API 说明文件检索大纲

1、运动建模接口总体介绍

1.1RflySim 平台无人系统运动模型的设计思想

1.1.1 从不同系统提取相似组件

图 1 1 不同类型无人载具系统的共同系统结构

不同类型的无人系统（例如无人车、无人机、无人船等）的外形和运行环境不一样，但是它们从系统结构图上有非常多建模与仿真系统可以大量复用的共同的特点。因此，这里的模型框架采用了模块化的方法将整个无人模型系统划分为若干子系统，使最大化这些公共因素，使复杂的建模问题简单化。同时，这种方法也有助于不同类型的无人系统之间共享相同的模型，并可在基于模型的设计软件（如 MATLAB/Simulink）中自动化地实现，相应的环境配置见 2、环境配置。

1.1.2 从组件模型到整机模型

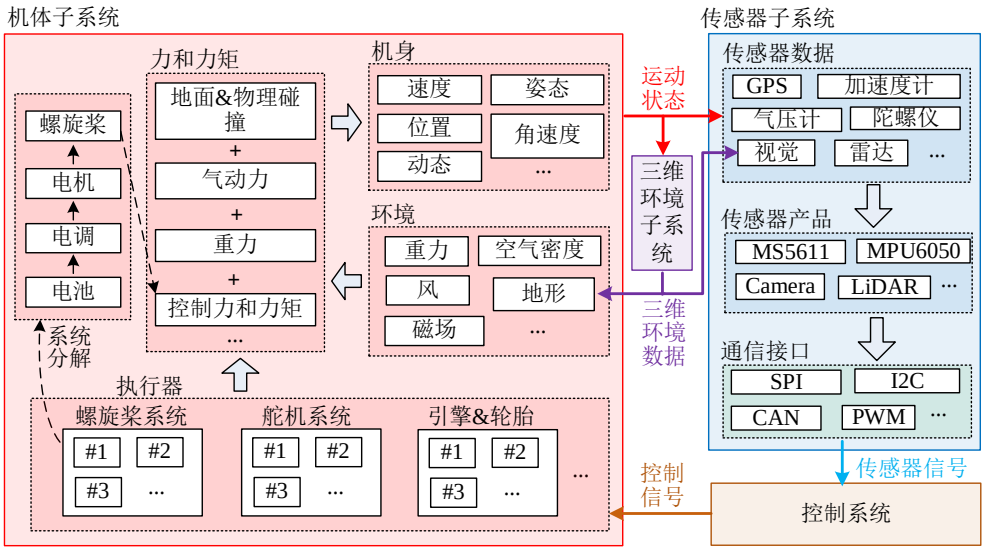


图 1 2 运动模型统一框架

无人系统载具运动模型可以通过物理与虚拟组件分解为图 1 2 所示的统一建模框架。在该框架中，运动模型系统整体可以细分为三个子系统：机体子系统、传感器子系统和三维环境子系统。

- 机体子系统包含了执行器、机身、运行环境、力与力矩等内部子系统模块，是对机体在环境的运动、能耗和故障特性的整体描述；
- 传感器模型主要用于描述控制软件之外的所有电子硬件模型，主要包含传感器数据、通信协议、连接接口等特性；
- 三维环境模型主要用于描述无人机飞行的三维视景环境（包括树木、障碍物、公路等），用于为自主控制系统提供视觉数据的模拟。

每个子系统又可以向内细分为更小的独立子系统模块，最终形成了上图所示的模块化统一建模框架。

1.1.3 通用建模接口

操作流程见 3、基本操作流程

通用输入输出接口见 5、DLL/SO 模型与通信接口

1.2RflySim 平台运动模型总体分类

1.2.1 动力学模型

不带控制器，仅根据无人系统的动力学特性建模，见 6、Simulink 建模模板介绍

1.2.2 综合模型

在原有动力学模型的基础上实现控制器，见 9. 综合模型

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，

安装步骤参考..\1.RflySimIntro\2.AdvExps\e6_VisualStudioInstall\Readme.pdf

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

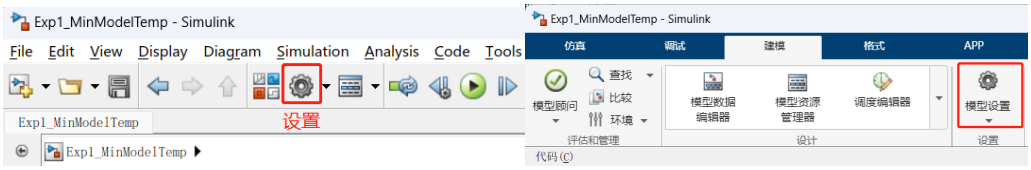
在 MATLAB 的命令行窗口中输入指令“mex -setup”，一般来说会自动识别并安装上 VS 2017 编译器，如右图所示显示“MEX 配置使用‘Microsoft Visual C++ 2017’以进行编译”说明安装正确。

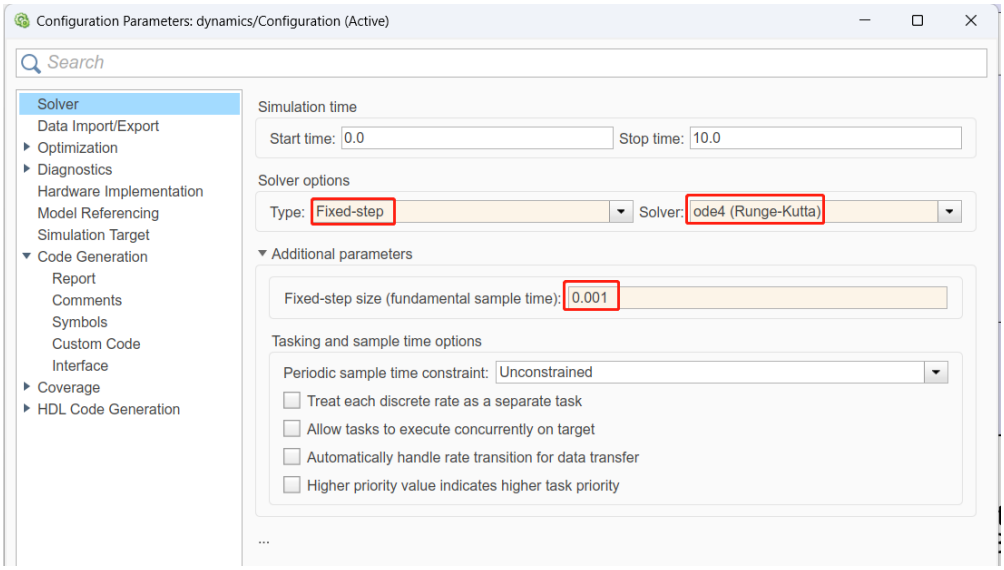
2.3 Simulink 代码生成设置

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。

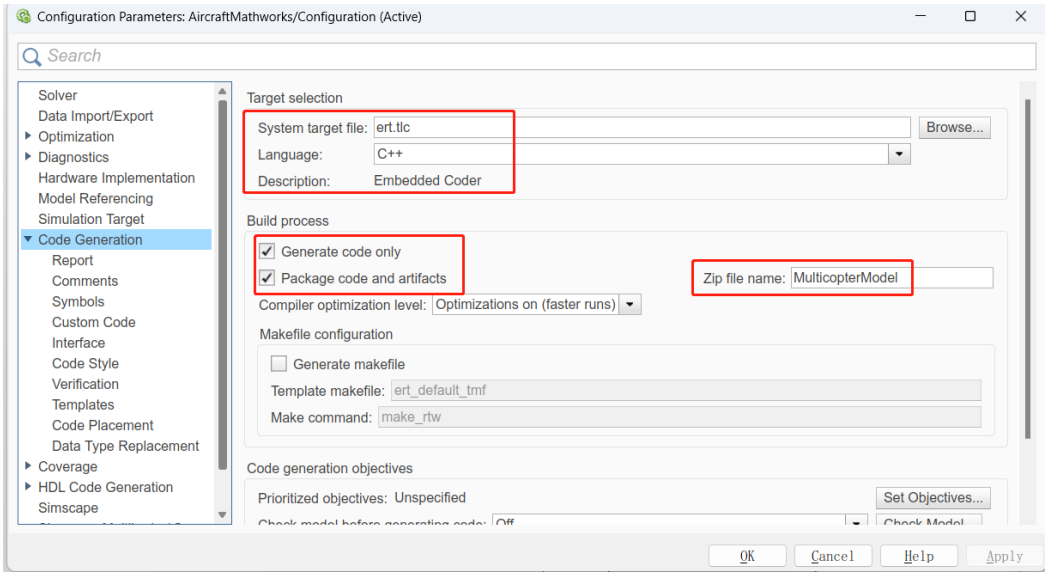
Simulink 模型编译设置步骤如下：

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）。

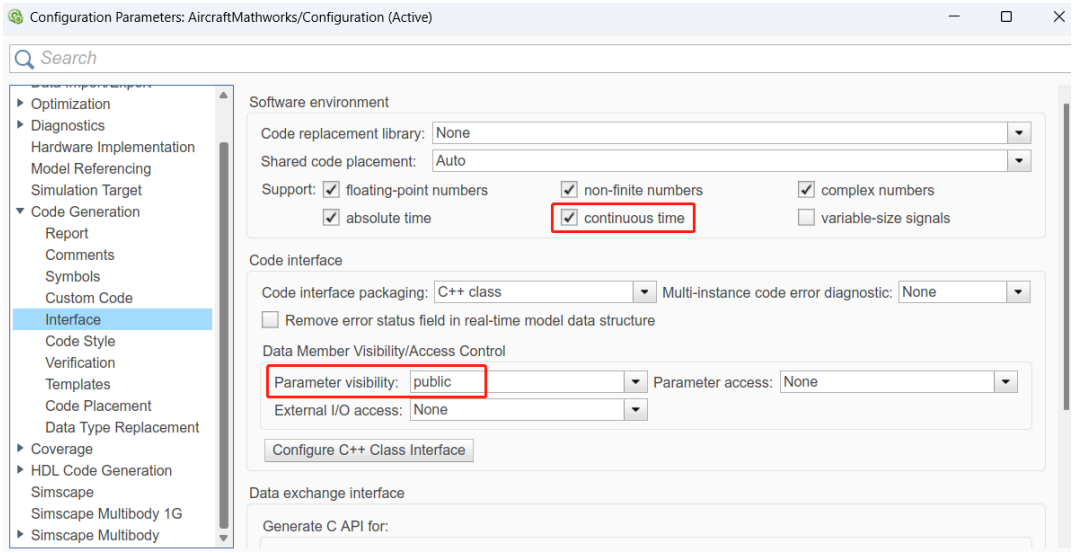




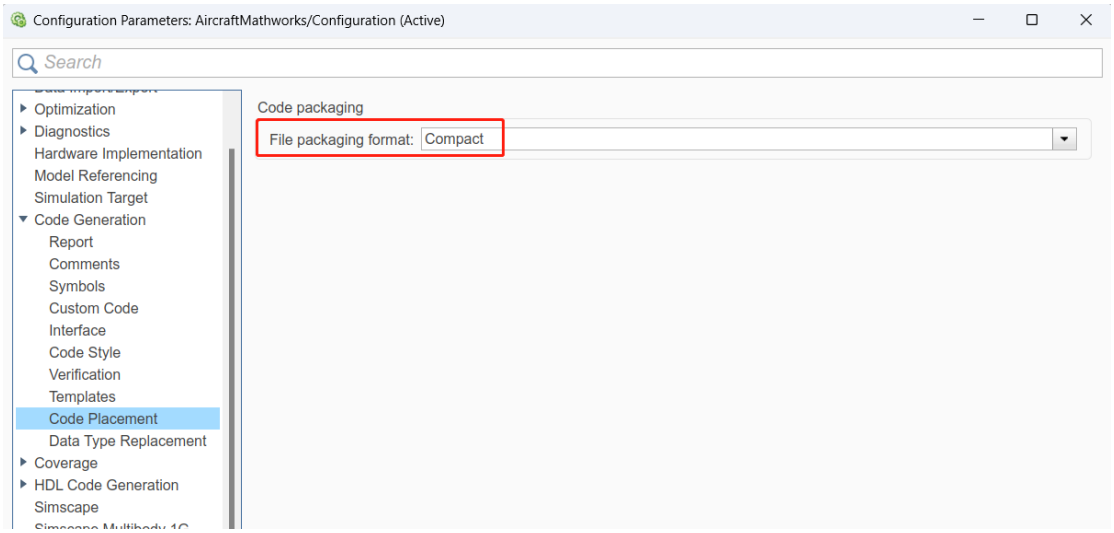
Step 2：代码生成方式选择 ert.tlc，可用于 windows， Linux 和各类嵌入式平台；语言选择 C++，便于通过继承方式调用生成代码；编译过程选择“代码和工具打包”，Zip 文件名设置为“MulticopterModel”。



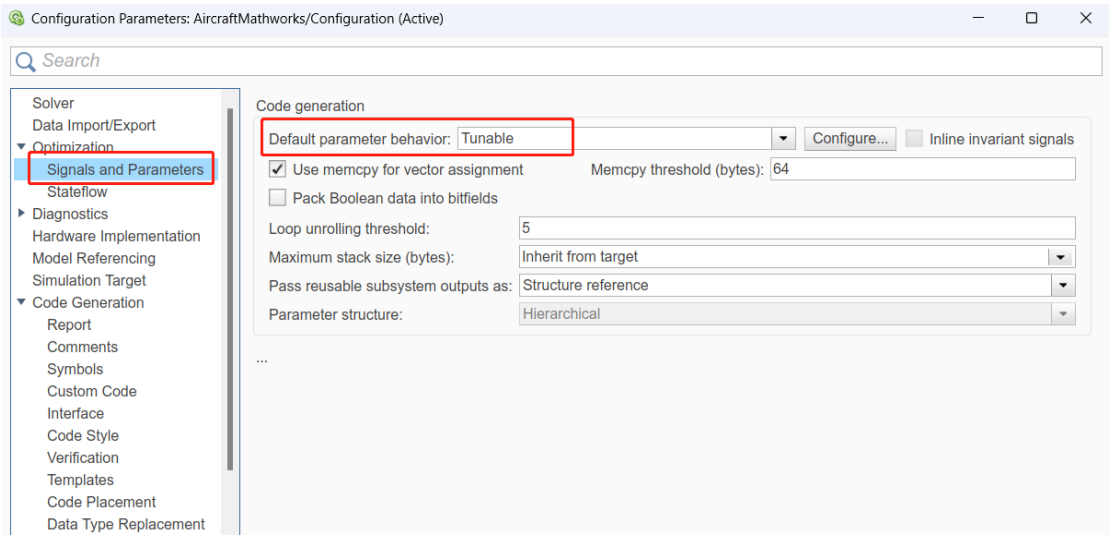
Step 3：因为包含连续模块（积分模块）因此需要勾选 continuous time，不然编译报错。此外将参数可见性 Parametervisibility 设为 public，参数结构体为共有变量，便于访问。



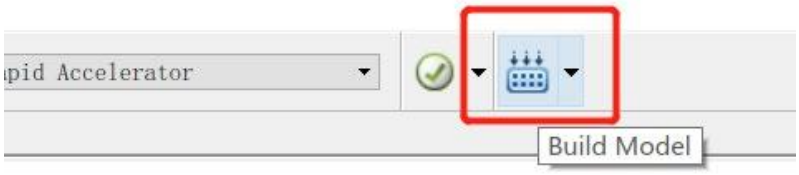
Step 4：在 Codeplacement 页设置文件打包类型为 compact，尽量避免生成多余文件，使得代码的可读性最强。



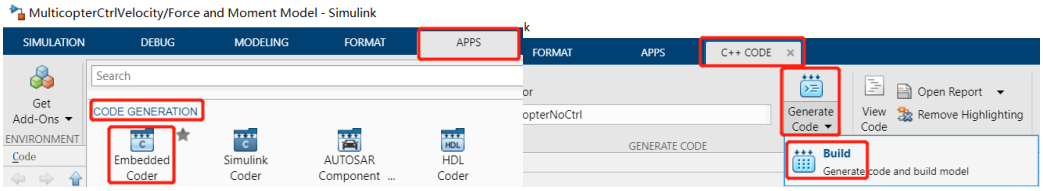
Step 5：设置参数为 Tunable(可调)，使得我们可以运行时修改参数。注： inline 形式更省内存，但是不便于访问参数，不便于实现参数实时修改或者模型故障注入。



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。



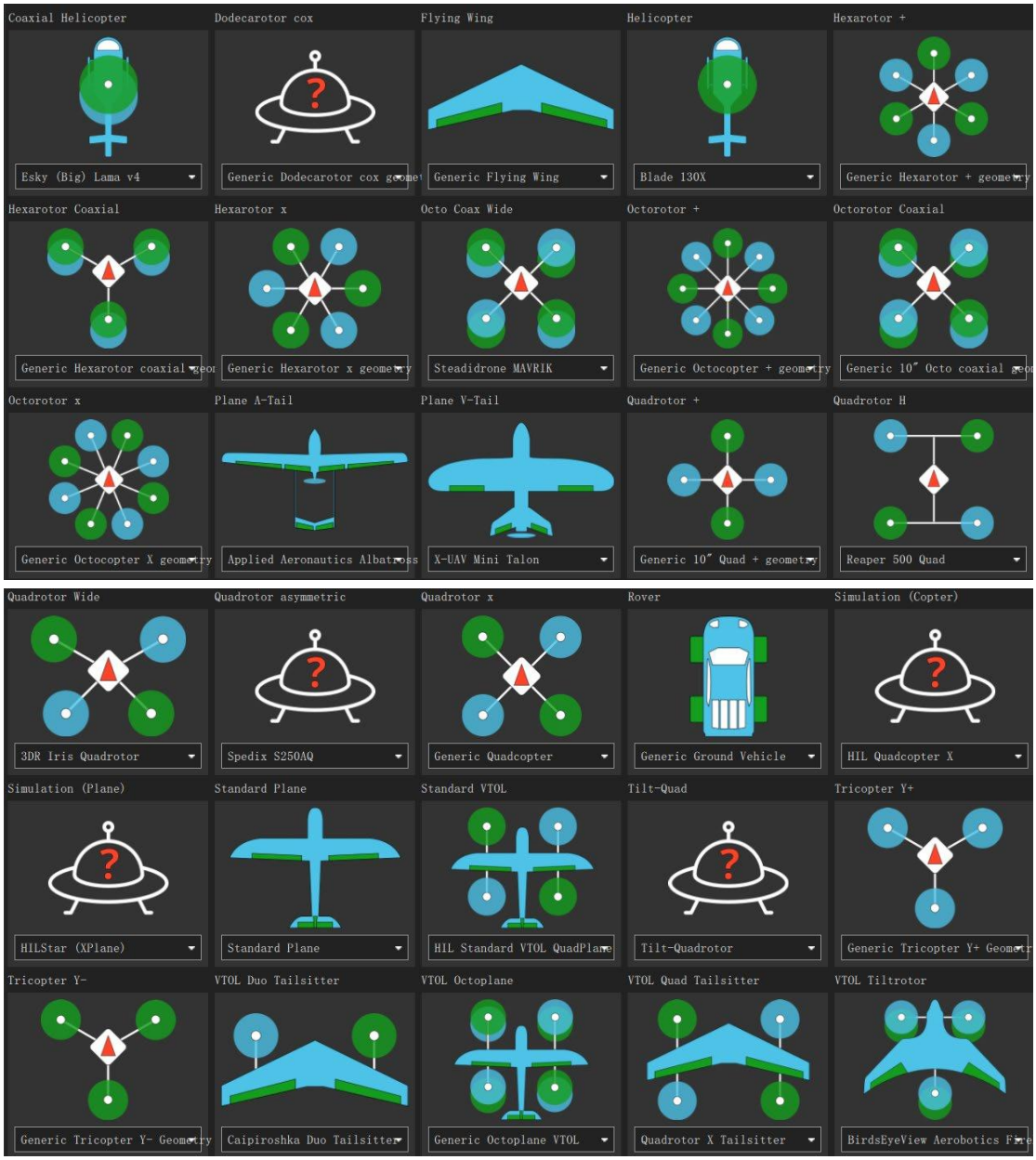
2.4 Linux 版环境配置

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。

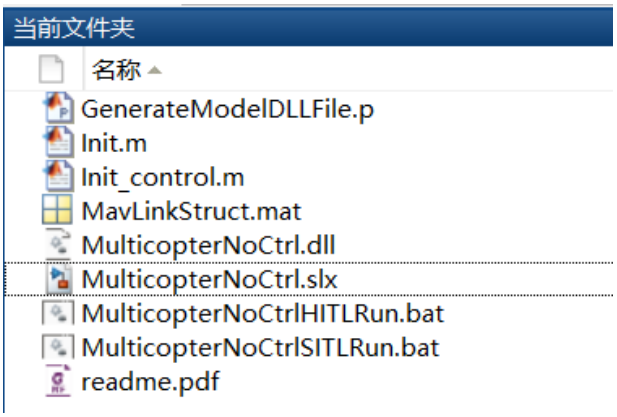


目前 RflySim 包含了旋翼、固定翼、无人车、无人船、标准垂直起降无人机、四旋翼尾座式垂起无人机和直升机，其他模型需要用户按照 RflySim 模型模板在 Simulink 中搭建。

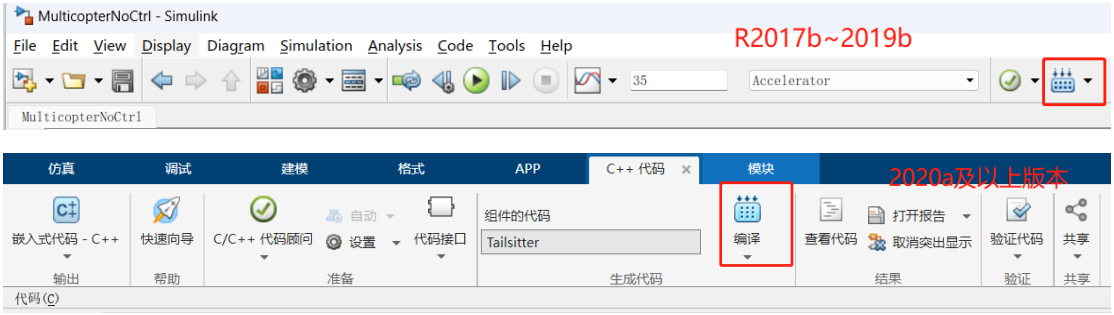
RflySim 模型模板见 RflySimAPIs3.0\4.RflySimModel\0.SourceCode\DLLModelTemp。评价模型开发完成的标准为：模型逻辑正确；运行不报错。

3.2 模型编译生成 C/C++文件

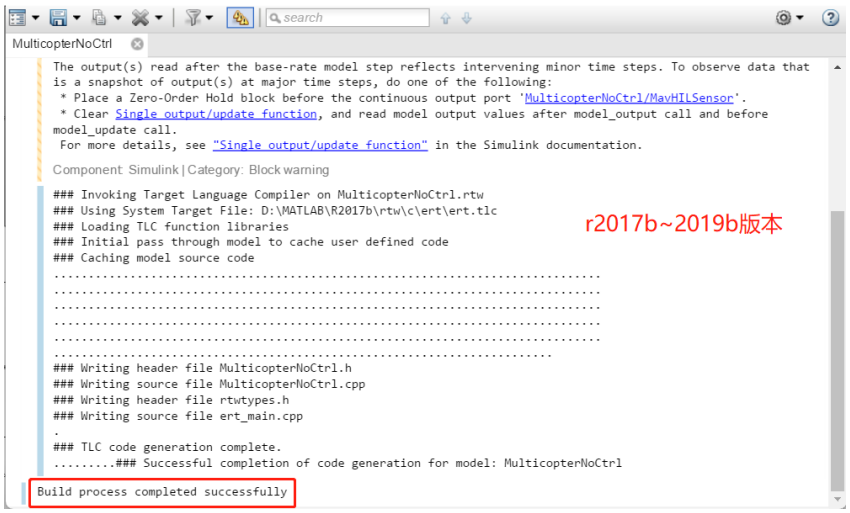
载具 Simulink 模型开发完成后，通过 MALTAB 打开模型源文件（例如，四旋翼为 MulticopterNoCtrl.slx）。



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）。

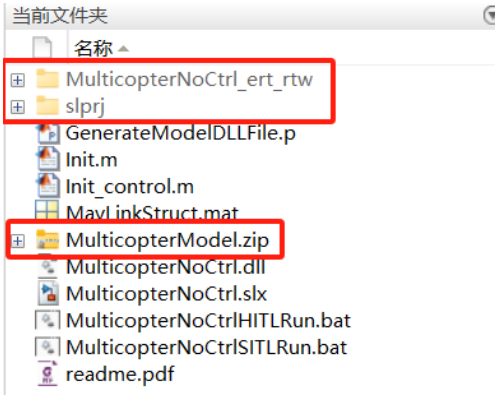


在 Simulink 的下方点击 View diagnostics 指令，即可弹出诊断对话框，可查看编译过程。在诊断框中弹出 Build process completed successfully，即表示编译成功。



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。

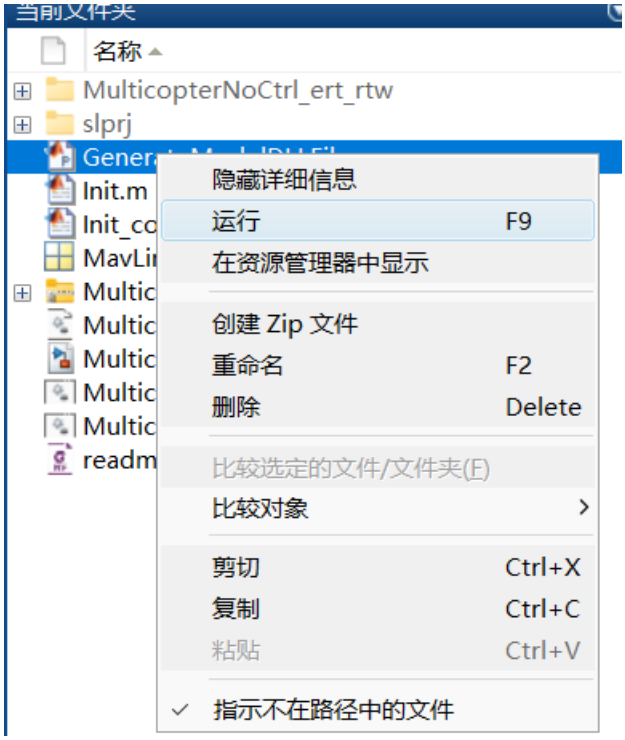


3.3 DLL/SO 模型生成

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
      cl 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

控制效率模型是指将飞行载具的控制量（如舵面偏转角度）与其产生的气动力或力矩（如升力、侧滑力、滚转力矩等）关联起来的数学模型。它可以用于描述飞行载具的稳定性和操纵性，以及不同控制量对飞行状态的影响。

控制分配是指将期望的气动力或力矩（如由 PID 控制器或其他算法计算出的期望横纵向力矩）分解为各个控制量（如各个舵面的偏转角度），并发送给相应的执行机构（如伺服电机）。它可以用于实现飞行载具的姿态和位置控制，以及优化控制性能和资源利用。

在 RflySim 平台中，控制效率模型和控制分配是通过混控文件来实现的。混控文件是一种文本文件，用于定义飞行载具的控制量名称、数量、范围、比例系数、极性等参数。这些参数决定了飞行载具的控制逻辑和特性。

为了生成混控文件，需要先确定飞行载具的机架类型，即不同控制量的布局和作用方向。PX4 官网提供了多种常见的机架类型，如四旋翼、六旋翼、飞翼、传统直升机等。用户可以根据自己的飞行载具选择合适的机架类型，或者自定义机架类型。如果选择官方提供的机架类型，那么可以直接使用官方提供的混控文件，或者根据需要进行修改。如果自定义机架类型，那么需要编写自己的混控文件。

混控文件的格式和语法可以参考 PX4 官方文档：混控器文件 | PX4 自动驾驶用户指南。混控文件的文件名通常以“.mix”为后缀，如“vtol_AAVVT_quad_x.mix”。混控文件可以放在>PX4-Autopilot/ROMFS/px4fmu_common/mixers/<目录下，或者其他任意位置。混控文件的路径和文件名需要在飞行载具的启动脚本中指定，如“set MIXER vtol_AAVVT_quad_x”。

混控文件的内容由若干个混控器组成，每个混控器对应一个控制量，如一个舵面或一个电机。每个混控器由一个标识符（如 R：旋翼或 Z：伪舵面）、一个输出通道索引（从 0

开始）、一个比例系数和一组控制输入组成。控制输入是指期望的气动力或力矩的分量，如滚转、俯仰、偏航、油门等。每个控制输入由一个符号（+或-）和一个缩放因子组成。符号表示控制量的正负方向，缩放因子表示控制量对控制输入的响应程度。例如，一个混控器可以写为：

R: 2- 10000 10000 0 -10000 10000

这表示这是一个旋翼混控器，输出通道索引为 2，比例系数为 10000，控制输入包括滚转（-）、俯仰（+）、偏航（-）和油门（+），它们的缩放因子分别为 1、1、0 和 1。这意味着这个旋翼会随着滚转指令反向旋转，随着俯仰和油门指令同向旋转，而不受偏航指令影响。

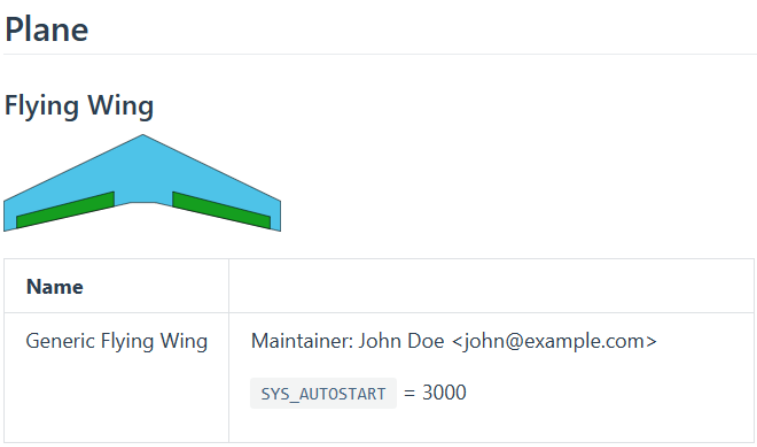
通过编写合适的混控文件，用户可以实现自己的控制效率模型和控制分配，从而控制飞行载具的飞行特性和性能。

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：

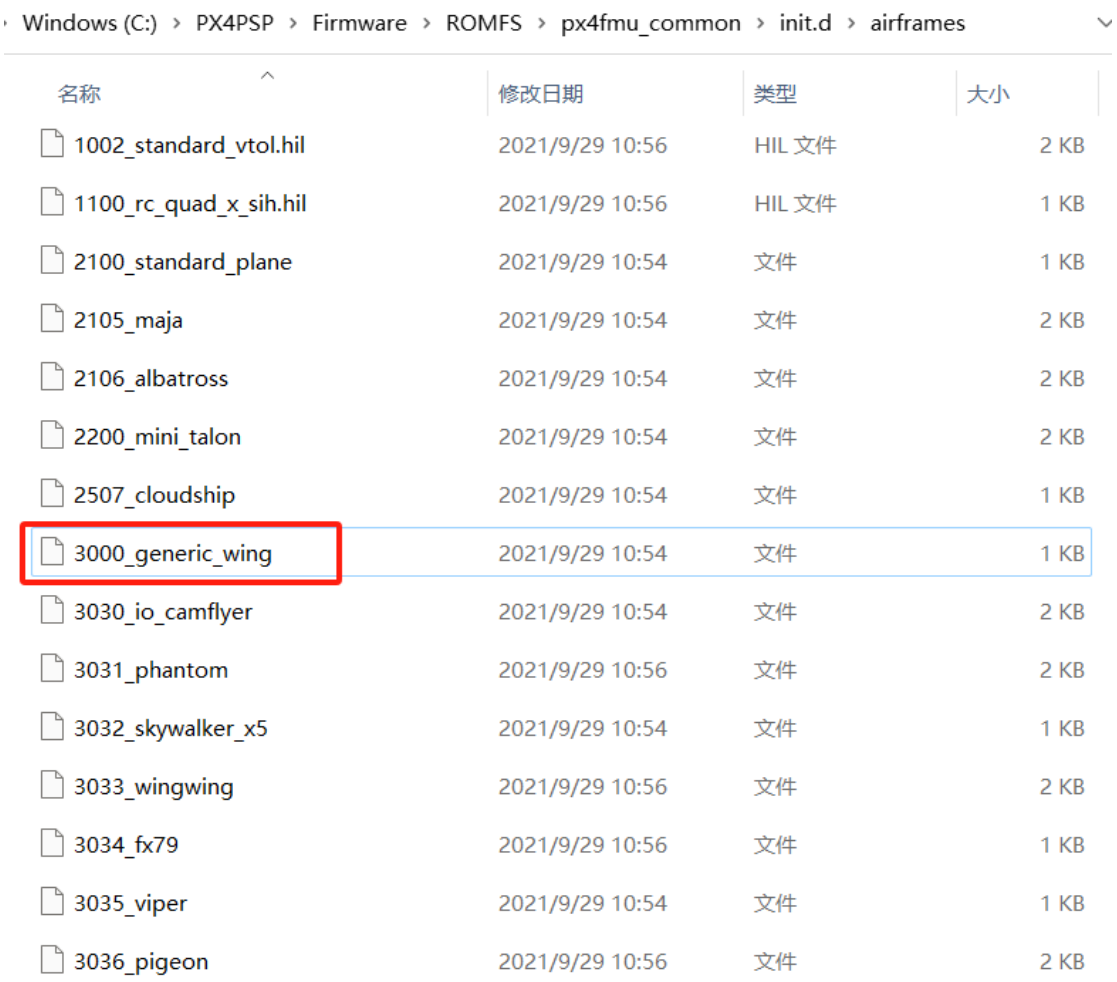
Step 1：

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。



Step 2:

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



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

```
#!/bin/sh
#
# @name Generic Flying Wing
#
# @type Flying Wing
# @class Plane
#
# @output MAIN1 left aileron
# @output MAIN2 right aileron
# @output MAIN4 throttle
#
# @output AUX1 feed-through of RC AUX1 channel
# @output AUX2 feed-through of RC AUX2 channel
# @output AUX3 feed-through of RC AUX3 channel
#
# @maintainer
#
# @board bitcraze_crazyflie exclude
#

. ${R}etc/init.d/rc.fw_defaults

set MIXER fw_generic_wing

```

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

```
Generic wing mixer
==========================

This file defines mixers suitable for controlling a delta wing aircraft.
The configuration assumes the elevon servos are connected to servo
outputs 0 and 1 and the motor speed control to output 3. Output 2 is
assumed to be unused.

Inputs to the mixer come from channel group 0 (vehicle attitude), channels 0
(roll), 1 (pitch) and 3 (thrust).

See the README for more information on the scaler format.

Elevon mixers
-------------
Three scalers total (output, roll, pitch).

On the assumption that the two elevon servos are physically reversed, the pitch
input is inverted between the two servos.

The scaling factor for roll inputs is adjusted to implement differential travel
for the elevons.

M: 2
S: 0 0  -8000  -8000      0 -10000  10000
S: 0 1   6000   6000      0 -10000  10000

M: 2
S: 0 0  -8000  -8000      0 -10000  10000
S: 0 1  -6000  -6000      0 -10000  10000

Output 2
--------
This mixer is empty.

Z:

Motor speed mixer
-----------------
Two scalers total (output, thrust).

This mixer generates a full-range output (-1 to 1) from an input in the (0 - 1)
range.  Inputs below zero are treated as zero.

M: 1
S: 0 3      0  20000 -10000 -10000  10000

```

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。

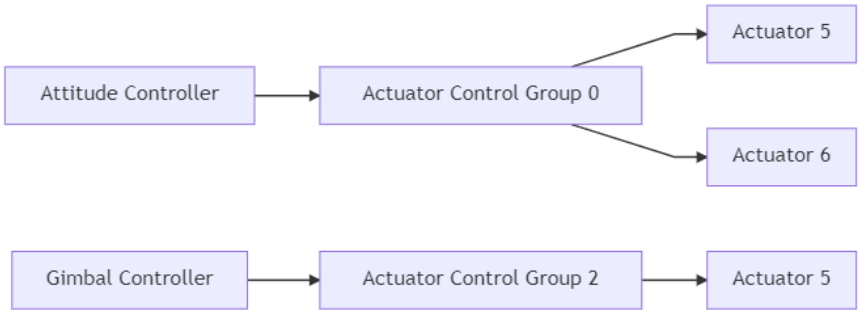
PX4 混控规则：

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>
O: <-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—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 O: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:
```

Multirotor Mixer—多旋翼混控器

Multirotor 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> <idlespeed>
```

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 5000。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 仿真一键启动 bat 脚本

3.5.1 硬件在环仿真启动脚本

```
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 TOTOAL_COPTER=8 （总飞机数量，仅在多机联机仿真才需要幅值，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 软件在环仿真启动脚本

```
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）
```

3.5.3 初始化姿态、高度仿真启动脚本

（1）脚本介绍

SITL/HITLPosAtt.bat、SITL/HITLPosAttStr.bat 支持在启动仿真时配置无人载具的初始位置和姿态。

SITL/HITLPosAtt.bat 脚本支持通过交互页面分别输入期望初始化 x、y、z（北东地坐标系，向下为正）、yaw、roll 和 pitch。

```
    REM Set Copters' x&y Position vector on the map
    SET /P PosXStr=Please enter the PosX (m) list:
    SET /P PosYStr=Please enter the PosY (m) list:
    SET /P YawStr=Please enter the Yaw (degree) list:

    SET /P AltStr=Please enter the PosZ (m) list:

    SET /P RollStr=Please enter the Roll (degree) list:
    SET /P PitchStr=Please enter the Pitch (degree) list:
```

SITL/HITLPosAttStr.bat 支持在脚本中设定初始化位置和姿态值，并支持设定多架无人载具的初始化位置和姿态值，通过“,”进行排列。例如，以下是以自定义期望位置和姿态启动 2 架无人载具的 zSITLPosStr.bat 设定方式，按以下修改 bat 脚本保存并运行，CopterSim 会按照 bat 脚本中位置姿态信息对无人载具进行初始化。

```
    REM Or comment the above code to use the following form
    SET PosXStr=1,2
    SET PosYStr=0,0
    SET YawStr=10,0
    SET AltStr=-10,-20
    SET RollStr=0,0
    SET PitchStr=0,0
```

（2）实现原理

Dll 模型中开放了初始化位置和姿态参数接口，CopterSim 支持传入初始化位置和姿态参数，如下所示，bat 语句中设定了在启动 CopterSim 时传入该参数。因此，在运行 SITL/HITLPosAtt.bat 后，CopterSim 以期望初始化位置和姿态值调用了 DLL 模型，同时启动了 RflySim3D 和 QGC，并完成了对应端口设置。

```
    start                                /realtime                                CopterSim.exe
1  %cntr%  %portNum%  %DLLModel%  %SimMode%  %UE4_MAP%  %IS_BROADCAST%  %xPos%  %yPos%  %yawAng%
1 %UDPSIMMODE% %IsSysID% %alt%,%rollAng%,%pitchAng%
```

同理，可实现以期望初始化位置和姿态值启动硬件在环仿真。

（3）应用场景

1）无人机初始化在空中

在 DLL 模型中对无人载具（多旋翼、固定翼、无人车、水下潜航器等）实现了基于机理的编程，以多旋翼来说，多旋翼 DLL 模型中包括了电机模块、力和力矩模块、物理碰撞引擎模块、传感器&GPS 模块和三维显示模块，由于力和力矩模块、物理碰撞引擎模块的

存在，以及初始化时没有额外的电机控制输入，因此当多旋翼初始化在空中时，多旋翼会以自由落体的方式往下掉，直到掉至地面上。当多旋翼具备初始化俯仰和滚转角，由于力和力矩模块，多旋翼的俯仰和滚转角会最终趋近于 0。偏航角可以保持。

0.ApiExps\21.RunPosAttBat\1.MultiModelCtrl\Readme.pdf

2）水下潜航器初始化在水下

UUV DLL 模型中包括了电机模块、力和力矩模块、传感器&GPS 模块和三维显示模块，力和力矩模块中，对水下潜航器浮力、重力、阻尼力、科里奥利力、电机所产生的力以及对应的力矩进行了建模，与多旋翼不同的是，水下潜航器在水下时，当浮力=重力时，水下潜航器能维持当前状态。因此，通过该脚本将水下潜航器初始化在水下并设定期望姿态值，水下潜航器能保持该状态。

0.ApiExps\21.RunPosAttBat\2.UUV\Readme.pdf

3）综合模型以特定高度、姿态直接开始任务

综合模型仿真区别于 PX4 软硬件在环仿真，综合模型中既有运动学模型，也有控制器，综合模型接收的是上层控制指令，如期望位置、期望速度等。

因此，通过 SITL/HITLPosAtt.bat、SITL/HITLPosAttStr.bat 启动综合模型仿真，同时通过 UDP 的方式向综合模型发送指令，可以实现无人载具以特定高度、姿态初始化后直接开始任务。

0. ApiExps\21.RunPosAttBat\3.CopterSimSILNoPX4\Readme.pdf

3.5.4 初始化 GPS 配置脚本（限完整版）

（1）脚本介绍

SITL/HITLPosStrLLA.bat 支持在启动仿真时用 GPS 坐标（经纬高）配置无人载具的初始位置。isPosGps 设置为 0 时，PosXStr，PosYStr 分别表示北东地坐标系下 xy 坐标；isPosGps 设置为 1 时，PosXStr，PosYStr 分别表示纬度和经度，并支持设定多架无人载具的初始位置，通过“,”进行排列。例如，以下是以自定义期望经纬度和偏航启动 2 架无人载具的设定方式，按以下修改 bat 脚本保存并运行，CopterSim 会按照 bat 脚本中位置姿态信息对无人载具进行初始化。

```
REM Or comment the above code to use the following form
SET PosXStr=40.100001,40.100002
SET PosYStr=116.2,116.2
SET YawStr=0,0
REM set isPosGps to 1 if PosXStr and PosYStr are gps lat and long value
SET isPosGps=1
```

同时 SITL/HITLPosStrLLA.bat 还支持在启动仿真时设定 GPS 原点（这个 GPS 原点作用在于作为参考点最终将北东地坐标系下飞机的坐标转换为 GPS 坐标，坐标转换过程参考 ModelParam_GPSLatLong）。isBatLLAOrin 设为 1 启用 bat 脚本配置 GPS 原点，LatLongAlt 填上 GPS 坐标。这样就不需要去修改地图校准文件 txt 或 dll 模型初始参数 ModelParam_GPSLatLong 来配置 GPS 原点了。

```
REM set isBatLLAOrin to 1 if use LatLongAlt instead of LLA in model or map/txt
```

```
SET isBatLLAOrin=1
REM use LatLongAlt to the GPS origin if isBatLLAOrin is set to 1
SET LatLongAlt=22.3170295,114.1601504,50
```

（2）实现原理

Dll 模型中开放了 GPS 初始化位置和 GPS 参考点接口，CopterSim 支持传入 GPS 初始化位置和 GPS 参考点参数，如下所示，bat 语句中设定了在启动 CopterSim 时传入 GPS 位置设置模式 isPosGps、GPS 参考点相关参数 isBatLLAOrin 和 isBatLLAOrin，汇总成逗号分隔的字符串，作为第 15 个输入参数。因此，在运行 SITL/HITLPosStrLLA.bat 后，CopterSim 以期望初始化位置和 GPS 参考点调用了 DLL 模型，同时启动了 RflySim3D 和 QGC，并完成了对应端口设置。

```
start /realtime CopterSim.exe 1 %cntr% %portNum% %DLLModel% %SimMode% %UE4_MAP% %IS_BROADCAST% %xPos% %yPos% %yawAng% 1 %UDPSIMMODE% %IsSysID% %IsAltAng% %isPosGps%,%isBatLLAOrin%,%LatLongAlt%
```

同理，可实现以期望初始化位置和 GPS 参考点启动硬件在环仿真。

（3）应用场景

1）集群场景中用 GPS 表示飞机初始位置

相隔较远的飞机从不同地点起飞（如北京和长沙）

3.6 仿真控制接口 ReqCopterSim.py

RflySim 提供了 ReqCopterSim.py 接口库来获取 CopterSim 电脑 IP，发送指令对 CopterSim 相关参数进行信息重置。因此用户在使用 RflySim 平台进行仿真时，除了通过 SITL/HITLRun.bat 脚本对初始化参数进行设置以外，还可以通过 python 的方式进行初始化参数设置，下面对 ReqCopterSim.py 接口库中关键函数进行介绍。

3.6.1 DLL 模型重置函数

CopterSim 所调用 DLL 模型的重置函数有两种，第一种为通过 DLL 模型名称进行重置：

```
def sendReSimDllName(self,CopterID=1,name=''):
功能：该函数可以通过 DLL 模型名称修改 CopterSim 所调用的 DLL。
说明：
Self：表明 ReqCopterSim 共享对象会影响该函数的行为；
CopterID：CopterSim ID 号，默认为 1.
name：DLL 模型名称。
注：要注意的是，这里默认搜索的 DLL 模型名称路径为 C:\PX4PSP\CopterSim\external\model（以平台安装在 C 盘为例进行说明），因此在使用该接口修改 DLL 模型时要确保 C:\PX4PSP\CopterSim\external\model 路径下是否存在新的 DLL 模型。
```

第二种为通过 DLL 模型序号进行重置：

```
def sendReSimDllIdx(self,CopterID=1,index=-1):
功能：该函数可以通过 DLL 序号来修改 CopterSim 所调用的 DLL。
说明：
Self：表明 ReqCopterSim 共享对象会影响该函数的行为；
CopterID：CopterSim ID 号，默认为 1.
index：选项 ID 序号，这里对应了 DLL 的序号。
```

3.6.2 三维场景重置函数

三维场景重置函数有两种，一种为通过三维场景名称进行重置：

```
def sendReSimMapName(self,CopterID=1,name=''):
功能：该函数可以通过地图名称来修改三维引擎仿真场景。
说明：
Self：表明 ReqCopterSim 共享对象会影响该函数的行为；
CopterID：CopterSim ID 号，默认为 1.
name：地图名称。
```

第二种为通过地图序号进行重置：

```
def sendReSimMapIdx(self,CopterID=1,index=-1):
功能：该函数可以通过地图序号来修改三维引擎仿真场景。
说明：
Self：表明 ReqCopterSim 共享对象会影响该函数的行为；
CopterID：CopterSim ID 号，默认为 1.
index：选项 ID 序号，这里对应了地图的序号。
```

3.6.3 指定 CopterID 数据获取函数

该函数能指定本机或联机的 CopterID，并请求该 Copter ID 将数据回传至本电脑。

```
def sendReSimIP(self,CopterID=1):
功能：请求指定 CopterSim 将数据回传到本电脑，适用于联机设置 Coptersim 消息。
说明：
Self：表明 ReqCopterSim 共享对象会影响该函数的行为；
CopterID：指定 CopterSim ID 号。
```

3.6.4 xyYaw 重设函数

```
def sendReSimXYyaw(self,CopterID=1,xyYaw=[0,0,0]):
功能：请求指定 CopterSim 更换 x、y、yaw 为指定值，x、y 单位为米，北东地，yaw，单位为度
说明：
Self：表明 ReqCopterSim 共享对象会影响该函数的行为；
CopterID：指定 CopterSim ID 号。
xyYaw=[0,0,0]：分别为期望 x、y、yaw。
```

3.6.5 初始化位置、姿态重设函数

```
def sendReSimXyzRPYaw(self,CopterID=1,Xyz=[0,0,0],RPYaw=[0,0,0]):
功能：请求指定 CopterSim 更换 xyz 为指定值，单位 m，北东地，roll、pitch、yaw 为指定值，单位度。
说明：
Self：表明 ReqCopterSim 共享对象会影响该函数的行为；
CopterID：指定 CopterSim ID 号。
Xyz=[0,0,0]：期望 xyz。
RPYaw=[0,0,0]：期望滚转、俯仰、偏航。
对应 CopterSim 界面
```

3.6.6 GPS 坐标转换参考原点重设函数（限完整版）

```
    def sendReGPSOrin(self,CopterID=1,LLA=[0,0,0]):
功能：请求指定 CopterSim 更换 GPS 原点为指定值
说明：
LLA -> lat (degree), lon (degree), alt (m, 向上为正)
对应 CopterSim 界面
```

3.6.7 载具 GPS 初始化位置重设函数（限完整版）

```
    def sendReGPSPos(self,CopterID=1,lat=0,lon=0,yaw=0):
功能：请求指定 CopterSim 更换坐标为指定 GPS 值
对应 CopterSim 界面
```

3.6.8 指定 CopterSim 坐标显示模式为 GPS（限完整版）

```
    def sendEnGpsMode(self,CopterID=1):
功能：请求指定 CopterSim 更换 GPS 显示模式
对应 CopterSim 界面
```

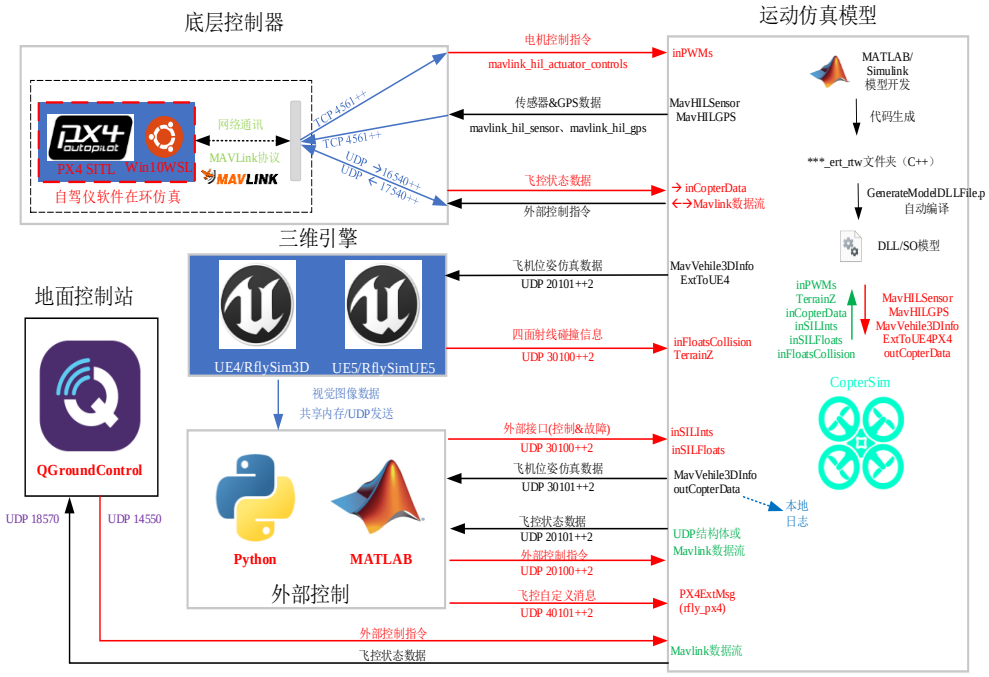
3.6.9 指定 CopterSim 坐标显示模式为 xy 坐标（限完整版）

```
def sendEnXyMode(self,CopterID=1):
```

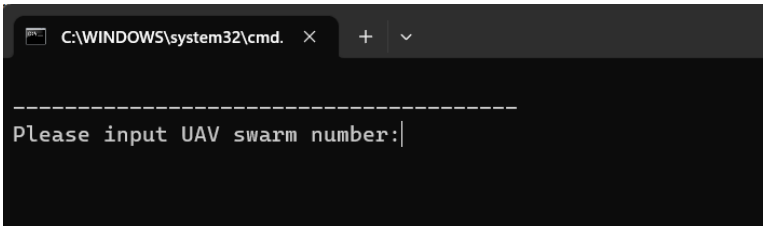
功能：请求指定 CopterSim 更换 xy 显示模式

3.7 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 PX4PSP\Firmware\src\modules\simulator\simulator_mavlink.cpp 来处理这些消息。



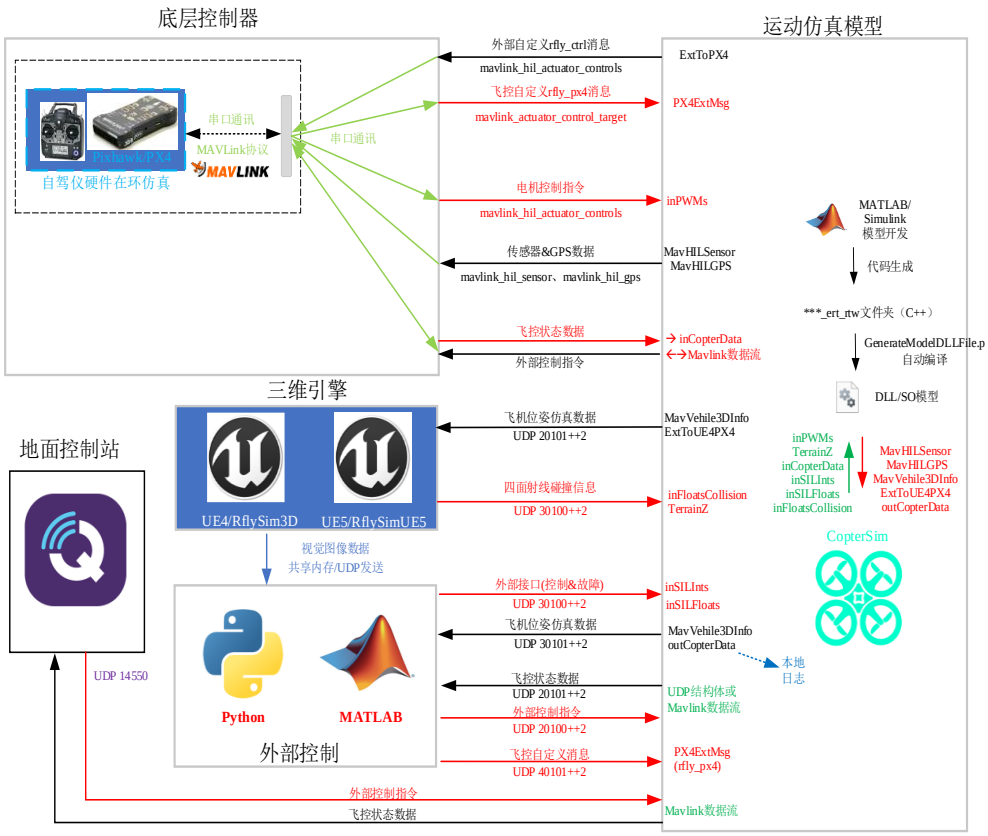
运行指定 SITLRun.bat 脚本后，会弹出提示框，在光标处输入数字，即可在三维场景中创建对应数量的无人载具，并完成初始化。单独 1 台电脑支持多机仿真，可同时仿真的无人载具数量由计算机性能和通信负载决定。



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。

3.8 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：

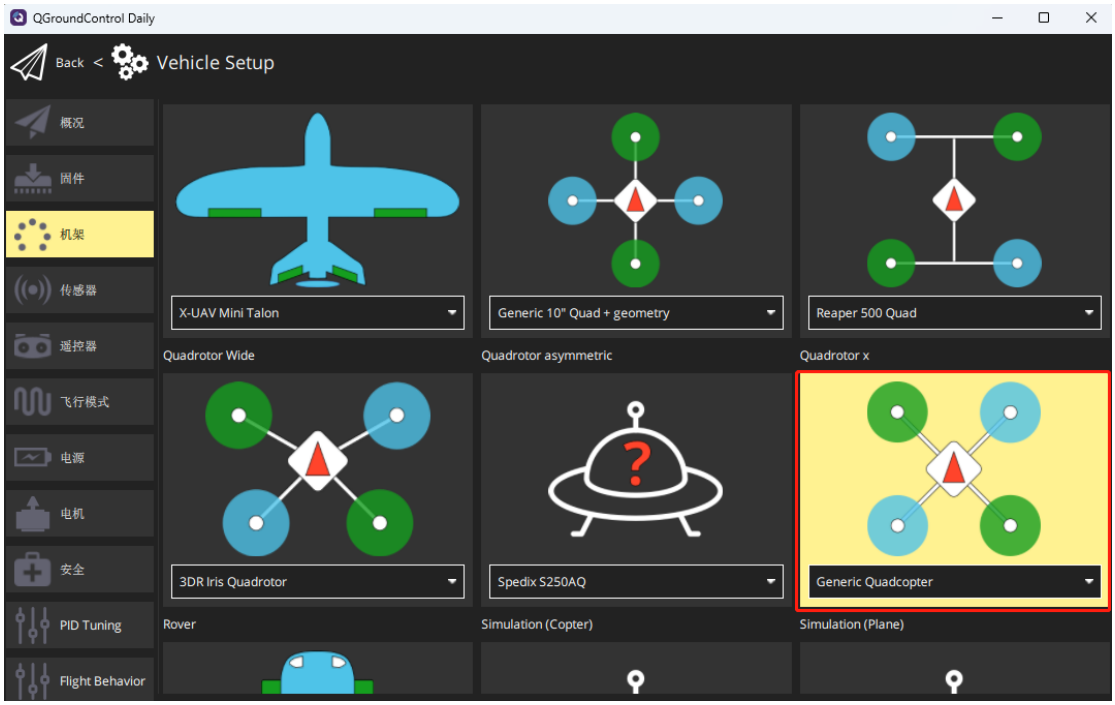
注：例程路径为*:\PX4PSP\RflySimAPIs\4.RflySimModel\2.AdvExps\e2_MultiModelCtrl\2.MultiModelCtrl

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。

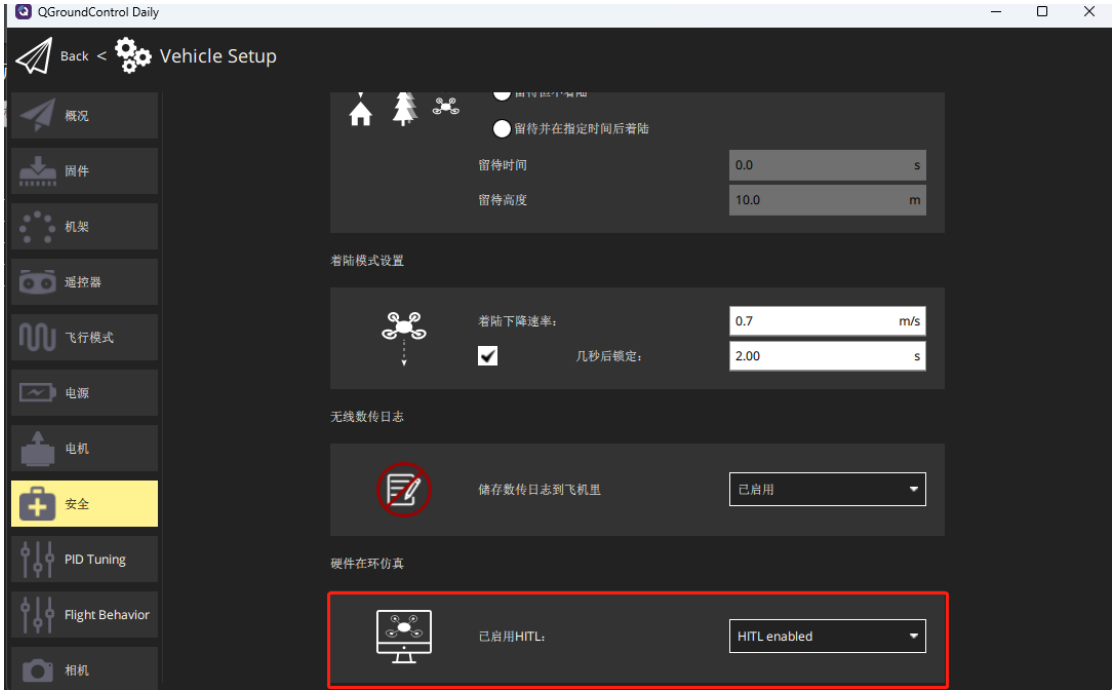
Step 2：通过 USB-TypeC 线将飞控连接至电脑 USB 端口。

Step 3：在 Rflytools 文件夹中打开 QGC 地面站。

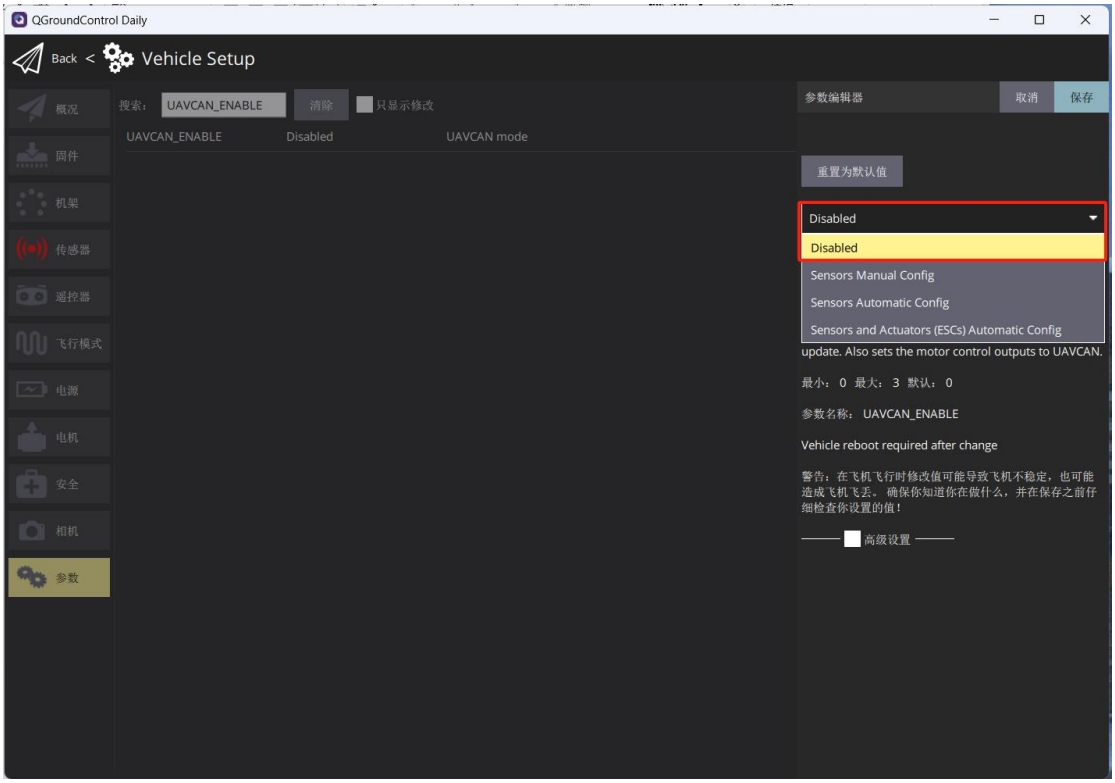
在机架界面设置机架型号为“Generic Quadcopter”（机架由仿真机型及对应的官方机架文件决定），设置完毕后点击右侧“应用并重启”。



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。



3.9 通过 QGC 或遥控器进行仿真测试

3.10 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图 5 1。

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。

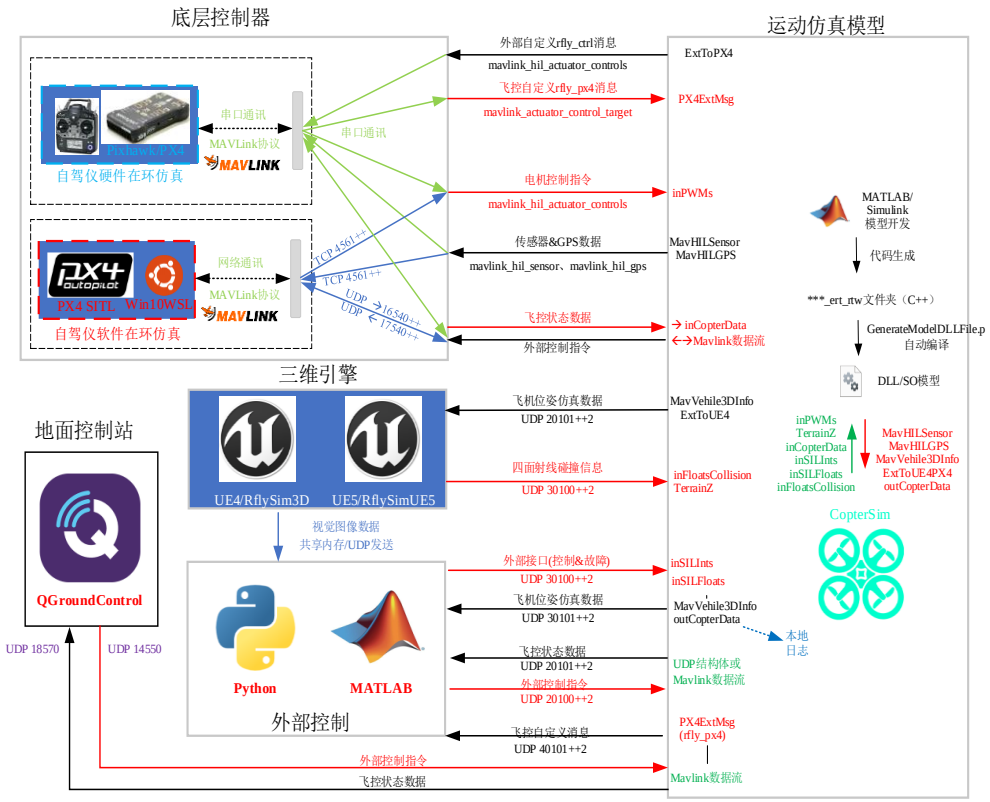


图 5 1 运动仿真模型与其他模块数据交互图解

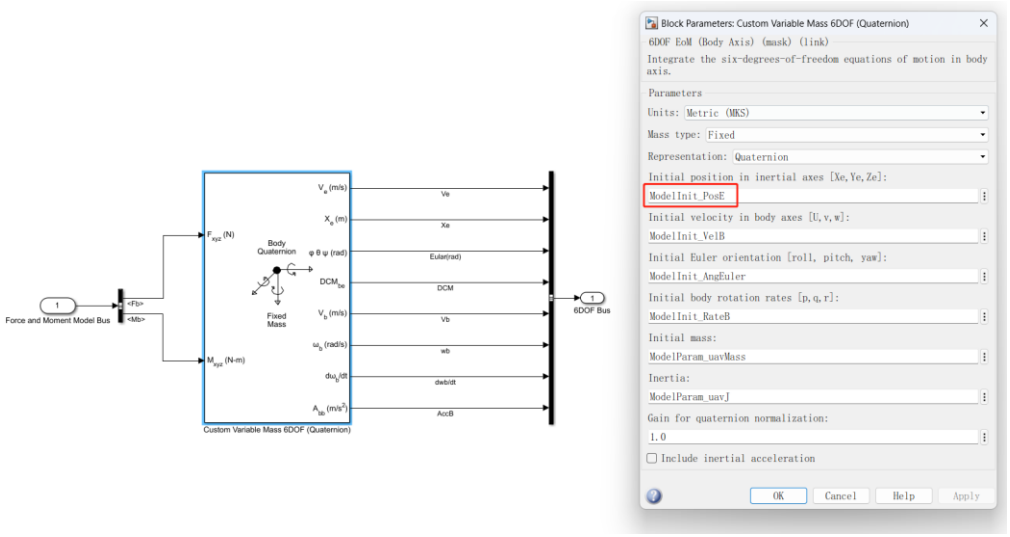
5.2 重要参数

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到的参数，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（惯性系（NED）载具初始位置坐标）

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。

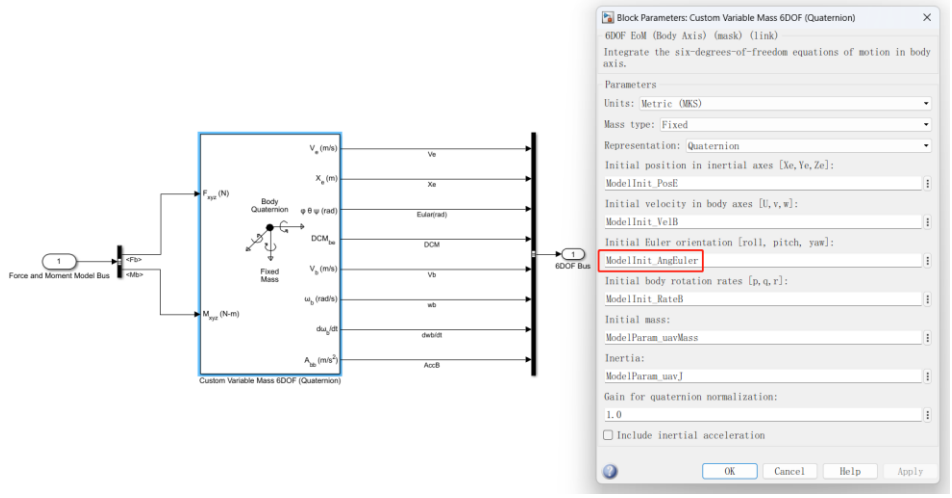
```
ModelInit_PosE=[0,0,0];
```



5.2.2 ModelInit_AngEuler（载具初始姿态角）

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。

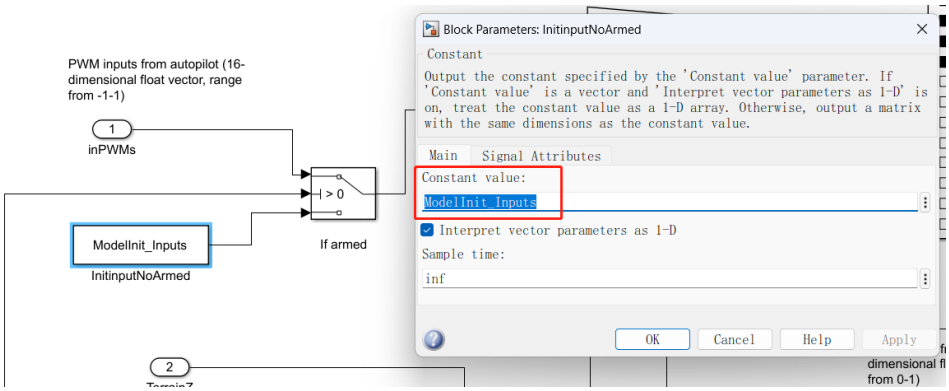
```
ModelInit_AngEuler=[0,0,0];
```



5.2.3 ModelInit_Inputs（初始化执行器参数）

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。

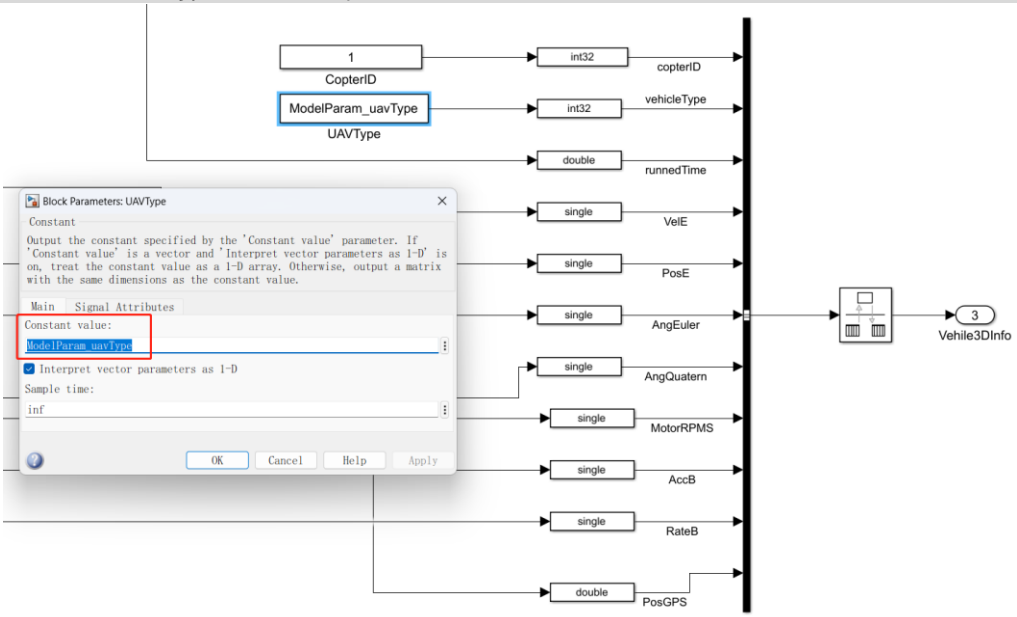
```
ModelInit_Inputs = [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0];
```



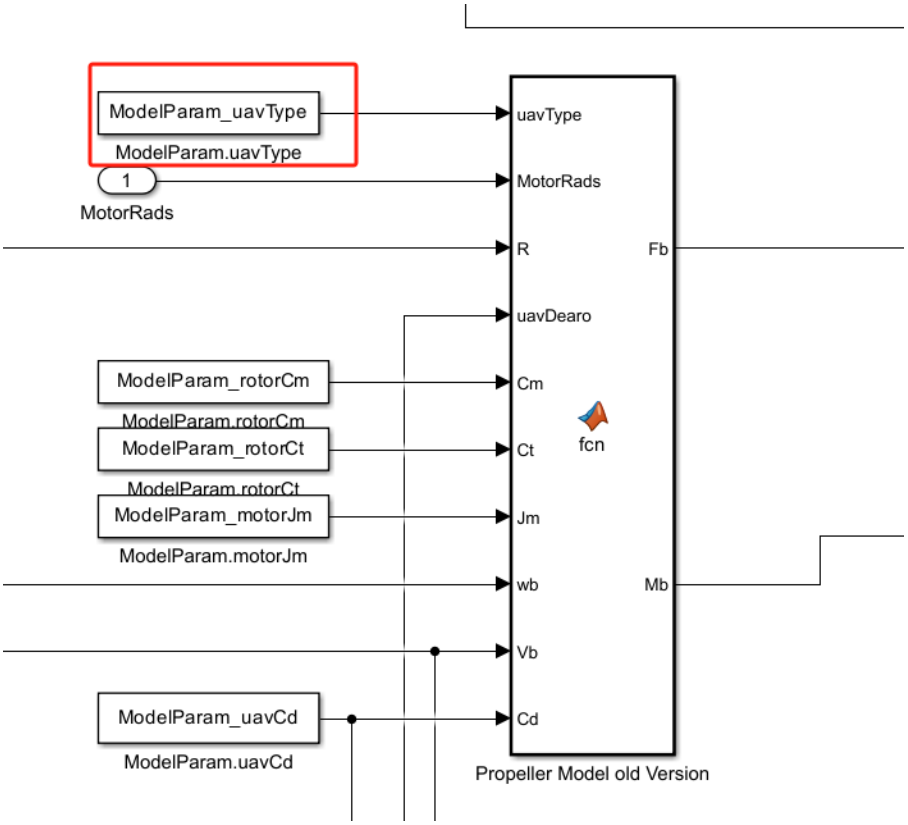
5.2.4 ModelParam_uavtype（机型定义参数）

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);
```

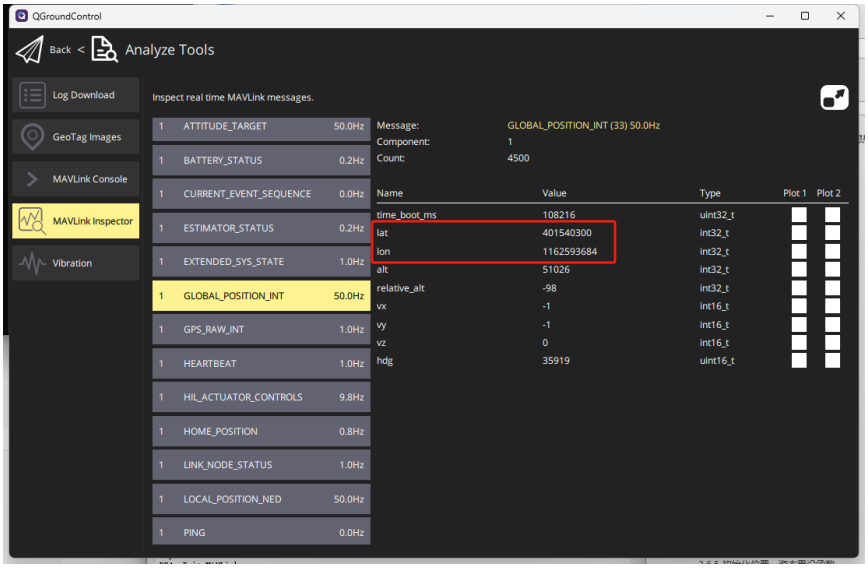


同时，针对多旋翼的螺旋桨模型，ModelParam_uavType 还会用于计算机架和力矩分配，具体可参见 6.2 Force and Moment Model 力和力矩模块,***S***-fm.

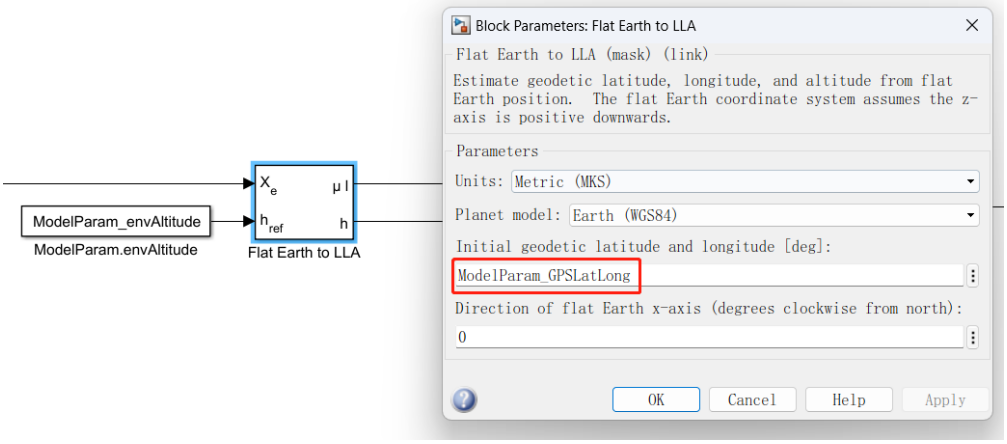


5.2.5 ModelParam_GPSLatLong（初始参考点经纬度）

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]，单位：度，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];
```



Simulink 模块 "Flat Earth to LLA" (从平面地球模型转换到经纬度高度，LLA 是 Latitude, Longitude, Altitude 的缩写) 是用于将平面地球模型（当地水平面坐标系：NED）的位置信息转换为地理坐标（纬度、经度）和高度的模块。

该模块的转换基于简化的地球模型，假定地球是一个完美的椭球体。转换算法需要考虑初始参考点的经纬度 ModelParam_GPSLatLong，以及从该参考点到当前位置的北向和东向距离。然后，通过一系列几何和三角计算，确定当前位置的经纬度和高度。

$$\begin{bmatrix} N \\ E \end{bmatrix} = \begin{bmatrix} \cos\psi & -\sin\psi \\ \sin\psi & \cos\psi \end{bmatrix} \begin{bmatrix} p_x \\ p_y \end{bmatrix},$$

$$R_N = \frac{R}{\sqrt{1-(2f-f^2)\sin^2\mu_0}}$$

$$R_M = R_N \frac{1-(2f-f^2)}{1-(2f-f^2)\sin^2\mu_0}$$

$$d\mu = \frac{dN}{R_M}$$

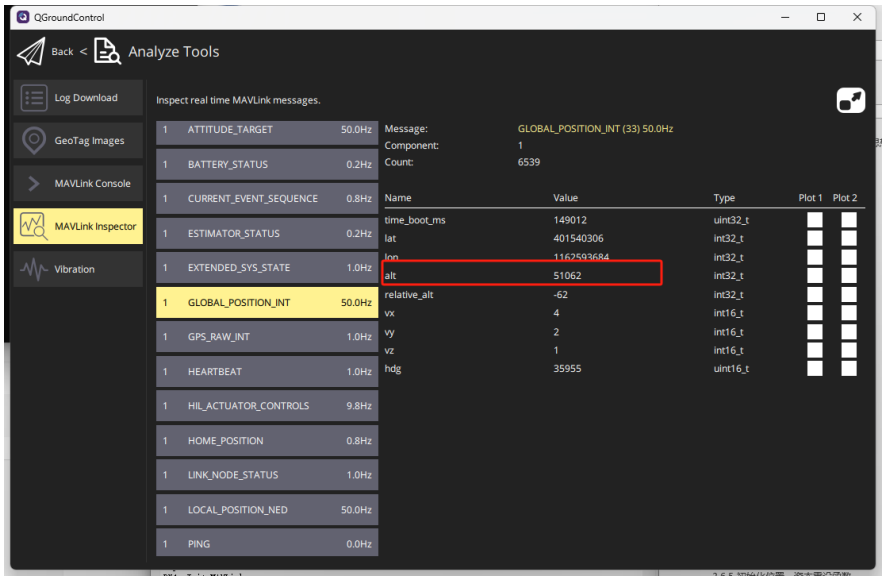
$$d\boldsymbol{l} = \frac{dE}{(R_N \cos\mu\, 0)}.$$

$$\mu = \mu_0 + d\mu$$
$$\iota = \iota 0 + d\iota$$

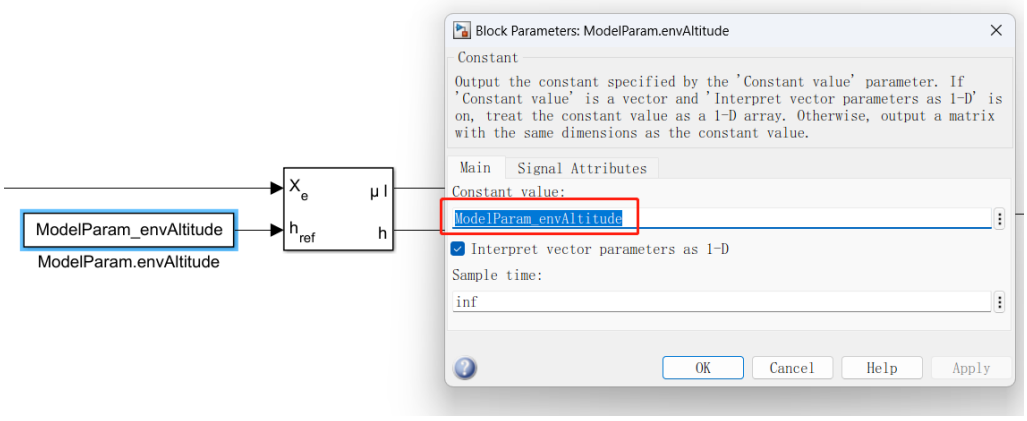
关于坐标转换的详细处理可参考：[Axes Transformations - MATLAB & Simulink - MathWorks 中国]

5.2.6 ModelParam_envAlitude（初始参考点海拔高度）

该参数用来配置飞机的海拔高度信息，z轴向下为正，单位：m，通过该参数可以调整飞机在QGC中的海拔显示。这会与场景地图的高程信息共同决定飞机的初始海拔高度信息，初始海拔不能为负值。

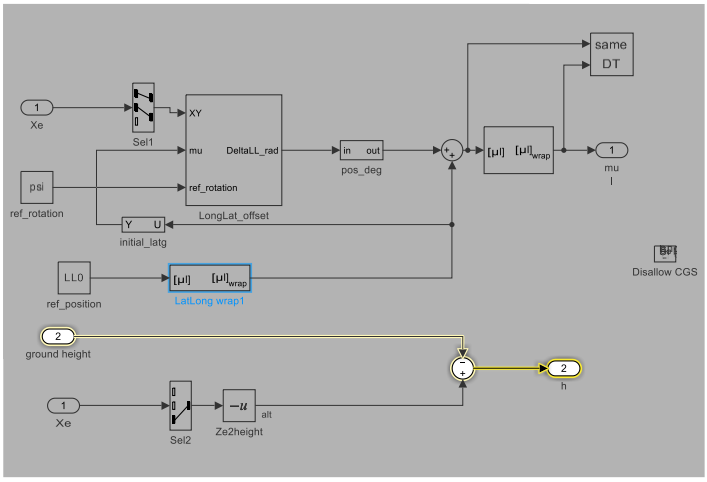



```
ModelParam_envAltitude = -50
```



"Flat Earth to LLA"中海拔高度计算如下

$$(h_{ref}): h = -p_z - h_{ref}.$$



其中初始 z 坐标来自 CopterSim 读取的三维场景的地形高程文件（见 [5.3.6.4 TerrainZ（地形高度输入）](#)），由此可知三维场景中的地下部分一旦超过 50m，这里算出的海拔高度就为负值，这会导致后续的气压计等传感器计算出错误的数据，从而飞控中滤波不能收敛，使得仿真失败。

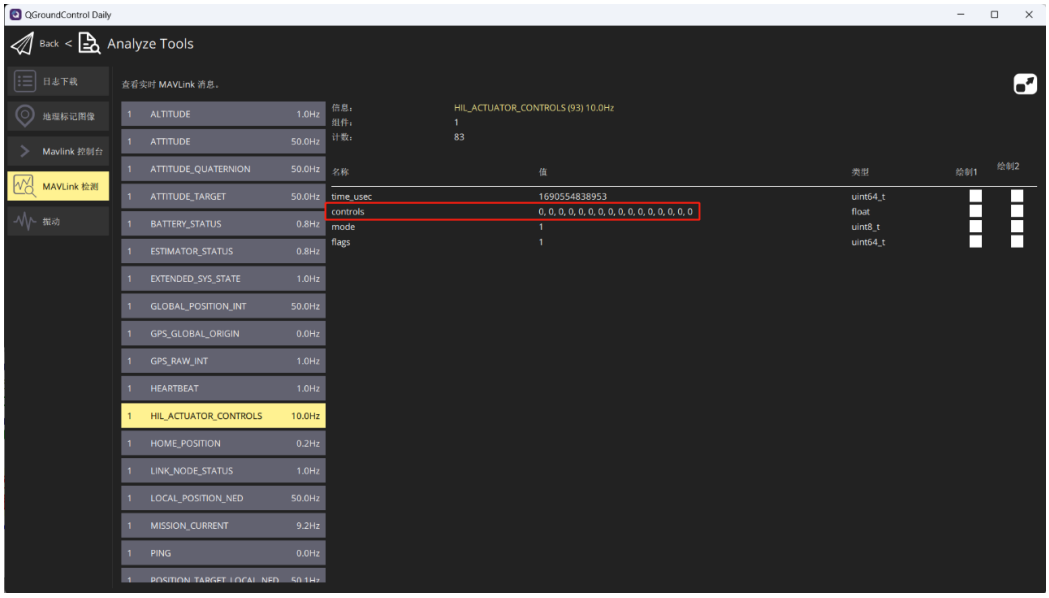
5.3 数据协议

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**错误!未找到引用源。**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。



5.3.1.2 inCopterData（飞控状态量输入）

inCopterData 是 32 维 double 型数据，前 8 维存储 PX4 的状态，目前 1-6 维数据，依次为：

- inCopterData(1)：PX4 的解锁标志位
- inCopterData(2)：接收到的 RC 频道总数。当没有可用的 RC 通道时，该值应为 0。
- inCopterData(3)：仿真模式标志位，0：HITL，1：SITL，2：SimNoPX4。
- inCopterData(4)：CoperSim 中的 3D fixed 标志位。
- inCopterData(5)：来自 PX4 的 VTOL_STATE 标志位。
- inCopterData(6)：来自 PX4 的 LANDED_STATE 标志位。

9-24 维接收 ch1-ch16 RC 通道信号（遥控器输入），25-32 维监听 rfly_px4 uORB 消息。

5.3.2 飞控仿真输出接口

5.3.2.1 HILSensor30d（传感器接口集合）

该输出信号是模型发送给飞控的各种传感器数据的集合，对应了 MAVLink 的 mavlink_hil_sensor_t 消息。输出信号中包括了加速度传感器的加速度值、陀螺仪传感器的角速度值、磁罗盘传感器的磁场值，气压和空速传感器的气压值等。

```
struct MavHILSensor {
    uint64_t time_usec; /*< 时间戳 (微秒，同步到 UNIX 时间或自系统启动后)*/
    float xacc; /*< X 轴加速度 (米/秒^2)*/
    float yacc; /*< Y 轴加速度 (米/秒^2)*/
    float zacc; /*< Z 轴加速度 (米/秒^2)*/
    float xgyro; /*< X 轴角速度 (弧度/秒)*/
    float ygyro; /*< Y 轴角速度 (弧度/秒)*/
    float zgyro; /*< Z 轴角速度 (弧度/秒)*/
    float xmag; /*< X 轴磁场强度 (高斯)*/
    float ymag; /*< Y 轴磁场强度 (高斯)*/
    float zmag; /*< Z 轴磁场强度 (高斯)*/
    float abs_pressure; /*< 绝对气压 (毫巴)*/
```

```
        float diff_pressure; /*< 差分气压 (风速) (毫巴)*/
        float pressure_alt; /*< 基于气压计算的高度*/
        float temperature; /*< 温度 (摄氏度)*/
        uint32_t fields_updated; /*< 更新字段的位掩码，位 0 = xacc，位 12: 温度，位 31: 在仿真中执行了姿态/位置/速度等的完全重置。*/
    };

    // Bit 1-15: 由上述 MAVLink 结构定义
    // Bit 16-30: 保留未来使用
```

5.3.2.2 HILGPS30d（GPS 接口）

该输出信号是模型发送给飞控的 GPS 数据值，它对应了 MAVLink 消息的 mavlink_hil_gps_t 结构体。输出信号中包含了经纬高、水平竖直精度、地速、北东地的速度、偏航角、定位状态和卫星数量等数据。

需要注意的是，这些传感器的值在仿真时由平台模型提供，在真机飞行时由真实传感器芯片提供。从图 1 得知，软件在环仿真时，传感器和 GPS 数据分别是从运动仿真模型的 MavHILSensor、MavHILGPS 接口发出、以 MAVLink 协议的形式通过 TCP 4561 系列端口发送到 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。

```
    struct MavHILGPS {
        uint64_t time_usec; /*< 时间戳 (微秒，自 UNIX 纪元以来或自系统启动以来)*/
        int32_t lat; /*< 纬度 (WGS84)，单位为十百万分之一度*/
        int32_t lon; /*< 经度 (WGS84)，单位为十百万分之一度*/
        int32_t alt; /*< 高度 (AMSL，非 WGS84)，单位为千分之一米 (向上为正)*/
        uint16_t eph; /*< GPS 水平位置精度衰减 (HDOP)，单位为厘米 (米*100)。如果未知，设置为 65535*/
        uint16_t epv; /*< GPS 垂直位置精度衰减 (VDOP)，单位为厘米 (米*100)。如果未知，设置为 65535*/
        uint16_t vel; /*< GPS 地面速度，单位为厘米/秒。如果未知，设置为 65535*/
        int16_t vn; /*< GPS 速度，单位为厘米/秒，北向，在地球固定的 NED 坐标系中*/
        int16_t ve; /*< GPS 速度，单位为厘米/秒，东向，在地球固定的 NED 坐标系中*/
        int16_t vd; /*< GPS 速度，单位为厘米/秒，向下，在地球固定的 NED 坐标系中*/
        uint16_t cog; /*< 地面航向 (非航首航向，而是移动方向)，单位为百分之一度，范围 0.0 至 359.99 度。如果未知，设置为 65535*/
        uint8_t fix_type; /*< 定位类型，0-1: 无定位，2: 二维定位，3: 三维定位。除非此字段至少为 2，否则某些应用程序不会使用此字段的值。*/
        uint8_t satellites_visible; /*< 可见卫星数量。如果未知，设置为 255*/
    };

    // Bit 1-13: 由上述 MAVLink 结构定义
    // Bit 14-30: 保留未来使用
```

5.3.3 仿真数据输出接口

5.3.3.1 VehileInfo60d（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。

```
    struct MavVehileStateInfo {
        int copterID;        // 无人机 ID
        int vehicleType;     // 无人机类型
        double runnedTime;   // 当前时间戳，单位为秒
```

```
        float VelE[3];       // NED 坐标系中无人机在地球坐标系下的速度（米/秒），分别为北向、东向和下向速
度
        float PosE[3];       // NED 坐标系中无人机在地球坐标系下的位置（米），分别为北向、东向和下向位置
        float AngEuler[3];   // 无人机的欧拉角（弧度），分别为滚转、俯仰和偏航角
        float AngQuatern[4]; // 无人机的姿态四元数，提供了另一种表示姿态的方式
        float MotorRPMS[8];  // 各电机的转速（RPM）
        float AccB[3];       // 无人机在机体坐标系中的加速度（米/秒²），分别为 X、Y、Z 方向
        float RateB[3];      // 无人机在机体坐标系中的角速度（弧度/秒），分别为 X、Y、Z 方向
        double PosGPS[3];    // 无人机的 GPS 位置，包括经度、纬度（度）和高度（米）
    };

    // Bit 1-33: 由上述 MAVLink 结构定义
    // Bit 34-60: 保留未来使用
```

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。

inSILInts 和 inSILFloats 接口在 CopterSim 中数据结构体定义为：

```
    struct PX4SILIntFloat{
```

```
    int checksum;//1234567897
    int CopterID;//飞机的 ID
    int inSILInts[8];//int 标志位
    float inSILFLoats[20];//float 参数位
};
```

5.3.6.3 inFromUE（RflySim3D 数据输入）

32 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。

其数据结构体定义为：

```
struct UEToCopterDataD{
    int checksum; //1234567899 为校验 ID
    int CopterID; //发出本消息的飞机 ID
    double inFromUE[32]; //通过蓝图发出的数据
};
```

5.3.6.4 TerrainIn15d（地形交互输入）

第 1 维是地形高度信号，由于这里的地球固联坐标系（地面坐标系）为 NED，垂直地面向下为正值。可以决定模型的初始位置高度。

1:地形高度(m)

2: haslop(0 或 1)

3:螺距(rad)

4:偏航(rad)

5: hasFric(0 或 1)

6: FrictionFactor

7: isMoveObj(0 或 1)

8: objVx (m/s)

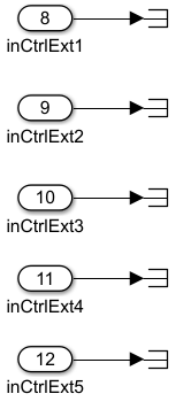
9: objVy (m/s)

10: objYaw (rad)

11-15:预留

5.3.6.5 inCtrlExt（浮点型数据输入）

包括 inCtrlExt1-inCtrlExt5 系列接口，要求数据维度为 28 维，数据类型为 double。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。目前主要用于故障注入。



5.3.6.6 inDoubCtrls(双精度浮点型数据输入)

28 维 double 型数据输入，接收来自 UDP 30100++2 系列端口的数据，主要用于大场景下的综合模型仿真，其数据结构体定义为：

```
struct DllInDoubCtrls{
    int checksum;//校验码 1234567897
    int CopterID; // 飞机的 ID
    double inDoubCtrls[28];//28 维的 double 型输入
};
```

5.3.6.7 inSIL28d（双精度浮点数据输入）

28 维 double 型输入，和 inSILIntFloats 接口功能一致。

5.3.6.8 inCollision20d（外部碰撞数据输入）

20 维 double 型输入，和 inFloatsCollision 接口功能一致，可以通过 UDP 网络从 UE4 传输，该端口为碰撞模型预留。

5.3.7 实时参数修改接口—FaultParamsAPI

5.3.7.1 FaultParamAPI.FaultInParams（故障注入参数修改接口）

实现方式	例程路径
Matlab	0.ApiExps\5.ParamAPI\2.FaultInParams\1.FaultParamsAPI_sim\Readme.pdf
Matlab	0.ApiExps\5.ParamAPI\2.FaultInParams\2.Matlab\Readme.pdf
Python	0.ApiExps\5.ParamAPI\2.FaultInParams\3.Python\Readme.pdf

5.3.7.2 FaultParamAPI.InitInParams（参数初始化接口）

实现方式	例程路径
Python	0.ApiExps\5.ParamAPI\1.initParams\2.initParamsAPI_py\readme.pdf
Csv 表格	0.ApiExps\5.ParamAPI\1.initParams\1.initParamsAPI_csv\Readme.pdf

5.3.7.3 FaultParamAPI.DynModiParams（参数动态修改接口）

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。

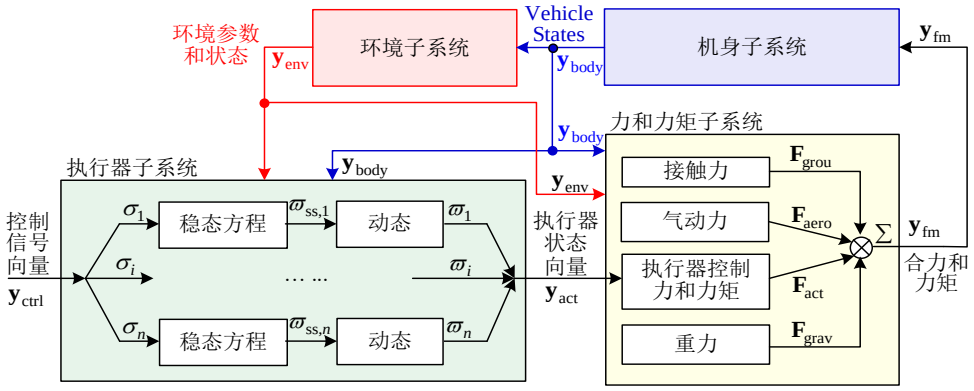
6、Simulink 建模模板介绍

无人系统载具运动模型可以通过物理与虚拟组件分解为**错误!未找到引用源。**所示的统

一建模框架。在该框架中，运动模型系统整体可以细分为三个子系统：机体子系统$\boldsymbol{S}_{\text{vehi}}$、三维环境模型子系统$\boldsymbol{S}_{\text{3d}}$和传感器模型子系统$\boldsymbol{S}_{\text{sens}}$。这三个子系统需要连接控制系统$\boldsymbol{S}_{\text{ctrl}}$形成一个完整的闭环，可以实现对室外真实环境的所有场景的模拟。它们之间的输入输出与信号连接关系可用子系统简化形式表示如下

$$\boldsymbol{y}_{\text{ctrl}} = \boldsymbol{S}_{\text{ctrl}}(\boldsymbol{u}_{\text{ctrl}}), \boldsymbol{u}_{\text{ctrl}} = \boldsymbol{y}_{\text{sens}}$$
$$\boldsymbol{y}_{\text{vehi}} = \boldsymbol{S}_{\text{vehi}}(\boldsymbol{u}_{\text{vehi}}), \boldsymbol{u}_{\text{vehi}} = \{\boldsymbol{y}_{\text{ctrl}}, \boldsymbol{y}_{\text{3d}}\}$$
$$\boldsymbol{y}_{\text{3d}} = \boldsymbol{S}_{\text{3d}}(\boldsymbol{u}_{\text{3d}}), \boldsymbol{u}_{\text{3d}} = \boldsymbol{y}_{\text{vehi}}$$
$$\boldsymbol{y}_{\text{sens}} = \boldsymbol{S}_{\text{sens}}(\boldsymbol{u}_{\text{sens}}), \boldsymbol{u}_{\text{sens}} = \{\boldsymbol{y}_{\text{vehi}}, \boldsymbol{y}_{\text{3d}}\}$$

- 机体子系统$\boldsymbol{S}_{\text{vehi}}$包含了执行器$\boldsymbol{S}_{\text{act}}$、机身$\boldsymbol{S}_{\text{body}}$、运行环境$\boldsymbol{S}_{\text{env}}$、力与力矩$\boldsymbol{S}_{\text{fm}}$等内部子系统模块，是对机体在环境的运动、能耗和故障特性的整体描述；



输入输出可描述如下

$$\boldsymbol{y}_{\text{body}} = \boldsymbol{S}_{\text{body}}(\boldsymbol{u}_{\text{ctrl}}), \boldsymbol{u}_{\text{ctrl}} = \boldsymbol{y}_{\text{fm}}$$
$$\boldsymbol{y}_{\text{fm}} = \boldsymbol{S}_{\text{fm}}(\boldsymbol{u}_{\text{fm}}), \boldsymbol{u}_{\text{fm}} = \{\boldsymbol{y}_{\text{body}}, \boldsymbol{y}_{\text{act}}, \boldsymbol{y}_{\text{env}}\}$$
$$\boldsymbol{y}_{\text{env}} = \boldsymbol{S}_{\text{env}}(\boldsymbol{u}_{\text{env}}), \boldsymbol{u}_{\text{env}} = \{\boldsymbol{y}_{\text{body}}, \boldsymbol{y}_{\text{3d}}\}$$
$$\boldsymbol{y}_{\text{act}} = \boldsymbol{S}_{\text{act}}(\boldsymbol{u}_{\text{act}}), \boldsymbol{u}_{\text{act}} = \{\boldsymbol{y}_{\text{ctrl}}, \boldsymbol{y}_{\text{body}}, \boldsymbol{y}_{\text{env}}\}$$

- 传感器模型主要用于描述控制软件之外的所有电子硬件模型，主要包含传感器数据、通信协议、连接接口等特性；
- 三维环境模型主要用于描述无人机飞行的三维视景环境（包括树木、障碍物、公路等），用于为自主控制系统提供视觉数据的模拟。

下面以多旋翼模型为例，介绍 Simulink 建模模板的几个功能模块

6.1 Motor Model 电机模块$\boldsymbol{S}_{\text{act}}$

执行器子系统$\boldsymbol{S}_{\text{act}}$主要用于根据控制系统发送的控制指令$\boldsymbol{y}_{\text{ctrl}}$来生成执行器的输出状态$\boldsymbol{y}_{\text{act}}$。在真实系统中，大多数执行器通常都自带控制单元来进行反馈控制，以确保执行器的运动规律遵循预先编程的特性。例如，无人机常用的电调产品通常具有一定的速度反馈功能，能够保证输出的转速与输出的油门满足线性关系；无人车常用的引擎系统和转向系统也通常由电子控制单元（Electronic Control Unit，ECU）来控制，使得引擎的输出与油门的输入满足特定的编程关系。考虑到执行器的输入输出关系遵循着人工预设的编程规律，而不是自然模型规律，利用数学的方法去推导其模型是非常困难的，通常需要借助系统辨识的方法。

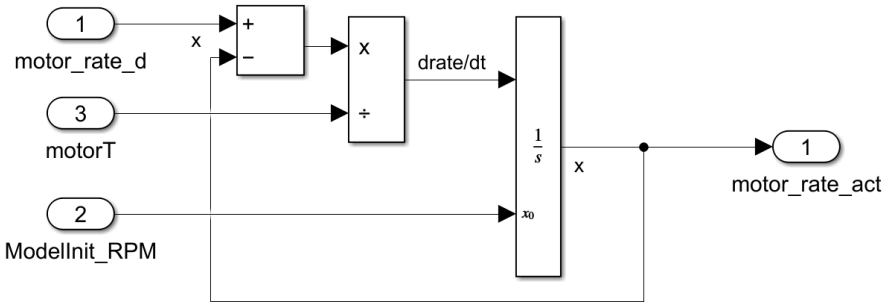
一个复杂的执行器模型可以将其在额定运行状态下，分解为一个稳态过程 $f_{ss,i}(\cdot)$和一个动态过程$G_{ss,i}(s)$。这里的额定运行状态，对于多旋翼来说是悬停飞行状态，对于车辆来说是额定速度向前行驶的状态，对于固定翼飞行器来说是以额定速度巡航飞行的状态。

1) 多旋翼动力单元模型

对于多旋翼的动力系统中的某一个动力单元（电调+电机+螺旋桨），它可以简化为一个一阶（或二阶）惯性环节叠加上一个线性稳态函数的形式，对应的传递函数可以表示为

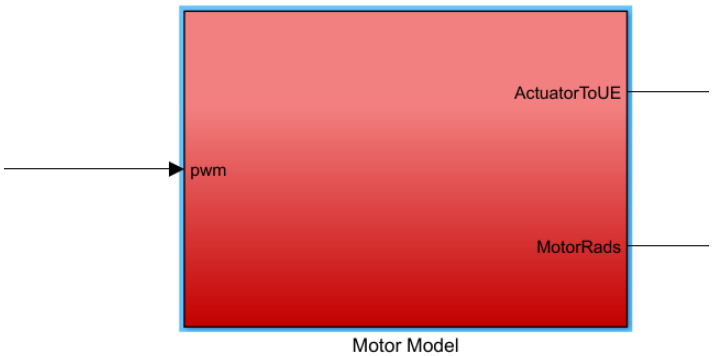
$$\delta_i = G_{ss,i}(s) \cdot f_{ss,i}(\sigma_i) = \frac{1}{k_1 s + 1}(k_2 \sigma_i + k_3)$$

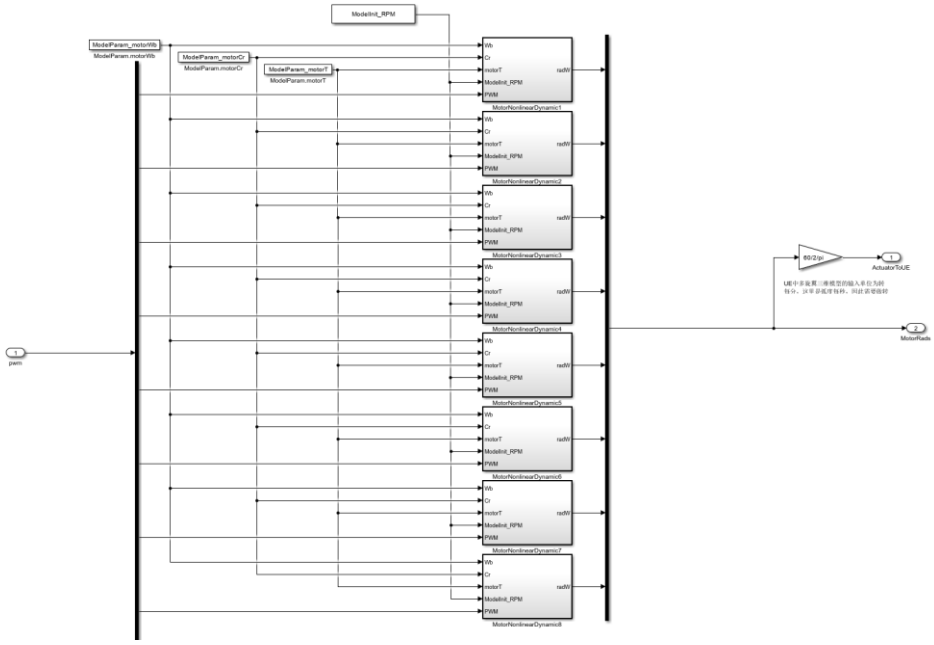
其中，s是拉普拉斯算子，σ_i表示该执行器的输入控制信号，δ_i表示该执行器输出的状态值，k_1, k_2, k_3是需要辨识的常数参数，对应 simulink 模块如下：



2) 总体输入输出

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S_{fm}

力和力矩子系统$\boldsymbol{S}_{\text{fm}}$主要用于综合执行器状态$\boldsymbol{y}_{\text{act}}$、飞行环境状态$\boldsymbol{y}_{\text{env}}$和机体运动状态$\boldsymbol{y}_{\text{body}}$，来计算出机身子系统$\boldsymbol{S}_{\text{body}}$受到的合力和力矩$\boldsymbol{y}_{\text{fm}} \triangleq \{{}^{b}\boldsymbol{F},\ {}^{b}\boldsymbol{M}\}$。为了方便六自由度机体运动模型的使用，合力和力矩通常需要统一映射（坐标转换）到机体坐标系中。

以下介绍合力的建模过程，合力矩建模过程与之类似。机体受到的合力来源可以表示为如下叠加形式：

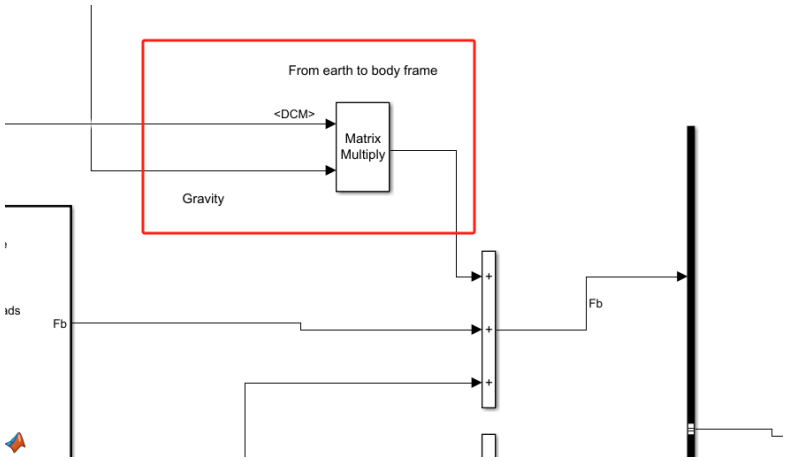
$$ {}^{b}\boldsymbol{F} = {}^{b}\boldsymbol{F}_{\text{aero}} + {}^{b}\boldsymbol{F}_{\text{grav}} + {}^{b}\boldsymbol{F}_{\text{cont}} + \sum {}^{b}\boldsymbol{F}_{\text{act},i} $$

其中，${}^{b}\boldsymbol{F}_{\text{aero}} \in \mathbb{R}^3$表示气动力向量，${}^{b}\boldsymbol{F}_{\text{grav}} \in \mathbb{R}^3$表示重力向量，${}^{b}\boldsymbol{F}_{\text{cont}} \in \mathbb{R}^3$表示接触力向量（来源于地面支撑力或者障碍物的物理碰撞力），${}^{b}\boldsymbol{F}_{\text{act},i} \in \mathbb{R}^3$表示某一个执行器生成的驱动力向量。

1) 机体坐标系下重力${}^{b}\boldsymbol{F}_{\text{grav}}$

$$ {}^{b}\boldsymbol{F}_{\text{grav}} = (\boldsymbol{R}_{b}^{e})^{-1} \begin{bmatrix} 0 \\ 0 \\ m \cdot g \end{bmatrix} $$

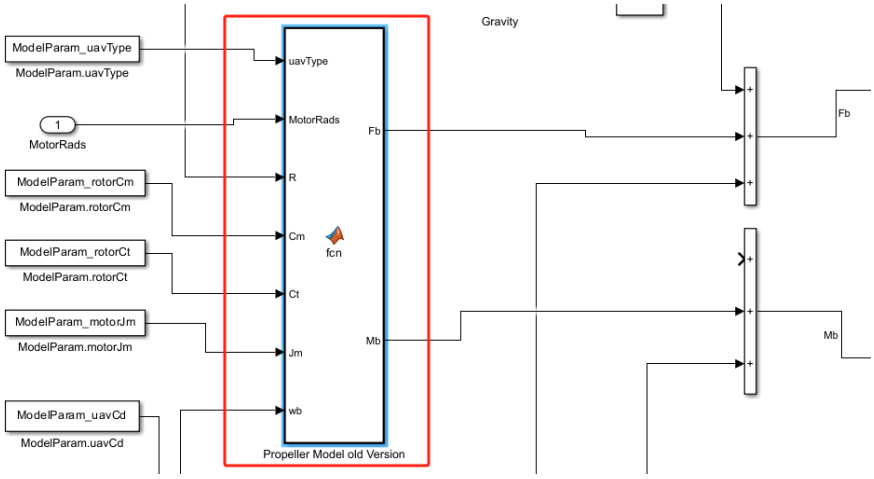
其中，g表示重力加速度，m表示机体质量，对应 simulink 模块如下



2) 机体坐标系下螺旋桨驱动力${}^{b}\boldsymbol{F}_{\mathbf{act},i}$（控制效率模型）

$$ {}^{b}\boldsymbol{F}_{\mathrm{act},i} = \boldsymbol{f}_{\mathrm{act},i}\left(\boldsymbol{\Phi}_{\mathrm{fm}}, \boldsymbol{y}_{\mathrm{env}}, \boldsymbol{y}_{\mathrm{body}}, \delta_i\right) $$

其中，$\delta_i \in \boldsymbol{y}_{\mathrm{act}}$表示此执行器当前的实时状态（例如，螺旋桨转速、舵机偏转角度、轮胎的驱动扭矩等），该状态量可以从执行器子系统$\boldsymbol{S}_{\mathrm{act}}$中获取；表达式$\boldsymbol{f}_{\mathrm{act},i}(\cdot)$与机体的运动状态$\boldsymbol{y}_{\mathrm{env}}$、环境状态$\boldsymbol{y}_{\mathrm{env}}$、和执行器的位置与方向分布都直接相关，其具体的建模方法可以参考文献。例如，单一螺旋桨输出力$\boldsymbol{T}$和力矩$\boldsymbol{M}$关于螺旋桨转动速度的函数关系，将力$\boldsymbol{T}$和力矩$\boldsymbol{M}$根据其旋翼安装信息，分解为机体坐标系下的三维向量即可求出该螺旋桨执行器的作用力向量${}^{b}\boldsymbol{F}_{\mathrm{act},i}$。不同类型的载具系统的最关键的区别在于其执行器作用力的位置分布与方向不相同。因此对于不同类型飞行器，只需要通过上式来建立不同的执行器作用力模型。这样就可以使本小节提出的机体子系统可以用于任意类型的飞行器上，



3) 机体坐标系下气动力${}^{b}\boldsymbol{F}_{\mathbf{aero}}$（多旋翼简化气动模型）

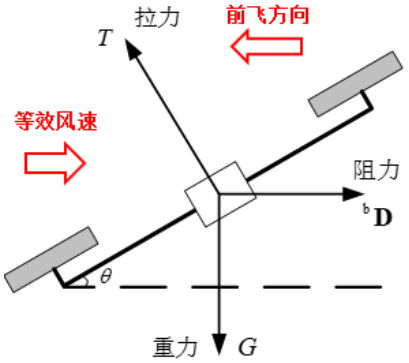
$$ {}^{b}\boldsymbol{F}_{\mathrm{aero}} = \boldsymbol{f}_{\mathrm{aero}}(\boldsymbol{\Phi}_{\mathrm{fm}}, \boldsymbol{\Phi}_{\mathrm{aero}}, \boldsymbol{y}_{\mathrm{env}}) $$

常规多旋翼的气动力主要为空气阻力与空气阻尼力矩，分别表示阻止飞行器向前运动阻力与阻止飞行器转动的力矩。

- 只考虑阻力作用，即气动力的方向与相对风速（来流）的方向相同（注：升力是垂直于来流方向的）；
- 理论上螺旋桨会受到来流的影响，导致拉力大小降低。因为常规多旋翼前飞速度

较低，该影响不显著，我们假设螺旋桨拉力始终不变，将螺旋桨因来流产生的拉力损失纳入到阻力系数中考虑；

- 多旋翼不同俯仰角时对应的迎风面积有区别，因此阻力系数会随着俯仰角（和滚转角）变化而改变



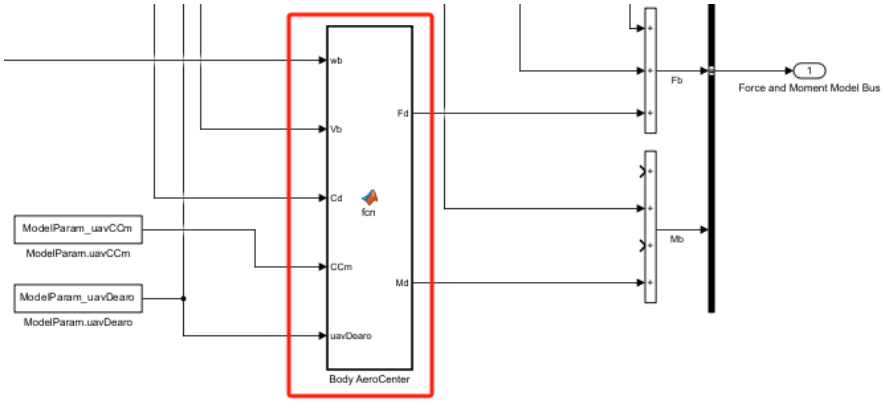
气流相对流速 vb：$^{b}\boldsymbol{v}_a \triangleq [u \quad v \quad w]^T$

空气阻力向量 Fd：$^{b}\boldsymbol{F}_d = C_d \cdot [u^2 \quad v^2 \quad w^2]^T$

气流相对转速 wb：$^{b}\boldsymbol{\omega}_a \triangleq [\omega_x \quad \omega_y \quad \omega_z]^T$

阻尼力矩向量 Md：$^{b}\boldsymbol{M}_d = C_{\mathrm{md}} \cdot \left[\omega_x^2 \quad \omega_y^2 \quad \omega_z^2\right]^T$

其中，C_d, C_{md}是标称状态下（额定前飞速度与高度）的平均阻力与阻力系数。更复杂地，可以将系数扩展为与迎角相关的曲线，同时三个轴向取不同系数。



4) 机体坐标系下机体的碰撞力$^{b}\boldsymbol{F}_{\mathrm{cont}}$

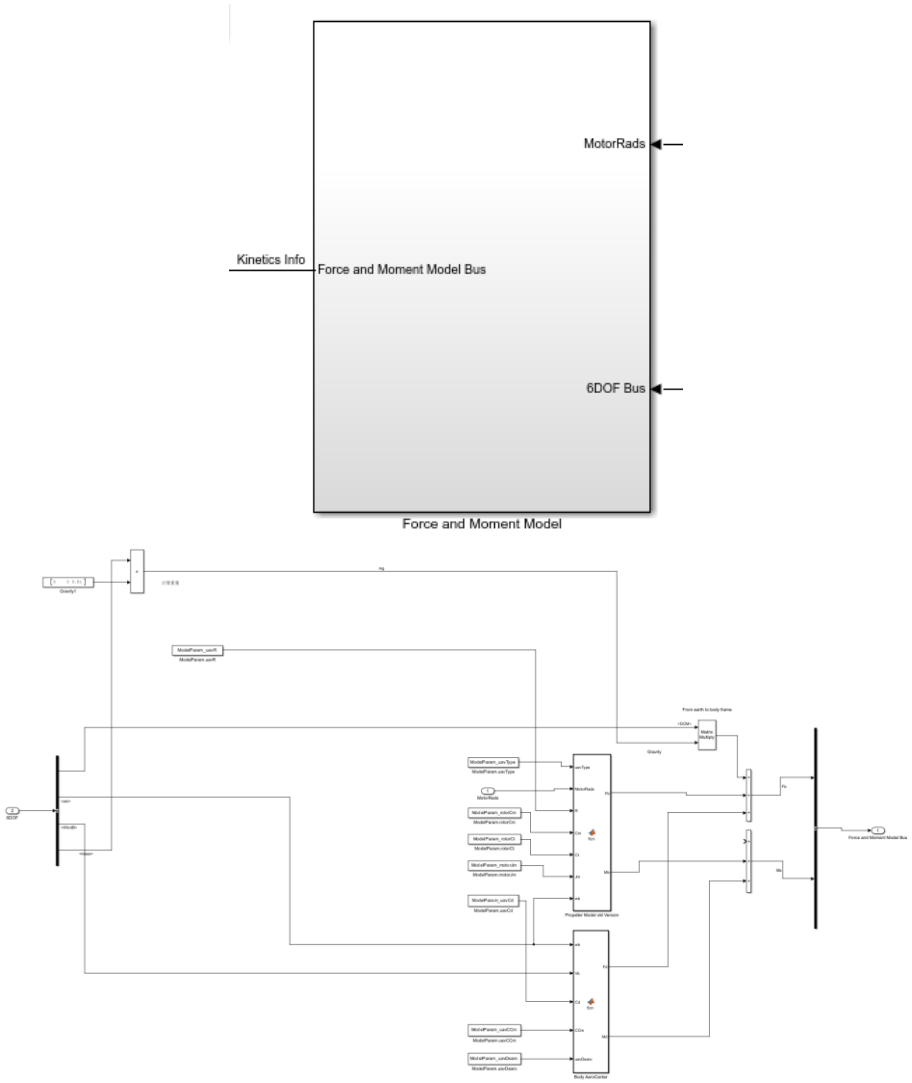
详细参见如下两个模块：

PhysicalCollisionModel 物理碰撞模块

GroundSupportModel 地面支撑模块

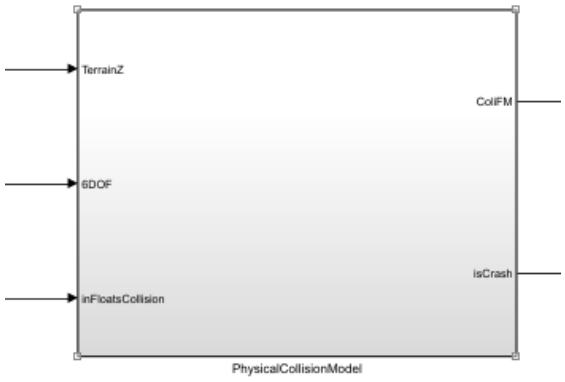
5) 总体输入输出

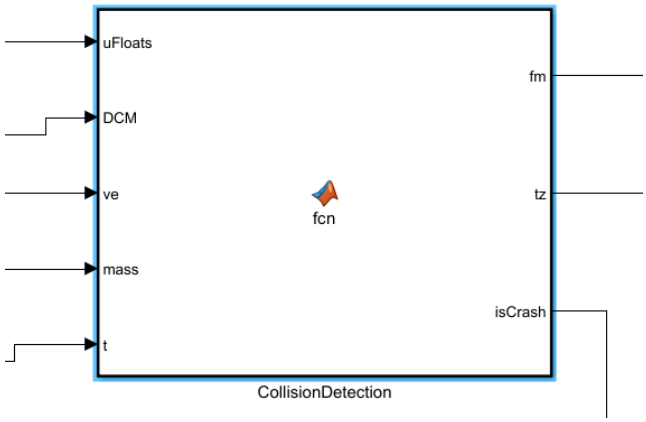
对无人机所受到的外部力和力矩进行模拟，例如，多旋翼在该模块中对螺旋桨拉力、机身气动力、自身重力以及地面支撑力等所有的外部力和力矩进行建模。该模块输入为电机转速 MotorRads、飞机运动学姿态 6DOF 和地形高度输入 tZ，输出为多旋翼合力、合力矩 Force and Moment Model Bus。



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。

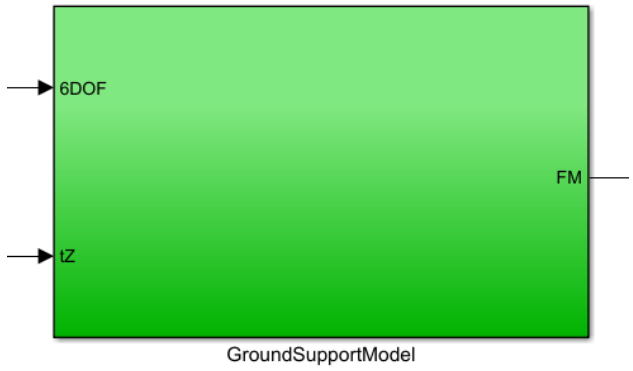
其中 uFloats 为 20 维外部输入浮点信号，该端口为碰撞模型预留，可以通过 UDP 网络从 UE4 传输。DCM 为方向余弦矩阵，ve 为碰撞时的速度，mass 为多旋翼质量，t 为时间戳。输出 fm 为力和力矩直接作用到 6DOF 对机体运动产生影响，tz 则表示多旋翼离地面的高度和 XYZ 的坐标，isCrash 则为碰撞判断，如果碰撞发生则三个电机损坏。具体碰撞逻辑可以点击进入该模块详细了解。

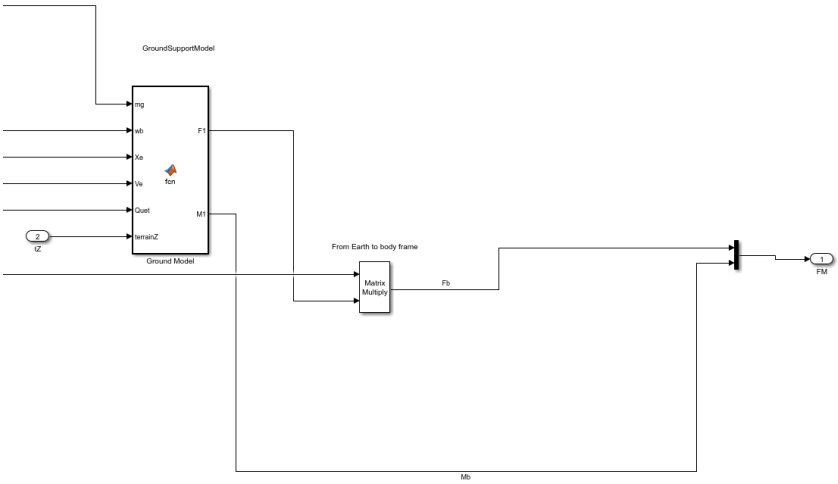
6.4 GroundSupportModel 地面支撑模块

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：

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

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。上式中，$k_1 > 0$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 刚体模块S_{body}

机身子系统$\boldsymbol{S}_{\text{body}}$主要的作用是根据机身受到的总力和力矩$\boldsymbol{y}_{\text{fm}}$来计算飞机的运动状态$\boldsymbol{y}_{\text{body}}$。在实际飞行器建模中，通常会使用两个基本假设：首先是平面地球假设，即假设飞机运动的范围较小，可忽略地球表面的曲率；其次是刚体假设，即假设飞行器的机体是刚性的，不会随意发生形变。上述假设对大多数的无人载具系统都适用。在此基础上，飞行器机体的运动通常可以用基于四元数的六自由度的动态方程表示如下

$$\begin{cases} \dot{{}^{e}p} = {}^{e}v = R_b^e \cdot {}^{b}v \\ \dot{{}^{b}v} = -[{}^{b}\boldsymbol{\omega}]_\times \cdot {}^{b}v + \dfrac{{}^{b}F}{m} \\ \dot{q}_0 = -\dfrac{1}{2} q_v^T \cdot {}^{b}\boldsymbol{\omega} \\ \dot{q}_v = \dfrac{1}{2}(q_0 I_3 + [q_v]_\times){}^{b}\boldsymbol{\omega} \\ J \cdot \dot{{}^{b}\boldsymbol{\omega}} = -{}^{b}\boldsymbol{\omega} \times (J \cdot {}^{b}\boldsymbol{\omega}) + {}^{b}M \end{cases}$$

${}^{e}\boldsymbol{p} \in \mathbb{R}^3$是定义在地球坐标系下的飞行器的位置向量；${}^{b}\boldsymbol{v} \in \mathbb{R}^3$是定义在机体坐标系下的飞行器的速度向量；${}^{b}\boldsymbol{\omega} \in \mathbb{R}^3$是定义在机体坐标系下的飞机的角速度向量；$\boldsymbol{R}_b^e \in \mathbb{R}^{3\times3}$是将向量从机体坐标系转换到地球坐标系的旋转矩阵；$\boldsymbol{J} \in \mathbb{R}^{3\times3}$和$m \in \mathbb{R}^+$是飞机的转动惯量矩阵和质量，$[\,]_\times$表示一个叉乘矩阵算子，例如设${}^{b}\boldsymbol{\omega} \triangleq [w_1, w_2, w_3]^T$，则其叉乘矩阵算子

的定义如下

$$[{}^b\boldsymbol{\omega}]_\times \triangleq \begin{bmatrix} 0 & -w_3 & w_2 \\ w_3 & 0 & -w_1 \\ -w_2 & w_1 & 0 \end{bmatrix}$$

定义四元数$\boldsymbol{q}_e^b \in \mathbb{R}^4$，令$\boldsymbol{q}_e^b = [q_0 \ q_1 \ q_2 \ q_3]^T = [q_0 \quad {q_v}^T]^T$

则有

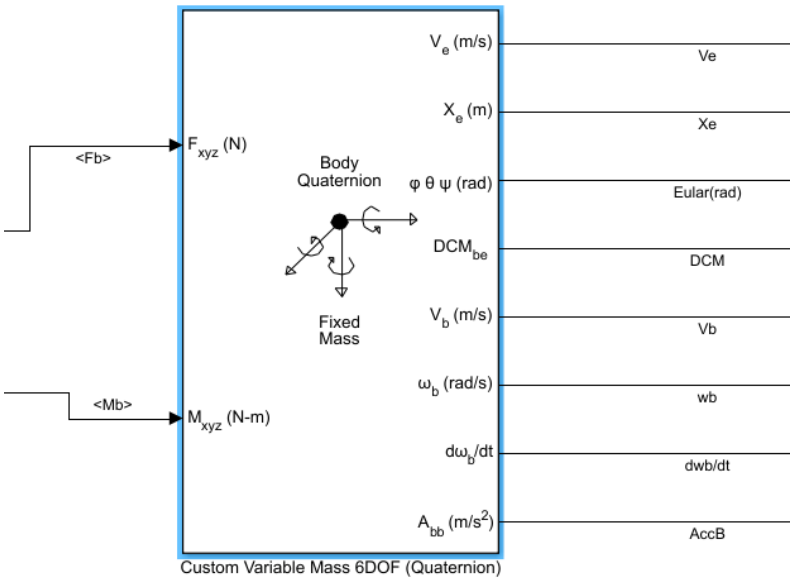
$$\dot{\boldsymbol{q}}_e^b = \frac{1}{2}\begin{bmatrix} 0 & -{}^b\boldsymbol{\omega}^{\boldsymbol{T}} \\ {}^b\boldsymbol{\omega} & -[{}^b\boldsymbol{\omega}]_\times \end{bmatrix} \boldsymbol{q}_e^b \xrightarrow{\boldsymbol{q}_e^b=[q_0 \quad {q_v}^T]^T} \begin{cases} \dot{q}_0 = -\frac{1}{2} q_v^T \cdot {}^b\boldsymbol{\omega} \\ \dot{q}_v = \frac{1}{2}(q_0 I_3 + [q_v]_\times)^b\boldsymbol{\omega} \end{cases}$$

在实际中，角速度${}^b\boldsymbol{\omega}$可由三轴陀螺仪近似测得,此时以上微分方程为线性

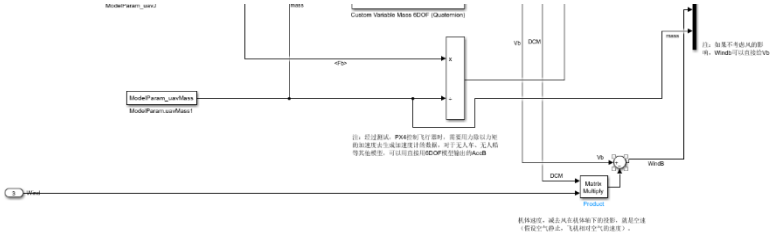
$$R_b^e = C(\boldsymbol{q}_e^b) = \begin{bmatrix} 1-2(q_2^2+q_3^2) & 2(q_1q_2-q_0q_3) & 2(q_1q_3+q_0q_2) \\ 2(q_1q_2+q_0q_3) & 1-2(q_1^2+q_3^2) & 2(q_2q_3-q_0q_1) \\ 2(q_1q_3-q_0q_2) & 2(q_2q_3+q_0q_1) & 1-2(q_1^2+q_2^2) \end{bmatrix}$$

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（旋转矩阵），角速度与角加速度。



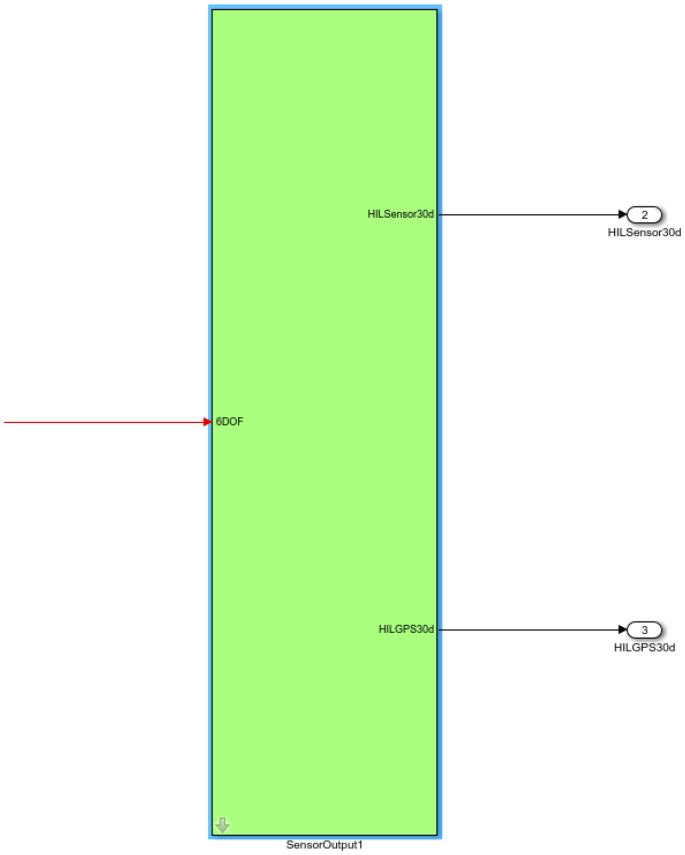
该模块通过对这些自由度进行数学建模，可以推导出无人机在空中运动时的动力学方程，从而可以进行控制系统设计、路径规划和飞行模拟等应用。此外，还可以根据实际需求对模型进行扩展，考虑更多的因素，如飞行器的非线性特性、气动力和惯性矩等。



6.6 SensorOutput 传感器输出模块S_{sens}

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 HILSensor 和 HILGPS。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S_{3d}

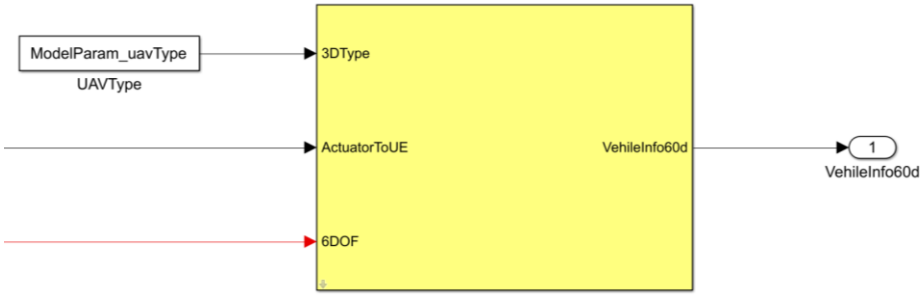
该模块输入为：

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。

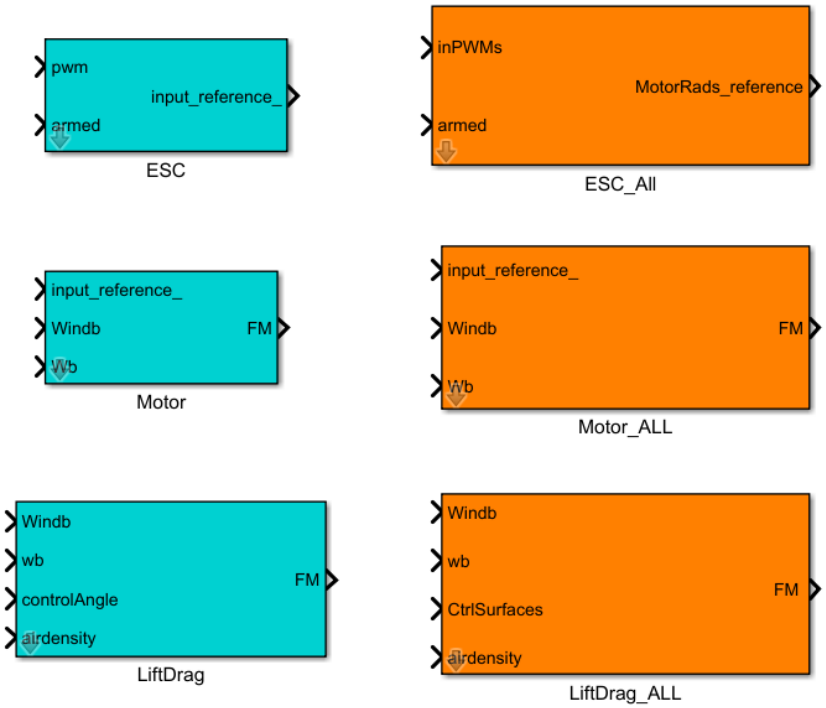
3）6DOF：来自 6DOF 模型的 6DOF Bus 输入，接收无人载具系统的位置、速度、姿态和加速度等信息。

输出为 VehileInfo60d：在三维显示模块中，会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

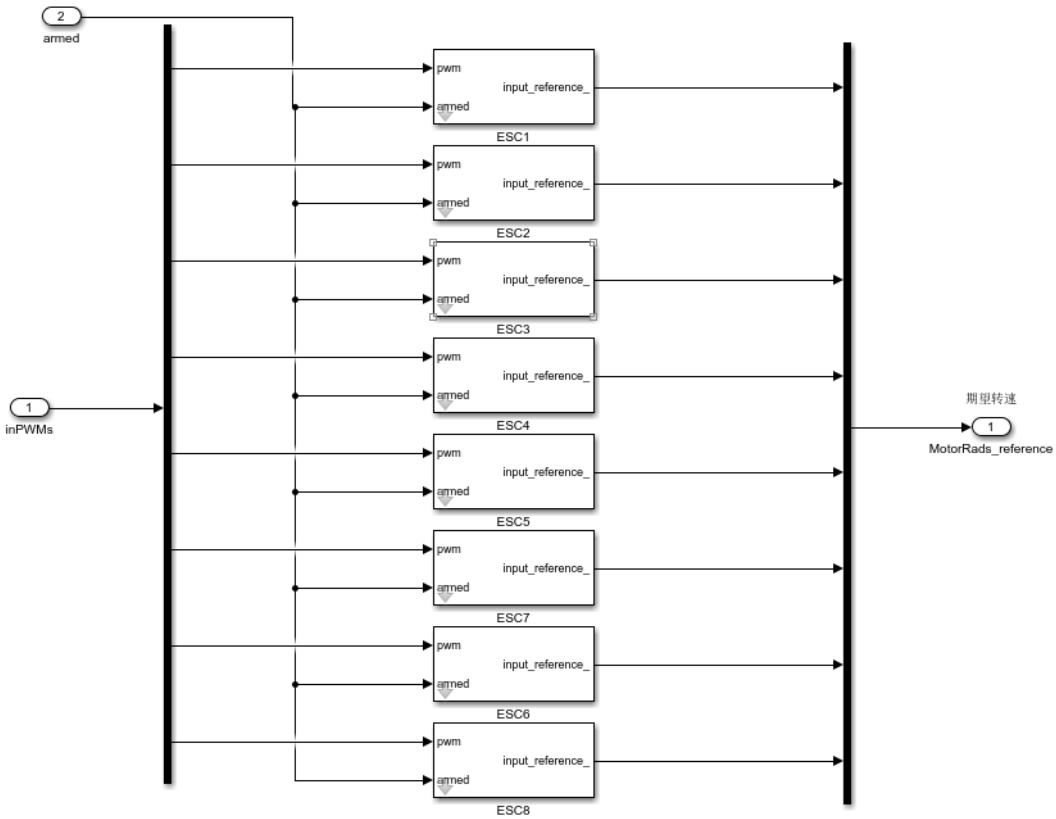
模块输入

（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。

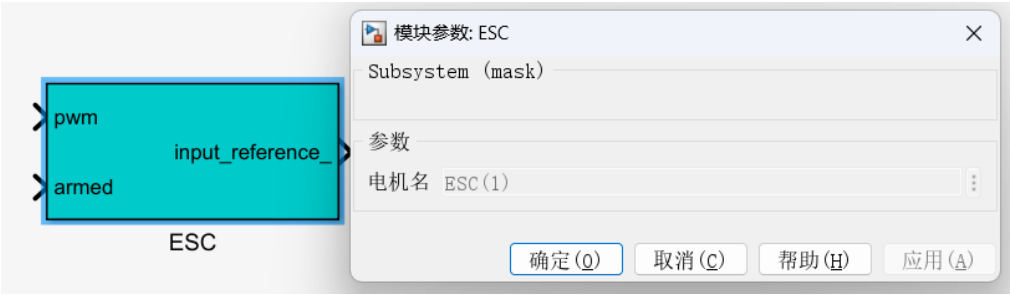
“ModelName_init.m”文件中的“ESC”结构体向量如下。

```
% 电调参数模板
ESCTmp.isEnable= 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“ModelName_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。

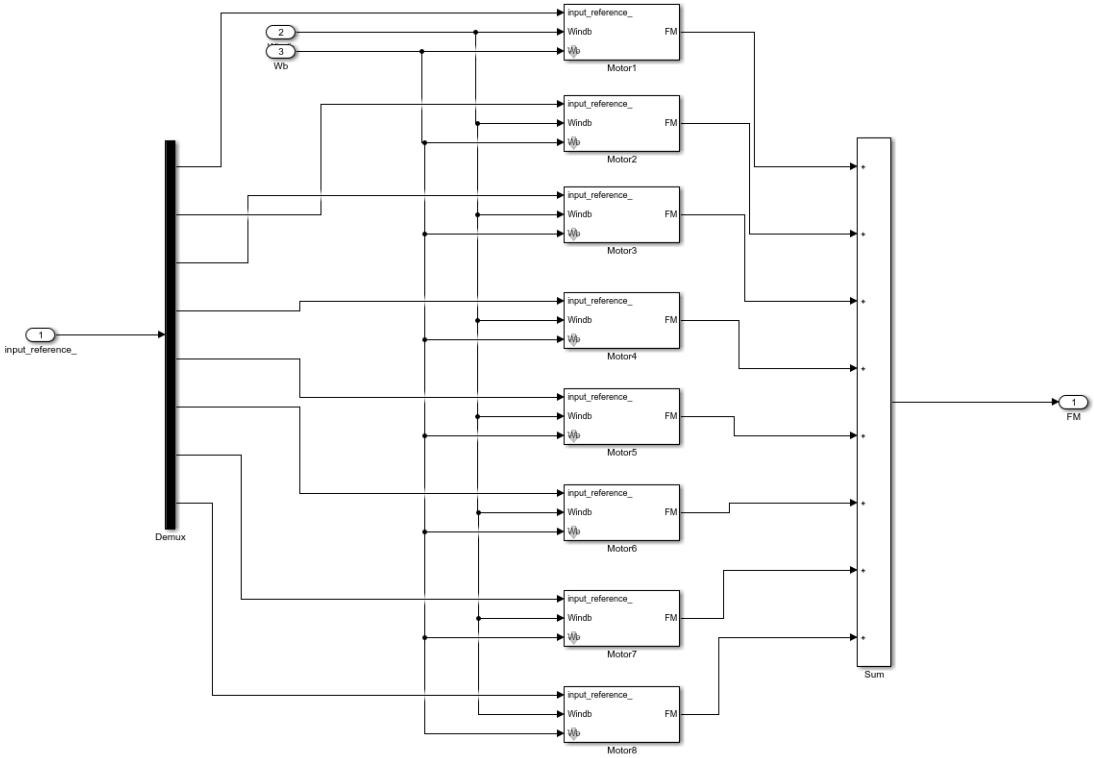
（3）wb：从 6DOF Bus 外部输入，代表机体转动角速度。

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。

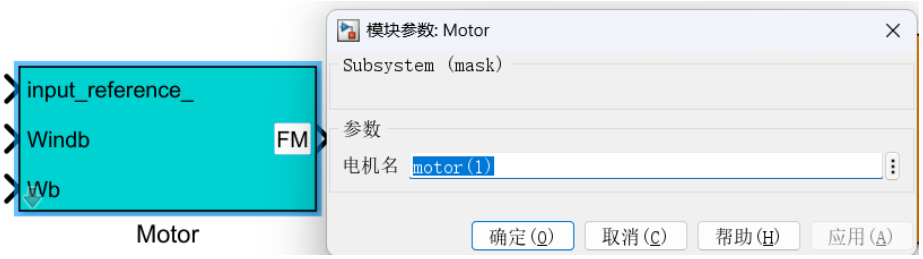
2）模块参数

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。



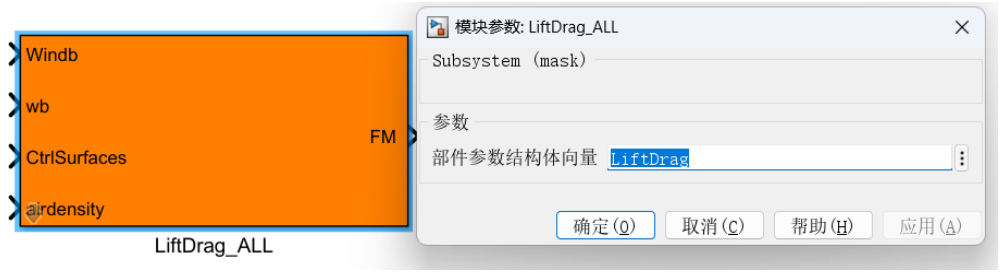
6.8.4 Motor 模块

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）。

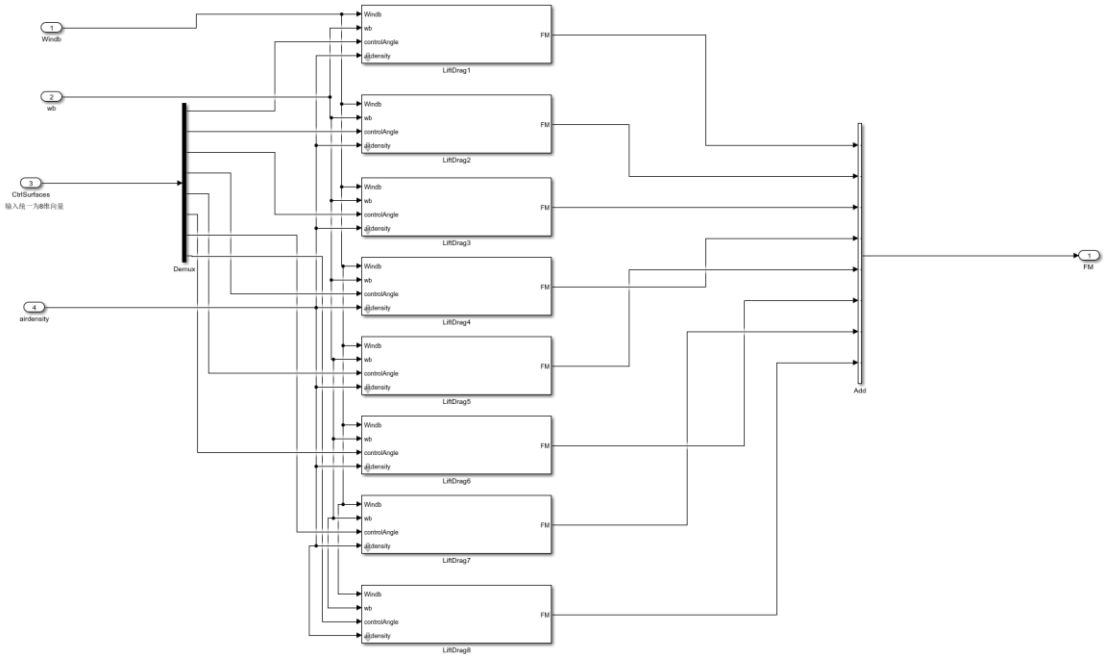
（4）airdensity：空气密度，用于计算动压，$q = \frac{1}{2} rho * V^2$。

输出：

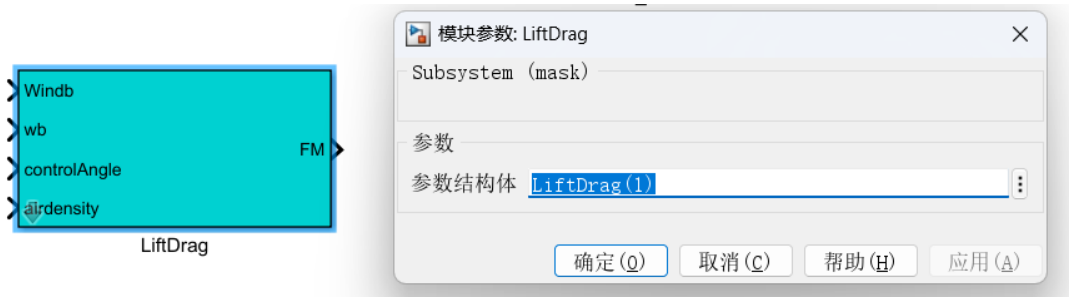
FM：升降舵、方向舵等舵面产生的机体坐标系下的力和力矩，6 维向量，前 3 维为合力，后 3 维为合力矩。

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 多旋翼模型

任意多旋翼模型的建模仿真案例都是基于上文中提出过的平台统一 Simulink 建模模板构建，它们之间唯一的区别在于将所有旋翼的力与力矩映射（大小和方向的投影与求和）到机身合力的控制效率模型。这在平台建模模板中是通过 ModelParam_uavType 参数来计算机架和力矩分配实现的，同时要实现不同的多旋翼模型还需适配下表中的参数

表格 1 模型常数参数对照表

参数名称	公式中参数名称	.m 文件参数名称
总质量	m	ModelParam.uavMass

重力加速度	g	ModelParam.envGravityAcc
转动惯量矩阵	J	ModelParam.uavJ
多旋翼机身半径	$\frac{d}{2}$	ModelParam.uavR
螺旋桨拉力系数	c_T	ModelParam.rotorCt
螺旋桨力矩系数	c_M	ModelParam.rotorCm
油门到电机稳态转速曲线斜率	C_R	ModelParam.motorCr
油门到电机稳态转速曲线零点	ω_b	ModelParam.motorWb
电机螺旋桨转动惯量	J_{RP}	ModelParam.motorJm
电机响应时间常数	T_m	ModelParam.motorT
阻力系数	C_d	ModelParam.uavCd
阻尼力矩系数	C_{dm}	ModelParam.uavCCm

7.1.1 四旋翼

7.1.1.1 建模原理

1) 控制效率模型（力与力矩）

将所有旋翼的力与力矩映射（大小和方向的投影与求和）到机身合力。关于多旋翼的控制效率模型，我们主要将其分为两个方面，单个螺旋桨拉力和反扭矩模型和整机拉力和力矩模型。

① 单个螺旋桨拉力和反扭矩模型

当多旋翼在无风情况下悬停时，ϖ_i表示螺旋桨i的转速，$\dot{\varpi}_i$是螺旋桨i的角速度变化率（理论上无风情况下悬停时为转速无变化，但电池电量的变化、螺旋桨的磨损等，可能会使螺旋桨的角速度有微小的变化），c_T与c_M为常数且可以通过实验确定

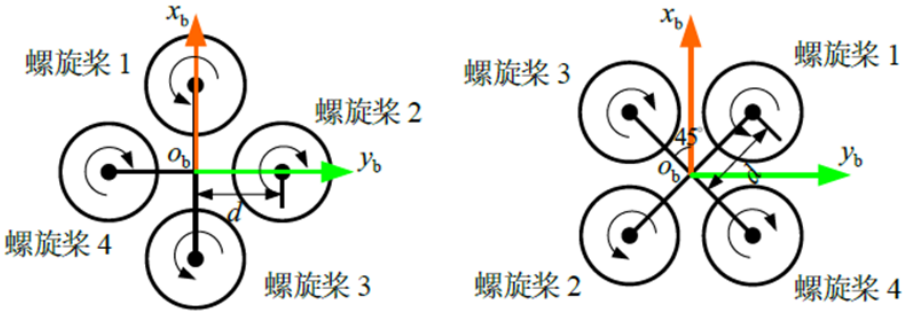
螺旋桨拉力可表示为: $T_i = c_T \varpi_i^2$

静态反扭力矩大小可表示为：$M_i = c_M \varpi_i^2$

动态反扭力矩大小可表示为：$M_i = c_M \varpi_i^2 + J_{RP}\dot{\varpi}_i$

② 整机拉力和力矩模型

对于多旋翼，为了实现控制分配，首先需要确定所有电机在机体坐标系下的位置。



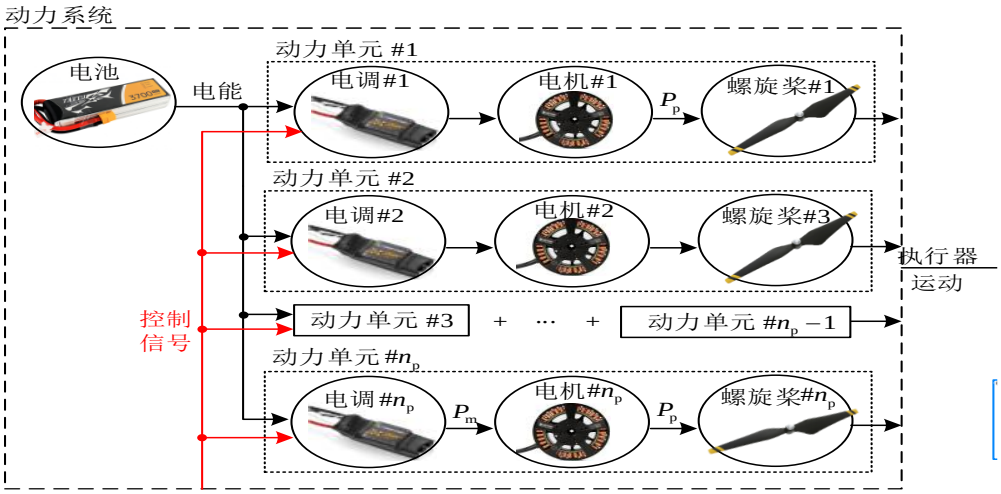
机体$o_b x_b$轴与每个电机所在的支撑臂之间的夹角为$\varphi_i \in \mathbb{R}_+ \cup \{0\}$，机体中心与电机的距离记为$d \in \mathbb{R}_+ \cup \{0\}$。

对于 X 字形四旋翼，螺旋桨产生的拉力和力矩可以表示：

$$\begin{bmatrix} f \\ \tau_x \\ \tau_y \\ \tau_z \end{bmatrix} = \begin{bmatrix} c_T & c_T & c_T & c_T \\ -\frac{\sqrt{2}}{2}dc_T & \frac{\sqrt{2}}{2}dc_T & \frac{\sqrt{2}}{2}dc_T & -\frac{\sqrt{2}}{2}dc_T \\ \frac{\sqrt{2}}{2}dc_T & -\frac{\sqrt{2}}{2}dc_T & \frac{\sqrt{2}}{2}dc_T & -\frac{\sqrt{2}}{2}dc_T \\ c_M & c_M & -c_M & -c_M \end{bmatrix} \begin{bmatrix} \varpi_1^2 \\ \varpi_2^2 \\ \varpi_3^2 \\ \varpi_4^2 \end{bmatrix}$$

其中f为总拉力，τ_x、τ_y、τ_z分别为 x、y、z 方向上的合力矩。

2) 动力单元模型



动力单元模型是以无刷直流电机、电调和螺旋桨为一组的整个动力机构。输入是 0～1 的电机油门指令，输出是螺旋桨转速。

电调接收油门指令σ和电池输出电压U_b后产生等效平均电压为$U_m = \sigma U_b$。首先，输入一个电压信号，电机能够转动到一个稳态转速ϖ_{ss}。这种关系通常是线性的，表示为

$$\varpi_{ss} = C_b U_m + \varpi_b = C_R \sigma + \varpi_b$$

其中 $C_R = C_b U_b$，C_b和ϖ_b为常参数。其次，当给定一个油门指令，电机到达稳态转速ϖ_{ss}需要一段时间，该时间决定了电机的动态响应，记为T_m。通常情况下，无刷直流电机的动态过程可以简化为一阶低通滤波器，其传递函数可以写为：

$$\varpi = \frac{1}{T_m s + 1}\varpi_{ss}$$

也就是说，当给定一个期望的稳态转速ϖ_{ss}时，电机转速并不能立即达到ϖ_{ss}，而需要经过一段时间的调整。结合上面两个式子可得完整的动力单元模型如下：

$$\varpi = \frac{1}{T_m s + 1}(C_R \sigma + \varpi_b)$$

7.1.1.2 建模仿真案例

1) 基于最大模板的四旋翼模型

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

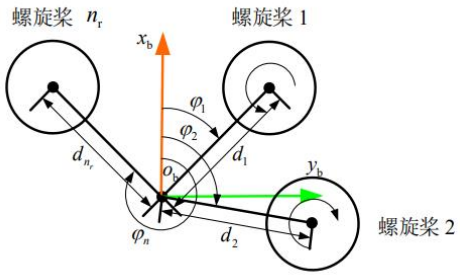
2) 带碰撞检测的四旋翼模型

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

7.1.2 六旋翼

7.1.2.1 建模原理

与四旋翼的唯一区别在于控制效率模型，对于多旋翼，为了实现控制分配，首先需要确定所有电机在机体坐标系下的位置。



对于有n_r个螺旋桨的多旋翼，顺时针方向从$i = 1$到$i = n_r \geq 5$依次标记螺旋桨，如上图所示。机体$o_b x_b$轴与每个电机所在的支撑臂之间的夹角为$\varphi_i \in \mathbb{R}_+ \cup \{0\}$，机体中心与第$i$个电机的距离记为$d_i \in \mathbb{R}_+ \cup \{0\}, i = 1,2,\dots,n_r$。

螺旋桨产生的拉力和力矩可以表示为：

$$\begin{bmatrix} f \\ \tau_x \\ \tau_y \\ \tau_z \end{bmatrix} = \begin{bmatrix} c_T & c_T & \dots & c_T \\ -d_1 c_T \sin\varphi_1 & -d_2 c_T \sin\varphi_2 & \dots & -d_{n_r} c_T \sin\varphi_{n_r} \\ -d_1 c_T \cos\varphi_1 & -d_2 c_T \cos\varphi_2 & \dots & -d_{n_r} c_T \cos\varphi_{n_r} \\ c_M \delta_1 & c_M \delta_2 & \dots & c_M \delta_{n_r} \end{bmatrix} \begin{bmatrix} \varpi_1^2 \\ \varpi_2^2 \\ \vdots \\ \varpi_{n_r}^2 \end{bmatrix}$$

7.1.2.2 建模仿真案例

1) 基于最大模板的六旋翼模型

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

7.1.3 四轴八旋翼

7.1.3.1 建模原理

参考六旋翼和四旋翼的建模原理

7.1.3.2 建模仿真案例

1) 基于最大模板的四轴八旋翼模型

*:\PX4PSP\RflySimAPIs\4.RflySimModel \2.AdvExps\e2_MultiModelCtrl\5.OctoCoxRotor\Readme.pdf

7.1.4 八旋翼

7.1.4.1 建模原理

参考六旋翼和四旋翼的建模原理

7.1.4.2 建模仿真案例

1) 基于最大模板的八旋翼模型

*:\PX4PSP\RflySimAPIs\4.RflySimModel \2.AdvExps\e2_MultiModelCtrl\6.OctoX\Readme.pdf

7.2 小型固定翼

7.2.1.1 建模原理

1)　气动力与力矩

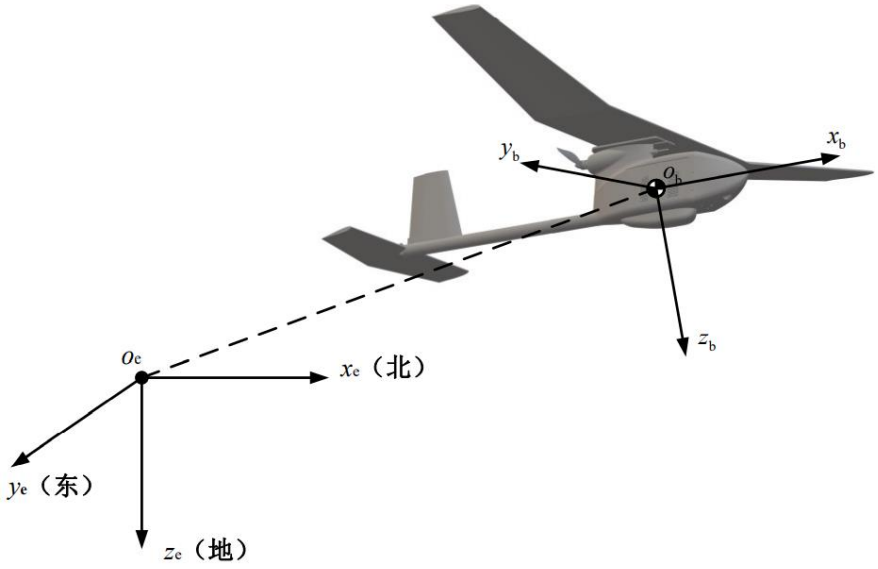
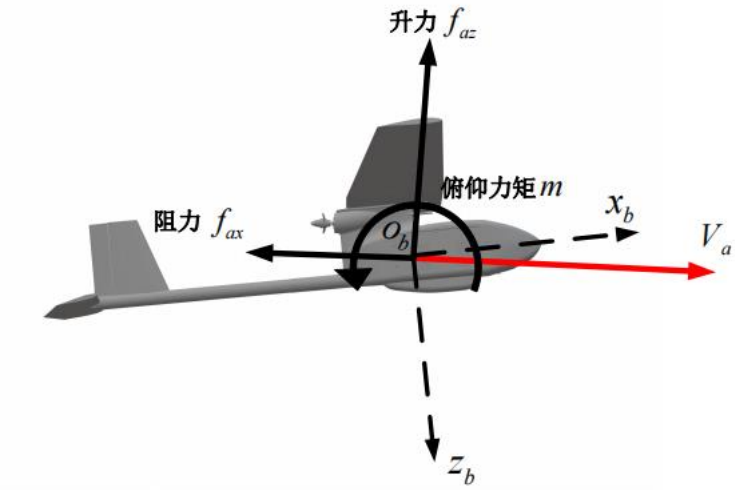


图 0.1 惯性坐标系与机体坐标系

①　纵向空气动力学

纵轴气动力和力矩包括升力、阻力和俯仰力矩，在其影响下机体将在$o_e x_e z_e$平面（惯性坐标系：北东地）内运动，该平面也被称为俯仰平面



升力、阻力和俯仰力矩都主要受到迎角α变化的影响。也受到机体俯仰速率ω_{y_b}以及升降舵δ_e的影响。因此升力、阻力和俯仰力矩的公式可以写为

升力$f_{az} \approx \frac{1}{2}\rho {V_a}^2 S C_L(\alpha, \omega_{y_b}, \delta_e)$

阻力$f_{ax} \approx \frac{1}{2}\rho {V_a}^2 S C_D(\alpha, \omega_{y_b}, \delta_e)$

俯仰力矩$M_{ay} \approx \frac{1}{2}\rho {V_a}^2 S c C_m(\alpha, \omega_{y_b}, \delta_e)$

模型简化：对上式进行一阶泰勒展开，再对近似线性化处理后的偏导数无量纲化

$$C_L \xrightarrow{\text{泰勒展开}} C_{L_0} + \frac{\partial C_L}{\partial \alpha}\alpha + \frac{\partial C_L}{\partial q}\omega_{y_b} + \frac{\partial C_L}{\partial \delta_e}\delta_e \xrightarrow{\text{无量纲化}} C_{L_0} + C_{L_\alpha}\alpha + C_{L_q}\frac{c}{2V_a}\omega_{y_b} + C_{L_{\delta_e}}\delta_e$$

最终，纵向气动力和力矩表示为

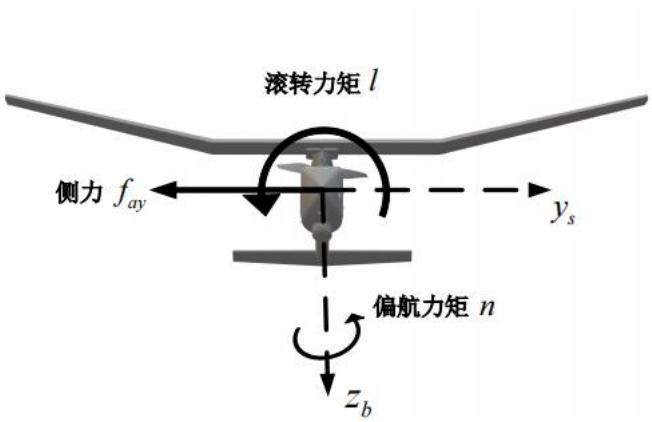
$$f_{az} = \frac{1}{2}\rho {V_a}^2 S\left(C_{L_0} + C_{L_\alpha}\alpha + C_{L_q}\frac{c}{2V_a}\omega_{y_b} + C_{L_{\delta_e}}\delta_e\right)$$

$$f_{ax} = \frac{1}{2}\rho {V_a}^2 S\left(C_{D_0} + C_{D_\alpha}\alpha + C_{D_q}\frac{c}{2V_a}\omega_{y_b} + C_{D_{\delta_e}}\delta_e\right)$$

$$M_{ay} = \frac{1}{2}\rho {V_a}^2 S c\left(C_{m_0} + C_{m_\alpha}\alpha + C_{m_q}\frac{c}{2V_a}\omega_{y_b} + C_{m_{\delta_e}}\delta_e\right)$$

② 横向空气动力学

横航向气动力和力矩，使得固定翼无人机沿$o_b y_b$轴方向（机体坐标系）运动，同时也引起滚转和偏航运动



横航向气动力主要受到侧滑角的影响，同时也受到滚转速率、偏航速率、副翼、方向舵的影响，表示为

侧力$f_{ay} \approx \frac{1}{2}\rho {V_a}^2 S C_Y(\beta, \omega_{x_b}, \omega_{z_b}, \delta_a, \delta_r)$

滚转力矩$M_{ax} \approx \frac{1}{2}\rho {V_a}^2 S b C_l(\beta, \omega_{x_b}, \omega_{z_b}, \delta_a, \delta_r)$

偏航力矩$M_{az} \approx \frac{1}{2}\rho {V_a}^2 S b C_n(\beta, \omega_{x_b}, \omega_{z_b}, \delta_a, \delta_r)$

模型简化：对上式进行一阶泰勒展开，再对近似线性化处理后的偏导数无量纲化，最终可以得到横航向气动力和力矩的线性表示为

$$f_{ay} = \frac{1}{2}\rho {V_a}^2 S\left(C_{Y_0} + C_{Y_\beta}\beta + C_{Y_P}\frac{b}{2V_a}\omega_{x_b} + C_{Y_r}\frac{b}{2V_a}\omega_{z_b} + C_{Y_{\delta_a}}\delta_a + C_{Y_{\delta_r}}\delta_r\right)$$

$$M_{ax} = \frac{1}{2}\rho {V_a}^2 Sb\left(C_{l_0} + C_{l_\beta}\beta + C_{lY_P}\frac{b}{2V_a}\omega_{x_b} + C_{l_r}\frac{b}{2V_a}\omega_{z_b} + C_{l_{\delta_a}}\delta_a + C_{l_{\delta_r}}\delta_r\right)$$

$$M_{az} = \frac{1}{2}\rho {V_a}^2 Sb\left(C_{n_0} + C_{n_\beta}\beta + C_{n_P}\frac{b}{2V_a}\omega_{x_b} + C_{n_r}\frac{b}{2V_a}\omega_{z_b} + C_{n_{\delta_a}}\delta_a + C_{n_{\delta_r}}\delta_r\right)$$

2) 动力系统推力

在建模过程中，认为固定翼无人机的动力系统沿机体 $o_b x_b$ 轴安装，螺旋桨动力系统在空速为零的情况下，所产生的推力为

$$T = C_T\rho\left(\frac{N}{60}\right)^2 {D_P}^4$$

因此,当空速大小为 V_a 时,动力系统推力在机体坐标系下表示为

$${}^bT = \begin{bmatrix} T - K\dfrac{V_a}{\sqrt{T}} \\ 0 \\ 0 \end{bmatrix}$$

7.2.1.2 建模仿真案例

1) 基于最小输入输出接口的固定翼模型

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

2) 带碰撞检测的固定翼模型

*:\PX4PSP\RflySimAPIs\4.RflySimModel\2.AdvExps\e3_FWingModelCtrl\1.FixWingModelCtrlColl\Readme.pdf

3) 固定翼位置控制

*:\PX4PSP\RflySimAPIs\4.RflySimModel\2.AdvExps\e3_FWingModelCtrl\2.FWPosCtrlAPI\Readme.pdf

4) 固定翼姿态控制

*:\PX4PSP\RflySimAPIs\4.RflySimModel\2.AdvExps\e3_FWingModelCtrl\3.FWAttCtrlAPI\Readme.pdf

5) 固定翼速度高度偏航控制

*:\PX4PSP\RflySimAPIs\4.RflySimModel\2.AdvExps\e3_FWingModelCtrl\4.VelAltYawCtrlAPI_Py\Readme.pdf

*:\PX4PSP\RflySimAPIs\4.RflySimModel\2.AdvExps\e3_FWingModelCtrl\5.VelAltYawCtrlAPI_Mat\Readme.pdf

7.3 无人车

7.3.1 精细化无人车模型

7.3.1.1 建模原理

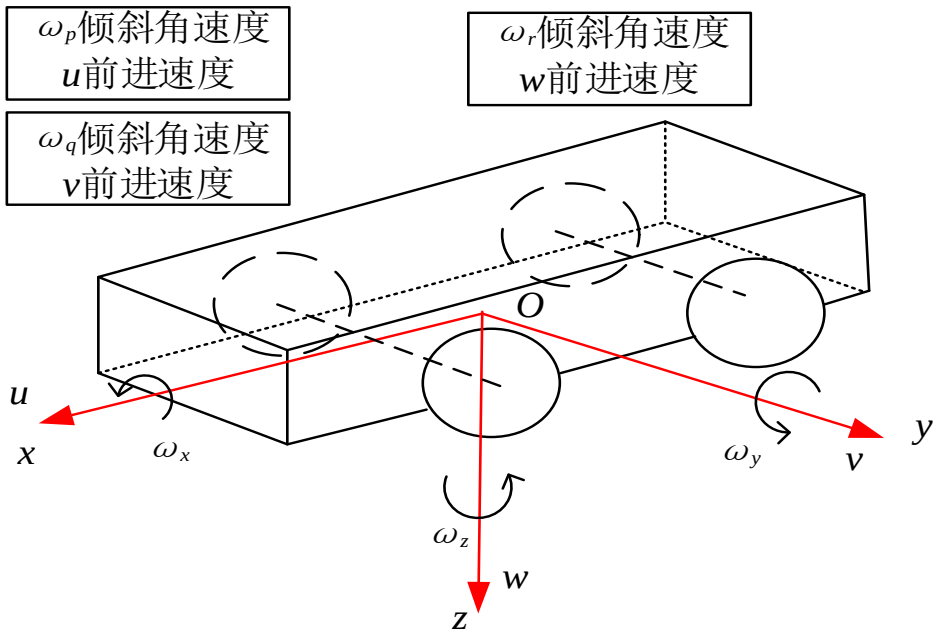


图 0.2 车辆坐标系示意图

1) 轮胎的力与力矩模型

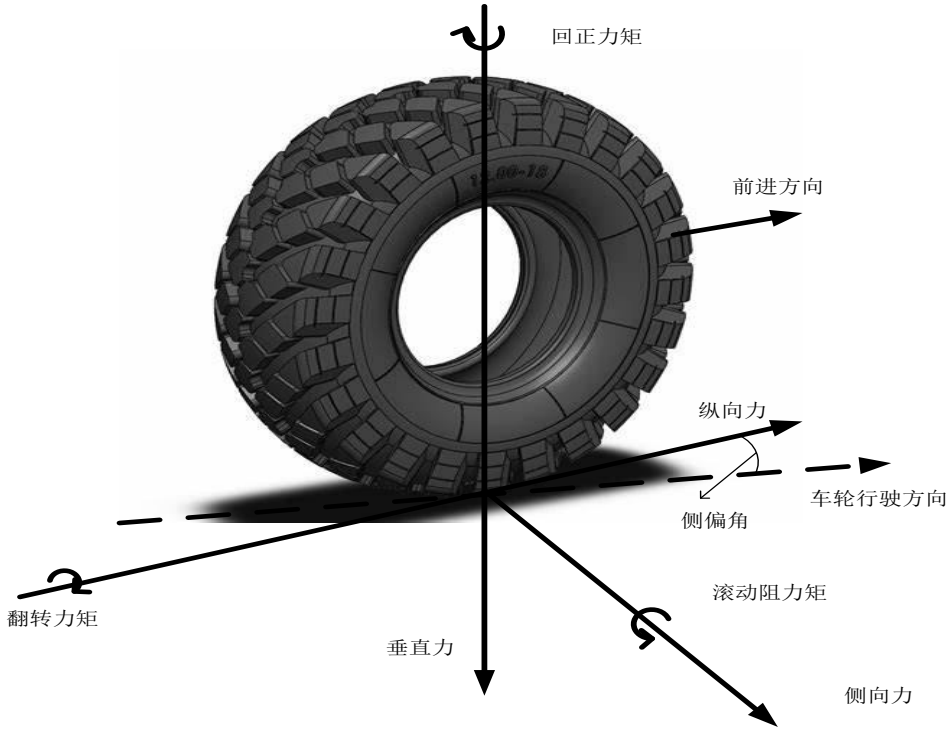


图 0.3 轮胎模型图

这里基于魔术公式轮胎模型讨论车辆轮胎的力学特性，轮胎模型魔术公式条件如图所示，其输入为纵向滑移率，侧偏角，外倾角，车轮垂直载荷；输出为纵向力，侧向力，翻转力矩，滚动阻力矩与回正力矩。

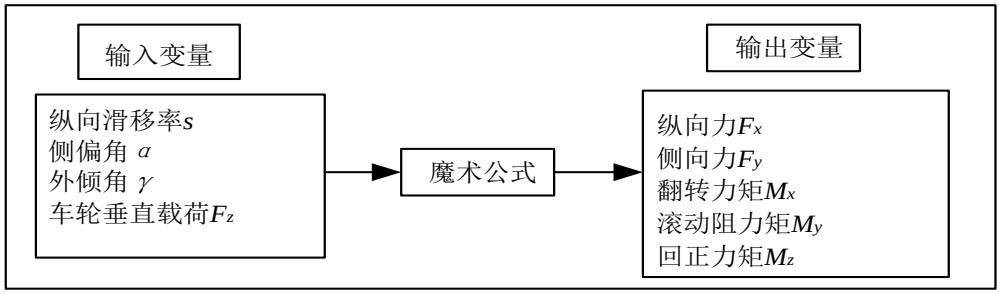


图 0.4 **魔术公式示意图**

① 纵向滑移率 s

轮胎纵向滑移率是指车辆行驶过程中轮胎在纵向方向上的滑动程度。它是通过比较轮胎实际滑动速度与理论滑动速度之间的差异来计算得出的。轮胎纵向滑移率的值越大，表示轮胎在纵向方向上的滑动程度越大，其表达式如下：

$$s = (Vx - R\omega)/Vx$$

其中， V_x 是车辆的纵向速度（单位：m/s）， R 是轮胎的半径（单位：m）， ω是轮胎的角速度（单位：rad/s）。

② 侧偏角 α

由图 0.4 可知，侧偏角是指车辆行驶过程中，轮胎与垂直方向之间的夹角。它描述了轮胎在转弯或曲线行驶时，与车辆行进方向的偏离程度。轮胎的侧偏角对车辆的操控性能和稳定性有着重要影响。

③ 外倾角 γ

轮胎外倾角是指车轮相对于垂直方向的倾斜角度。它是车辆悬挂系统中一个重要的参数，可以对车辆的操控性、行驶稳定性和轮胎磨损等方面产生影响。

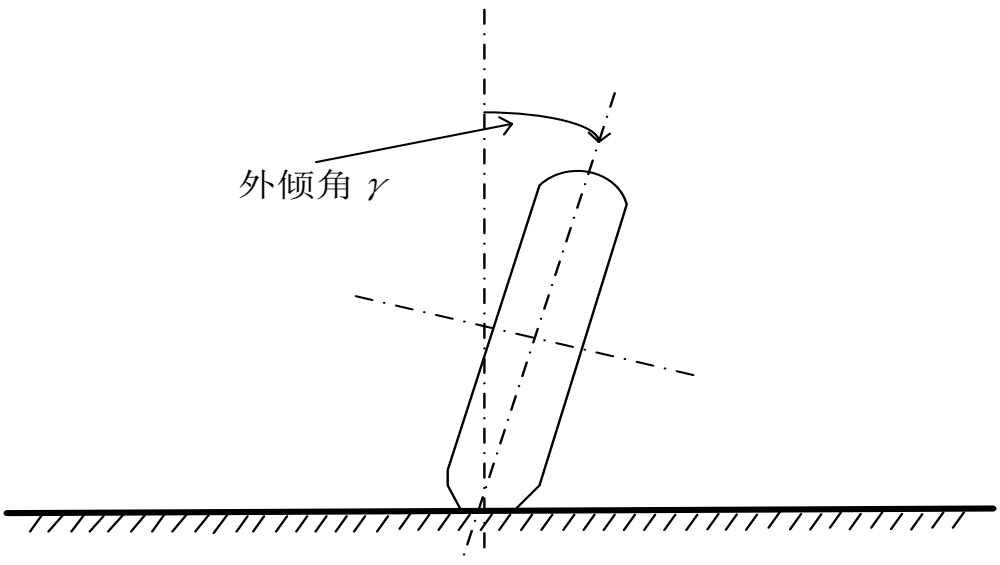


图 0.5 **外倾角示意图**

④ 车轮垂直载荷 F_z

车轮垂直载荷为车辆与载荷的重量分配到单个轮胎上的重量，一般处在平面上平稳运行或静止的车辆，其每个车轮上的垂直载荷相同。

⑤ 纵向力 F_x

轮胎纵向力是指轮胎在车辆行驶过程中产生的与车辆纵向运动相关的力，它主要包括牵引力和制动力，牵引力是指轮胎向前推动车辆的力，使车辆能够加速或保持匀速行驶。当驱动力施加在轮胎上时，轮胎与地面之间会产生摩擦力，这个摩擦力就是牵引力的来源。制动力是指轮胎在制动时产生的阻力，使车辆减速或停止。

$$\begin{cases} F_x = \left(D\,sin\left(C \times arctan\left(BX_1 - E(BX_1 - arctan(BX_1))\right)\right)\right) + s_v \\ X_1 = s + s_h \\ C = B_0 \\ D = B_1F_z^2 + B_2F_Z \\ B = (B_3F_z^2 + B_4F_Z) \times e^{-B_5F_z}/(C \times D) \\ s_h = B_9F_z + B_{10} \\ s_v = 0 \\ E = B_6F_z^2 + B_7F_z + B_8 \end{cases}$$

式中，s_h为曲线的水平方向漂移，s_v为曲线的垂直方向漂移。C 为曲线的形状因子，D 为曲线峰值因子，B 为刚度因子，E 为曲线的曲率因子。

⑥ 侧向力 F_y

汽车在行驶过程中，由于路面的侧向倾斜，曲线行驶时的离心力等作用，车轮中心沿车轴方向产生一个侧向力。 因为车轮是有弹性的，所以，在侧向力未达到车轮与地面间的最大摩擦力时，侧向力使轮胎产生变形，使车轮倾斜，导致车轮行驶方向偏离预定的行驶路线。

$$\begin{cases} F_y = \left(D\,sin\left(Carctan\left(BX_1 - E(BX_1 - arctan(BX_1))\right)\right)\right) + s_v \\ X_1 = \alpha + s_h \\ C = A_0 \\ D = A_1F_z^2 + A_2F_z \\ B = A_3\,sin\left(2\,arctan\frac{F_z}{A_4}\right) \times (1 - A_5|\gamma|)/(C \times D) \\ s_h = A_9F_z + A_{10} + A_8\gamma \\ s_v = A_{11}F_z\gamma + A_{12}F_z + A_{13} \\ E = A_6F_z + A_7 \end{cases}$$

⑦ 回正力矩 M_z

回正力矩是轮胎发生侧偏时，产生的作用于轮胎绕 OZ 轴的力矩。 圆周行驶时，回正力矩是使车轮恢复到直线行驶位置的主要力矩之一。

$$\begin{cases} M_z = (Dsin(Carctan(BX_1 - E(BX_1 - arctan(BX_1)))) + S_v) \\ X_1 = \alpha + s_h \\ C = C_0 \\ D = C_1 \cdot F_z^2 + C_2 \cdot F_z \\ B = (C_3 \cdot F_z^2 + C_4 \cdot F_z) \cdot (1 - C_6 \cdot |\gamma| \cdot e \cdot (-C_5) \cdot F_z)/(C \cdot D) \\ s_h = C_{11} \cdot \gamma + C_{12} \cdot F_z + C_{13} \\ s_v = (C_{14} \cdot F_z^2 + C_{15} \cdot F_z) \cdot \gamma + C_{16} \cdot F_z + C_{17} \\ E = (C_7 \cdot F_z^2 + C_8 \cdot F_z + C_9) \cdot (1 - C_{10} \cdot |\gamma|) \end{cases}$$

⑧ 翻转力矩 M_x

翻转力矩是轮胎发生侧偏时，产生的作用于轮胎绕 OX 轴的力矩。

$$M_x = -F_z \cdot D_e$$

其中，$D_e = F_y/L_s$ 为侧向形变，L_s 为轮胎侧偏刚度，其在轮胎模型中常被假定是一个常量。

⑨ 滚动阻力矩 M_y

轮胎所受到的与滚动方向相反的力矩即为轮胎滚动阻力。

$$M_y = F_z \cdot R_e \cdot R_p$$

这里，R_e 为轮胎的滚动半径；R_p 为滚动阻力系数。

表格 2 魔术公式参数取值表

A0	A1	A2	A3	A4	A5	A6	A7	A8	A9
1.65	-34	1250	3036	12.8	0.005	-0.021	0.7739	0.0022	0.0134
A10	A11	A12	A13	B0	B1	B2	B3	B4	B5
0.0037	19.1656	12.1356	6.262	2.3737	-9.46	1490	130	276	0.0886
B6	B7	B8	B9	B10	C0	C1	C2	C3	C4
0.0040	0.0615	1.2	0.0299	-0.176	2.34	1.4950	6.4166	-3.574	-0.0877
C5	C6	C7	C8	C9	C10	C11	C12	C13	C14
0.0984	0.00276	-0.0001	0.1	-1.3332	0.0255	-0.0235	0.0302	-0.0647	0.0211
C15	C16	C17							
0.8946	-0.0994	-3.3369							

2) 整车的力与力矩模型

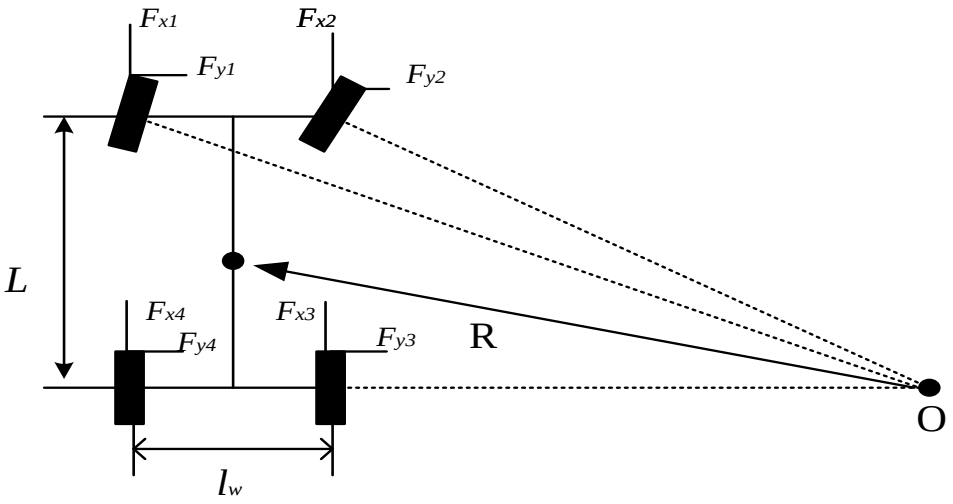


图 0.6 整车受力示意图

车辆的力与力矩模型可表达为如下模式：

$$\begin{bmatrix} F_X \\ F_Y \\ M_X \\ M_Y \\ M_Z \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ \frac{l_w}{2} & -\frac{l_w}{2} & -\frac{l_w}{2} & \frac{l_w}{2} \\ \frac{L}{2} & \frac{L}{2} & -\frac{L}{2} & -\frac{L}{2} \\ \frac{L}{2} & -\frac{L}{2} & -\frac{L}{2} & \frac{L}{2} \end{bmatrix} \begin{bmatrix} F_{x1} & F_{y1} & F_{z1} & F_{z1} & F_{y1} \\ F_{x2} & F_{y2} & F_{z2} & F_{z2} & F_{y2} \\ F_{x3} & F_{y3} & F_{z3} & F_{z3} & F_{y3} \\ F_{x4} & F_{x4} & F_{z4} & F_{z4} & F_{y4} \end{bmatrix}$$

其中，F_X，F_Y分别为车辆的纵向力与侧向力，M_X,M_Y,M_Z为整车所受三个方向的力矩；F_{xi}(i=1,2,3,4)，F_{yi}(i=1,2,3,4)分别为每个轮胎所受的纵、侧向力，已由轮胎模型魔术公式求得。

7.3.1.2 建模仿真案例

3.CustExps\e4_TrailerModelCtrl\Readme.pdf

7.3.2 阿卡曼底盘无人车

7.3.2.1 建模仿真案例

1) 基于最小输入输出接口的阿卡曼底盘无人车模型

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

2) 阿卡曼底盘无人车位置控制

*:\PX4PSP\RflySimAPIs\4.RflySimModel\2.AdvExps\e5_CarAckermanCtrl\1.CarAckermanPosCtrl_Py\Readme.pdf

3) 阿卡曼底盘无人车速度控制

*:\PX4PSP\RflySimAPIs\4.RflySimModel\2.AdvExps\e5_CarAckermanCtrl\3.CarAckermanVelCtrl_Py\Readme.pdf

7.3.3 差动无人车

7.3.3.1 建模仿真案例

1) 基于最小输入输出接口的差动无人车模型

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

其余见：2.AdvExps\e6_CarR1DiffCtrl\Readme.pdf

7.4 垂直起降无人机

7.4.1 4+1 垂起

7.4.1.1 建模仿真案例

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

7.4.2 四旋翼尾座式垂起

7.4.2.1 建模仿真案例

*:\PX4PSP\RflySimAPIs\4.RflySimModel\2.AdvExps\e4_VTOLModelCtrl\2.TailsitterMode

lCtrl\Readme.pdf

7.5 无人船

需补充

7.6 直升机

7.6.1.1 建模仿真案例

2.AdvExps\e8_Helicopter\Readme.pdf

7.7 水下无人艇

7.7.1.1 建模仿真案例

2.AdvExps\e9_UUV\Readme.pdf

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

PX4MavCtrlV4.py 库中的运动状态控制接口

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 targe 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 targe attitude to PX4 in the body forward-rightward-downward
(FRD) 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)
frame 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)
frame 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)
frame 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)
```

DllSimCtrlAPI.py 库中的 dll 模型外部数据输入接口

sendSILIntFloat

发送到 DLL 模型的 inSILInts and inSILFLoats 接口

```
def sendSILIntFloat(self,inSILInts=[0]*8,inSILFLoats=[0]*20,copterID=-1):
```

结构体如下

```
struct PX4SILIntFloat{
    int checksum;//1234567897
    int CopterID;//飞机的 ID
    int inSILInts[8];//int 标志位
    float inSILFLoats[20];//float 参数位
};
```

函数参数

- inSILInts：默认为一个 8 维的整数数组，所有元素初始化为 0。
- inSILFLoats：默认为一个 20 维的浮点数组，所有元素初始化为 0。
- copterID：目标无人机的 ID，默认值为-1，表示发送到自己。

函数逻辑

```
checkSum=1234567897
ID=copterID
if copterID<=0:
    ID=self.CopterID
PortNum = 30100+(ID-1)*2
buf = struct.pack("10i20f",checkSum,ID,*inSILInts,*inSILFLoats)
self.udp_socket.sendto(buf, (self.ip, PortNum))
```

- checkSum 设置为固定值 1234567897。
- 根据 copterID 的值确定目标无人机的 ID（ID）。如果 copterID 小于或等于 0，则使用默认的 CopterID。
- 计算端口号 PortNum，基于 30100 加上 (ID-1)*2 的结果。这意味着每个无人机的数据发送端口按照特定规则计算得出。
- 使用 struct.pack 函数将数据打包成一个二进制格式。格式字符串 "10i20f" 表示首先打包 10 个整数，随后是 20 个浮点数。这里的整数包括 checkSum、ID 和数组 inSILInts 的 8 个值；浮点数为数组 inSILFLoats 的 20 个值。
- 通过 UDP 套接字发送打包后的数据到指定的 IP 地址和端口（PortNum）。

sendSILIntDouble

发送到 DLL 模型的 inDoubCtrls 接口

```
def sendSILIntDouble(self,inSILInts=[0]*8,inSILDoubs=[0]*20,copterID=-1):
```

结构体如下

```
struct DllInDoubCtrls{
    int checksum;//校验码 1234567897
    int CopterID; // 飞机的 ID
    double inDoubCtrls[28];//28 维的 double 型输入
};
```

函数参数

- inDoubCtrls：默认为一个 28 维的双精度浮点数组，所有元素初始化为 0。
- copterID：目标无人机的 ID，默认值为-1，表示发送到自己。

函数逻辑

```
        checkSum=1234567897
        ID=copterID
        if copterID<=0:
            ID=self.CopterID
        PortNum = 30100+(ID-1)*2
        buf = struct.pack("2i28d",checkSum,ID, *inSILInts,*inSILDoubs)
        self.udp_socket.sendto(buf, (self.ip, PortNum))
```

- checkSum 设置为固定值 1234567897。
- 根据 copterID 的值确定目标无人机的 ID（ID）。如果 copterID 小于或等于 0，则使用默认的 CopterID。
- 计算端口号 PortNum，基于 30100 加上 (ID-1)*2 的结果。这意味着每个无人机的数据发送端口按照特定规则计算得出。
- 使用 struct.pack 函数将数据打包成一个二进制格式。格式字符串 "2i28d" 表示首先打包 2 个整数，随后是 28 个双精度浮点数。这里的整数包括 checkSum 和 ID 的 2 个值；双精度浮点数为实际结构体 inDoubCtrl 的 28 个值。
- 通过 UDP 套接字发送打包后的数据到指定的 IP 地址和端口（PortNum）。

sendInDoubCtrls

发送到 DLL 模型的 inDoubCtrls 接口

```
def sendInDoubCtrls(self,inDoubsCtrls=[0]*28,copterID=-1):
```

结构体如下

```
struct DllInDoubCtrls{
    int checksum;//校验码 1234567897
    int CopterID; // 飞机的 ID
    double inDoubCtrls[28];//28 维的 double 型输入
};
```

函数参数

- inDoubCtrls：默认为一个 28 维的双精度浮点数组，所有元素初始化为 0。
- copterID：目标无人机的 ID，默认值为-1，表示发送到自己。

函数逻辑

```
        checkSum=1234567897
        ID=copterID
        if copterID<=0:
            ID=self.CopterID
        PortNum = 30100+(ID-1)*2
        buf = struct.pack("2i28d",checkSum,ID,*inDoubsCtrls)
        self.udp_socket.sendto(buf, (self.ip, PortNum))
```

- checkSum 设置为固定值 1234567897。
- 根据 copterID 的值确定目标无人机的 ID（ID）。如果 copterID 小于或等于 0，则

使用默认的 CopterID。

- 计算端口号 PortNum，基于 30100 加上 (ID-1)*2 的结果。这意味着每个无人机的数据发送端口按照特定规则计算得出。
- 使用 struct.pack 函数将数据打包成一个二进制格式。格式字符串 "2i28d" 表示首先打包 2 个整数，随后是 28 个双精度浮点数。这里的整数包括 checkSum 和 ID 的 2 个值；双精度浮点数为实际结构体 inDoubCtrl 的 28 个值。
- 通过 UDP 套接字发送打包后的数据到指定的 IP 地址和端口（PortNum）。

sendModelInParams

方法定义和参数

```
def sendModelInParams(self, Bitmask, InParams, copterID=-1):
```

1.Bitmask: 一个 32 位的无符号整数，用来指示 InParams 数组中哪些参数需要被模拟器修改。

2.InParams: 一个包含 32 个双精度浮点数的数组，这些浮点数代表要发送到模拟器的参数值。

3.copterID: 无人机的标识符。如果这个参数小于或等于 0，将使用对象属性 self.CopterID 作为无人机 ID。

参数处理和计算

```
checkSum=1234567891
ID=copterID
if copterID &lt;= 0:
    ID=self.CopterID
PortNum = 30100 + (ID-1) * 2
```

4.checkSum：一个固定的整数值，可能用于数据包校验，确保数据的完整性。

5.ID：最终使用的无人机 ID，若传入的 copterID 有效则使用之，否则使用对象属性 self.CopterID。

6.PortNum：计算出的端口号，基于无人机 ID。每个无人机有一个基于其 ID 的独立端口，以 30100 为起始端口，每个无人机端口号间隔 2。

数据打包和发送

```
buf = struct.pack("2i32d", checkSum, Bitmask, *InParams)
self.udp_socket.sendto(buf, (self.ip, PortNum))
```

7.struct.pack("2i32d", checkSum, Bitmask, *InParams)：使用 struct 模块将 checkSum、Bitmask 和 32 个双精度浮点数打包成一个二进制字符串。格式字符串 "2i32d" 表示打包结构为两个整数和 32 个双精度浮点数。这里的 *InParams 表示将 InParams 列表中的元素作为参数展开。

8.self.udp_socket.sendto(buf, (self.ip, PortNum))：通过 UDP 套接字发送打包好的数据到

指定的 IP 地址和端口号。self.ip 是模拟器的 IP 地址，PortNum 是基于无人机 ID 计算得到的端口号。

sendInitInParams

结构体 PX4ModelInParams

```
struct PX4ModelInParams{
    int checksum; //1234567892 for InitInParams
    uint32 Bitmask;
    double InParams[32];
};
```

1.int checksum: 一个整数，用于校验数据包的完整性和正确性。对于初始化参数（InitInParams），校验和被设置为 1234567892，以区别于其他可能的数据包类型。

2.uint32 Bitmask: 一个 32 位的无符号整数，用作一个位掩码。每一位对应 InParams 数组中的一个元素，如果该位被设为 1，则对应的参数会被模拟器读取并可能被修改。

3.double InParams[32]: 一个包含 32 个双精度浮点数的数组，这些数值是要发送到模拟器的参数。

方法 sendInitInParams

```
def sendInitInParams(self, Bitmask=0, InParams=[0]*32, copterID=-1):
    checkSum=1234567892
    Bitmask=ctypes.c_uint32(Bitmask).value
    ID=copterID
    if copterID &lt;= 0:
        ID = self.CopterID
    PortNum = 30100 + (ID-1) * 2
    buf = struct.pack("iI32d", checkSum, Bitmask, *InParams)
    self.udp_socket.sendto(buf, (self.ip, PortNum))
```

4.Bitmask: 默认值为 0，表示不修改任何参数，除非明确指定。

5.InParams: 默认值为一个长度为 32 的列表，所有元素初始化为 0，代表参数的默认设置。

6.copterID: 默认为-1，表示如果不提供具体的无人机 ID，则使用类中存储的 self.CopterID。

7.ID: 最终用于计算 UDP 端口的无人机 ID。

8.PortNum: 计算得出的端口号，每个无人机有独立的端口号，基于 ID 计算。

9.ctypes.c_uint32(Bitmask).value: 将 Bitmask 参数确保为无符号 32 位整数。

10.struct.pack("iI32d", checkSum, Bitmask, *InParams): 使用 struct 库打包数据为二进制格式，"iI32d" 表示格式为一个整数，一个无符号整数和 32 个双精度浮点数。

11.self.udp_socket.sendto(buf, (self.ip, PortNum)): 通过 UDP 套接字将打包的数据发送到指定的 IP 地址和端口。

9. 综合模型

9.1 综合模型协议

9.1.1 背景

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。

模型参数：包含模型参数和控制器参数。CflightModel 中需要考虑增加控制器参数设置的接口。

输入接口：考虑综合模型发送给 CopterSim 或者从 CopterSim 接收的消息。

命令输入：用于控制无人机解锁、起飞、降落等基本过程。命令输入放入 InSIL 里面。

命令	是否支持
解锁	支持
起飞	支持
降落	支持
返航	支持
盘旋	适用于固定翼
设置当前期望位置	只要支持该命令即可完成轨迹控制

碰撞输入：由碰撞检测方案给出。

```
typedef struct {
  real_T TerrainZ;
  real32_T inFloatsCollision[20];
} Collision_Multicopter_T;
```

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 MVLink 设定的位置、速度、姿态指令。Offboard 模式主要用于执行复杂的空中机动，例如自动路径跟踪、目标追踪等。对于起飞、降落和返航等任务，通常会使用专门的飞行模式，如 AUTO.TAKEOFF、AUTO.LAND 和 AUTO.RTL 等

旋翼机 Mavlink Offboard 消息

SET_POSITION_TARGET_LOCAL_NED 期望位置(x, y, z)、期望速度(v_x, v_y, v_z)、期望加速度(a_{fx}, a_{fy}, a_{fz})。期望速度被加到位置控制器的输出，作为速度控制器的输入。期望加速度被加到速度控制器的输出，用于计算推力矢量。支持的坐标系包括 MAV_FRAME_LOCAL_NED，MAV_FRAME_BODY_NED。 SET_POSITION_TARGET_GLOBAL_INT 期望位置(lat_int, lon_int, alt)、期望速度(v_x, v_y, v_z)、期望加速度(a_{fx}, a_{fy}, a_{fz})。期望速度被

加到位置控制器的输出，作为速度控制器的输入。期望加速度的融入尚不完善。支持的坐标系 MAV_FRAME_GLOBAL。 SET_ATTITUDE_TARGET 1 姿态 SET_ATTITUDE_TARGET.q+油门 SET_ATTITUDE_TARGET.thrust 2 角速率 SET_ATTITUDE_TARGET body_roll_rate, body_pitch_rate, body_yaw_rate+油门 SET_ATTITUDE_TARGET.thrust。

如上是 PX4 官方支持的 Offboard 控制消息，坐标系和各种组合模式较为复杂。本 Offboard 控制的消息主要分为两类，一类是位置类，常用于编队飞行；一类是姿态类，常用于特技动作。

```
    # 设置期望位置、速度、加速度消息、偏航角、偏航角速率，对于位置使用纬经高时会化为整数
    set_position_target_local_ned_send(self, time_boot_ms, target_system, target_component, coordinate_frame, type_mask, x, y, z, vx, vy, vz, afx, afy, afz, yaw, yaw_rate, force_mavlink1=False)
    # 设置期望姿态、角速率、油门
    set_attitude_target_send(self, time_boot_ms, target_system, target_component, type_mask, q, body_roll_rate, body_pitch_rate, body_yaw_rate, thrust, force_mavlink1=False)
```

9.1.2 综合模型 Offboard 控制设计

综合模型支持的 Offboard 控制方式。

终端	是否支持
python Offboard	支持
Matlab Offboard	支持

9.1.3 输入输出接口说明

表 1 CopterSim DLL 模型输入接口说明

DLL 输入	CopterSim → DLL	Redis→CopterSim	结构体定义

inSILInts /inSILFloats	UDP 30100+index*2	Redis Key：key{30100+index*2}	struct PX4SILIntFloat{
			int checksum;//1234567897
			int CopterID;//飞机的 ID
			int inSILInts[8];//int 标志位
			float inSILFLoats[20];//float 参数位
			};

inDoubCtrls	UDP 30100+index*2	Redis Key：key{30100+index*2}	struct DllInDoubCtrls{
			int checksum;//校验码 1234567897
			int CopterID; // 飞机的 ID
			double inDoubCtrls[28];//28 维的 double 型输入
			};

inFromUE	UDP 30100+index*2	Redis Key：key{30100+index*2}	struct UEToCopterDataD{
			int checksum; //1234567899 为校验 ID
			int CopterID; //发出本消息

			的飞机 ID
			double inFromUE[32]; //通过蓝图发出的数据
			};

inCtrlExt1-5	UDP 30100+index*2	Redis Key：key{30100+index*2}	struct PX4SILIntFloat{
			int checksum;
			// port=1 2 3 4 5，对应 1234567881/2/3/4/5
			int CopterID;//飞机的 ID
			int inSILInts[8];//int 标志位
			float inSILFLoats[20];//float 参数位
			};
			//port=1 2 3 4 5，则 checkSum=1234567881/2/3/4/5，CopterSim 收到后，会按照新规则，将 inSILInts[8]（强制转为 float，这里会损失精度）和 inSILFloats[20] 合并为 28 维的 inCtrlExt1/2/3/4/5

表 2 CopterSim DLL 模型输出接口说明

DLL 输出	CopterSim → DLL	Redis→CopterSim	结构体定义

MavVehile3DInfo	UDP 20101+index*2	Redis Key：key{20101+index*2}	struct SOut2Simulator {
			int copterID; //Vehicle ID
			int vehicleType; //Vehicle type
			double runnedTime; //Current 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)
			};

outCopterData	UDP 30101+index*2	Redis Key：key{30101+index*2}	`struct outCopterStruct{`
			`  int checksum;`
			`//1234567890`
			`  int CopterID;`
			`  double data[32];`
			`//data`
			`};`

9.1.4 UE 小场景下协议

1．输入

1）inSILInts 协议

inSILInts 的第 0 个数字用于表征和修改状态，相应位为 1 时表明系统为相应的状态。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:Hor	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:
Offboad 位置控制系列	Offboad 位置姿态系列						
24:	25:	26:	27:	28:	29:	30:	31:
毁伤标志							

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:hasPos	1:hasVel	2:hasAcc	3:hasYaw	4:hasYawRate	5:	6:	7:
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 NED	位置和速度 Global						
24:	25:	26:	27:	28:	29:	30:	31:

2）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）inCtrlExt1-5 协议

包含了 inCtrlExt 1-5 系列接口，28 维 float 型数据输入。将 inCtrlExt1 的前 8 维转为 int 型数据，作为标志类指令；剩余接口作为故障类消息输入。

inCtrlExt1[0] 故障注入标志位

0:	1:	2:	3:	4:	5:	6:	7:
电机执行效率故障	螺旋桨故障	电池失效故障	低电压故障	低电量故障	负载故障	负载漂移故障	负载泄露故障
8:	9:	10:	11:	12:	13:	14:	15:
常风故障	阵风故障	紊流风故障	切向风故障	加速度计噪声干扰	陀螺仪噪声干扰	磁力计噪声干扰	气压计噪声干扰
16	17	18:	19:	20:	21:	22:	23:
GPS 故障							
24:	25:	26:	27:	28:	29:	30:	31:

故障注入类消息协议如下：

```
inCtrlExt1:
inCtrlExt1[9-16]: #1~#x 号电机执行效率比（取值范围 0~1，预留到 8 个电机）
inCtrlExt1[17-24]: #1~#x 号螺旋桨执行效率比（取值范围 0~1，预留到 8 个电机）
inCtrlExt1[25]: 电压失效比（0~1）
inCtrlExt1[26]: 电量失效比（0~1）
inCtrlExt1[27]: 重量泄露比（0~1）
inCtrlExt2:
inCtrlExt2 [0-2]: x,y,z 的泄露因子（0~1）
inCtrlExt2 [3-5]: X,y,z 轴的风速
inCtrlExt2 [6]: 阵风强度
```

```
inCtrlExt2 [7]：风到达时间
inCtrlExt2[8]：紊流风强度
inCtrlExt2[9]：切向风强度
inCtrlExt2[10]：加速度计噪声增益
inCtrlExt2[11]：陀螺仪噪声
inCtrlExt2[12]：磁力计噪声
inCtrlExt2[13]：气压计噪声
inCtrlExt2[14]：GPS 噪声增益
inCtrlExt2[15]：3D 方式
inCtrlExt2[16]：星数
```

2．输出

1）MavVehile3DInfo 协议固定，具体内容参照 SOut2Simulator 结构体。

2）outCopterData 协议

32 维 double 型数据，支持自定义，作为预留接口，目前能想到会使用到的地方：

1） UE 内置模型，作为控制量的转发。

2） UE 调用 DLL 模型。

3） 客户需求，比如电池电量、续航时间

```
outCopterData [0-2] =pos;
outCopterData [3-5]=vel; // 速度|遥控器俯仰、滚转、偏航信号|inSILFloats[3]可以用作速率
outCopterData [6-8]=acc;
outCopterData [9-11]=att;  // 姿态控制使用欧拉角，对于用户来说更加直观。
outCopterData [12-14]=attRate;
outCopterData [15]=thrust; // 油门
outCopterData[16] //电池电量
outCopterData[17] //续航时间
```

9.1.5 UE 大场景下协议

1．输入

1）inDoubCtrls 协议

28 维 double 型数据输入，这里将前 8 维转为 int 作为标志类指令，后 20 维作为消息类指令，拆分成 inSILInts 和 inSILDoubs。

协议与 inSILInts、inSILFloats 一致，唯一区别在于消息类指令（对应小地图协议的 inSILFloats）精度从 float 提升到双精度 double。

2）inCtrlExt1-5 协议

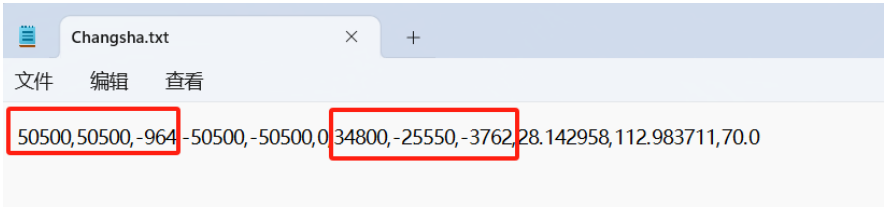
同小场景下 inCtrlExt1-5 协议，用作故障注入标志位和消息输入。

2.输出

同小场景下的输出协议。

大场景下的经纬高坐标原点通过 png 和 txt 设置，其中 png 为地形高程信息，png 地形文件实际上是以图片形式存储的二维矩阵，包含了场景的高程图。以 png 格式存储矩阵能够很好的实现高程矩阵的压缩，便于节省空间。png 的高程文件并不包含坐标原点、缩放尺度、场景范围等信息，因此需要一个校正文件， RflySim 平台采用 txt 格式输入 9 维数组或 12 维数组（多出经纬高地理基准信息） 传入校正信息。

txt 文件为地形校准数据，平台中 txt 校正文件存储的是右上角三维坐标点（ xy 全为正，z 向上为正）、左下角三维坐标点（ xy 全为负， z 向上为正）、第 3 点三维坐标点，单位均为厘米。前两个点的目的是为了确认地形的范围和中心坐标，第 3 点坐标可自行选取，理论上需要尽量在高度上与前两个点有落差，用于校正高度尺度。



除此之外，当在 RflySim3D 中使用 cesium 全球大场景（仅个人高级版以上） 时， 在 txt 中，还可以上述 9 维数据之后输入三维 GPS 地形数据，按纬度、经度、高度顺序加入，可配置 QGroundControl 中对应的显示基准坐标。

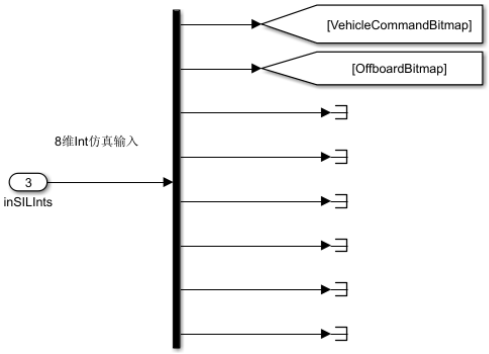
9.2 综合模型实现

综合模型采用 MATLAB 自动代码生成 DLL 模型实现，DLL 模型封装了必要的接口，供 CopterSim 加载和调用。

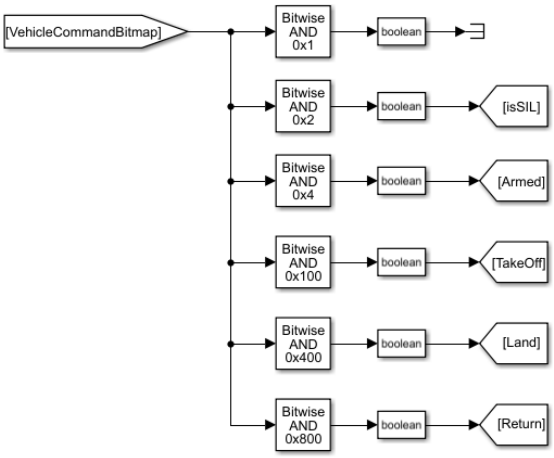
9.2.1 旋翼机综合模型实现

9.2.1.1 协议解析

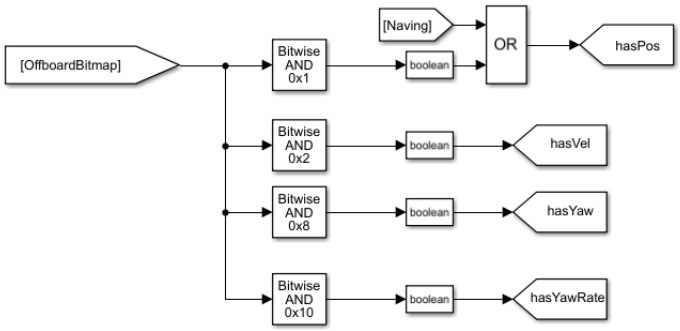
协议解析是将 CopterSim 收到的网络包解析成 3.2 节的指令。inSILInts 是一个 8 维的输入，当前仅使用第 0 个数指令和第 1 个数 Offboard 模式。后续第 6 个数和第 7 个数将作为 Global 坐标系下的纬度和经度的整数表示。下图中，将 inSILInts 向量，分解为了 8 个单独的数。



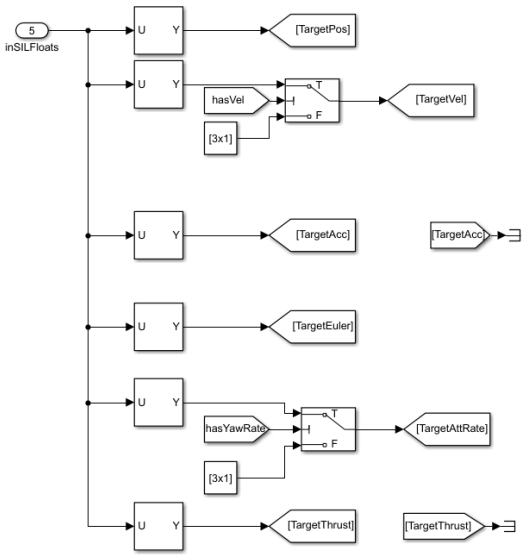
指令的每一位都有相应的含义，所以还需进一步对每一位进行解析。如下图使用 Bitwise 模块对位进行解析，第 0 位暂时没有使用。第 1 位标识是否是仿真模式，当该位为 1 时表示使用仿真模式，当该位为 0 时表示硬件在环模式，只有仿真模式时综合模型中的控制器才生效。第 2 位表示是否解锁，其它位还包括起飞、降落、返航功能。



如下是标识有哪些 offboard 控制信息的标志。3.2.2 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。

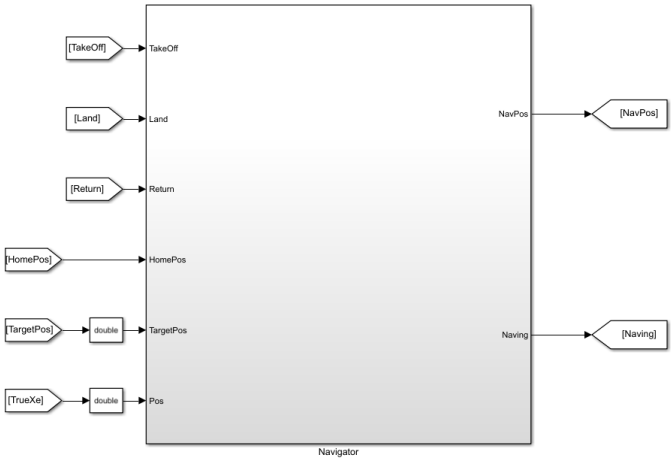


进一步，如下图所示解析 offboard 发送的实际位置、速度值等。在上图中是为了标识相应的值存不存在，而下图则是具体的数值。当前仅使用位置、速度、偏航、偏航角速率。在下图的解析中，使用了 hasVel 标志和 hasYawRate 标志。当这两个标志为假时，表明没有速度和偏航角速率的输入，这时将切换为遥控器模式。在遥控器模式下，1500 代表期望速度或期望偏航角速率为 0。



9.2.1.2 Navigator

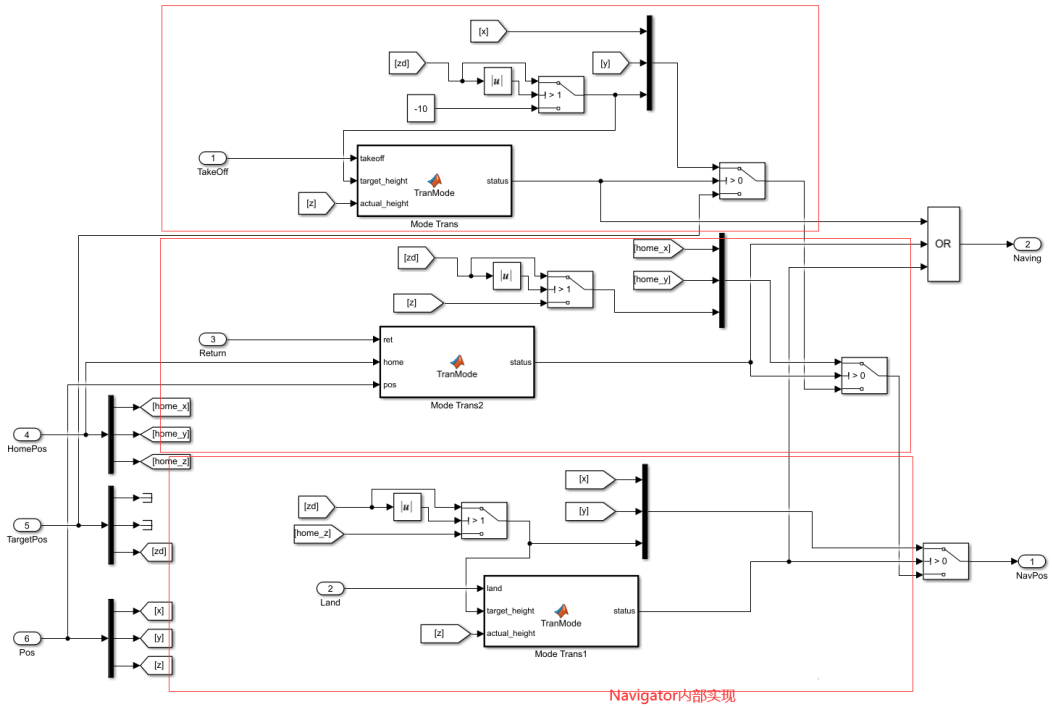
为了让综合模型能够接收上层指令，如起飞、降落、返航等功能，需要将这些上层指令转换为一系列的期望位置。这个工作由 Navigator 模块完成。Navigator 首先需要接收这些指令，然后在得到期望位置时需要依赖当前位置信息和用户设定的期望位置信息。举例而言，用户在起飞、返航、着落时都可以设置高度。此外用户发送起飞指令从设计上只需发送一次，那么 Navigator 就需要自己去判断起飞是否完成，这样就需要同时知道期望位置和实际位置。



如下图，是 Navigator 的内部实现。最上面是起飞模块，起飞模块的优先级最低。也就是同时收到起飞、返航、降落的优先级关系为：起飞<返航<降落。当用户没有指定高度时，其默认高度是 NED 坐标下-10m。考虑到从设计上来讲，只需成功接收到一次起飞指令，后续 Navigator 就会持续生成期望位置直至起飞到指定高度。所以设计了 Mode Trans 模块，该模块的作用是当接收到起飞命令时切换到起飞模式，当飞机到达指定高度时将切换到位置模式。在起飞过程中，飞机不会响应用户设定的位置。

在返航时，飞机将从当前位置保持高度不变返航到 home 点位置。返航也可设置返航高度，当没有指定高度时将以上一时刻的高度作为期望高度，即做高度保持。因为默认情况下不设置高度时，高度值为 0，考虑到 GPS 定位精度一般不超过 1m，所以将 1m 作为是否设置了高度的判据。home 点是模型运行时，第一次运行到位置采集时记录的初始位置。

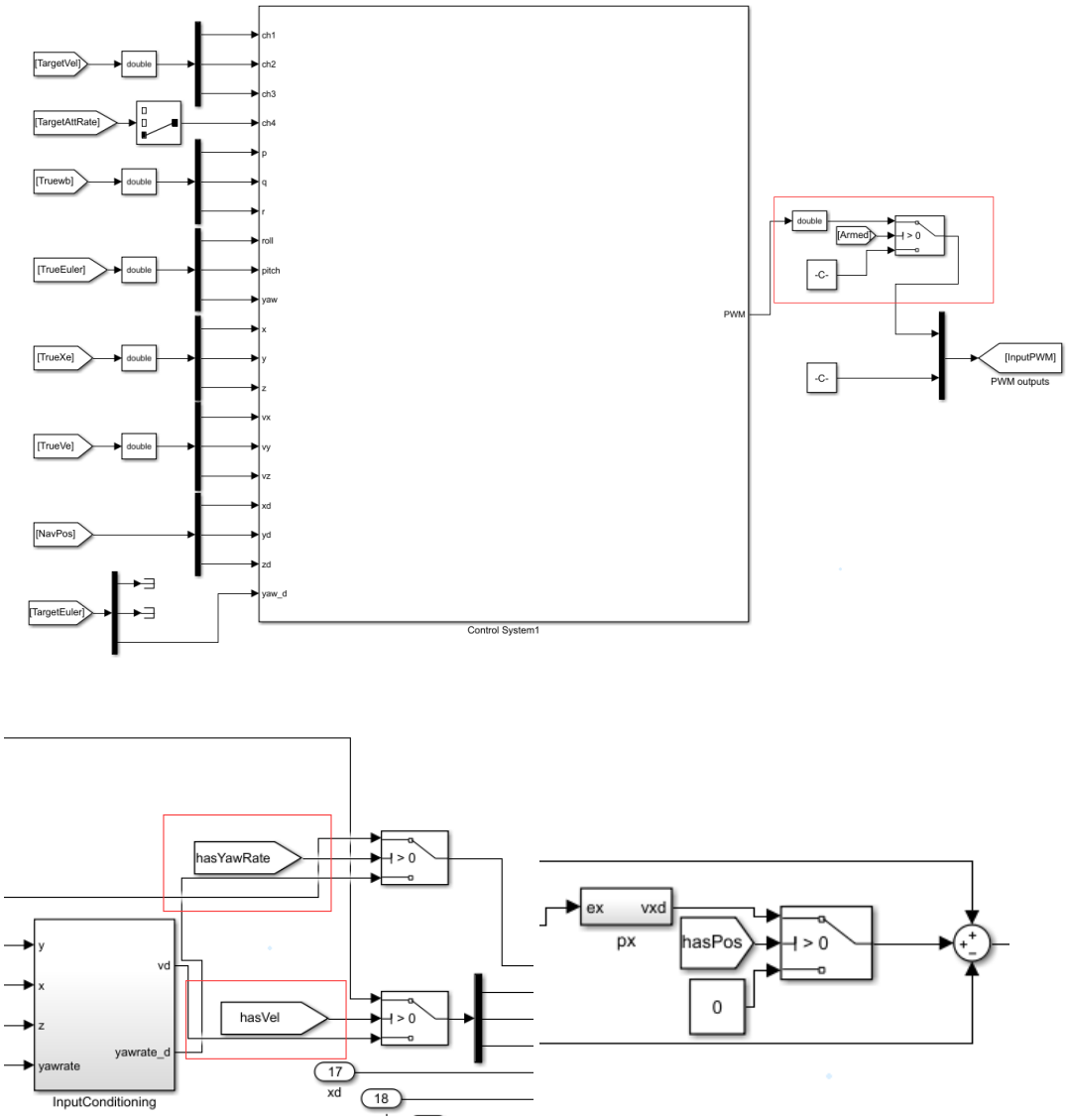
着陆是优先级最高的指令，着陆时同样可以指定高度。这是考虑到，着陆并不一定在 home 点位置着陆。其它外部模块可以检测当前离地面的距离，这样就可以确定着陆高度，所以着陆时预留了设置高度接口。



Navigator 的输出是 NavPos，是指期望的位置信息。Naving 是指 Navigator 处于执行指令的过程中，当 Naving 为 True 时，会把 hasPos 设置为真。

9.2.1.3 控制器

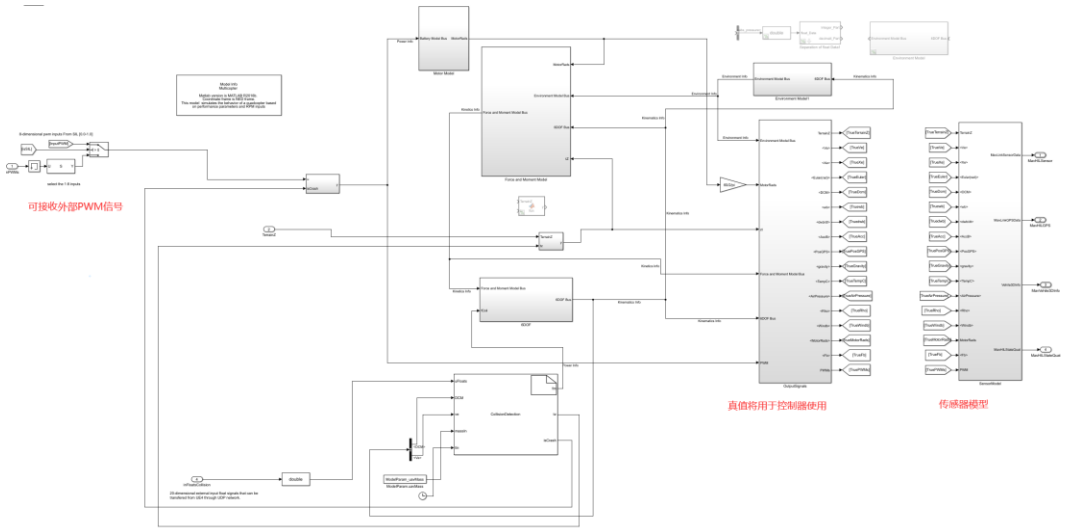
旋翼机综合模型是一个四串级的 PID 控制器，包含位置环、速度环、姿态环和角速度环。当前支持位置、速度、偏航角、偏航角速率的控制。当不进行位置控制时，hasPos 标志为假，位置控制器的输出为 0，这可在下图中观察到。当没有收到速度信号或者偏航角速度信号时，将进入遥控器控制模式。当前只是预留了遥控器控制接口，遥控器的值设定为 1500，没有真正启用遥控器。



9.2.1.4 模型

如下图所示是四旋翼的模型。该模型可以接收内部控制器的输入，也可以接收外部控制器的输入。这两种模式通过 inSIL 进行控制，当该标志为 true 时，表明是仿真模式，从内部控制器获得 PWM 波脉宽。当该标志为 false 时将从外部获得 PWM 波。

模型中包含的主要模块：电机模型、6DOF 动力学模型、6DOF 运动学模型、环境模型、碰撞模型、传感器模型。对于内部控制，由于没有设计滤波器，所以直接使用真值。而输出到 CopterSim 的传感器数据是带了噪声的，环境模型的影响会体现在传感器上。



9.2.2 固定翼综合模型实现

9.2.2.1 协议解析

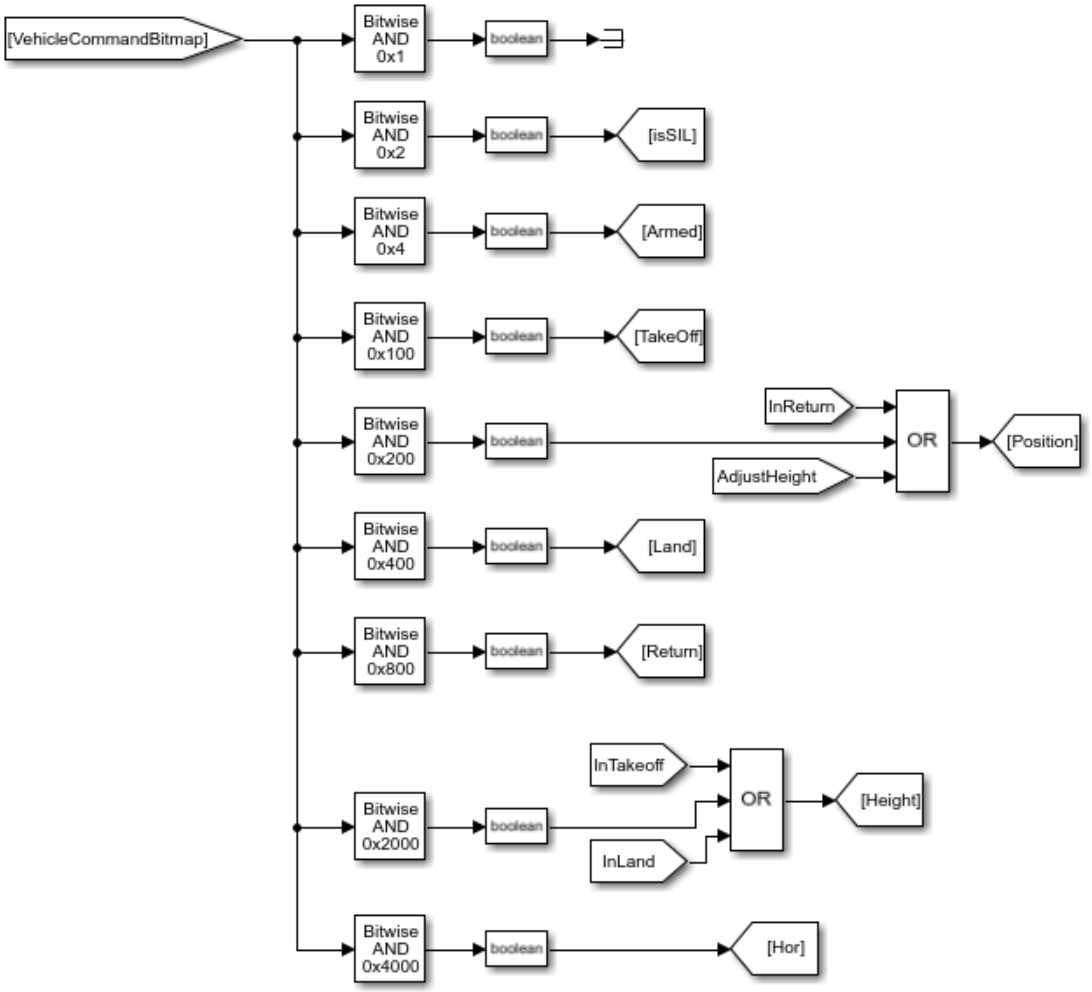
固定翼和旋翼相比遵循相互兼容的协议，但是固定翼的模式与旋翼有所不同，因为固定翼并不能像四旋翼那样悬停。如下图所示，是固定翼综合模型内的协议解析。增加了 Positon、Height、Hor，而且起飞、返航、降落的行为也与旋翼机有所不同。

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° 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。

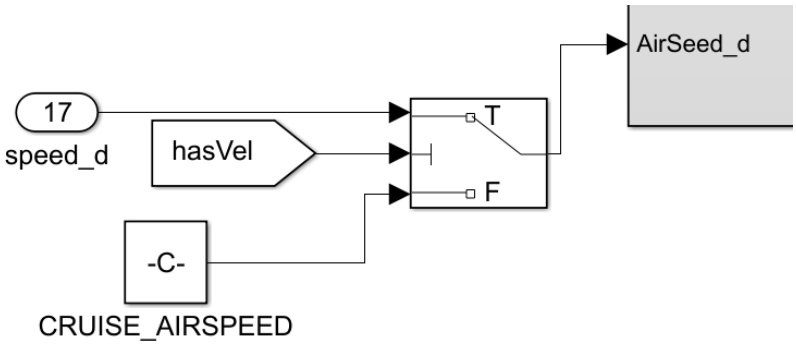
Position，同时控制水平位置和高度。用户可以通过 inSILInts 直接指定模式为位置控制，当返航或者降落时系统也可以自动触发位置模式。在位置模式下，如果设定了相应的位置，先飞到相应的位置。到达指定位置之后，如果没有指定下一个位置，将自动盘旋。在返航模式时，本质上就是水平位置回到出发点并且支持指定返航高度，所以这个功能可以使用 Position 模式进行实现。而在降落时，首先会调整飞机的高度，图中 AdjustHeight 就是描述的这一过程。调整飞机的高度是通过盘旋的方式完成的，在盘旋时水平位置也会发生变化，所以也采用位置模式进行控制。

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。

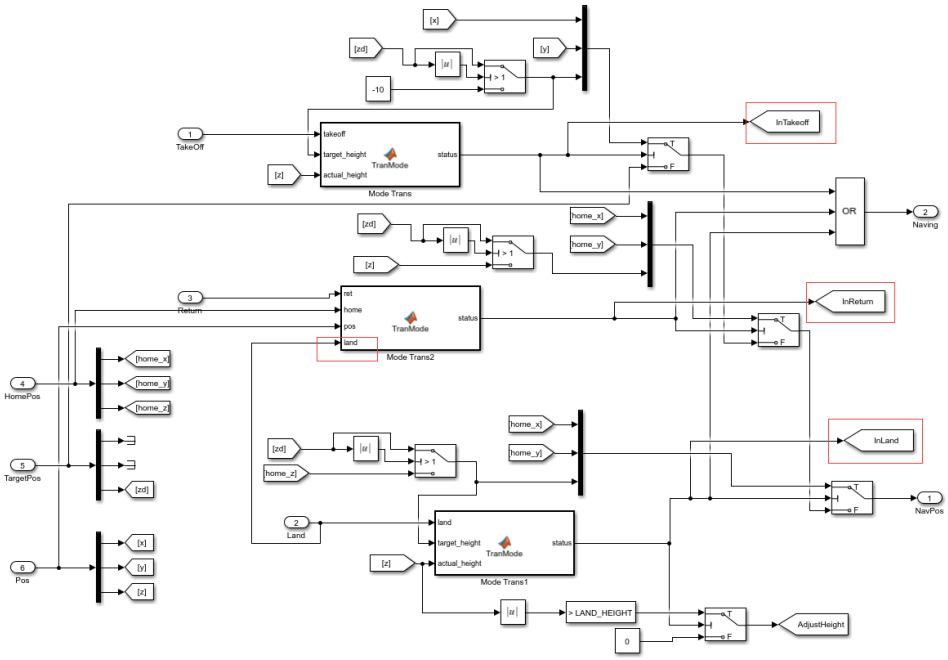


9.2.2.2 Navigator

固定翼 Navigator 的实现与旋翼机类似，但是新增了 InTakeoff、InReturn、InLand、AdjustHeight 等标志。因为固定翼在起飞、返航、降落的过程中有不同的操作，所以用这些标志来进行区分。在起飞时，屏蔽 TECS 控制器俯仰的输出，直接设置为一个固定角度 15°。而在返航过程中，需要自动触发完整的位置控制。降落时，则分为两个阶段一个是

高度调整阶段，用 AdjustHeight 标识，此阶段触发完整的位置控制；另一个是降落阶段，仅对高度进行控制，由于地面摩擦力没有建模，所以降落时的速度暂时也没有进行控制。

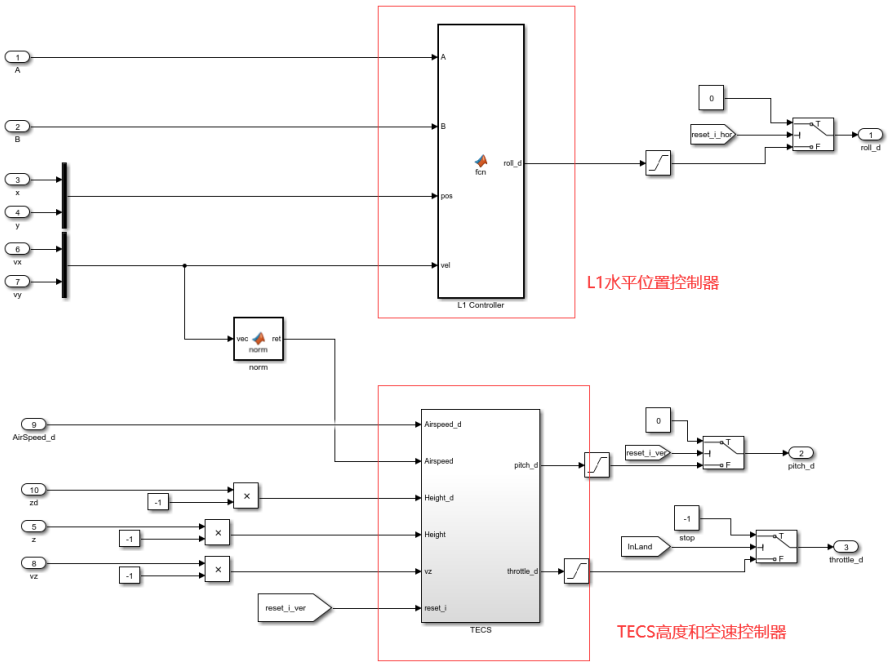
此外，返航模式执行后，只能执行降落模式。下面的模式切换中，只有收到降落指令才会从返航模式中退出。



9.2.2.3 控制器

固定翼的位置控制器和旋翼机的位置控制器大相径庭，原因是固定翼一旦起飞就需要保持一定的速度。

如下图，使用 L1 控制器控制水平位置。这里不解释控制器的内部实现，只关注控制器的输入输出。L1 控制器需要根据当前的期望水平位置、上一个期望水平位置、当前的水平位置、当前水平速度联合计算出期望的滚转角。由此可见，L1 控制器输入很多，但是输出只有一个滚转角。L1 控制器并不会在整个飞行过程中都生效，采用 switch 的这种方式简化了 L1 控制器的内部实现。



使用 TECS 控制器进行高度和速率控制。对于固定翼而言只控制器空速，不能任意控制各个方向的速度。TECS 提供了一种解决方案，即根据能量而不是初始设定值来反映问题。一架飞行器的总能量是飞行器动能和势能之和。推力（通过油门控制）可以增加飞机的总能量。一个给定的总能量状态可以通过势能和动能的任意组合来实现。 换句话说，飞行器在高海拔以低空速飞行和在低海拔以高空速飞行时的总能量是等价的。我们称这种情况叫做比能量平衡，它是根据当前高度和真实空速设定值计算的。 可以通过控制俯仰角来控制飞行器的比能量平衡。俯仰角增加将动能转变为势能，俯仰角减少则情况相反。这样，通过将初始空速和海拔设定值转化为能量大小（空速和海拔存在耦合，而能量大小可以独立控制），就可以把控制问题解耦。我们利用油门调节飞行器的特定总能量，利用俯仰角来维持势能（高度）和动能（真空速）的特定平衡点。

TECS 控制器输入期望的速率和高度值，并输入当前的高度和速率值，最后输出期望的俯仰角和油门值。

10. 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.

[3].