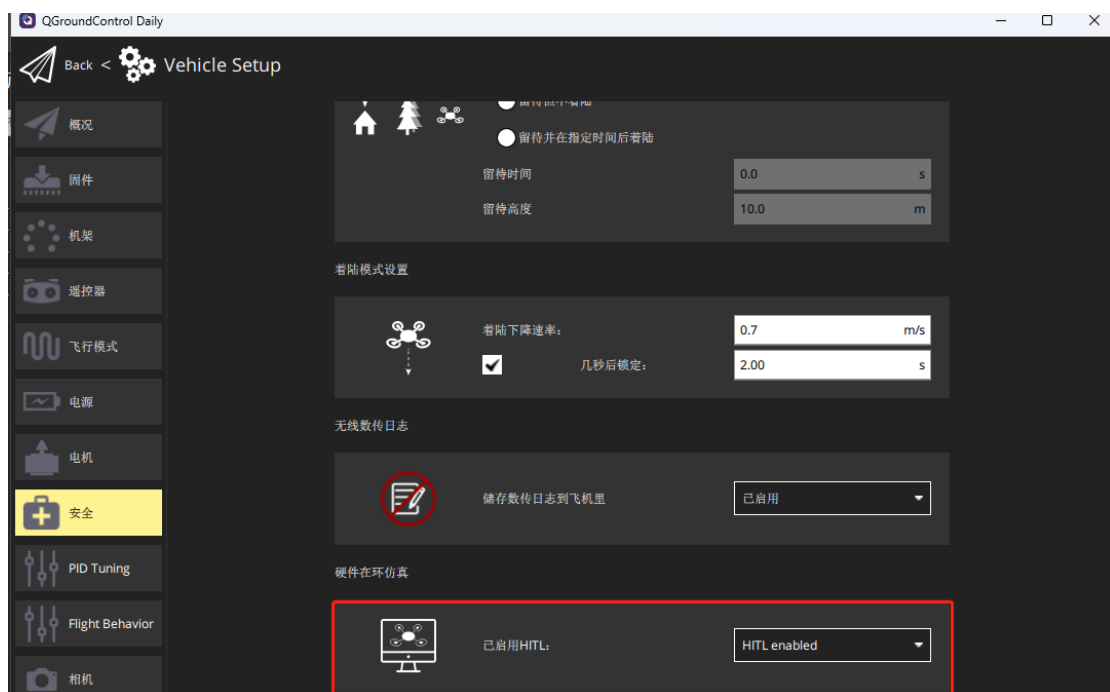
Step 3: 在 Rflytools 文件夹中打开 QGC 地面站。

| | | | |
|---|-----------------|------|------|
|  3DDisplay | 2023/7/27 15:02 | 快捷方式 | 1 KB |
|  CopterSim | 2023/7/27 15:02 | 快捷方式 | 1 KB |
|  FlightGear-F450 | 2023/7/27 15:02 | 快捷方式 | 2 KB |
|  HITLRun | 2023/7/27 15:02 | 快捷方式 | 2 KB |
|  Python38Env | 2023/7/27 15:02 | 快捷方式 | 2 KB |
|  QGroundControl | 2023/7/27 15:02 | 快捷方式 | 1 KB |
|  RflySim3D | 2023/7/27 15:02 | 快捷方式 | 1 KB |
|  RflySimAPIs | 2023/7/27 15:02 | 快捷方式 | 1 KB |
|  RflySimUE5 | 2023/7/27 15:02 | 快捷方式 | 1 KB |
|  SITLRun | 2023/7/27 15:02 | 快捷方式 | 2 KB |
|  Win10WSL | 2023/7/27 15:02 | 快捷方式 | 2 KB |

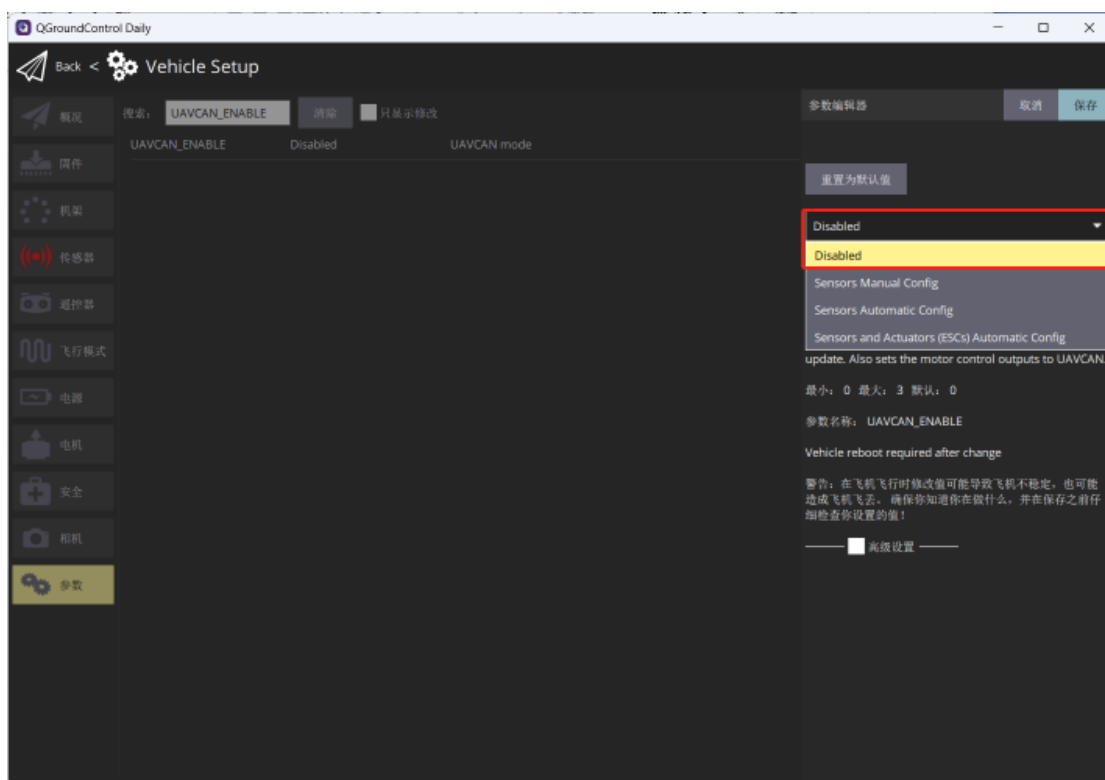
在机架界面设置机架型号为“Generic Quadcopter”(机架由仿真机型及对应的官方机架文件决定)，设置完毕后点击右侧“应用并重启”。



Step 4: 在“安全”界面，选择“HITL enabled”启动硬件在环仿真，重新插拔飞控。



Step 5: 点击“参数”，在搜索栏中输入“UAVCAN_ENABLE”，在弹出框中设置为“Disabled”，保存后重新插拔飞控即完成硬件在环仿真前的配置。



3.8 通过 QGC 或遥控器进行仿真测试

3.9 外部接口通信调试

4、DLL 文件生成脚本-GenerateModelDLLFile.p

5、DLL/SO 模型与通信接口

5.1 总体介绍

从实现机制的角度分析，可将 RflySim 平台分为运动仿真模型、底层控制器、三维引擎、外部控制和地面控制站五部分。

基于 MATLAB/Simulink 完成模型开发后，自动代码生成 C++ 文件并通过平台 GenerateModelDLLFile.p 接口生成 DLL 模型，在使用 RflySim 平台进行软硬件在环仿真时，会将 DLL 模型导入到 CopterSim，形成运动仿真模型。运动仿真模型拥有多个输入输出接口与底层控制器、三维引擎、地面控制站和外部控制进行数据交互，具体数据链路、通信协议及通信端口号见[错误!未找到引用源。](#)。

其中，软件在环仿真时，底层控制器和运动仿真模型之间以网络通讯的方式进行数据交互，而硬件在环仿真时，底层控制器和运动仿真模型之间以串口通讯的方式进行数据交互。

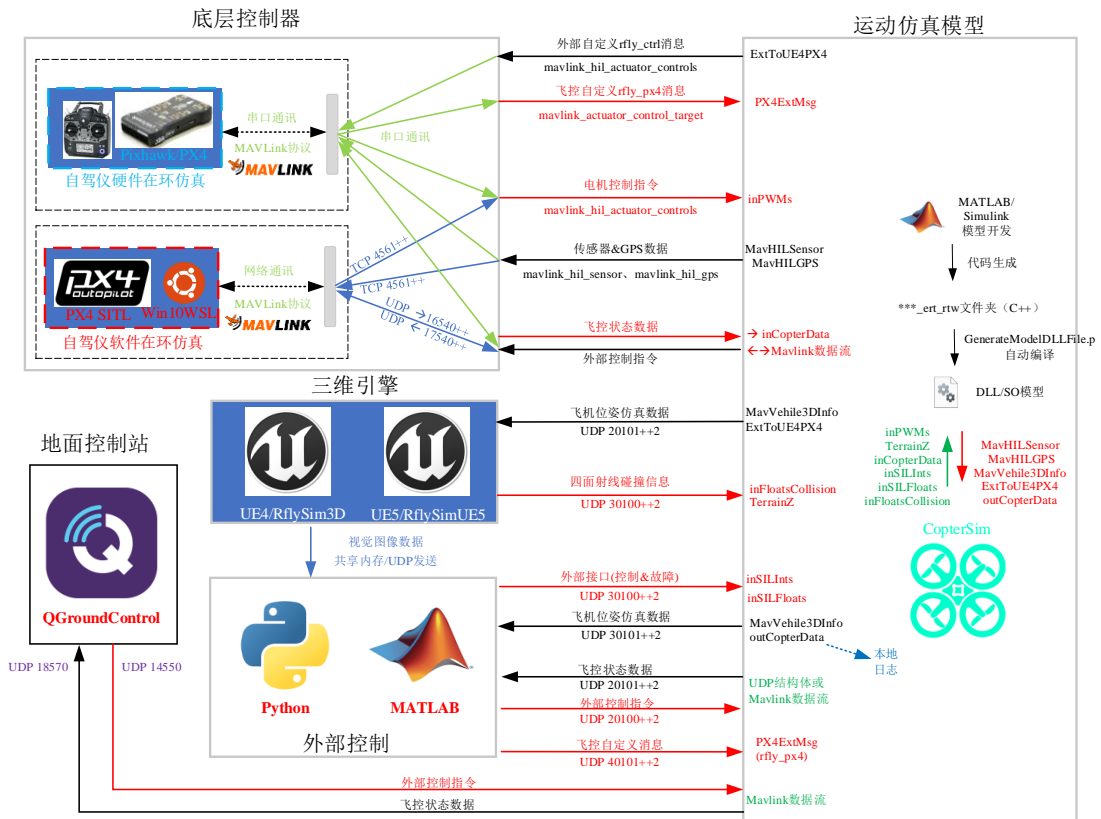


图 1 运动仿真模型与其他模块数据交互图解

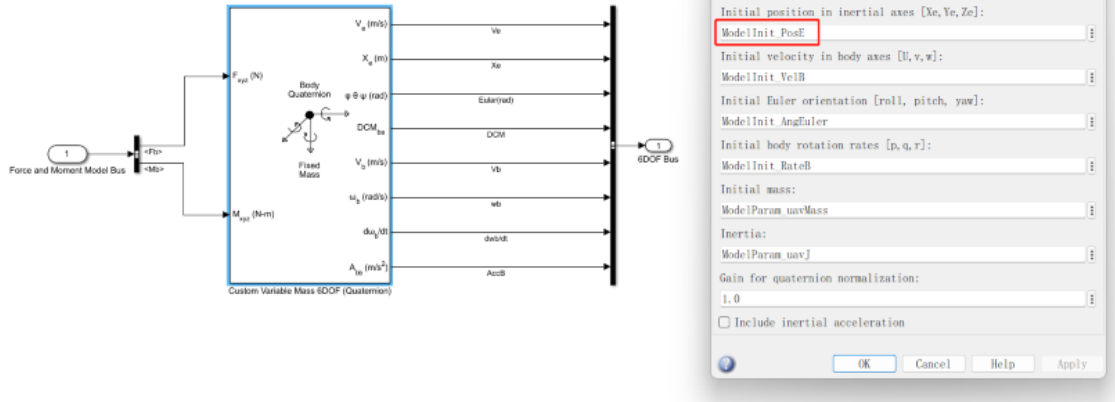
5.2 重要参数

RflySim 运动仿真模型均有对应的 `Init.m` 文件，在该文件中定义了运动仿真模型所需要用的参数，包括模型公式系数（如多旋翼中的螺旋桨拉力系数、力矩系数等）、噪声系数、传感器系数以及会影响 RflySim3D 显示的相关变量等，下面对模型中的重要参数进行介绍。

5.2.1 ModelInit_PosE

该参数为运动仿真模型世界坐标系下位置初始化参数，三维数据，分别为 $[x, y, z]$ ，通过该参数，RflySim 可以在仿真开始前初始化无人机显示在 RflySim3D 地图中的指定 X、Y 位置。

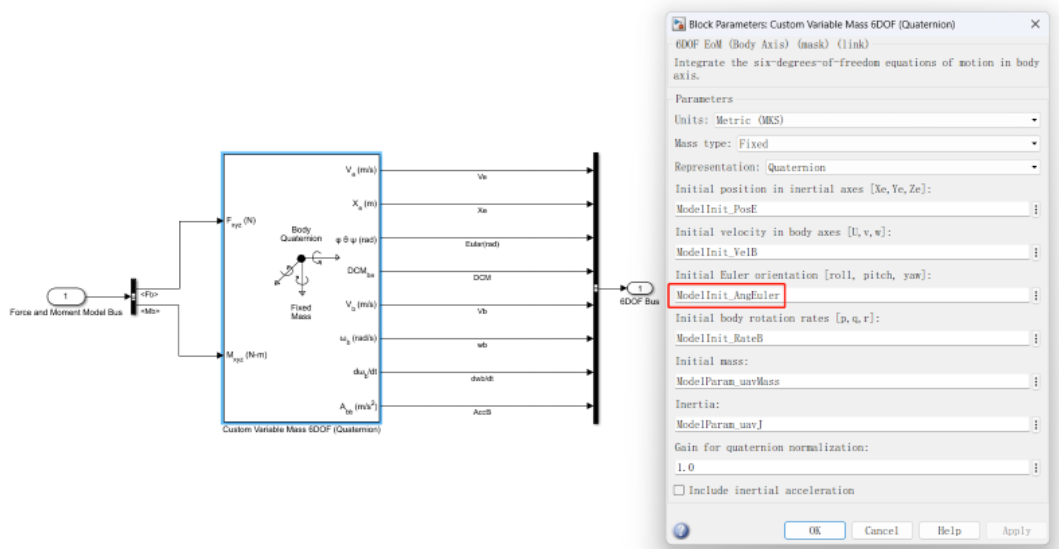
```
ModelInit_PosE=[0,0,0];
```



5.2.2 ModelInit_AngEuler

该参数为运动仿真模型姿态初始化参数，三维数据，分别为 $[\varphi, \theta, \psi]$ ，通过该参数，RflySim 可以在仿真开始前让无人机以指定偏航角在 RflySim3D 中初始化显示。

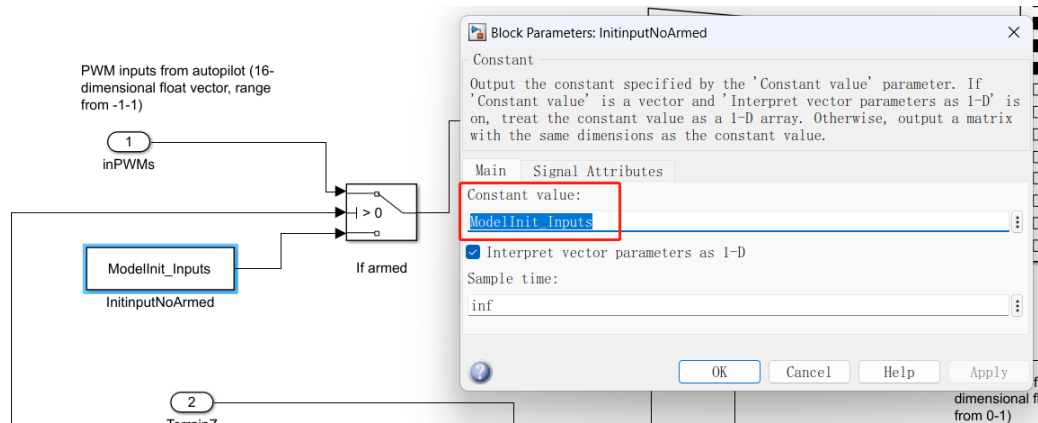
```
ModelInit_AngEuler=[0,0,0];
```



5.2.3 ModelInit_Inputs

该参数为运动仿真模型执行器的输入初始化参数，为 16 维数据，对于有特定需求的飞行器，比如油门初始状态需要处于最小值 (-1)，即需要该参数来修改执行器输入初始化值。

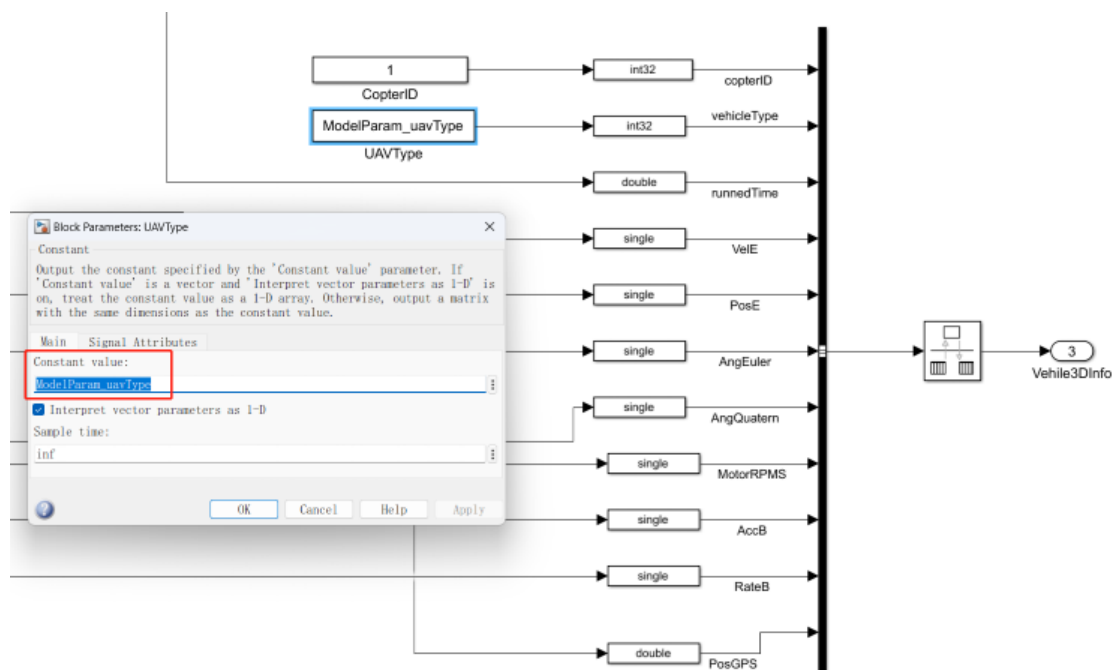
```
ModelInit_Inputs = [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0];
```



5.2.4 ModelParam_uavtype

该参数决定了仿真时调用的 RflySim3D 中的显示模型，比如，ModelParam_uavType 为 3 时，RflySim3D 中的显示模型为四旋翼，ModelParam_uavType 为 100 时，RflySim3D 中的显示模型为常规小型固定翼。

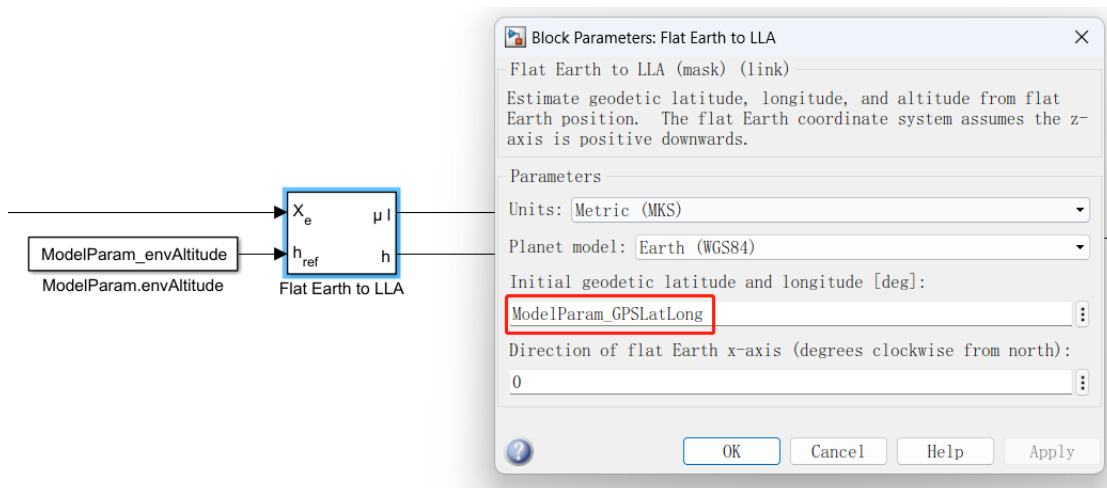
```
ModelParam_uavType = int16(3);
```



5.2.5 ModelParam_GPSLatLong

该参数用来配置飞机的经纬度信息，二维数据，分别为[ModelParam_envLatitude, ModelParam_envLongitude]，通过该参数可以调整飞机在 QGC 中的显示坐标。

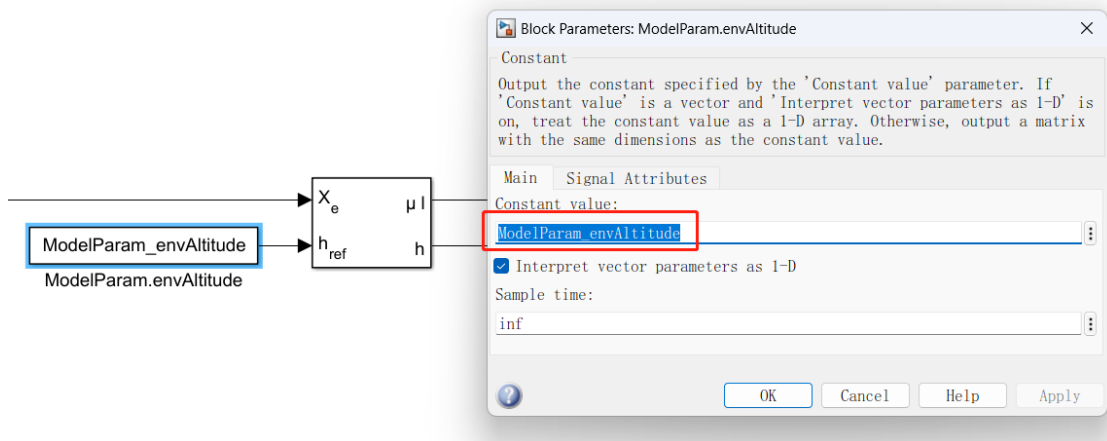
```
ModelParam_envLongitude = 116.259368300000;
ModelParam_envLatitude = 40.1540302;
ModelParam_GPSLatLong = [ModelParam_envLatitude ModelParam_envLongitude];
```



5.2.6 ModelParam_envAltitude

该参数用来配置飞机的高度信息，z轴向下为正，通过该参数可以调整飞机在QGC中的海拔显示。

`ModelParam_envAltitude = -50`



5.3 数据协议

5.3.1 飞控仿真输入接口

5.3.1.1 inPWMs（电机控制量输入）

16 维执行器控制量输入，已归一化到-1 到 1 尺度，它的数据来自飞控回传的电机控制 MAVLink 消息 `mavlink_hil_actuator_controls_t` 的 `controls`，软硬件在环仿真过程中可以通过 QGroundControl 中 Analyze Tools 里的 MAVLink 检测功能实时查看 `controls` 变化。

从图 1 得知，软件在环仿真时，电机控制指令从 PX4 SITL 控制器通过 TCP 4561 系列端口以 MAVLink 协议发送到运动仿真模型的 inPWMs 接口，而硬件在环仿真时，该指令是从飞控通过串口以 MAVLink 协议发送到运动仿真模型的 inPWMs 接口。

发送到 PX4 SITL 控制器的，而硬件在环仿真时，这些数据是通过串口发送到飞控的。

5.3.3 仿真数据输出接口

5.3.3.1 MavVehicle3Dinfo（真实仿真数据输出）

该输出信号是模型发送给 RflySim3D 的真实仿真数据，是平滑的理想值，这些数据可用于 Simulink 下的飞控与模型进行软件仿真测试。由于模型真值在真机实验时是不可获取的，只能用 PX4 自驾仪的状态估计值（存在延迟、噪声和干扰），这就导致 Simulink 控制器往 PX4 在环仿真和真机实验时效果变差，需要进行调整。

5.3.3.2 outCopterData（自定义日志输出）

32 维 double 型，里面的内容可自定义发送数据。发往本接口的数据，一方面会写入到本地的 log 日志中（在 C:\PX4PSP\CopterSim 下新建 CopterSim*.csv，才会开始记录*号飞机的数据，注意这里*要换成飞机的 ID）。另一方面，本数据会通过 UDP 传输到 30101 系列端口（补充 readme）。

5.3.3.4 ExtToUE4（自定义显示数据输出）

16 维 double 型数据，通过 20100 系列端口发送给 RflySim3D 作为第 9-24 维执行器控制消息显示（补充 readme）。

5.3.4 自动代码生成控制器通信接口

5.3.4.1 ExtToPX4（自定义 uORB 数据输出）

16 维 float 型数据，以串口的方式发送给 PX4 的 uORB 消息 rfly_ext，用于传输其他传感器或必要数据给飞控（补充 readme）。

5.3.4.2 inCopterData（uORB 数据输入）

32 维，其中后 8 维接收 PX4 消息，数据来自 uORB msg rfly_px4.control[0:7]。

5.3.5 碰撞数据接收接口—inFloatsCollision

利用 inFloatsCollision 实现了一个简单地物理引擎，可以根据 RflySim3D 回传的四周距离数据，实现碰到障碍物的回弹、碰到其他飞机便坠毁等功能（补充 readme）。

5.3.6 外部数据传入接口

5.3.6.1 inSILInts（整型数据输入）

8 维 Int32 型输入，通过 UDP 协议获取，来自 30100++2 系列端口号，软硬件在环仿真时，可通过该端口向模型输入一些量；同时，该接口是实现综合模型的关键接口。

5.3.6.2 inSILFloats（浮点型数据输入）

20 维 float 型输入，通过 UDP 协议获取，来自 30100++2 系列端口号，软硬件在环仿真时，可通过该端口向模型输入一些量；同时，该接口是实现综合模型的关键接口。

5.3.6.3 inFromUE（RflySim3D 数据输入）

16 维 double 型数据，来自三维引擎（Rflysim3D/RflySimUE5），可用于实现地面交互、

碰撞引擎等需要与三维引擎进行数据交互的相关功能。

5.3.7 实时参数修改接口—FaultParamsAPI

5.4 通信接口

5.4.1 20100++2 系列端口

20100++2 系统端口为 CopterSim 的 UDP 收端口，主要接收外部控制指令。

5.4.2 20101++2 系列端口

20101++2 系列端口为 CopterSim 的 UDP 发端口，其发出的数据主要包括：

- 1) 仿真时，发给三维引擎的飞机位姿仿真数据。
- 2) 外部控制时，发出飞控状态数据。

5.4.3 30100++2 系列端口

30100++2 系列端口为 CopterSim 的 UDP 收端口，其接收的数据主要包括：

- 1) 来自三维引擎的四面射线碰撞信息，用于实现碰撞功能。
- 2) 来自外部控制（Python/MATLab）的数据，包括控制或故障注入等。

5.4.4 30101++2 系列端口

30101++2 系列端口为 CopterSim 的 UDP 收端口，其发出的数据主要包括：

- 1) 飞机位姿仿真数据。
- 2) 来自运动仿真模型 outCopterData 接口的自定义日志数据。

5.4.5 TCP 端口

TCP 端口为软件在环仿真时 PX4 控制器和运动仿真模型之间的通信端口，仿真时，PX4 控制器通过 TCP 4561 系列端口将电机控制指令发送给运动仿真模型的 inPWMs 接口，而模型通过 TCP 4561 系列端口将传感器和 GPS 数据反馈给 PX4 控制器，形成仿真闭环。

5.4.6 飞控 USB 串口

硬件在环仿真时，PX4 飞控和运动仿真模型之间通过串口进行通讯，其中，PX4 飞控发出的数据主要包括：

- 1) 通过 MavLink 消息 mavlink_hil_actuator_control 发出的电机控制指令。
- 2) 通过 MavLink 消息 mavlink_actuator_control_target 发出的飞控自定义消息 rfly_px4 消息。
- 3) 飞控状态数据。

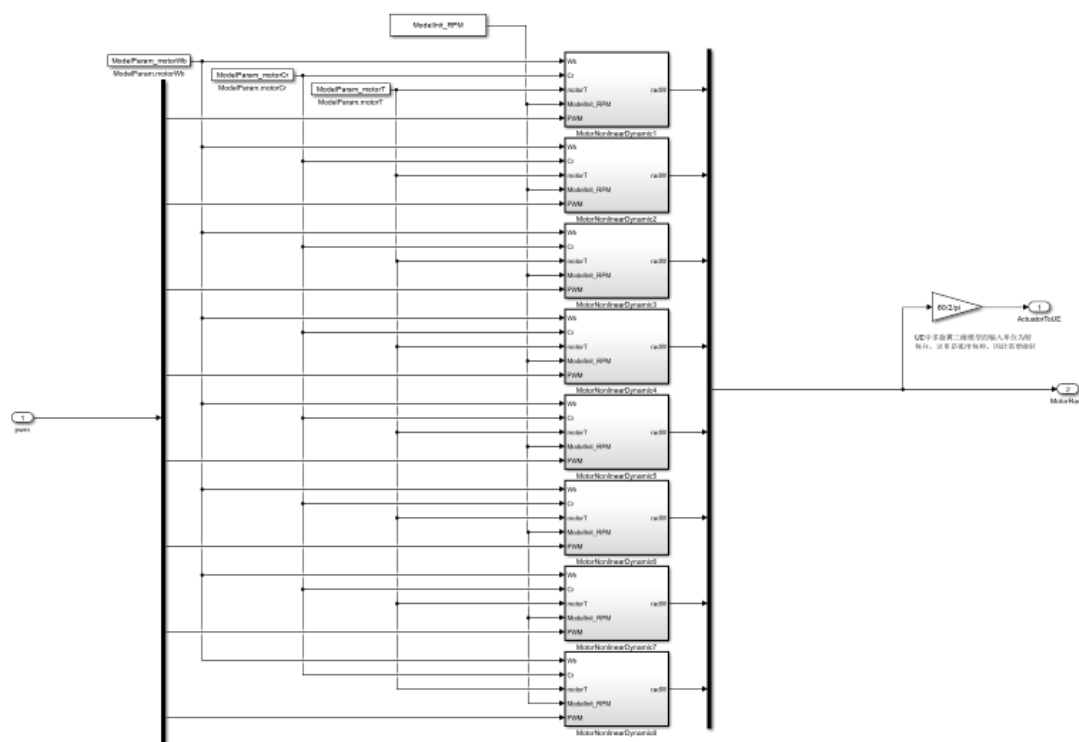
模型发出的数据主要包括：

- 1) 通过 MavLink 消息 mavlink_hil_sensor、mavlink_hil_gps 发出的传感器和 GPS 数据。
- 2) 外部自定义 rfly_ctrl 消息。
- 3) 外部控制指令。

6、Simulink 建模模板介绍

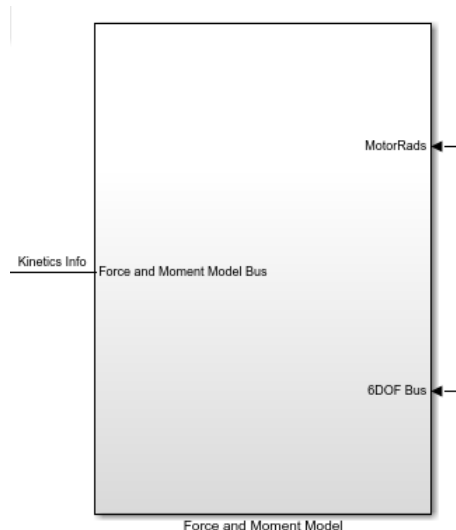
6.1 Motor Model 电机模块

四旋翼电机通常由电机本体、转子、定子和控制器组成。转子上的螺旋桨通过电机转动产生升力和推力，而控制器则负责调节电机的速度和方向。从而实现无人机的稳定悬停、前进、后退、转向等动作。在该模块中输入为 PWM 值，经过各电机的非线性动力学模型后得到各电机转速，该模块的输出分别为输入给力和力矩模型的电机转速（弧度每秒）；输入给 UE 的电机转速（转每分），由于 UE 中多旋翼的三维模型输入单位为转每分，所以在传输给 UE 的转速上做了单位转换。



6.2 Force and Moment Model 力和力矩模块

对无人机所受到的外部力和力矩进行模拟，例如，多旋翼在该模块中对螺旋桨拉力、机身气动力、自身重力以及地面支撑力等所有的外部力和力矩进行建模。该模块输入为电机转速 MotorRads、飞机运动学姿态 6DOF 和地形高度输入 tZ，输出为多旋翼合力、合力矩 Force and Moment Model Bus。

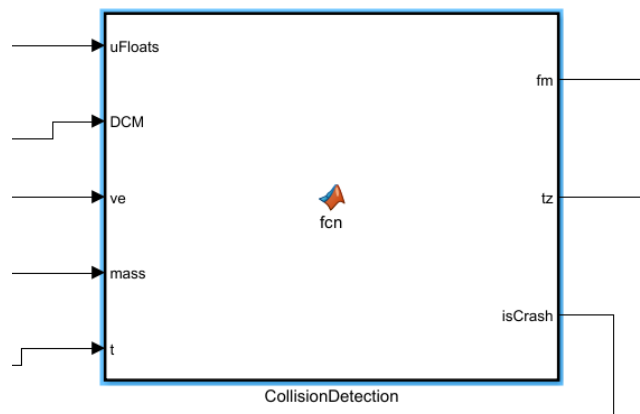


6.3 PhysicalCollisionModel 物理碰撞模块

碰撞模块能检测多旋翼飞行过程中是否碰撞到了物体，以及所碰撞物体的类型并做出符合物理规律的反应。开启碰撞模式后，在碰撞到飞机时多旋翼的速度满足冲量定理，在碰撞到房屋等固定物体时，多旋翼会朝着障碍物反向的反弹回去，反弹速度为原速度的 1/10。

由图可知，碰撞模块的输入为 `uFloats`、`DCM`、`ve`、`mass`、`t`，输出为 `fm`、`tz`、`isCrash`。

其中 `uFloats` 为 20 维外部输入浮点信号，该端口为碰撞模型预留，可以通过 UDP 网络从 UE4 传输。`DCM` 为方向余弦矩阵，`ve` 为碰撞时的速度，`mass` 为多旋翼质量，`t` 为时间戳。输出 `fm` 为力和力矩直接作用到 6DOF 对机体运动产生影响，`tz` 则表示多旋翼离地面的高度和 XYZ 的坐标，`isCrash` 则为碰撞判断，如果碰撞发生则三个电机损坏。具体碰撞逻辑可以点击进入该模块详细了解。



6.4 GroundSupportModel 地面支撑模块

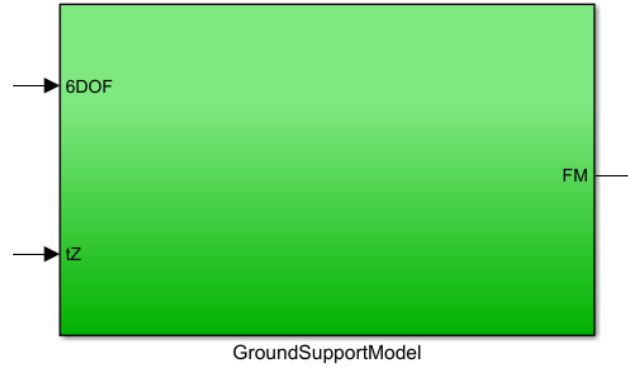
机体的碰撞力主要来源于地面的支撑力与摩擦力，以及在有障碍物时与障碍物发生物理碰撞时的作用力。由于物体的外形具有一定的复杂性，针对复杂形状去求解物理接触点并求解碰撞力是非常困难的，而且在大多数情况下如此高精度的碰撞受力也不是必须。目前，大多数物理引擎采用的是一种简化的物理受力求解方法，即将所有物体简化为较为简

单的基本几何体（例如圆柱体或者长方体）来计算其与地面或其他物体之间的物理接触受力。每个物体的各个表面下方都带有一个弹簧缓冲模型来模拟实际物体的接触、碰撞与缓冲。通过调整弹簧的刚度，可以模拟不同物体的表面柔软程度，或者模拟地面的缓冲作用。

本模型中使用的是一个典型的地面支撑力 F_z 的缓冲接触，可以表示如下：

$$F_z = \begin{cases} 0, & \Delta z > 0 \\ -k_1 \Delta z - k_2 \Delta \dot{z}, & \Delta z \leq 0 \end{cases}$$

其中， Δz 表示机体距离地面表面的 z 方向的位移； $\Delta z > 0$ 表示机体位于地面上方，未发生接触因此受力为 0； $\Delta z < 0$ 表示物体位于地表下方（陷入地面当中），此时地面产生一个反馈控制器，生成一个支撑力试图将位移 Δz 控制到 0，从而实现了地面接触与缓冲的模拟。式错误!未找到引用源。中， $k_1 > 0$ 和 $k_2 > 0$ 是弹簧的缓冲系数，数值越大，代表恢复形变的作用力越强，物体表面的硬度越大，碰撞时的瞬时反作用力也就越大。

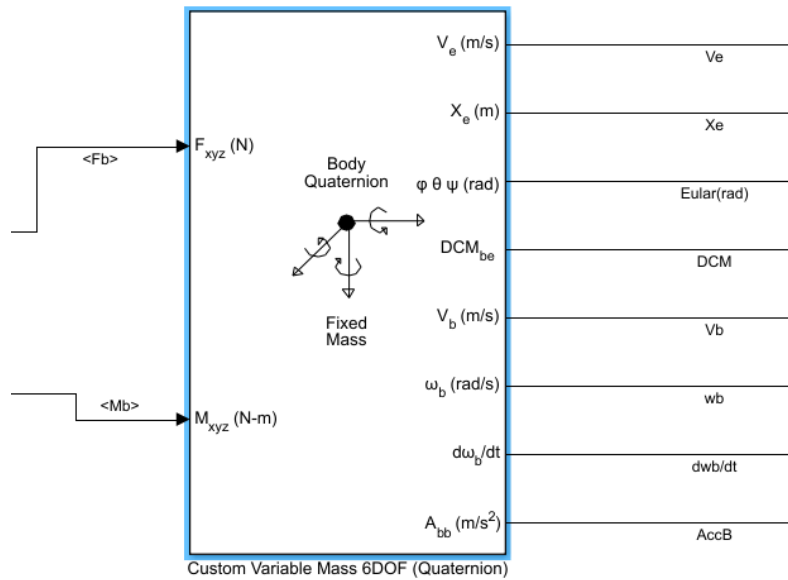


6.5 6DOF 刚体运动学模块

无人机的六自由度模块是用于描述无人机在空中运动时的姿态和位置变化。这个模型基于刚体动力学原理，将无人机视为一个刚体，并考虑了无人机在三个坐标轴上的旋转运动（俯仰、横滚和偏航）以及机体与地球坐标系上的平移运动（前后、左右和上下）。

该模型的输入为机体所受的力和力矩，输出为机体坐标系下的速度与加速度，地球坐标系下的速度，位置，欧拉角，方向余弦矩阵，角速度与角加速度。

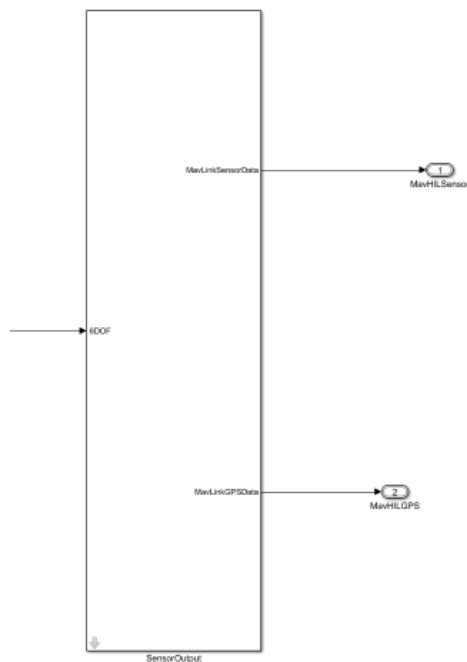
该模块通过对这些自由度进行数学建模，可以推导出无人机在空中运动时的动力学方程，从而可以进行控制系统设计、路径规划和飞行模拟等应用。此外，还可以根据实际需求对模型进行扩展，考虑更多的因素，如飞行器的非线性特性、气动力和惯性矩等。



6.6 SensorOutput 传感器输出模块

该模块中包括了环境模型、传感器模型和 GPS 模型，其中环境模型对重力和大气压强对无人系统飞行产生的影响进行了模拟；传感器模型中不仅对磁力计、惯性导航进行了建模，同时加入了噪声模拟；GPS 模型用于计算 GPS 数据，在仿真时反馈回 PX4 控制器。

该模块输入为 6DOF Bus 结构体，输出为 MavHILSensor 和 MavHILGPS。平台对整体进行了 mask 封装，用户在使用时仅需要把该模块复制到无人载具模型并按 6DOF Bus 结构体输入数据至该模块，即可完成传感器和 GPS 建模。



6.7 3DOutput 三维显示模块

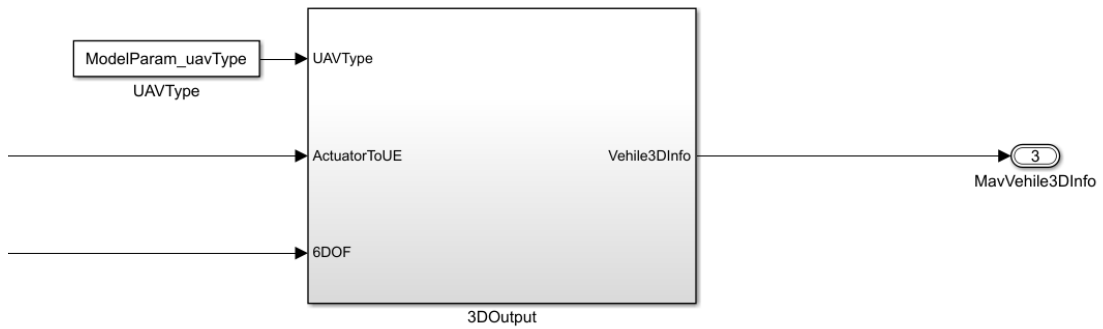
该模块输入为：

1) UAVType: 三维显示 ID, 来自***_init.m 中的 ModelParam_uavType, 由三维模型文件中的 XML 文件决定, 例如: 常规四旋翼的 UAVType 为 3, 小型固定翼的 UAVType 为 100。

2) ActuatorToUE: 来自电机模型, 决定了在三维引擎中无人载具系统电机/舵机的转动情况。

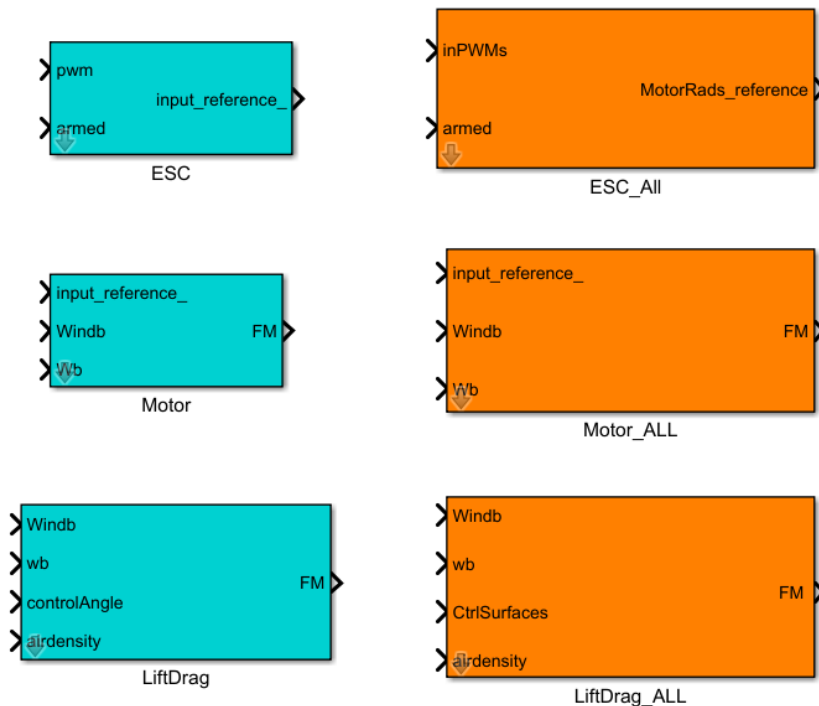
3) 6DOF: 来自 6DOF 模型的 6DOF Bus 输入, 接收无人载具系统的位置、速度、姿态和加速度等信息。

输出为 MavVehicle3DInfo: 在三维显示模块中, 会按协议对输入信息进行数据打包, 并通过该接口将数据发送至三维引擎, 实现可视化处理。



6.8 Gazebo 模型模块

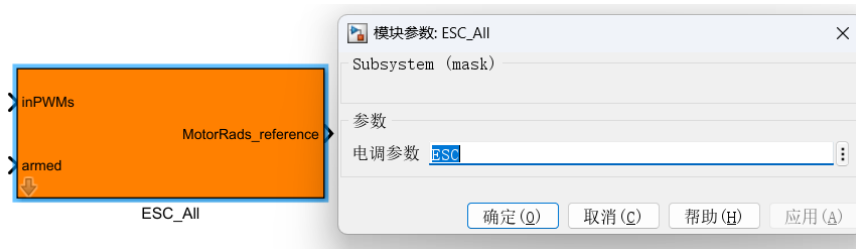
目前 RflySim 平台中已完成的 Gazebo 模型模块有 6 种, 分别为 ESC 模块、Motor 模块、LiftDrag 模块、ESC_All 模块、Motor_ALL 模块和 LiftDrag_ALL 模块。其中, ESC_All 模块是由 8 个 ESC 模块组成的合模块, 同理可得 Motor_ALL 模块和 LiftDrag_ALL 模块。



用户可利用 Gazebo 模型模块实现 Gazebo 平台支持仿真的机架类型在 RflySim 平台上的建模，如旋翼、固定翼、小车等。

6.8.1 ESC_ALL 模块

1) 功能简介



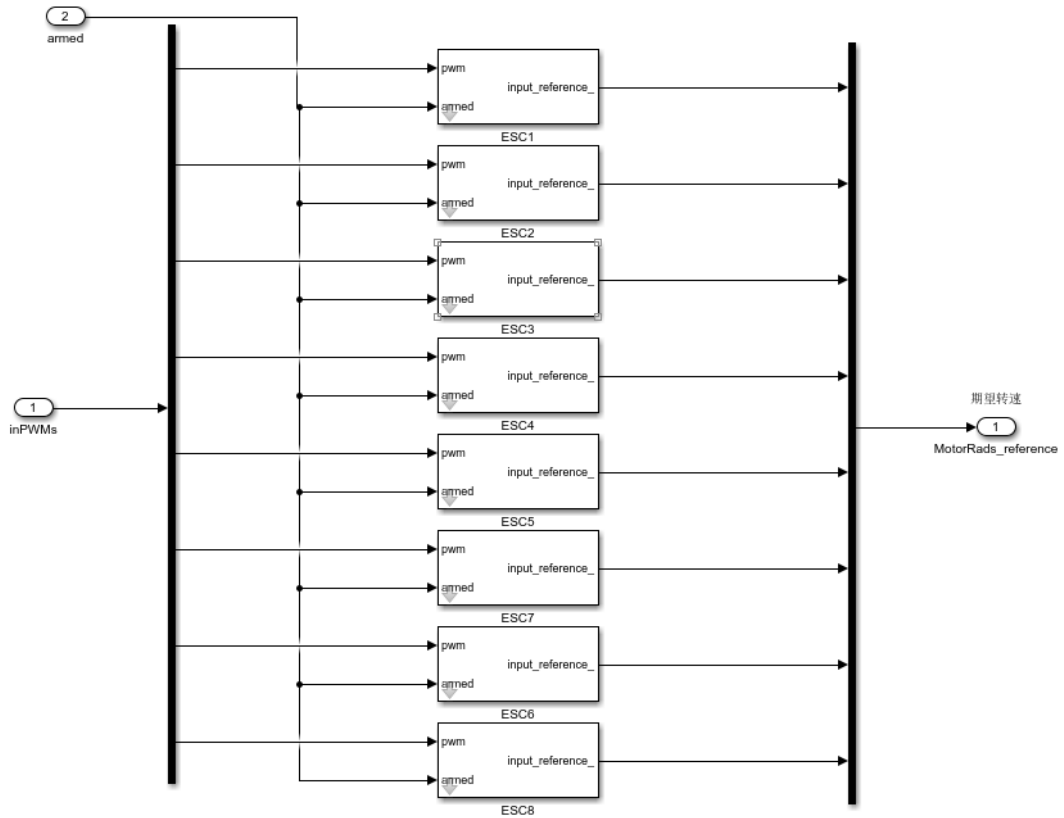
模块输入

(1) **inPWMs**: PWM 信号（8 通道，数据自 PX4 飞控回传的电机控制 MAVLink 消息 `mavlink_hil_actuator_controls_t` 的 `controls`，和 Gazebo 中对应的“xxx.sdf”文件最末尾的 `channel` 定义顺序保持一致）。

(2) **armed**: 外部输入的布尔逻辑变量，`False` 代表屏蔽 ESC_ALL 模块输出，输出为 0，反之则启用 ESC_ALL 模块输出。这里取值一般来自于 `inCopterData` 接口的第 1 维数据，该数据是来自于 `CopterSim` 的解锁标志位，这样可以保证仿真过程中未解锁时，飞机电机应该不转保持初始值，只有解锁后才能接通 `inPWMs` 输入，这样可以避免在某些情况下，开始仿真但未解锁是飞机乱动的情况。

模块输出

MotorRads_reference: 期望转速，8 个电机控制信号的期望数值，包含电机转速与升降舵等部件转动两种控制类型。`MotorRads_reference` 是一个 8 维的向量。



2) 模块参数

ESC_ALL 模块进行了 mask 封装，内部包含了 8 个 ESC 子模块。输入参数为“ModelName_init.m”文件中的“ESC”结构体向量，ESC(1)、ESC(2)、ESC(3)...等是针对每个 channel 通道的参数结构体，顺序与 Gazebo 模型对应“xxx.sdf”文件（文件路径为*PX4PSP\Firmware\Tools\sitl_gazebo\models）末尾的 channel 通道定义顺序一致。

“ModelName_init.m”文件中的“ESC”结构体向量如下。

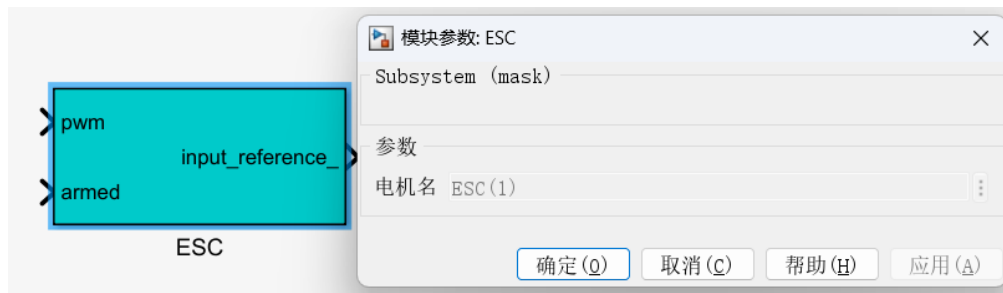
```
% 电调参数模板
ESCTmp.isEnabled= false;    %是否启用本电调
ESCTmp.input_offset = 0;    %输入偏移量
ESCTmp.input_scaling = 1;   %输入缩放系数
ESCTmp.zero_position_disarmed = 0;%零点位置解锁值
ESCTmp.zero_position_armed = 0;%零点位置锁定值
ESCTmp.joint_control_type=1;%关节控制类型，0:velocity,1:position,...

% 依据 xxx.sdf.jinja 文件末尾的 control_channels 定义 ESC 各通道顺序及 ESCTmp 参数取值。
ESC = [ESCTmp, ...
       ESCTmp, ...
       ESCTmp, ...
       ESCTmp, ...
       ESCTmp, ...
       ESCTmp, ...
       ESCTmp, ...
       ESCTmp]; %电调的数据结构体，扩充成 8 维
```

6.8.2 ESC 模块

下图为 Gazebo 模型模块里的 ESC 模块，该模块也进行了 mask 封装，输入参数为“M

odelName_init.m”文件中的“ESC(1)”结构体，一个ESC模块对应了1个电机/舵机。



输入：

(1) pwm: PWM 信号，仿真时数据来自模型 inPWMs 接口的其中一条通道，大小在-1~1 之间。

(2) armed: 解锁标志位，外部输入的布尔逻辑变量，同 ESC_ALL 模块的 armed。

输出：

input_reference_: 电机/舵机对应的期望转速，单位为转/分钟。

6.8.3 Motor_ALL 模块

1) 功能简介



输入：

(1) input_reference_: 经过 ESC 模块处理后的控制信号，期望电机转速控制类型信号输入至电机模块该接口。

(2) Windb: 从 Environment Model Bus 外部输入，代表机体相对风的速度，为 1x3 矩阵变量。

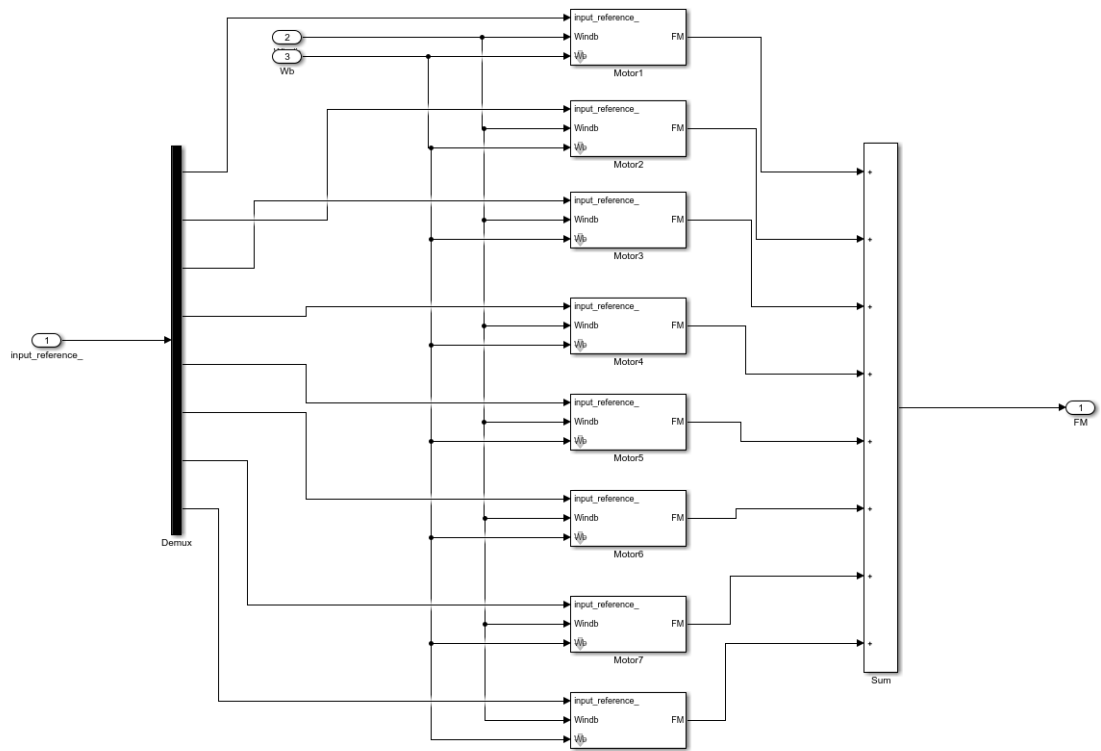
(3) wb: 从 6DOF Bus 外部输入，代表机体转动角速度。

输出：

FM: 机体坐标系下电机产生的力和力矩，6 维向量，前 3 维为电机模块产生的合力，后 3 维为电机模块产生的合力矩。

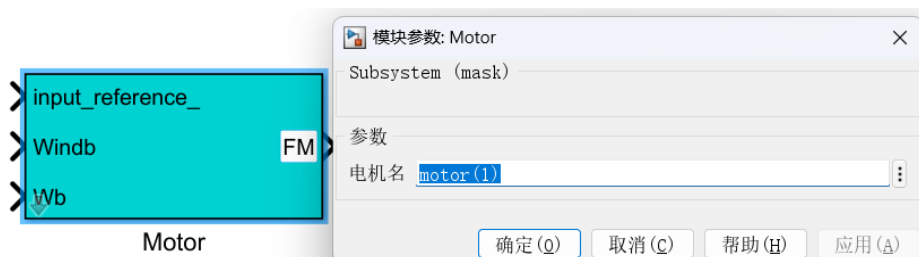
2) 模块参数

Motor_ALL 进行了 mask 封装，输入参数为“ModelName_init.m”文件中的“motor”结构体向量，参数取值参考 xxx.sdf.jinja 文件中的 motor_model 插件部分参数。内部包含 8 个 Motor 子模块，分别处理输入 input_reference_的其中一个通道控制信号，最多可支持无人机 8 个电机的控制。



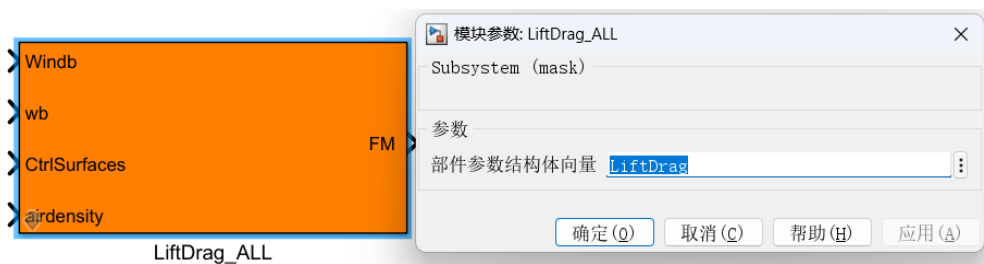
6.8.4 Motor 模块

电机子模块同样进行了 mask 封装，输入参数为“ModelName_init.m”文件中的“motor(1)”，在“ModelName_init.m”中定义并初始化，模块最终输出为单个 motor 模块的力和力矩，前 3 维为力，后 3 维为力矩。



6.8.5 LiftDrag_ALL 模块

1) 功能简介



输入：

(1) Windb: 从 Environment Model Bus 外部输入，代表机体相对风的速度，为 1x3 矩阵变量。

(2) **wb**: 从 6DOF Bus 外部输入，代表机体转动角速度。

(3) **CtrlSurfaces**: 8 维向量，来自 ESC 模块的输出信号（仅舵面控制信号）。

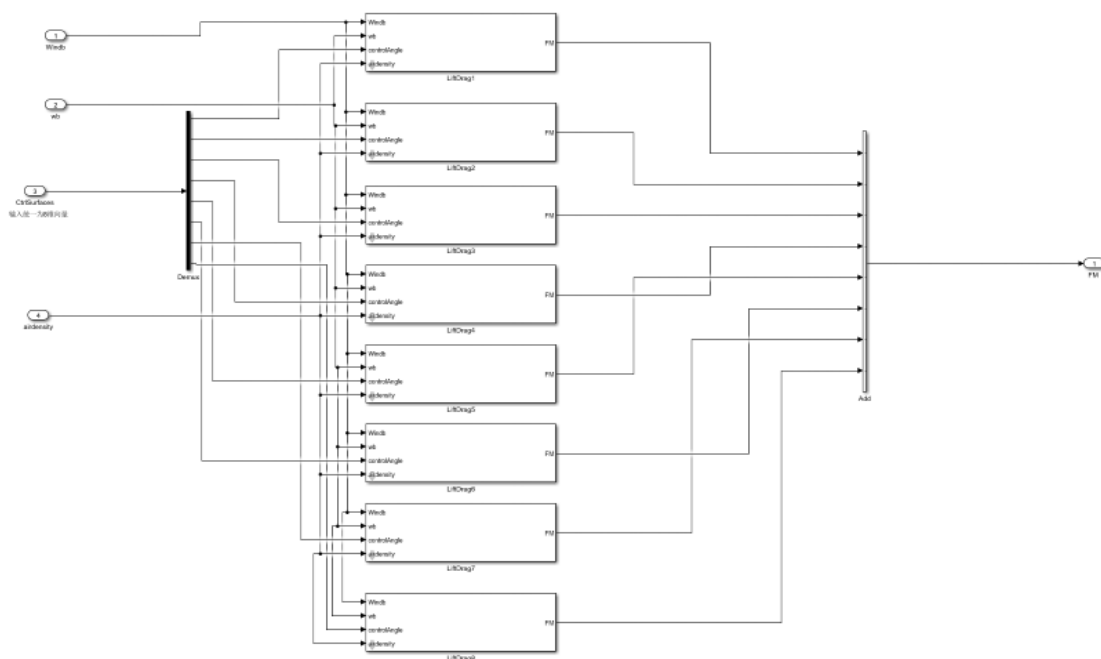
(4) **airdensity**: 空气密度，用于计算动压， $q = \frac{1}{2} \rho * V^2$ 。

输出：

FM: 升降舵、方向舵等舵面产生的机体坐标系下的力和力矩，6 维向量，前 3 维为合力，后 3 维为合力矩。

2) 模块参数

“LiftDrag_ALL” 模块经过了 mask 封装，输入参数为 “ModelName_init.m” 文件中的 LiftDrag 结构体变量，LiftDrag (1)、LiftDrag (2)、LiftDrag (3)···等是针对每个舵面，或者说升降舵、方向舵等不同部件的参数结构体，顺序与 Gazebo “xxx.sdf” 文件最后的 channel 通道定义顺序保持一致。



6.8.6 LiftDrag 模块



用于计算负责升降舵、方向舵、副翼等部件的升力、阻力及力矩，LiftDrag 模块输入参数来自 “ModelName_init.m” 文件中的 LiftDrag (1) 结构体，在 “ModelName_init.m” 中定义并初始化。

7、RflySim 已支持载具仿真操作介绍

7.1 四旋翼模型

*:\PX4PSP\RflySimAPIs\4.RflySimModel\2.AdvExps\e2_MultiModelCtrl\1.MultiModelCtrl\Readme.pdf

7.2 六旋翼

*:\PX4PSP\RflySimAPIs\4.RflySimModel \2.AdvExps\e2_MultiModelCtrl\4.HexModelCtrl\Readme.pdf

7.3 四轴八旋翼

需补充

7.4 小型固定翼

*:\PX4PSP\RflySimAPIs\4.RflySimModel\1.BasicExps\e2_FixWingModelCtrl\Readme.pdf

7.5 垂直起降无人机

7.5.1 4+1 垂起

*:\PX4PSP\RflySimAPIs\4.RflySimModel\2.AdvExps\e4_VTOLModelCtrl\1.VTOLModelCtrl\Readme.pdf

7.5.2 四旋翼尾座式垂起

*:\PX4PSP\RflySimAPIs\4.RflySimModel\2.AdvExps\e4_VTOLModelCtrl\2.TailsitterModeI Ctrl\Readme.pdf

7.6 无人车

7.6.1 阿卡曼底盘无人车

*:\PX4PSP\RflySimAPIs\4.RflySimModel\1.BasicExps\e3_CarAckermanModeCtrl\Readme.pdf

7.6.2 差动无人车

*:\PX4PSP\RflySimAPIs\4.RflySimModel\1.BasicExps\e4_CarR1DiffModelCtrl\Readme.pdf

7.7 无人船

需补充

7.8 直升机

需补充

8、外部控制接口

8.1 QGC

QGC (QGroundControl) 外部控制接口主要有以下几种:

MAVLink 消息接口: 通过 MAVLink 消息协议, 与飞控通信实现对飞行器的控制和监控。

UDP 命令接口: 通过 UDP 协议发送和接收控制指令, 实现对飞行器的控制。

TCP 命令接口: 通过 TCP 协议发送和接收控制指令, 实现对飞行器的控制。

WebSocket 接口: 通过 WebSocket 协议进行数据交换和通信, 实现对飞行器的控制和监控。

8.2 Simulink 控制接口

Simulink 中提供了多种方式来实现无人机的外部控制接口, 常用的方法有:

使用 Simulink Coder 将 Simulink 模型生成为 C 代码, 再通过 C 代码与外部控制程序进行交互。

使用 UDP 或 TCP/IP 协议在 Simulink 和外部控制程序之间进行通信。

使用 Simulink Real-Time Workshop 和硬件连接板实现实时控制。

8.3 Python 控制接口

Python 外部控制接口一般对载具的速度位置以及姿态等状态进行控制, 以下为常用接口:

固定翼起飞控制接口: “sendMavTakeOff”, 控制固定翼在指定地点起飞。

```
def sendMavTakeOff(self, xM=0, yM=0, zM=0, YawRad=0, PitchRad=0):  
    """ Send command to make aircraft takeoff to the desired local position (m)
```

解锁接口: “SendMavArm”, 载具解锁指令。

```
def SendMavArm(self, isArm=0):  
    """ Send command to PX4 to arm or disarm the drone
```

目标位置控制接口: “SendPosNED”, 在北东地坐标系下发送目标位置以及偏航角。

```
def SendPosNED(self, x=0, y=0, z=0, yaw=0):  
    """ Send vehicle target position (m) to PX4 in the earth north-east-down (NED) frame with yaw control (rad)  
    when the vehicle fly above the ground, then z < 0
```

固定翼巡航半径接口: “SendCruiseRadius”, 改变固定翼的巡航半径。

```
def SendCruiseRadius(self, rad=0):  
    """ Send command to change the Cruise Radius (m) of the aircraft
```

载具姿态发送接口: “SendAttPX4”, 在右前下坐标系下发送目标姿态。

```
def SendAttPX4(self, att=[0, 0, 0, 0], thrust=0.5, CtrlFlag=0, AltFlg=0):  
    """ Send vehicle target attitude to PX4 in the body forward-rightward-downward (FR
```

D) frame

固定翼巡航速度接口：“SendCruiseSpeed”，改变固定翼的巡航速度。

```
def SendCruiseSpeed(self, Speed=0):  
    """ Send command to change the Cruise speed (m/s) of the aircraft
```

目标位置控制接口：“SendPosNEDNoYaw”，在北东地坐标系下发送目标位置。

```
def SendPosNEDNoYaw(self, x=0, y=0, z=0):  
    """ Send vehicle targe position (m) to PX4 in the earth north-east-down (NED) fra  
me without yaw control (rad)  
    when the vehicle fly above the ground, then  $z < 0$ 
```

地面速度控制接口：“SendGroundSpeed”，控制固定翼地面速度。

```
def SendGroundSpeed(self, Speed=0):  
    """ Send command to change the ground speed (m/s) of the aircraft
```

目标速度控制接口：“SendVelNEDNoYaw”，在北东地坐标系下发送目标速度。

```
def SendVelNEDNoYaw(self, vx, vy, vz):  
    """ Send targe vehicle speed (m/s) to PX4 in the earth north-east-down (NED) fram  
e without yaw control  
    when the vehicle fly upward, the  $vz < 0$ 
```

目标速度控制接口：“SendVelNED”，在北东地坐标系下发送目标速度以及偏航角速度。

```
def SendVelNED(self, vx=0, vy=0, vz=0, yawrate=0):  
    """ Send targe vehicle speed (m/s) to PX4 in the earth north-east-down (NED) fram  
e with yawrate (rad/s)  
    when the vehicle fly upward, the  $vz < 0$ 
```

目标速度控制接口：“SendVelFRD”，在右前下坐标系下发送目标速度以及偏航角速度。

```
def SendVelFRD(self, vx=0, vy=0, vz=0, yawrate=0):  
    """ Send vehicle targe speed (m/s) to PX4 in the body forward-rightward-downward  
(FRD) frame with yawrate control (rad/s)  
    when the vehicle fly upward, the  $vz < 0$ 
```

目标速度控制接口：“SendVelNoYaw”，在右前下坐标系下发送目标速度。

```
def SendVelNoYaw(self, vx, vy, vz):  
    """ Send vehicle targe speed (m/s) to PX4 in the body forward-rightward-downward  
(FRD) frame without yawrate control (rad)  
    when the vehicle fly upward, the  $vz < 0$ 
```

目标位置控制接口：“SendPosFRD”，在右前下坐标系下发送目标位置和偏航角。

```
def SendPosFRD(self, x=0, y=0, z=0, yaw=0):  
    """ Send vehicle targe position (m) to PX4 in the body forward-rightward-downward  
(FRD) frame with yaw control (rad)  
    when the vehicle fly above the ground, then  $z < 0$ 
```

目标位置控制接口：“SendPosFRDNoYaw”，在右前下坐标系下发送目标位置。

```
def SendPosFRDNoYaw(self, x=0, y=0, z=0):  
    """ Send vehicle targe position (m) to PX4 in the body forward-rightward-downward  
(FRD) frame without yaw control (rad)  
    when the vehicle fly above the ground, then  $z < 0$ 
```

最大速度控制接口：“SendCopterSpeed”，设定旋翼的最大飞行速度。

```
def SendCopterSpeed(self, Speed=0):  
    """ send command to set the maximum speed of the multicopter
```

固定翼降落控制接口：“sendMavLand”，设定固定翼的期望将落位置。

```
def sendMavLand(self, xM, yM, zM):  
    """ Send command to make aircraft land to the desired local position (m)
```

9、参考资料

- [1]. 全权,杜光勋,赵峙尧,戴训华,任锦瑞,邓恒译.多旋翼飞行器设计与控制[M],电子工业出版社,2018.
- [2]. 全权,戴训华,王帅.多旋翼飞行器设计与控制实践[M],电子工业出版社,2020.