API specification file search outline

1、General introduction to motion modeling interfaces

1.1 Design idea of motion model for unmanned system on RflySim platform

1.1.1 Extracting similar components from different systems

Figure 1.1 Common system architecture of different types of unmanned vehicle systems11

Different types of unmanned systems (such as unmanned vehicles, unmanned aerial vehicles, unmanned ships, etc.) have different shapes and operating environments, but they have many common characteristics from the system structure diagram that can be reused in a large number of modeling and simulation systems. Therefore, the model framework here adopts a modular approach to divide the entire unmanned model system into several subsystems, so as to maximize these common factors and simplify the complex modeling problem. At the same time, this approach also helps to share the same model between different types of unmanned systems, and can be automatically implemented in model-based design software (such as MATLAB/Simulink). The corresponding environment configuration is shown in 2.

1.1.2 From component model to complete machine model

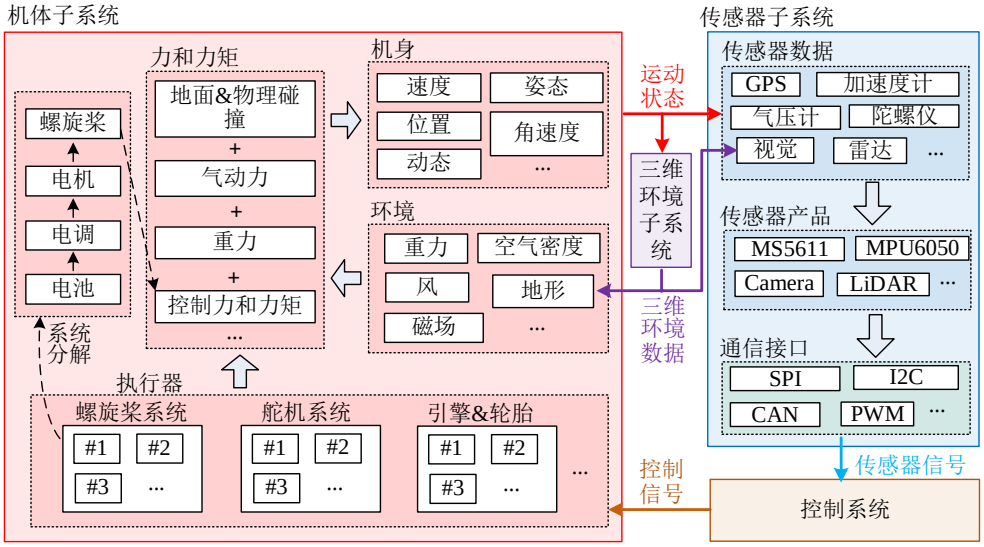


FIG. 1.2 Unified framework of motion model12

Unmanned system vehicle motion model can be decomposed into errors through physical and virtual components 错误!未找到引用源。Unified modeling framework shown. In this framework, the whole motion model system can be divided into three subsystems: the body subsystem, the sensor subsystem and the three-dimensional environment subsystem.

- The body subsystem includes actuator, body, operating environment, force and torque, which is the overall description of the body's motion, energy consumption and fault characteristics in the environment.
- The sensor model is mainly used to describe all the electronic hardware models except the control software, which mainly includes sensor data, communication protocol, connection interface and other characteristics.
- The 3D environment model is mainly used to describe the 3D visual environment of UAV flight (including trees, obstacles, roads, etc.), which is used to provide visual data simulation for the autonomous control system.

Each subsystem can be subdivided into smaller independent subsystem modules, and finally form a modular unified modeling framework as shown in the figure above.

1.1.3 General Modeling interface

See 3, Basic operation process

See 5, DLL/SO model and Communication interface for general input and output interface

1.2 Overall classification of RflySim platform motion models

1.2.1 Dynamic model

Without the controller, only according to the dynamic characteristics of the unmanned system

modeling, see 6, Simulink modeling template introduction

1.2.2 Synthesis model

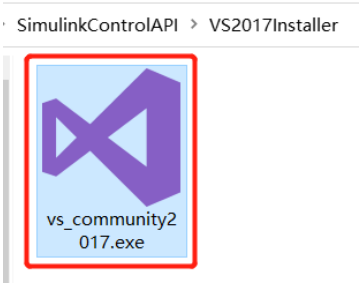
The controller is implemented based on the original dynamic model, see 9. Synthesis Model

2. Environment configuration

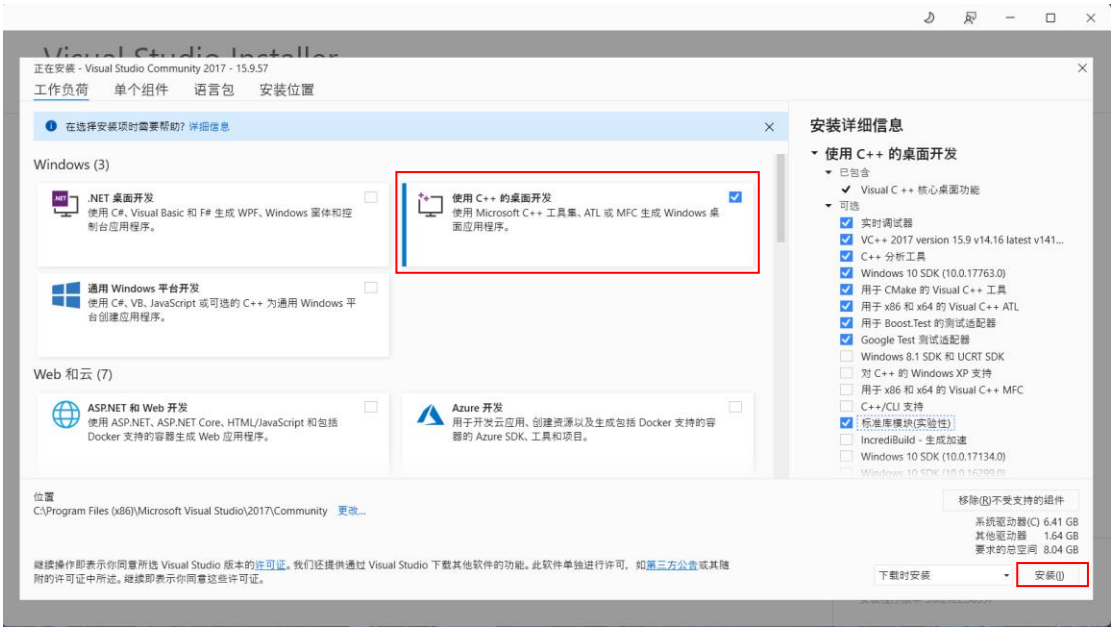
2.1 Visual Studio Installation

RflySim model development requires Visual Studio compiler, such as the use of MATLAB S-Function Builder module, Simulink automatic generation of C/C++ model code, etc. It is recommended to install Visual Studio 2017 here. The online installation steps (Internet connection required) are as follows:

Step 1: Double-click to run "RflySimAPIs3.0\4.RflySimModel\ 1.Basicexps \VS2017Installer\vs_community2017.exe";



Step 2: Select "Desktop Development using C++", click "Install" in the bottom right corner, and wait for the installation to complete.



Note: 1. Higher version of MATLAB can also install VS2019, but MATLAB can only recognize Visual Studio lower than its own version, so MATLAB2017b cannot recognize VS2019.

2, please do not change the default VS installation directory (such as installing to the D disk),

it will cause MATLAB can not recognize.

2.2 MATLAB compilation environment configuration

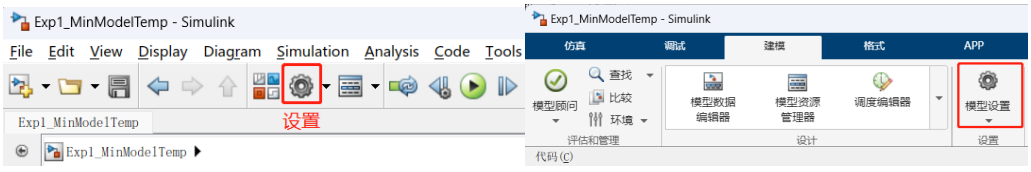
In the MATLAB command line window, enter the command "mex-setup", in general, the VS 2017 compiler will be automatically recognized and installed, as shown in the right picture, "MEX configuration uses' Microsoft Visual C++ 2017 'for compilation" shows that the installation is correct.

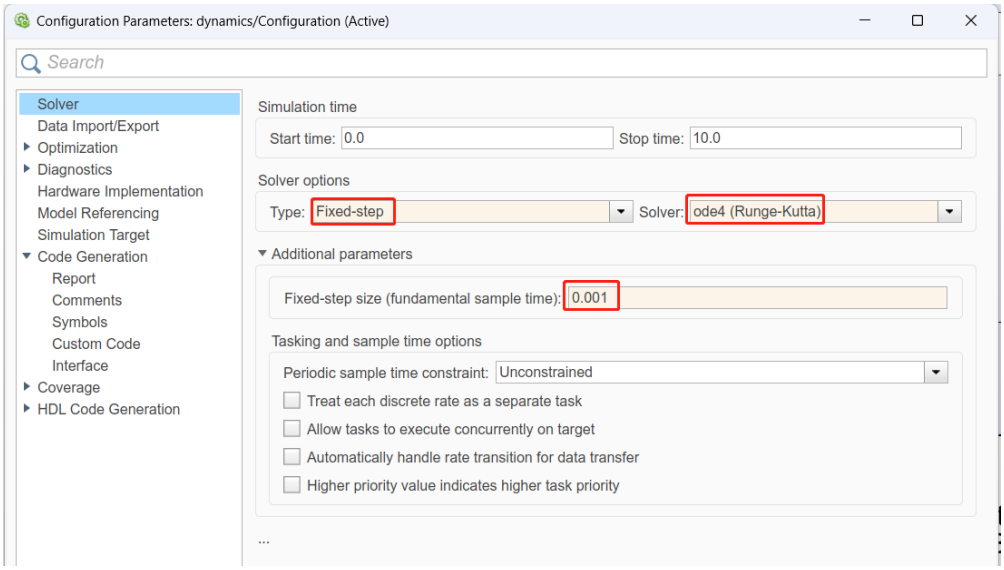
2.3 Simulink code generation setup

Before using the RflySim platform to simulate the hardware/software in the loop of the vehicle model, it is necessary to compile the Simulink source program of the model and generate the DLL file.

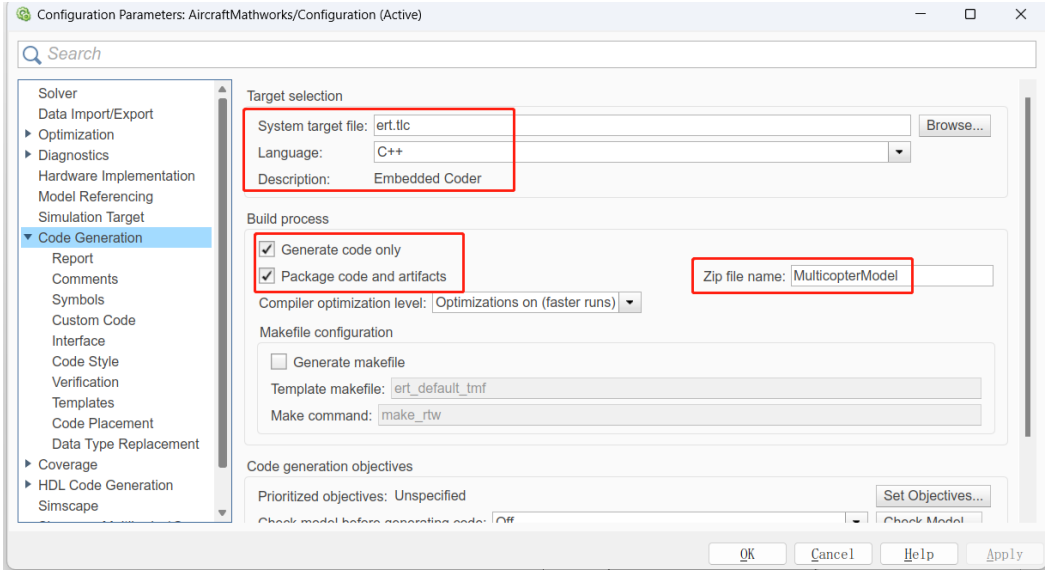
The steps of Simulink model compilation setup are as follows:

Step 1: Open the Simulink "Settings" page and set the simulation as a Fixed-step long, fourth-order Runge-Kutta solver with a step size of 0.001s (or other Settings can be set according to requirements).

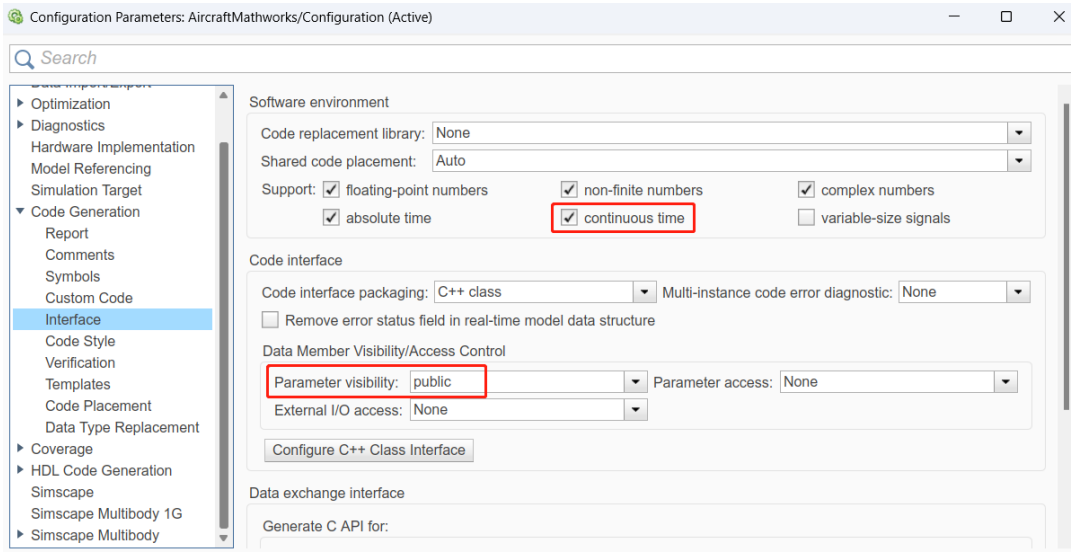




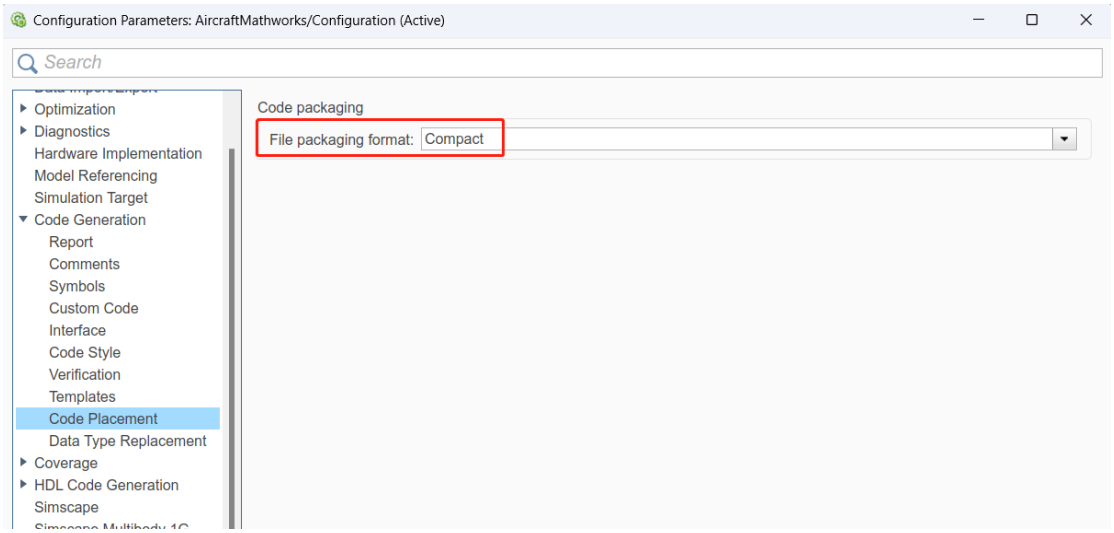
Step 2: Choose ert.tlc code generation method, which can be used in windows, Linux and various embedded platforms; Choose C++ language, easy to call through inheritance code generation; The compilation process selects "code and tools packaging", and the Zip file name is set to "MulticopterModel".



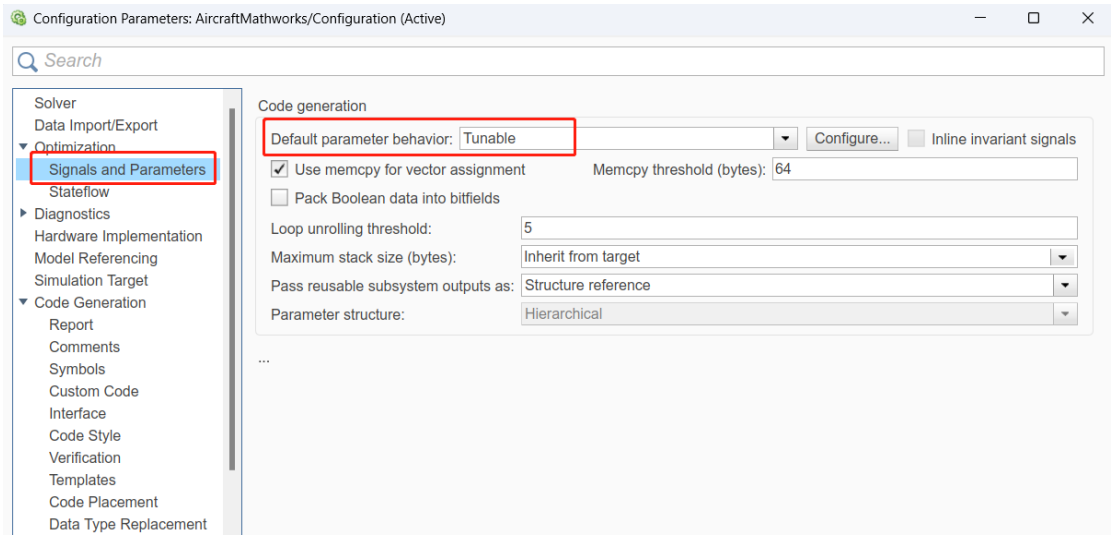
Step 3: Because it contains continuous modules (integral modules), it is necessary to check continuous time, otherwise the compilation will give an error. In addition, the Parametervisibility is set to public, and the parameter structure is a common variable for easy access.



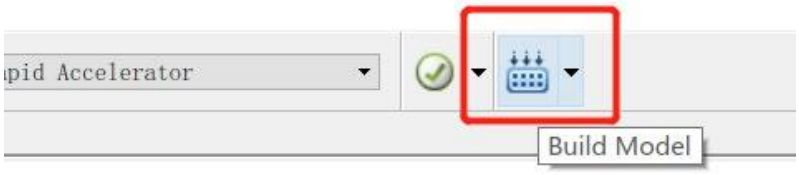
Step 4: In the Codeplacement page, set the file packing type to compact to avoid generating extra files and make the code the most readable.



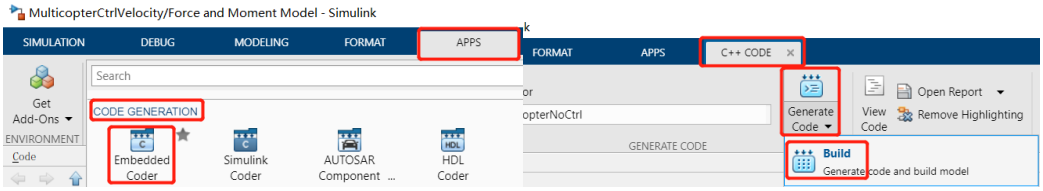
Step 5: Set the parameters to be Tunable so that we can modify the parameters at runtime. Note: inline form is more memory saving, but it is not easy to access parameters, it is not easy to realize real-time parameter modification or model fault injection.



After completing the vehicle Simulink model compilation setting, click the Simulink compilation button to generate C/C++ code, as follows: For MATLAB 2019a and previous versions, the toolbar style is shown below, directly click its compilation button "Build".



For 2019b and later versions, click Aps-Code GENERATION -- Embedded Coder to pop up the CODE GENERATION toolbar. In the toolbar, click "C++CODE" - "Generate Code" - "Build" button as shown below to compile and generate code.



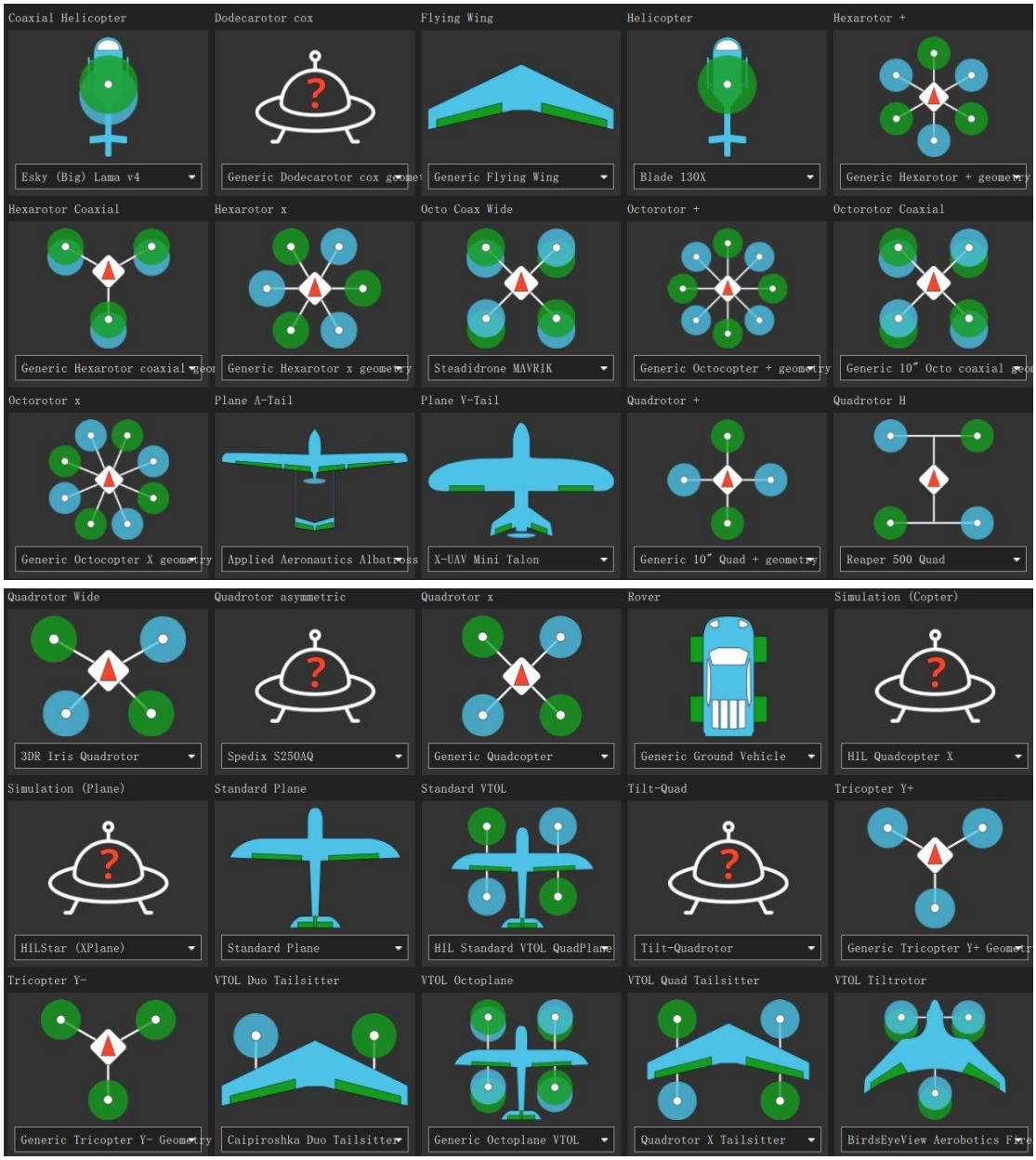
2.4 Environment Configuration for Linux version

3. Basic operation process

3.1 Vehicle Simulink model development

The RflySim vehicle model is developed based on MATLAB/Simulink, and the modular modeling idea is adopted. The dynamic motion model of unmanned vehicle is divided into motor module, force and torque module, link module, 6-DOF module and sensor module.

RflySim supports the software and hardware in the loop simulation of any PX4 controllable model. All supported models can be viewed from the Airframe page of QGroundControl, as shown in the following figure.

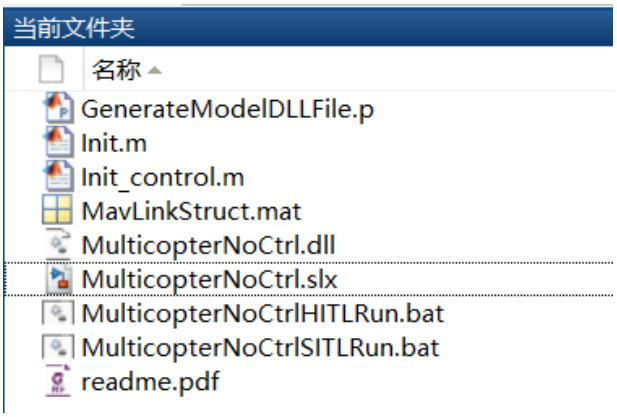


At present, RflySim includes rotor, fixed wing, unmanned vehicle, unmanned ship, standard vertical take-off and landing UAV, quadrotor tail type vertical UAV and helicopter. Other models need to be built by users in Simulink according to the RflySim model template.

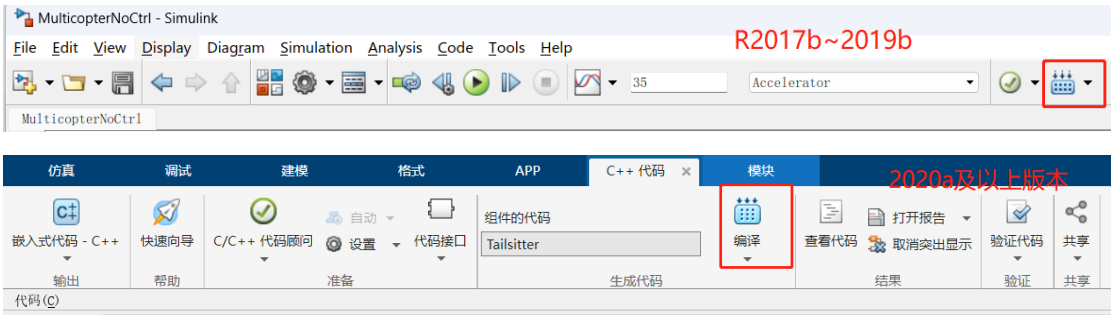
The RflySim model template is shown in RflySimAPIs3.0\4.RflySimModel\0.SourceCode\DLLModelTemp. The criteria for evaluating the completion of model development are: the model logic is correct; No errors were reported when the model was run.

3.2 The model is compiled to produce C/C++ files

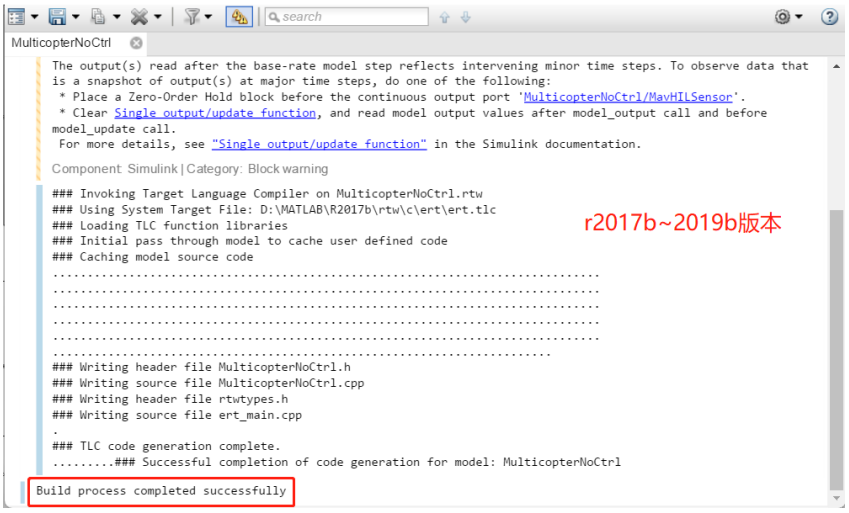
After the vehicle Simulink model is developed, open the model source file through MALTAB (for example, MulticopterNoCtrl.slx for quadrotor).



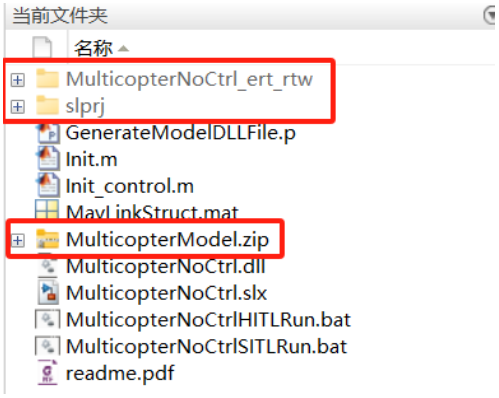
In the upper menu bar of Simulink, click compile command (MATLAB 2017b~2019b version can be directly click compile in the menu bar, 2022a and above version operation process is: APP-Embedded Coder-compile).



Click the View diagnostics command at the bottom of Simulink, and a diagnostic dialog box will pop up to view the compilation process. In the diagnostic box, "Build process completed successfully" will pop up, which means that the compilation is successful.



After the Simulink model has been compiled, the ***_ert_rtw folder and the MulticopterModel.zip compression package will be generated, indicating that the model has been compiled successfully.

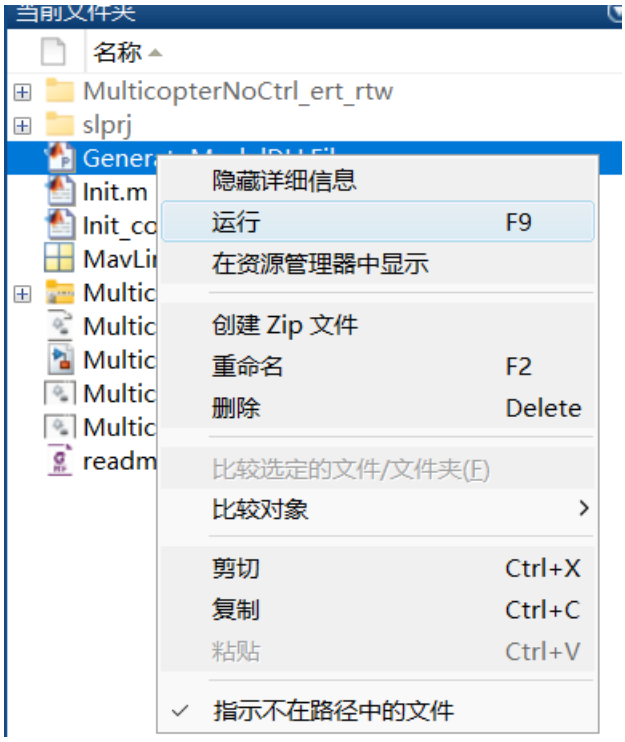


3.3 DLL/SO model generation

The DLL(under windows)/SO (under Linux) model needs to be imported into CopterSim to form the motion simulation model when the vehicle software and hardware are simulated in the loop using RflySim platform. Therefore, the C++ files corresponding to the model need to be packaged into the DLL/SO model after the model compilation is completed.

Under Windows system:

Once you have the ***_ert_rtw folder and the MulticopterModel.zip archive, run GenerateModelDLLFile.p to get the DLL model.



As shown below, the DLL model is successfully generated.

```
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 >>
```

On Linux:

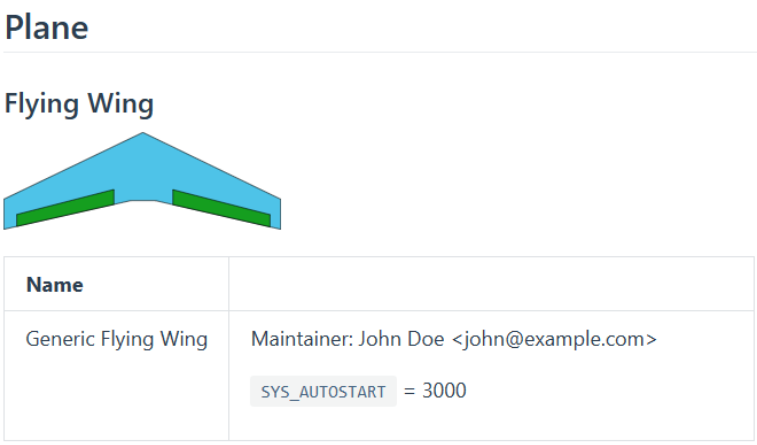
3.4 Introduction of PX4 mixed control rules

Users in the use of RflySim software and hardware in the loop simulation platform for vehicle, need to confirm the rack type, PX4 website frame definition to view [frame reference | PX4 autopilot user guide](#).

The flying wing model, for example, how to confirm the official flyer frame type and mixed control file:

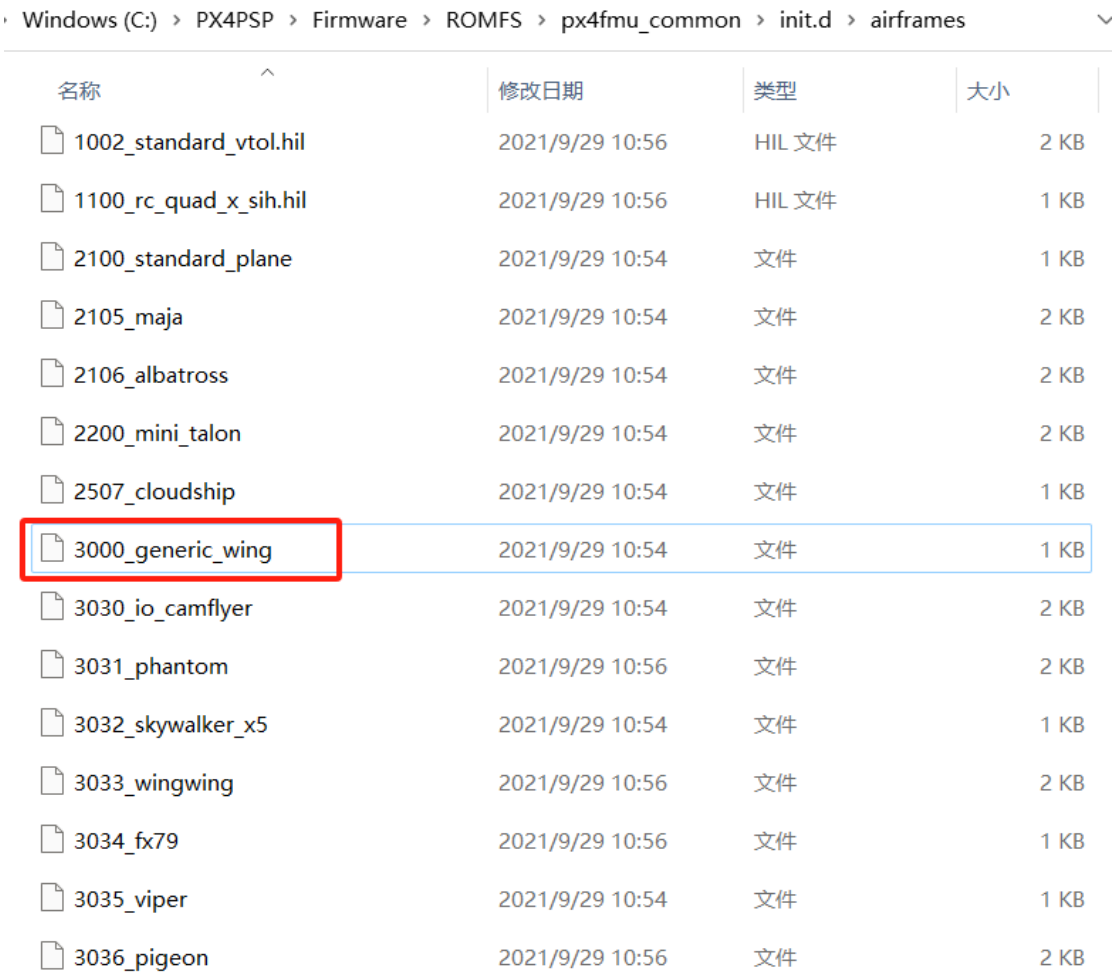
Step 1:

In the [frame of reference | PX4 autopilot find Flying Wing in the user guide], can be seen in the following figure, rack, called Generic Flying Wing, SYS_AUTOSTART = 3000.



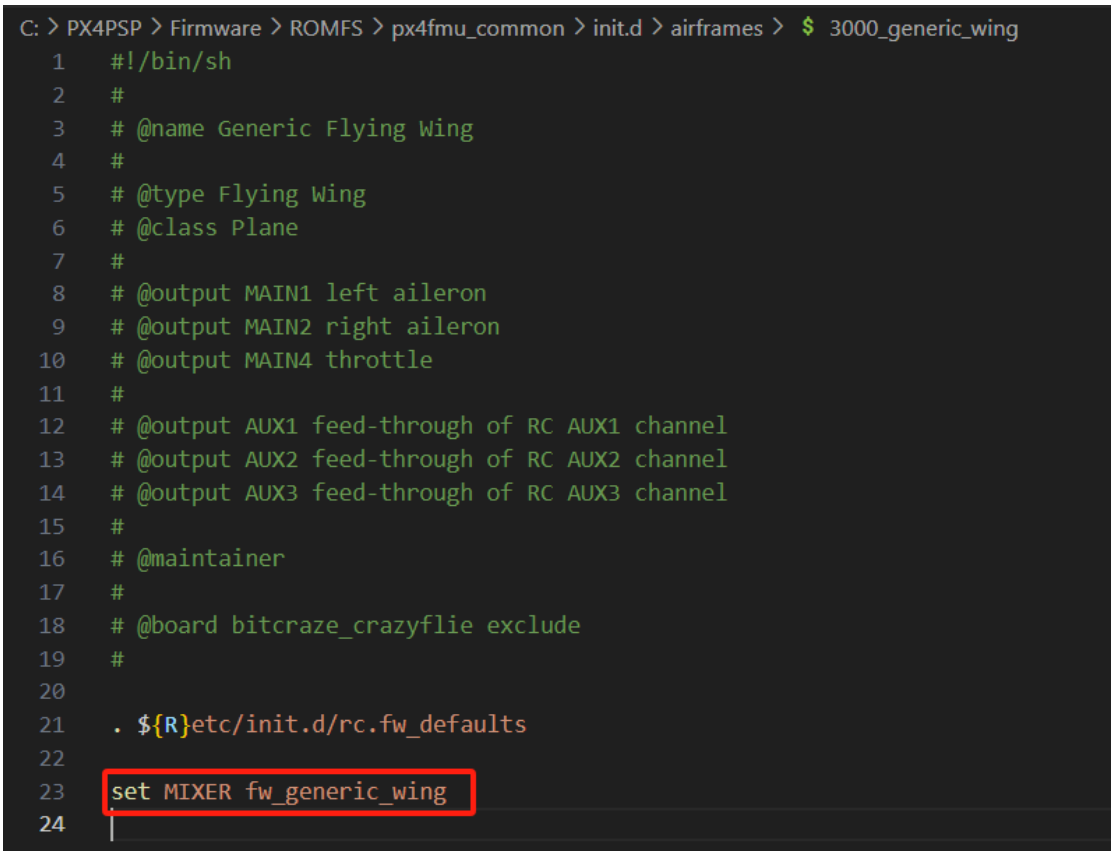
Step 2:

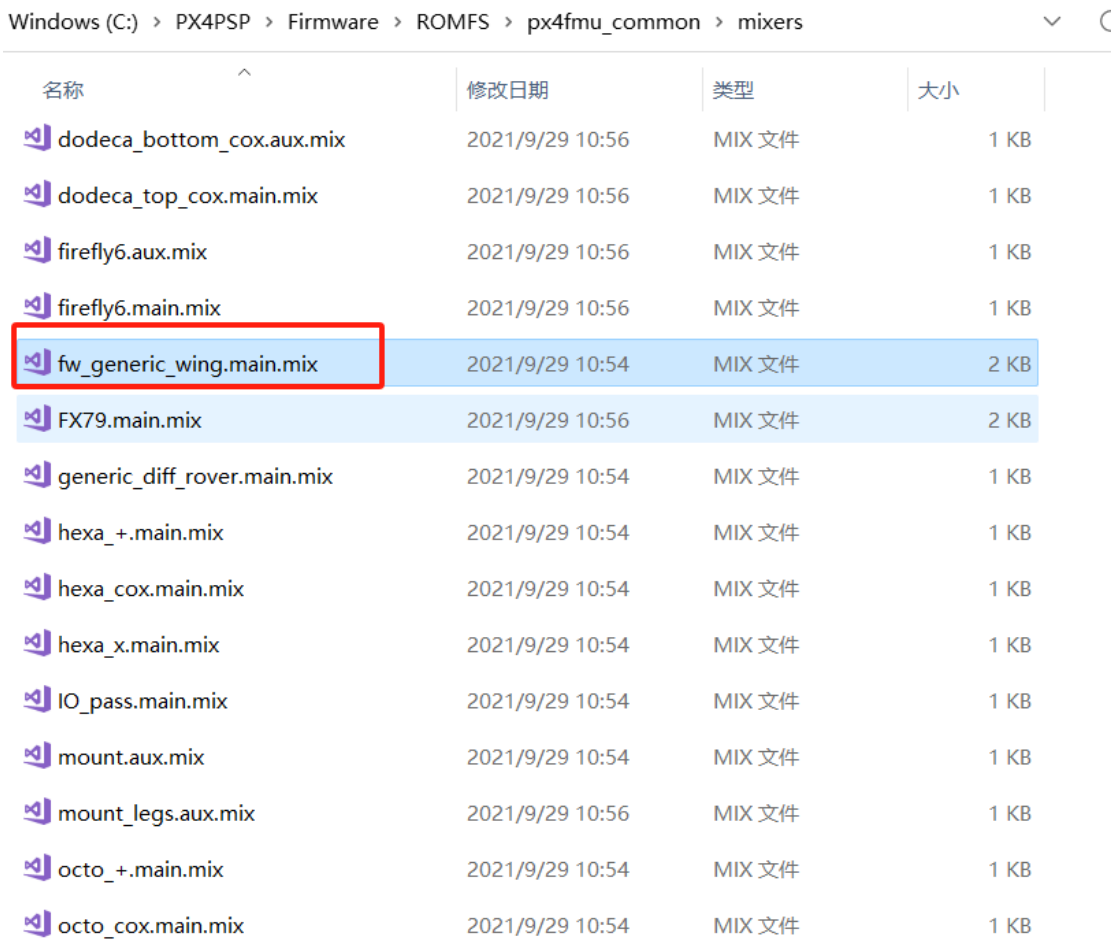
Open the path "PX4PSP\Firmware\ROMFS\px4fmu_common\init.d\airframes" to find the corresponding rack file.



Step 3:

Open the rack file 3000_generic_wing with Vs Code, you can see that the mix control file is set in the file.





Step 4:

Open the mix file fw_generic_wing.main.mix with Vs Code with the following content:

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

```

At this point, we have understood the PX4 wing rack file and mixing control file, the following introduces the PX4 mixing control rules.

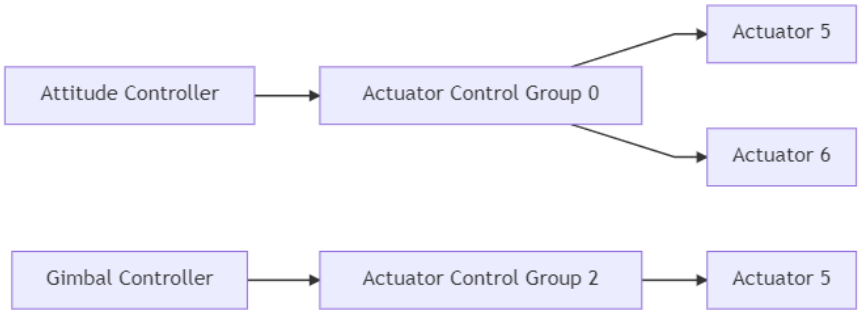
PX4 mixed control rules:

Note: detailed definition see [mixed controller and actuator | PX4 autopilot user guide]

PX4 architecture to ensure the core controller does not need to do special processing for fuselage layout. Hybrid control refers to the distribution of input commands (e.g., remote control to turn right) to the motor and actuator commands (e.g., steering or servo PWM) of the servo. In

the case of fixed wing aileron control, each aileron is controlled by a servo, then the meaning of mixed control is to control one aileron to lift and the other aileron to fall. Similarly, for a multirotor, the pitch operation requires changing the speed of all the motors. Separating the mixing logic from the actual attitude controller can greatly improve reusability.

The specific controller sends a specific normalized force or torque command (scaled to -1.. +1) to the mixer, which sets each individual actuator accordingly. Control output drivers (e.g. UART, UAVCAN, or PWM) put the output of the mixer in the native units when the actuator is actually running, e.g., output a PWM instruction with a value of 1300.



There are four main Control groups at the output of the PX4 control channel, which are:

- actuator_controls_0: The main control channel of flight control, used to output the control quantity of pitch, roll, yaw, throttle and other channels. It is defined as follows:

```
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: Alternate control channel, used in VTOL to output control output of fixed wing mode. It is defined as follows:

```
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: Pan/tilt control channel. It is defined as follows:

```
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: Remote control mapping channel. It is defined as follows:

```
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 mixer file syntax

A Mixer file is a text file that defines one or more Mixer definitions: a mapping between one or more inputs and one or more outputs. There are four main types of definitions: multirotor mixer, helicopter mixer, Large mixer, and null mixer.

- multirotor mixer: Defines a type + or x rotor vehicle with output 4, 6, or 8.
- helicopter mixer: Defines the output of the helicopter swash plate server and the main motor ESCs (the tail rotor is a separate mixer).
- Macromixer: Combines zero or more control inputs into a single actuator output. The inputs are scaled and the mixing function sums the results before applying the output scaler.
- Macromixer: Produces an actuator output with zero output (when not in fail-safe mode).

The amount of output produced by each mixer depends on the type and configuration of the mixer. For example: the multirotor mixer can have 4, 6, or 8 outputs depending on its model, while the Macromixer or Macromixer produces only one output. Multiple mixers can be specified in each file. The output order (which assigns mixers to actuators) is specific to the device that reads the mixer definition, and for PWM, the output order matches the declared order.

The statement defined at the beginning of each mixer file is:

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

Where tag indicates the selected mixer type, as follows:

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

Macromixer - Additive Mixer

The Large Mixer is used to control UAV actuators and servos. It combines zero or more control inputs into a single actuator output. The inputs are scaled and the blending function sums the results before applying the output scaler. The minimum actuator traversal time limit can also be specified in the output scalar (inverse of the slew rate). A simple mixer definition is as follows:

```
M: <control count>
O: <-ve scale> <+ve scale> <offset> <lower limit> <upper limit> <traversal time>
```

If `<control count>` is zero, the sum is effectively zero and the mixer will output a fixed value, which is constrained by `<lower limit>` and `<upper limit>`.

The second line defines the output scalars with the scalar parameters described above. While the computation is performed as a floating-point operation, the values stored in the definition file are scaled by a factor of 10,000; That is, an offset of -0.5 is encoded as -5000. The `<traversal time>`(optional) on the output scale is used for actuators that may damage the aircraft if it is too large - for example, the tilt actuator on a tilt-rotor VTOL aircraft. Can be used to limit the rate of change of the actuator (if not specified, no rate limit is applied). For example: a `<traversal time>` value of 20000 will limit the change rate of the actuator such that it takes at least 2 seconds from `<lower limit> and` <upper limit>, and vice versa.

Note 1: `<traversal time>` should only be used when the hardware requires it!

Note 2: Do not place any restrictions on actuators that control the attitude of the vehicle (such as servos for pneumatic surfaces), as this can easily lead to controller instability.

Go ahead and define the inputs to `<control count>` and their scaling in the form:

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

Note 3: `S:` must be below `O:`.

Note 4: Any mixer output with throttle input (`<group>=0 and <`index>=3 in `S:`) will not work in the unlocked or pre-unlocked state. For example: a server with four inputs (roll, pitch, yaw, and throttle) will not move in the unlocked state even with roll/pitch/yaw signals.

The `<group>` value identifies the control group to be read by the scaler, and the `<index>` value indicates the offset in that group. These values are specific to the device defined by the reading mixer. When used for hybrid vehicle control, mixer group 0 is the vehicle attitude control group, while 0 to 3 are usually roll, pitch, yaw, and thrust, respectively. The remaining fields are configured to control the scaler using the parameters discussed above. When the calculation is performed as a floating-point operation, the values stored in the definition file are scaled by a factor of 10,000; That is, an offset of -0.5 is encoded as -5000. An example of a typical mixer file is explained below. Example analysis in detail, please see: https://docs.px4.io/v1.13/en/dev_airframes/adding_a_new_frame.html#mixer-file.

Null Mixer - Null mixer

This mixer does not consume any control channels and produces a single actuator output whose value is always zero. Typically, Null Mixer is used as a placeholder in a collection of mixers to implement a particular pattern of actuator output. It can also be used to control the value of the output used for fail-safe devices (output is 0 in normal use; During fail-safe, the mixers are ignored and fail-safe values are used instead). The definition is as follows:

```
Z:
```

Multirotor Mixer - Multirotor mixer

The Multirotor Mixer combines four control inputs (roll, pitch, yaw, thrust) into a single set of actuator outputs that are used to drive the motor speed controller. It is defined as follows:

```
R: <geometry> <roll scale> <pitch scale> <yaw scale> <idlespeed>
```

The supported models are:

- 4x - Quadrotor x configuration
- 4+ - Quadrotor + type configuration
- 6x - Six rotor x configuration
- 6+ - Six rotor + type configuration
- 8x - ocrotor Type x configuration
- 8+ - octortor + type configuration

Roll, pitch, and yaw ratio values determine the ratio of roll, pitch, and yaw control relative to thrust control. When the calculation is performed as a floating-point operation, the values stored in the definition file are scaled by a factor of 10,000; For example: 0.5 is encoded as 5000. Roll, pitch, and yaw inputs range from -1.0 to 1.0, while thrust inputs range from 0.0 to 1.0. The output of each actuator ranges from -1.0 to 1.0.

The idling speed ranges from 0.0 to 1.0. Idling speed is relative to the maximum speed of the motor, which is the speed at which the motor is commanded to rotate when all control inputs are zero. In the case of actuator saturation, the values of all actuators are readjusted so that the saturated actuator limit is 1.0.

Helicopter Mixer - A helicopter mixer

The Helicopter Mixer combines three control inputs (roll, pitch, thrust) into four outputs (rotary swash plate and main motor ESC setting). The first output of the helicopter mixer is the throttle setting of the main motor. The subsequent output is the servo that rotates the swash plate. The tail rotor can be controlled by adding a simple mixer. Thrust control inputs are used for the main motor Settings as well as the collective pitch of the swash plate. It uses a throttle curve and a pitch curve, both made up of five points.

Note: The throttle and pitch curves map the "thrust" rod input position to the throttle value and the pitch value (respectively). This allows flight characteristics to be adjusted for different types of flight.

The Helicopter Mixer is defined as follows:

```
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:` The point that defines the throttle curve. `P:` The point that defines the pitch curve. Both curves contain five points between 0 and 10,000. For a simple linear change, the five values of the curve would be 0, 2500, 5000, 7500, 10,000.

Each swash server (3 or 4) is defined as follows:

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

<angle> is in degrees, 0 degrees is the direction of the nose. Positive angles are clockwise. <arm length> is the normalized length, that is, 10,000 equals 1. If all the servo arms are the same length, the value should all be 10,000. A larger arm length will reduce the amount of servo deflection, while a shorter arm length will increase it. The servo output is scaled by <scale> / 10000. After scaling, <offset> is applied and its value should be between -10000 and +10000. In the full servo range, <lower limit> and <upper limit> should be -10000 and +10000.

The tail rotor can be controlled by adding a large Mixer:

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

By mapping the tail rotor directly to the yaw command. This applies to the two servo-controlled tail rotor, as well as the tail rotor with dedicated motors.

The 130-blade helicopter mixer file looks like this:

```
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
```

- At 50% thrust, the slope of the throttle curve is slightly steeper, reaching 6000(0.6).
- At 100% thrust, reach 10,000 (1.0) with a smaller slope.
- The pitch curve is linear, but its entire range will not be used.
- At 0% throttle, the total distance to the joystick setting is already at 500(0.05).
- At maximum throttle, the total distance to the joystick is only 4500(0.45).
- Using higher values for this type of helicopter will stall the blades.
- The rotary swash plate system for this helicopter is located at 0, 140, and 220 degrees angles.
- The servo arms are not equal in length.
- Compared to the first servo, the second and third servos have an arm length of 1.3054.
- The servos are limited to -8000 and 8000 because they are mechanical constraints.

VTOL Mixer - VTOL Drone Mixer

The VTOL system uses a multi-rotor mixer as the output in multi-rotor mode and a sum mixer as the output in fixed-wing mode. Mixer systems for VTOL Uavs can either be combined into a single mixer, where all actuators are connected to IO or FMU ports, or split into separate mixer files for IO and AUX.

3.5 Modification of bat script related parameters

3.5.1 Hardware-in-the-loop simulation script

```
    The conventional hardware-in-the-loop simulation script supports the input of serial port sequence (separated by comma ",") to start the hardware-in-the-loop simulation of multiple computers
    Note: The line at the beginning of REM is a comment statement and will not be executed. Other bat script syntax rules can be searched and learned by yourself.
    Note: The position of the aircraft in this script is automatically generated by the script according to the rectangular queue. The control variables include:
    SET /a START_INDEX=1 (initial aircraft serial number, CopterID of the aircraft generated by this script, START_INDEX as the initial value, in turn incremented by 1)
    SET /a TOTOAL_COPTER=8 (Total number of aircraft, amplitude is only needed in the multi-aircraft online simulation, tell this script the actual total number of aircraft, to determine the side length of the rectangular queue)
    SET UE4_MAP=Grasslands (set map name)
    SET /a ORIGIN_POS_X=0 (origin X position of rectangular formation in meters, integer input only)
    SET /a ORIGIN_POS_Y=0 (origin Y position of rectangular formation, in meters, integer input only)
    SET /a ORIGIN_YAW=0 (yaw Angle of origin of rectangular formation, unit degree, integer input only)
    SET /a VEHICLE_INTERVAL=2 (Aircraft interval in rectangular formation, in meters, integer input only)
    SET /a UDP_START_PORT=20100 (UDP communication interface for receiving external control data, 2 is automatically added to CopterID, this usually does not need to be modified, only if the computer port is occupied)
    set DLLModel=0 (Used to set the name of the DLL model to be imported into CopterSim for hardware-in-loop simulation. It is determined by the filename of the DLL model generated by the Simulink model compiled through GenerateModelDLLFile.p. When set to 0, Quadrotor DLL model is used by default)
    set SimMode=0 (Simulation mode, here set to 0 or PX4_HITL for hardware-in-the-loop simulation)
    SET IS_BROADCAST=0 (online emulation or not, the destination IP address sequence can be entered here)
    SET UDPSIMMODE=0 (UDP_START_PORT port received data protocol, UDP mode transmission is the platform private structure, support Simulink control; MAVLink mode transmits MAVLink protocol and supports Python and mavros control modes)
```

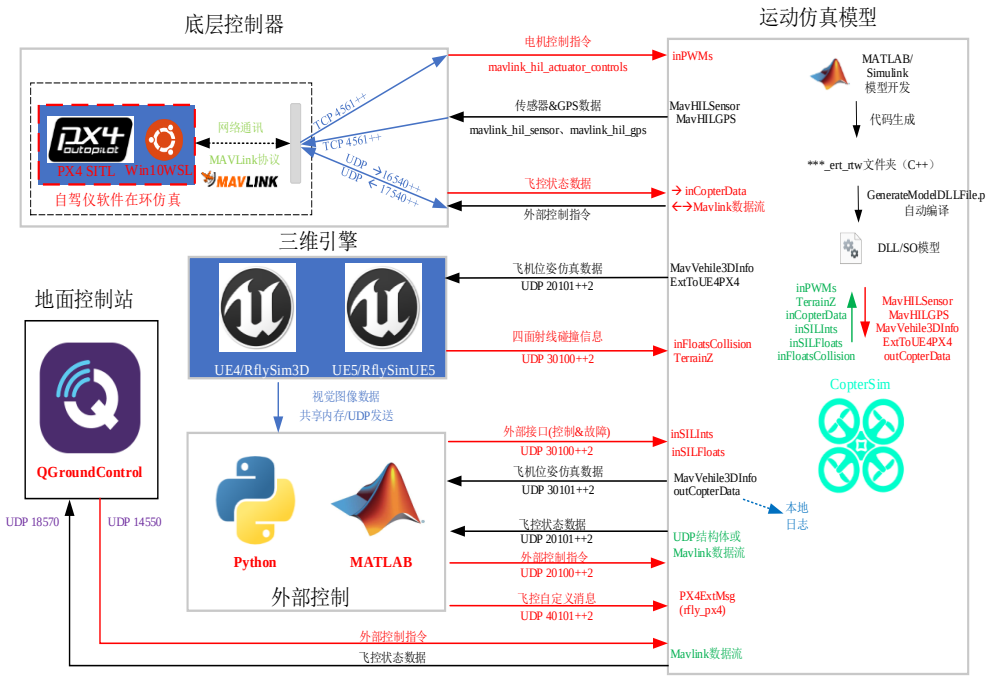
3.5.2 Software-in-the-loop simulation script

```
    Conventional software-in-the-loop script supports input of the number of aircraft and automatically starts multi-aircraft software-in-the-loop simulation
    Compared with HITLRun.bat, the key code is as follows
    set SimMode=2 (here set to software in loop mode, corresponding to the value of CopterSimUI)
    set PX4SitlFrame=iris (This is used to set the PX4 simulation rack type, set to the non-digital part of the rack file, for example, iris for quadrotor, generic_wing for flying wing, hexa_x for hexrotor, standard_plane for fixed wing, See 3.4 Introduction of PX4 mixed control rules for confirmation method of frame type)3.4 Introduction of PX4 mixed control rules
```

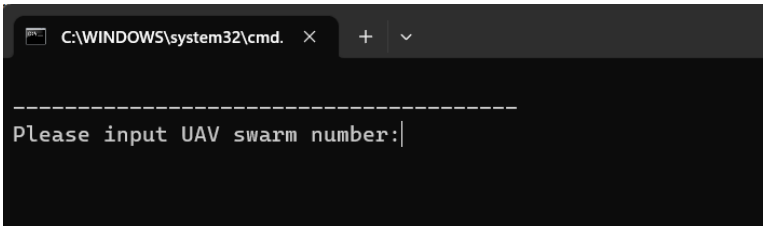
3.6 SITL simulation

The RflySim software-in-the-loop simulation system is available through the software-in-the-loop simulation script (e.g. After the script is run, the PX4 simulation environment, CopterSim,

QGroundControl and 3D engine (RflySim3D/RflySimUE5) will be started on the computer, and the communication port will be automatically configured. The RflySim platform (including CopterSim, QGroundControl and 3D engine) communicates with PX4 via MAVLink messages. During SITL simulation, these messages are processed by PX4PSP\Firmware\src\modules\simulator\ Simulator_mavlink-cpp.



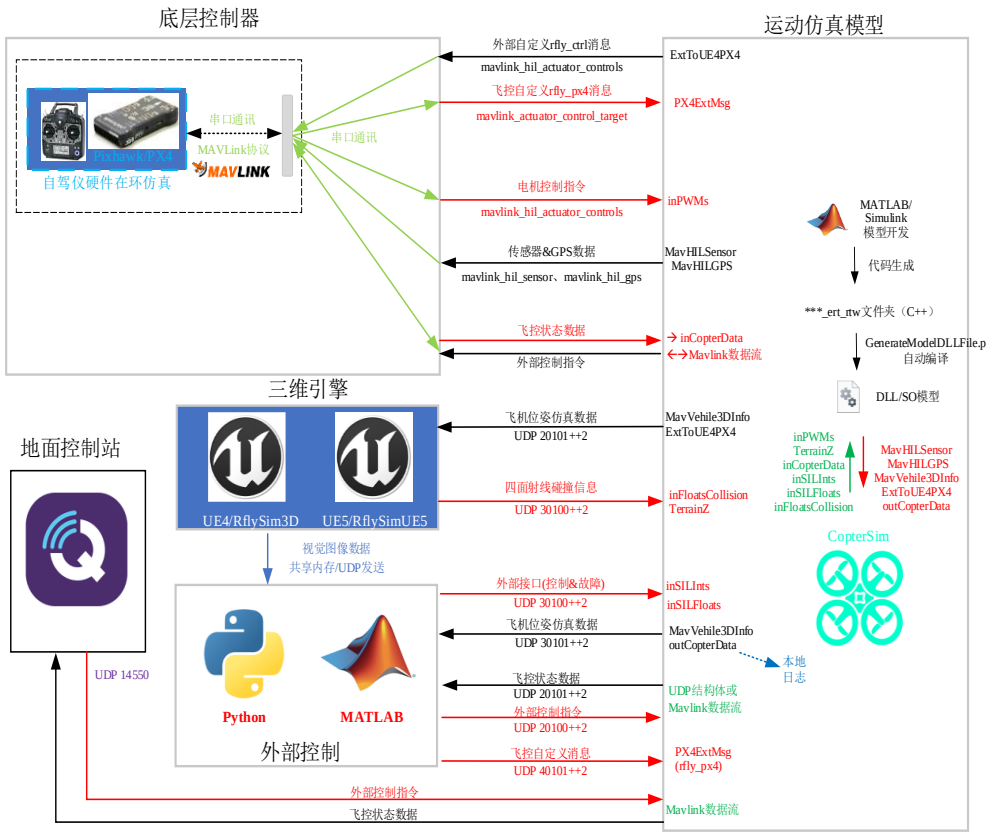
After running the specified SITLRun.bat script, a prompt box will pop up. You can create the corresponding number of unmanned vehicles in the 3D scene and complete the initialization by inputting the number at the cursor. A single computer supports multi-computer simulation, and the number of unmanned vehicles that can be simulated at the same time is determined by computer performance and communication load.



In the following, the four-rotor software in the ring startup script is explained. Enter 1 at the cursor of the prompt box and then enter. The system will launch QGroundControl, RflySim3D and CopterSim three software. When "GPS 3D fixed&EKF initialization finished" is indicated in the message prompt bar at the bottom left of CopterSim, it indicates that the software is changing the simulation initialization is completed and the simulation can be started.

3.7 HITL simulation

Hardware-in-the-loop simulation (HITL or HIL) is a simulation mode that runs PX4 firmware on real flight controller (i.e., flight control) hardware. During hardware-in-the-loop simulation, the flight controller is connected to the host through USB, and then the communication between the RflySim platform and the flight controller can be realized by means of serial port.



Taking the quadrotor model as an example, the general configuration steps of the flight control before the start of the loop simulation are introduced:

Note: The routine path is *:\PX4PSP\RflySimAPIs\4.RflySimModel\2.AdvExps\e2_MultiMo

delCtrl\2.MultiModelCtrl

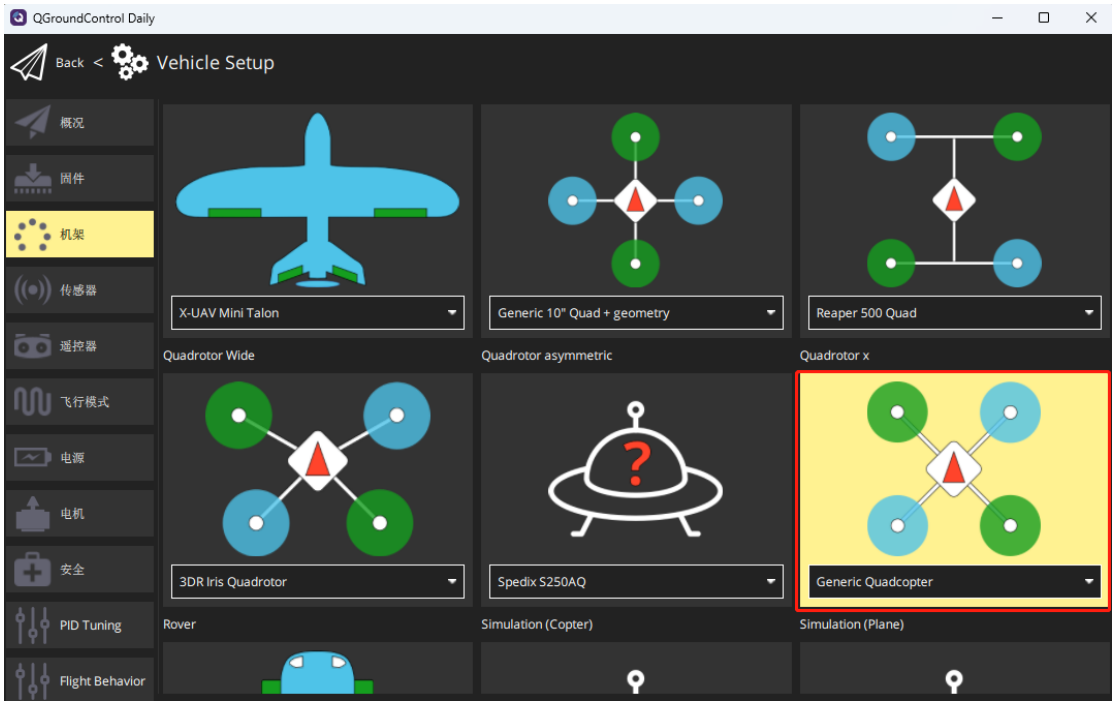
Step 1: Determine the type of flight control and firmware version used for simulation. The platform recommends the use of Pixhawk 6C flight control with firmware version 1.13.3.

Step 2: Connect the flight control to the USB port of the computer through the USB-TYPEC cable.

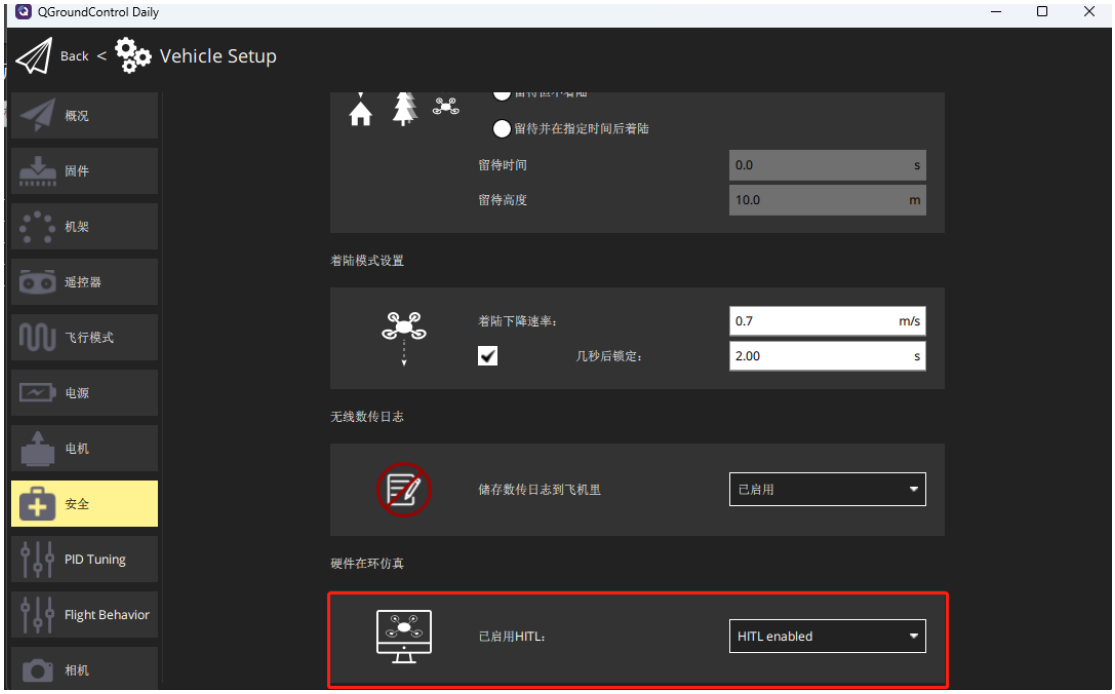
Step 3: Open the QGC ground station in the Rflytools folder.

Set the rack model as "Generic Quadcopter" in the rack interface (the rack is determined by the simulation model and the corresponding official rack file), and click "Apply and Restart" on

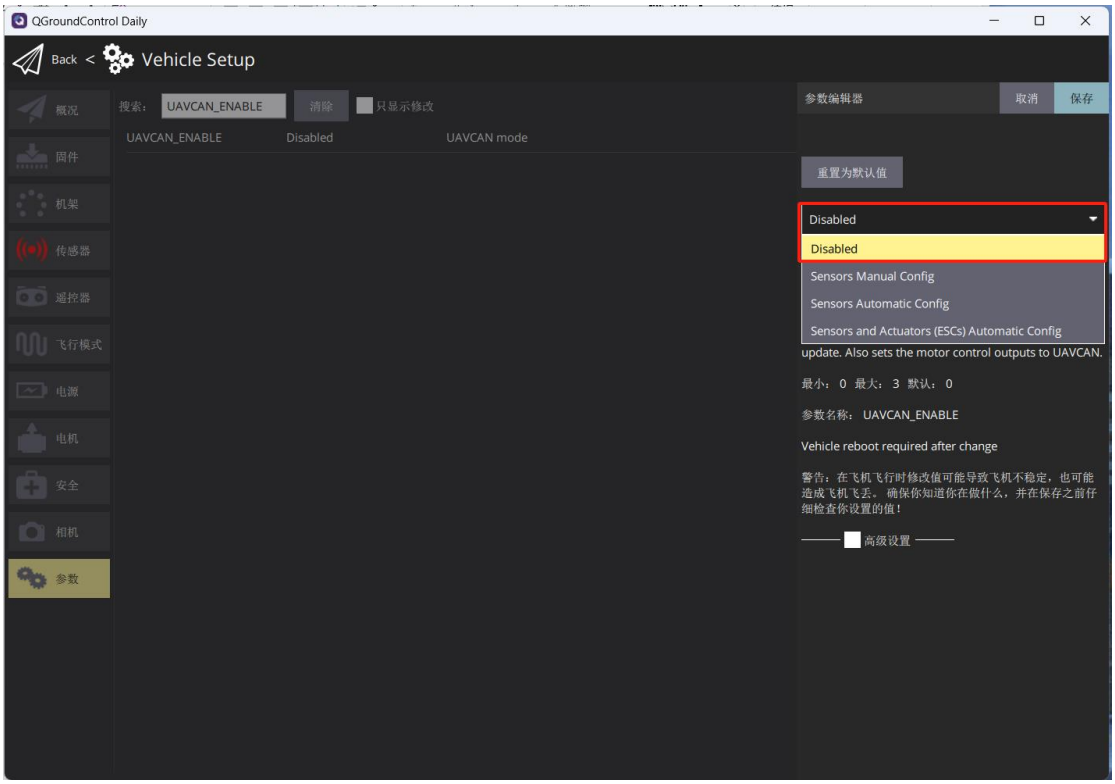
the right after setting.



Step 4: In the "Safety" interface, select "HITL enabled" to start the hardware-in-the-loop simulation and replug and unplug the flight control.



Step 5: Click "Parameters", enter "UAVCAN_ENABLE" in the search bar, and set it to "Disabled" in the pop-up box. After saving, replug and unplug the flight control to complete the configuration before the hardwire-in-the-loop simulation.



3.8 Perform simulation test through QGC or remote control

3.9 External interface communication debugging

4, DLL file generation script -GenerateModelDLLFile.p

5, DLL/SO model and communication interface

5.1 General Introduction

From the perspective of implementation mechanism, RflySim platform can be divided into five parts: motion simulation model, underlying controller, 3D engine, external control and ground control station.

After model development is completed based on MATLAB/Simulink, C++ files are automatically generated by code and DLL models are generated through the platform GenerateModelDLLFile.p interface. DLL models are imported into CopterSim when RflySim platform is used for software/hardware in-loop simulation. The motion simulation model is formed. The motion simulation model has multiple input and output interfaces to exchange data with the underlying controller, 3D engine, ground control station and external control. See **错误！未找到引用源。**The **reference source was not found.**

Among them, when the software is in the loop simulation, the data exchange between the bottom controller and the motion simulation model is in the way of network communication, and when the hardware is in the loop simulation, the data exchange between the bottom controller and the motion simulation model is in the way of serial communication.

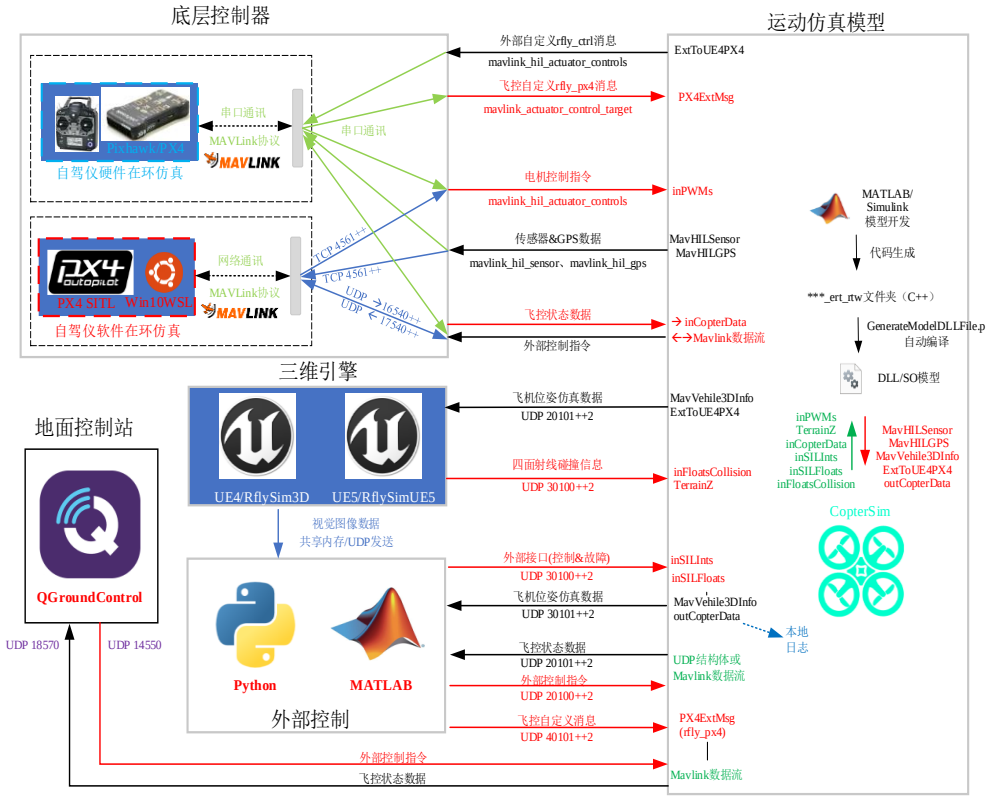


FIG. 0.1 Illustration of data interaction between motion simulation model and other modules01

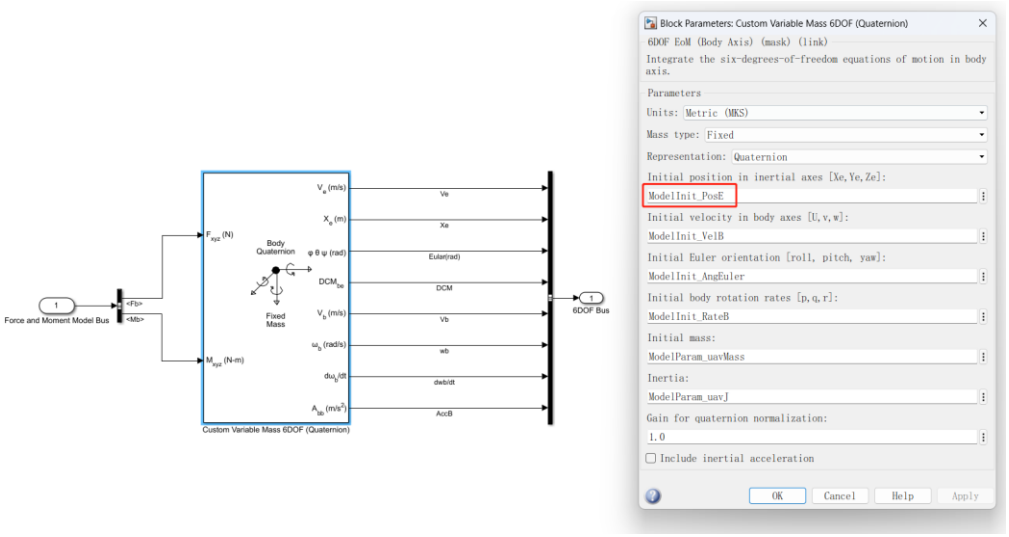
5.2 Important Parameters

The RflySim motion simulation model has the corresponding Init.m file, which defines the parameters needed for the motion simulation model, including the model formula coefficients (such as the propeller tension coefficient and torque coefficient in the multi-rotor), the noise coefficient, the sensor coefficient and the related variables that will affect the RflySim3D display, etc. The important parameters in the model are introduced in the following.

5.2.1 ModelInit_PosE

This parameter is the position initialization parameter in the world coordinate system of the motion simulation model, and the 3D data are [x,y,z], respectively. Through this parameter, RflySim can initialize the specified X and Y positions of the UAV displayed in the RflySim3D map before the simulation begins.

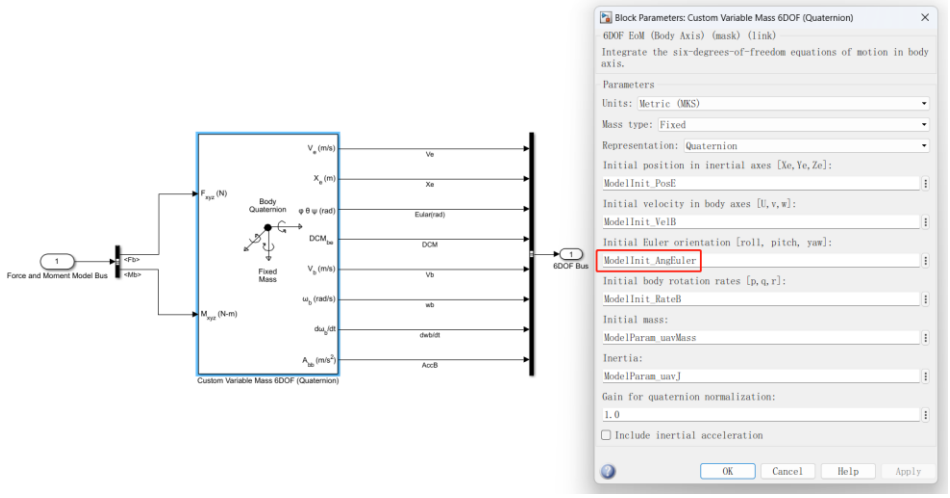
```
ModelInit_PosE = 0, 0;
```



5.2.2 ModelInit_AngEuler

This parameter is the attitude initialization parameter of the motion simulation model, 3D data, respectively, through this parameter, RflySim can initialize the display of the UAV in RflySim3D with a specified yaw Angle before the simulation begins.[φ, θ, ψ]

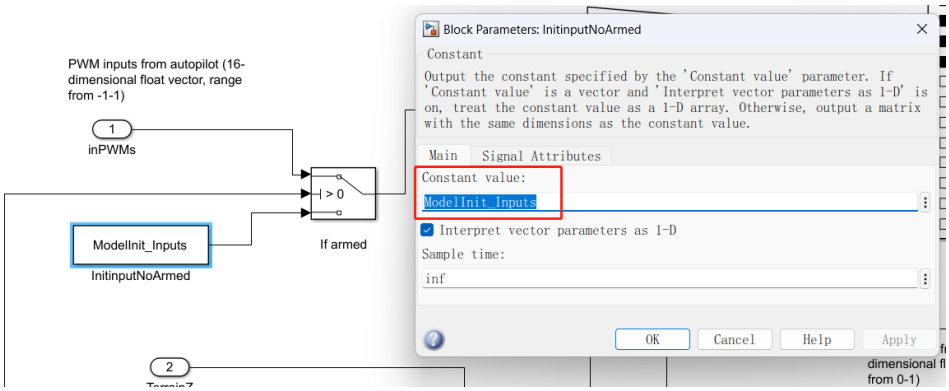
```
ModelInit_AngEuler = 0, 0;
```



5.2.3 ModelInit_Inputs

This parameter is the input initialization parameter of the motion simulation model actuator, which is a 16-dimensional data. For the aircraft with specific requirements, for example, the initial state of the throttle needs to be at the minimum value (-1), that is, this parameter is needed to modify the input initialization value of the actuator.

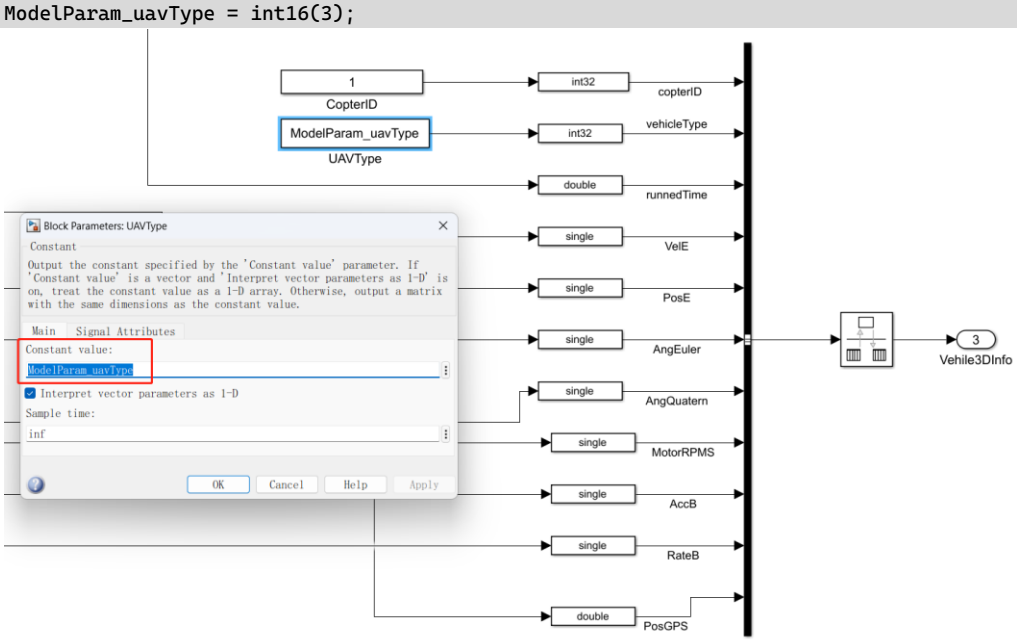
```
ModelInit_Inputs = [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0];
```



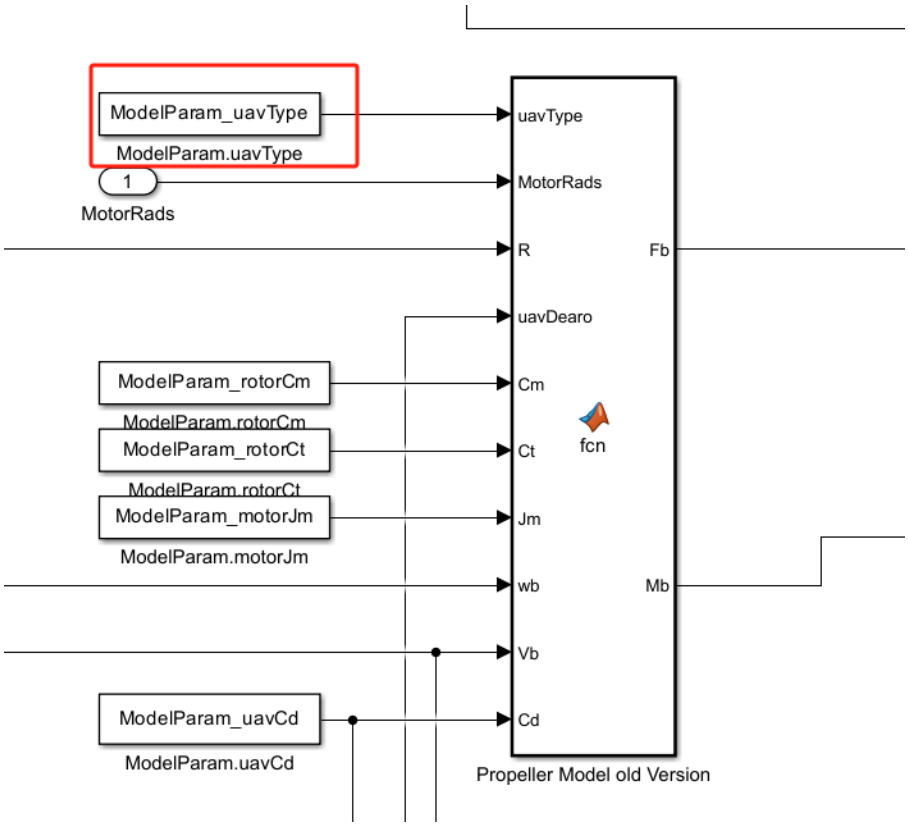
5.2.4 ModelParam_uavtype

This parameter determines the display model in RflySim3D to be called during simulation. For example, when ModelParam_uavType is 3, the display model in RflySim3D is quadrotor, and ModelParam_uavType is 100, The display model in RflySim3D is a conventional small fixed wing.

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



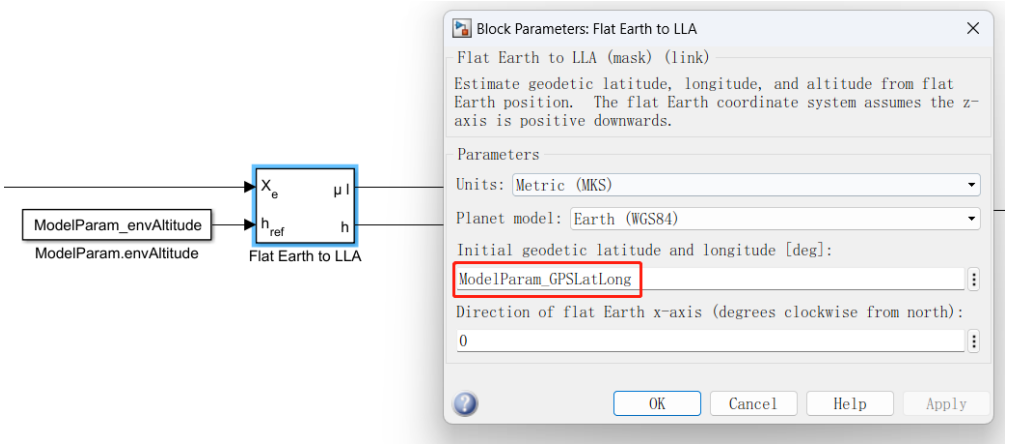
At the same time, in view of the propeller Model of the rotor, ModelParam_uavType can also used for computer rack and Moment distribution, specific see 6.2 Force and Moment Model Force and Moment module, ***S*** - FM.



5.2.5 ModelParam_GPSLatLong

This parameter is used to configure the longitude and latitude information of the aircraft, two-dimensional data, respectively [ModelParam_envLatitude, ModelParam_envLongitude], through which the aircraft display coordinates in QGC can be adjusted.

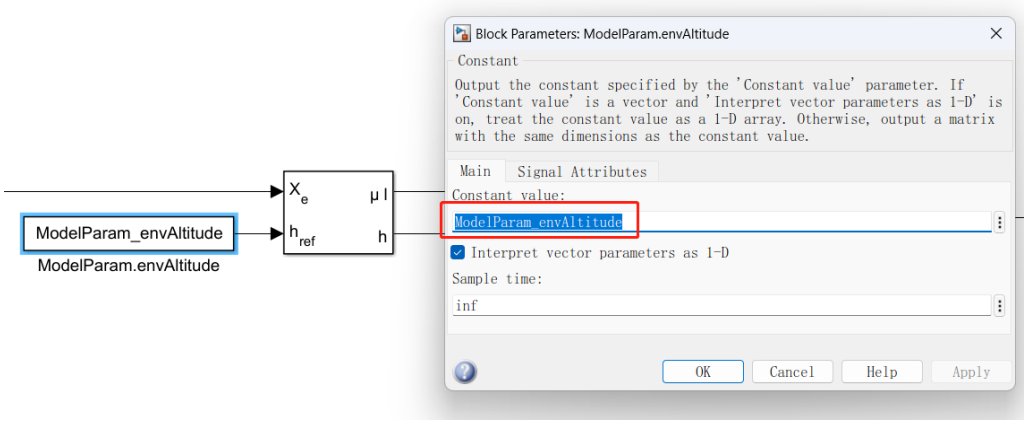
```
ModelParam_envLongitude = 116.259368300000;
ModelParam_envLatitude = 40.1540302;
ModelParam_GPSLatLong = [ModelParam_envLatitude ModelParam_envLongitude];
```



5.2.6 ModelParam_envAlitude

This parameter is used to configure the altitude information of the aircraft. The z-axis is positive downward, and the altitude display of the aircraft in QGC can be adjusted by this parameter.

```
ModelParam_envAltitude = -50
```



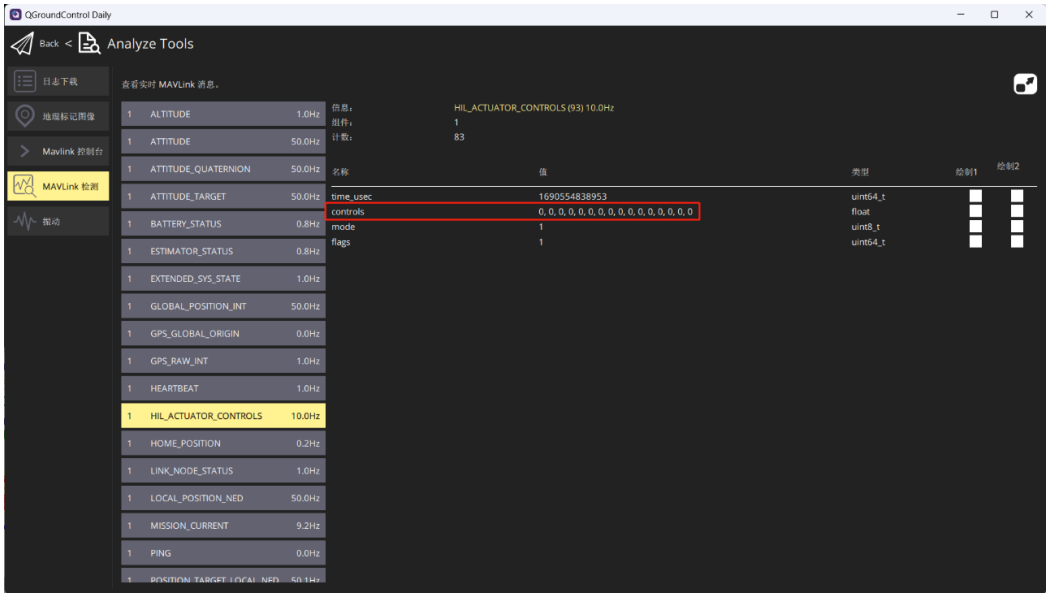
5.3 Data Protocol

5.3.1 Flight control simulation input interface

5.3.1.1 inPWMs (Motor Control Input)

The 16-dimensional actuator control input, which has been normalized to -1 to 1 scale, comes from controls on the motor control MAVLink message sent back by the flight control. During the loop simulation, the software and hardware can view the controls changes in real time through the MAVLink detection function in the Analyze Tools of QGroundControl.

Figure 0.1 shows that when the software is in the loop simulation, the motor control command is sent from the PX4 SITL controller through the TCP 4561 series port to the inPWMs interface of the motion simulation model by the MAVLink protocol, while the hardware is in the loop simulation, FIG. 0.1 Illustration of data interaction between motion simulation model and other modules01The command is sent from the flight controller to the inPWMs interface of the motion simulation model through the serial port with the MAVLink protocol.



5.3.1.2 inCopterData (Flight Control State Input)

inCopterData is a 32-dimensional double data. The first eight dimensions store the PX4 state, and the current 1-6 dimensions are as follows:

- inCopterData(1) : The unlock bit of PX4
- inCopterData(2) : The total number of RC channels received. This value should be 0 when there are no RC channels available.
- inCopterData(3) : Simulation mode flag bit, 0: HITL, 1: SITL, 2: SimNoPX4.
- inCopterData(4) : 3D fixed flag in CoperSim.
- inCopterData(5) : VTOL_STATE flag bit from PX4.
- inCopterData(6) : LANDED_STATE bit from PX4.

Dimensions 9-24 receive ch1-ch16 RC channel signals (remote control input), and dimensions 25-32 listen for rfly_px4 uORB messages.

5.3.2 Flight control simulation output interface

5.3.2.1 MavHILSensor

This output signal is a collection of various sensor data sent by the model to the flight control, which corresponds to the mavlink_hil_sensor_t message of MAVLink. The output signal includes the acceleration value of the acceleration sensor, the angular velocity value of the gyroscope sensor, the magnetic field value of the magnetic compass sensor, and the pressure value of the air pressure and airspeed sensors.

5.3.2.2 MavHILGPS (GPS interface)

This output signal is the GPS data value sent by the model to the flight control, which corresponds to the mavlink_hil_gps_t structure of the MAVLink message. The output signal contains the data of longitude and latitude height, horizontal and vertical accuracy, ground speed, northeast ground speed, yaw Angle, positioning status, and the number of satellites.

It should be noted that the values of these sensors are provided by the platform model in the simulation, and by the real sensor chip when the real aircraft is flying. Figure 1 shows that when the software is in the loop simulation, the sensor and GPS data are sent from the MavHILSensor and MavHILGPS interfaces of the motion simulation model and sent to the PX4 SITL controller in the form of MAVLink protocol through TCP 4561 series ports, while the hardware is in the loop simulation, These data are sent to the flight control through the serial port.

5.3.3 Simulation data output interface

5.3.3.1 MavVehile3Dinfo (Real Simulation Data Output)

The output signal is the real simulation data sent by the model to RflySim3D, which is the ideal value of smoothness. These data can be used for software simulation test of flight control

and model under Simulink. Because the true value of the model is not available in the real test, only the state estimation of the PX4 autopilot (with delay, noise and interference) can be used, which leads to the poor effect of the Simulink controller to the PX4 in the loop simulation and the real test, so it needs to be adjusted.

5.3.3.2 outCopterData (Custom log output)

32-dimensional double, the contents of which can be customized to send data. The data sent to this interface, on the one hand, will be written to the local log log (in C:\PX4PSP\CopterSim new CopterSim*.csv, will start to record the data of the * aircraft, note that the * should be replaced with the aircraft ID here). On the other hand, this data will be transmitted to the 30101 series port via UDP (supplement readme).

5.3.3.4 ExtToUE4 (Custom display data output)

The 16-dimensional double data is sent to RflySim3D via port 20100 series to be displayed as actuator control messages for dimensions 9-24 (supplementary readme).

5.3.4 Automatic code generation of controller communication interfaces

5.3.4.1 ExtToPX4 (Custom uORB Data Output)

16-dimensional float data, uORB message rfly_ext sent to PX4 via serial port, used to transfer other sensor or necessary data to flight control (supplement readme).

5.3.4.2 inCopterData (uORB Data input)

32 dimensions, of which the last 8 receive PX4 messages, data from uORB msg rfly_px4.control[0:7].

5.3.5 Collision Data Receiving Interface - inFloatsCollision

inFloatsCollision is used to implement a simple physics engine, which can realize the functions of bouncing back when hitting obstacles and crashing when hitting other planes according to the four distance data returned by RflySim3D (supplementary readme).

5.3.6 External data incoming interface

5.3.6.1 inSILInts (Integer Data Input)

8 dimensional Int32 type input, obtained through UDP protocol, from 30100++2 series port number, software and hardware in the ring simulation, can input some quantities to the model through this port; At the same time, this interface is the key interface to realize the synthesis model.

5.3.6.2 inSILFloats

20 dimensional float input, obtained through UDP protocol, from 30100++2 series port number, software and hardware in the ring simulation, can input some quantities to the model through this port; At the same time, this interface is the key interface to realize the synthesis model.

5.3.6.3 inFromUE (RflySim3D Data Input)

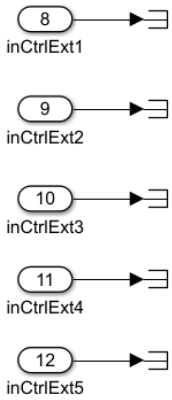
16-dimensional double data from 3D engine (Rflysim3D/RflySimUE5), which can be used to implement ground interaction, collision engine and other related functions that need to interact with 3D engine data.

5.3.6.4 TerrainZ (Terrain height input)

1D terrain height signal, since the earth fixed coordinate system (ground coordinate system) here is NED, the vertical ground downward is positive. The initial position height of the model can be determined.

5.3.6.5 inCtrlExt (Floating Point Data Input)

Includes inCtrlExt1-inCtrlExt5 series interface, requiring 28-dimensional data, data type is Si ngle. It is obtained through UDP protocol and comes from 30100++2 series port number. When th e software and hardware are in the ring simulation, they can input some quantities to the model thr ough this port.



5.3.7 Real-time parameter modification interface - FaultParamsAPI

Real-time parameter modification by means of Matlab: 0.ApiExps\10.FaultParamsDynMod\Readme.pdf

Live parameter modification via Pythond: 0.ApiExps\19.initParamModAPI_py\readme.pdf

Live parameter changes via csv: 0.ApiExps\18.initParamModAPI_csv\Readme.pdf

5.4 Communication interfaces

5.4.1 20100++2 series port

The 20100++2 system port is the UDP receiving port of CopterSim, which mainly receives external control instructions.

5.4.2 20101++2 series port

Port 20101++2 is the UDP initiator of CopterSim. The data sent from port 20101++2 mainly include:

1） During simulation, the simulation data of the aircraft position and attitude sent to the 3D engine.

2） For external control, the flight control status data is sent.

5.4.3 30100++2 Series port

The 30100++2 series port is the UDP receiving port of CopterSim, and the data it receives mainly includes:

1) The four-sided ray collision information from the 3D engine is used to implement the collision function.

2) Data from external control (Python/MATLab), including control or fault injection, etc.

5.4.4 30101++2 series port

Port 30101++2 series is the UDP receiving port of CopterSim, and the data sent by it mainly includes:

1) Simulation data of aircraft position and attitude.

2) Custom log data from the outCopterData interface of the motion simulation model.

5.4.5 TCP port

TCP port is the communication port between the PX4 controller and the motion simulation model when the software is in the loop simulation. During the simulation, the PX4 controller sends the motor control commands to the inPWMs interface of the motion simulation model through the TCP 4561 series port. And the model fed back the sensor and GPS data to the PX4 controller through TCP 4561 series port to form a simulation closed loop.

5.4.6 Flight control USB serial port

When the hardware is in the loop simulation, the PX4 flight control and the motion simulation model communicate through the serial port. The data sent by the PX4 flight control mainly include:

1) Motor control commands sent by MavLink message mavlink_hil_actuator_control.

2) the flight control custom message rfly_px4 message issued through the MavLink message mavlink_actuator_control_target.

3) Flight control status data.

The data sent by the model mainly include:

1） Sensor and GPS data sent through MavLink messages mavlink_hil_sensor and mavlink_hil_gps.

2） Externally customize the rfly_ctrl message.

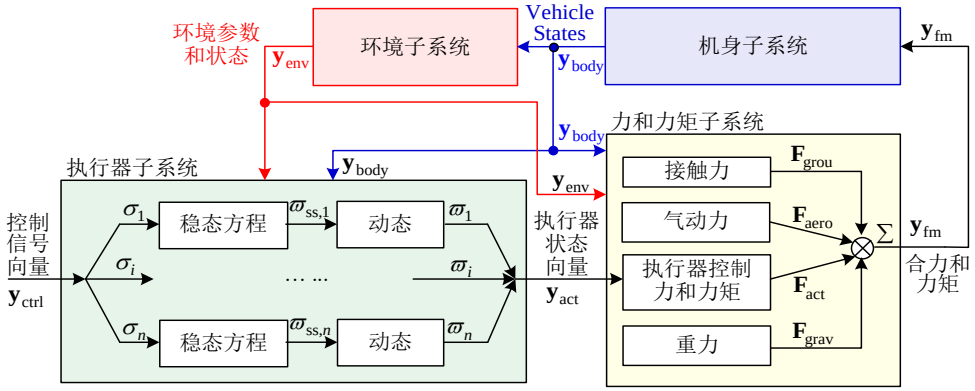
3） External control instructions.

6. Introduction of Simulink modeling template

The vehicle motion model of the unmanned system can be decomposed into the unified mode ling framework shown in Figure 1.2 through physical and virtual components. FIG. 1.2 Unified fra mework of motion model12In this framework, the motion model system can be divided into three subsystems: the body subsystem$\boldsymbol{S}_{\text{vehi}}$, $\boldsymbol{S}_{\text{3d}}$the 3D environment model subsystem$\boldsymbol{S}_{\text{sens}}$ and the senso r model subsystem. These three subsystems need to be connected with the control system to form a complete closed loop, which can realize the simulation of all scenes in the real outdoor environm ent. $\boldsymbol{S}_{\text{ctrl}}$The input, output and signal connection relationship between them can be expressed in th e simplified form of the subsystem as follows

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

- The body subsystem includes actuator, body, operating environment, force and torque, which is the overall description of the body's motion, energy consumption and fault characteristics in the environment.$\boldsymbol{S}_{\text{vehi}}\boldsymbol{S}_{\text{act}}\boldsymbol{S}_{\text{body}}\boldsymbol{S}_{\text{env}}\boldsymbol{S}_{\text{fm}}$



The input and output can be described as follows

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

- Sensor model is mainly used to describe all electronic hardware models except control software, mainly including sensor data, communication protocol, connection interface and other features;
- The 3D environment model is mainly used to describe the 3D visual environment of UAV flight (including trees, obstacles, roads, etc.), which is used to provide visual data simulation for the autonomous control system.

In the following, taking the multi-rotor model as an example, several functional modules of Simulink modeling template are introduced

6.1 Motor ModelS_{act}

The actuator subsystem is mainly used to generate the output state of the actuator according to the control instructions sent by the control system. $\boldsymbol{S}_{\text{act}}\boldsymbol{y}_{\text{ctrl}}\boldsymbol{y}_{\text{act}}$In real systems, most actuators usually have their own control unit for feedback control to ensure that the motion law of the actuator follows the pre-programmed characteristics. For example, the commonly used electric control products of UAV usually have a certain speed feedback function, which can ensure that the output speed and the output throttle meet the linear relationship; The engine system and steering system commonly used in unmanned vehicles are usually controlled by Electronic Control Unit (ECU), so that the output of the engine and the input of the throttle meet the specific programming relationship. Considering that the input-output relationship of the actuator follows the artificial preset programming law rather than the natural model law, it is very difficult to use the mathematical method to deduce its model. It usually needs to use the method of system identification.

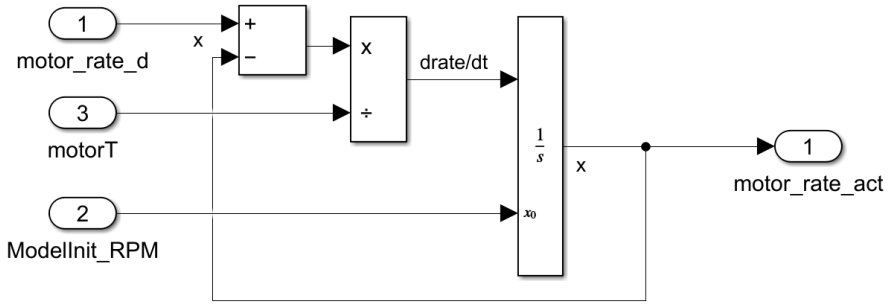
A complex actuator model can be decomposed into a steady-state process and a dynamic process under the rated operation state. $f_{\text{ss},i}(\cdot)G_{\text{ss},i}(s)$The rated operation state here, for the multi-rotor is the hovering flight state, for the vehicle, it is the state of driving forward at the rated speed, and for the fixed-wing vehicle, it is the state of cruising at the rated speed.

1) Model of the multirotor power unit

For a certain power unit (electric regulation + motor + propeller) in the multi-rotor power system, it can be simplified as a first-order (or second-order) inertial link superimposed on a linear steady state function, and the corresponding transfer function can be expressed as

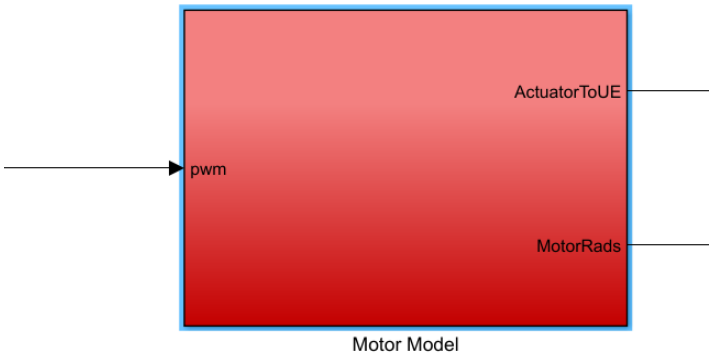
$$\delta_i = G_{\text{ss},i}(s) \cdot f_{\text{ss},i}(\sigma_i) = \frac{1}{k_1 s + 1}(k_2\sigma_i + k_3)$$

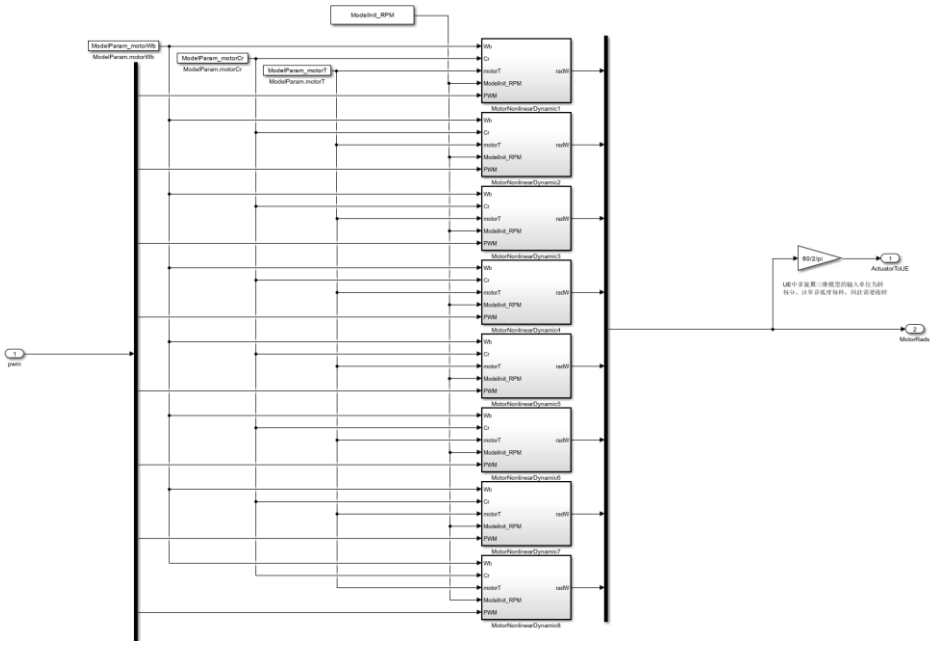
Where, is the Laplacian operator, represents the input control signal of the actuator, represents the output state value of the actuator, is the constant parameter to be identified, and the corresponding simulink module is as follows:$s\sigma_i\delta_i k_1, k_2, k_3$



2) Overall inputoutput

A quadrotor motor usually consists of the motor body, rotor, stator and controller. The propeller on the rotor generates lift and thrust by turning the motor, while the controller is responsible for adjusting the speed and direction of the motor. So as to realize the stable hovering, forward, backward, and steering actions of the UAV. In this module, the input is the PWM value, and the motor speed is obtained after the nonlinear dynamic model of each motor. The output of the module is the motor speed (radian per second) of the input force and torque model respectively. For the motor speed (RPM) input to UE, since the input unit of the 3D model of the multi-rotor in UE is RPM, the unit conversion is done on the speed transmitted to UE.





6.2 Force and Moment Model Force and moment moduleS_{fm}

The force and moment subsystem is mainly used to calculate the resultant force and moment of the airframe subsystem by integrating the actuator state, the flight environment state and the body motion state. $\boldsymbol{S}_{\text{fm}}\boldsymbol{y}_{\text{act}}\boldsymbol{y}_{\text{env}}\boldsymbol{y}_{\text{body}}\boldsymbol{S}_{\text{body}}\boldsymbol{y}_{\text{fm}} \triangleq \{{}^{b}\boldsymbol{F},\ {}^{b}\boldsymbol{M}\}$In order to facilitate the use of 6-DOF airframe motion model, the resultant forces and torques usually need to be uniformly mapped

(coordinate transformed) into the airframe coordinate system.

The modeling process of the resultant force is described in the following, and the modeling process of the resultant torque is similar. The source of the resultant force on the body can be expressed as the following superposition form:

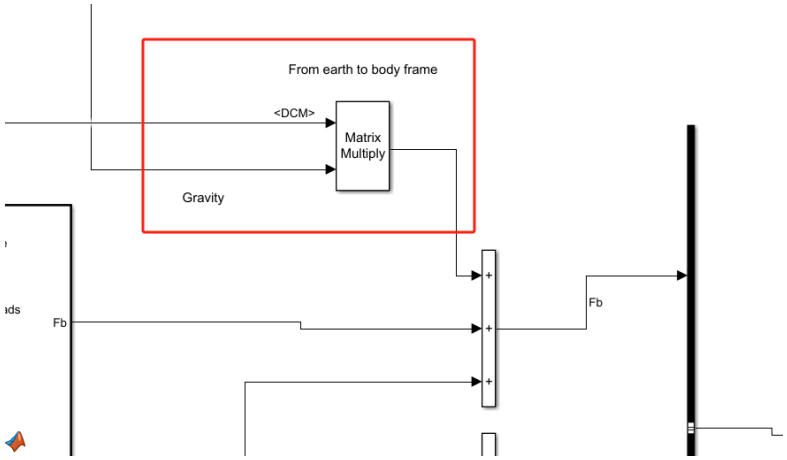
$$^{b}\boldsymbol{F} = {}^{b}\boldsymbol{F}_{\text{aero}} + {}^{b}\boldsymbol{F}_{\text{grav}} + {}^{b}\boldsymbol{F}_{\text{cont}} + \sum {}^{b}\boldsymbol{F}_{\text{act},i}$$

Where, represents the aerodynamic force vector, represents the gravity vector, represents the contact force vector (from the ground support force or the physical collision force of obstacles), represents the driving force vector generated by a certain actuator.$^{b}\boldsymbol{F}_{\text{aero}} \in \mathbb{R}^{3}\, ^{b}\boldsymbol{F}_{\text{grav}} \in \mathbb{R}^{3}\, ^{b}\boldsymbol{F}_{\text{cont}} \in \mathbb{R}^{3}\, ^{b}\boldsymbol{F}_{\text{act},i} \in \mathbb{R}^{3}$

1) The force of gravity in the body coordinate system$^{b}\boldsymbol{F}_{\text{grav}}$

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

Where, represents the gravity acceleration and represents the body mass, corresponding to the simulink module as followsgm

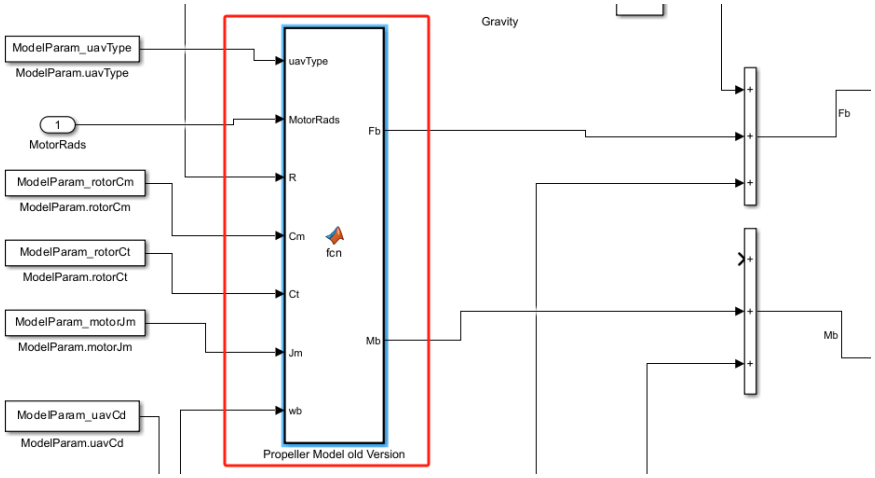


2) Propeller driving force in the body coordinate system$^{b}\boldsymbol{F}_{\text{act},i}$

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

Where, represents the current real-time state of this actuator (e.g., propeller speed, steering gear deflection Angle, tire driving torque, etc.), which can be obtained from the actuator subsystem; $\delta_{i} \in \boldsymbol{y}_{\text{act}}\boldsymbol{S}_{\text{act}}$The expression is directly related to the motion state of the body, the environment state, and the position and direction distribution of the actuator. The specific modeling method can be referred to the literature. $\boldsymbol{f}_{\text{act},i}(\cdot)\boldsymbol{y}_{\text{env}}\boldsymbol{y}_{\text{env}}$For example, the output force and torque of a single propeller as a function of the rotational speed of the propeller can be obtained by decomposing the force and torque into three-dimensional vectors in the body coordinate system according to the rotor installation information. $TMTM\,^{b}\boldsymbol{F}_{\text{act},i}$ The most important difference between different types of vehicle systems is that the position distribution and direction of the actuator force are not the same. Therefore, for different types of aircraft, it is

only necessary to establish different actuator force models through the above equation. In this way, the body subsystem proposed in this subsection can be used on any type of aircraft.

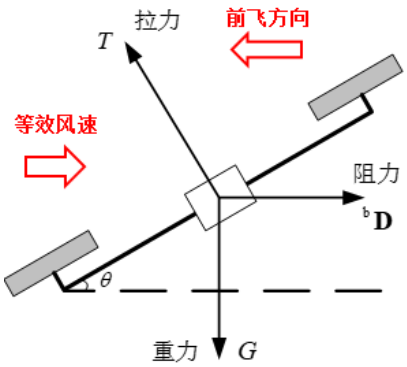


3) Aerodynamic force in body coordinate system (simplified aerodynamic model of multi-rotor) ${}^{b}\boldsymbol{F}_{\text{aero}}$

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

The aerodynamic forces of conventional multi-rotor are mainly air resistance and air damping torque, which respectively represent the resistance to prevent the aircraft from moving forward and the torque to prevent the aircraft from rotating.

- Only the drag effect is considered, that is, the direction of the aerodynamic force is the same as the direction of the relative wind speed (the incoming flow) (note: the lift is perpendicular to the incoming flow direction);
- Theoretically, the propeller will be affected by the incoming flow, resulting in a decrease in the size of the pull force. Since the forward flight speed of the conventional multi-rotor is low, this effect is not significant. We assume that the propeller tension is always the same, and the tensile loss of the propeller due to the incoming flow is included in the drag coefficient.
- The windward area corresponding to different pitch angles of the multirotor is different, so the drag coefficient will change with the change of pitch Angle (and roll Angle)



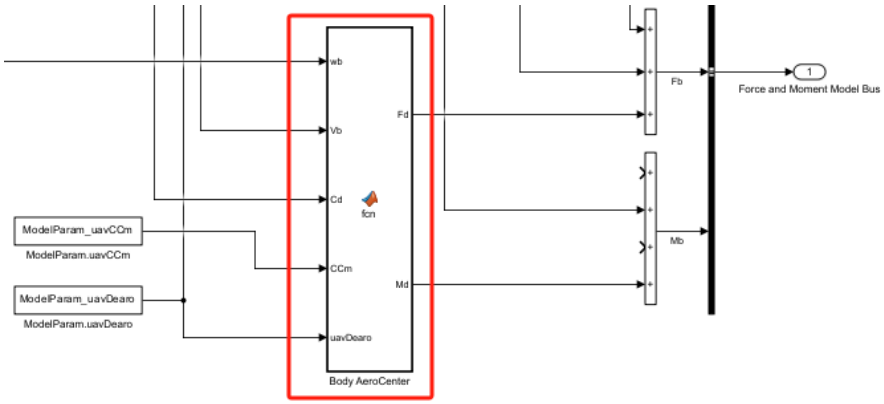
Relative velocity of air flow vb:$^{\boldsymbol{b}}\boldsymbol{v}_a \triangleq [u \quad v \quad w]^T$

Air resistance vector Fd:$^{\boldsymbol{b}}\boldsymbol{F}_d = C_d \cdot [u^2 \quad v^2 \quad w^2]^T$

Airflow relative speed wb:$^{\boldsymbol{b}}\boldsymbol{\omega}_a \triangleq [\omega_x \quad \omega_y \quad \omega_z]^T$

Damping torque vector Md:$^{\boldsymbol{b}}\boldsymbol{M}_d = C_{\text{md}} \cdot \left[\omega_x^2 \quad \omega_y^2 \quad \omega_z^2\right]^T$

Where, is the average drag and drag coefficient in nominal condition (rated forward flight speed and altitude). C_d, C_{md}More complicated, the coefficient can be extended to a curve related to the Angle of attack, and different coefficients can be taken in the three axes.



4) The impact force of the body in the body coordinate system$^{\boldsymbol{b}}\boldsymbol{F}_{\text{cont}}$

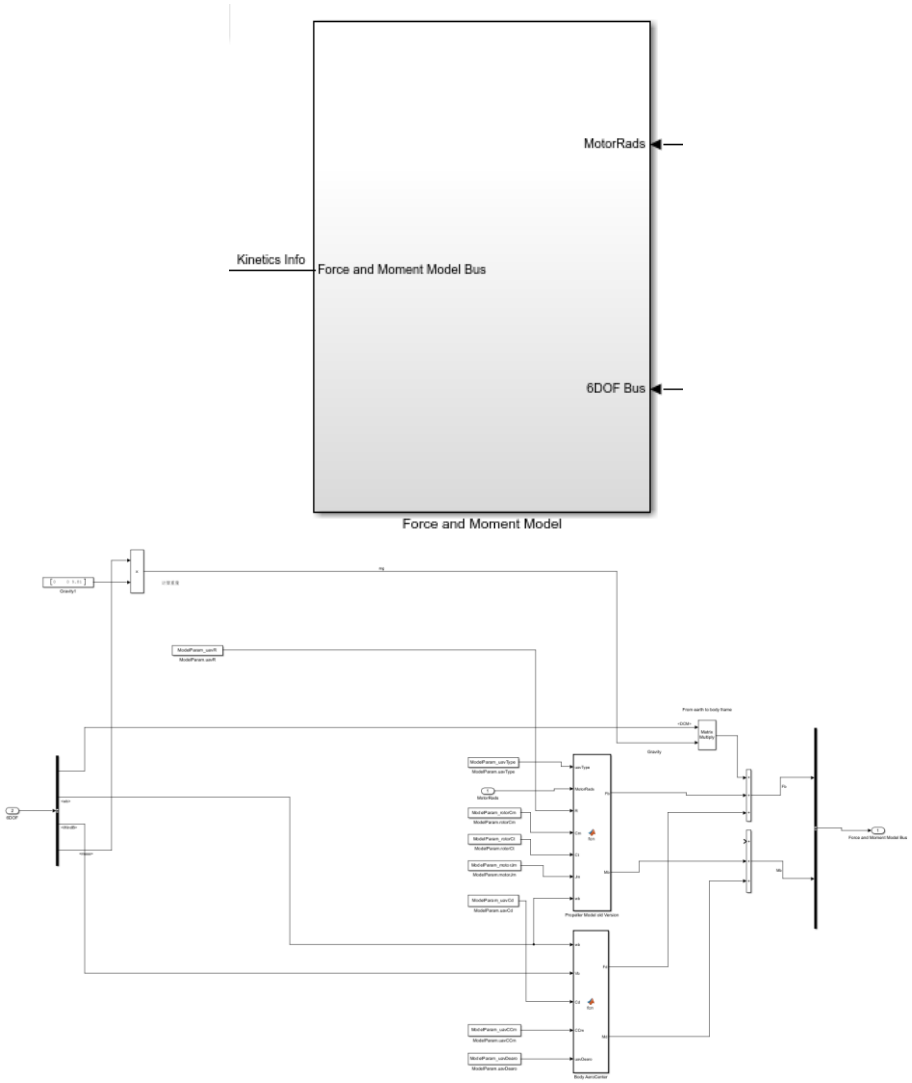
See the following two modules for details:

PhysicalCollisionModel

GroundSupportModel

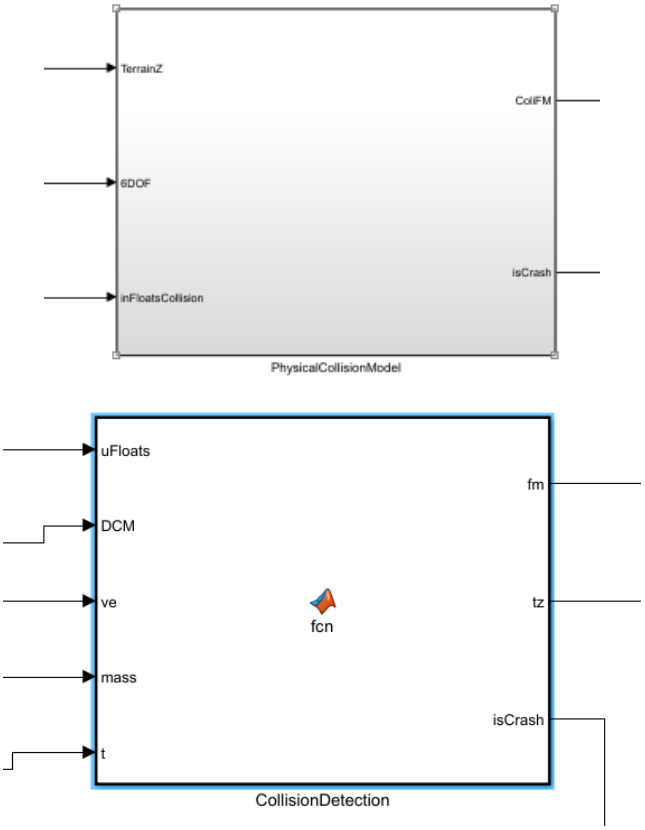
5) Overall input and output (control efficiency model)

Simulate the external forces and torques that the UAV is subjected to. For example, the multi-rotor models all the external forces and torques such as the propeller pull, the aerodynamic force of the fuselage, its own gravity, and the ground support in this module. The input of the module is the motor speed MotorRads, the aircraft kinematic attitude 6DOF and the terrain height input tZ, and the output is the multi-rotor resultant Force, combined torque Force and Moment Model Bus.



6.3 PhysicalCollisionModel (Collision detection function is only supported by personal advanced version or above)

The collision module can detect whether the multi-rotor flight has collided with the object, and the type of the collision object and make a response in accordance with the physical law. When the collision mode is turned on, the speed of the multi-rotor meets the impulse theorem when it collides with the plane. When it collides with the fixed object such as the house, the multi-rotor will bounce back toward the obstacle in the opposite direction, and the rebound speed is 1/10 of the original speed.



It can be seen from the figure that the input of the collision module is uFloats, DCM, ve, mass, t, and the output is fm, tz, isCrash.

Where uFloats are 20-dimensional external input floating point signals, this port is reserved for the collision model and can be transmitted from UE4 over a UDP network. DCM is the direction cosine matrix, ve is the velocity at the time of collision, mass is the mass of the multirotor, and t is the timestamp. The output fm force and torque directly act on the 6DOF to affect the body motion, tz represents the height of the multi-rotor from the ground and the coordinates of XYZ, isCrash is the collision judgment, if the collision occurs, the three motors are damaged. The specific collision logic can be clicked into the module for detailed understanding.

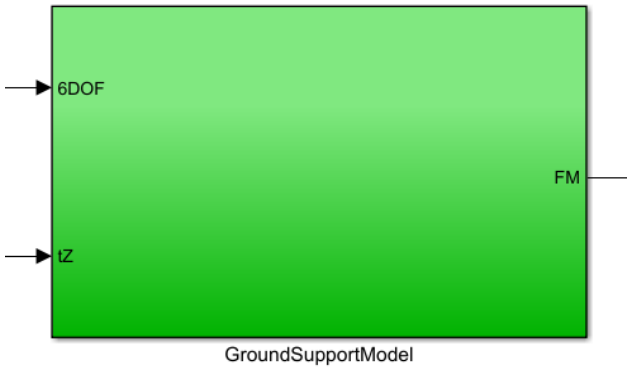
6.4 GroundSupportModel Ground support module

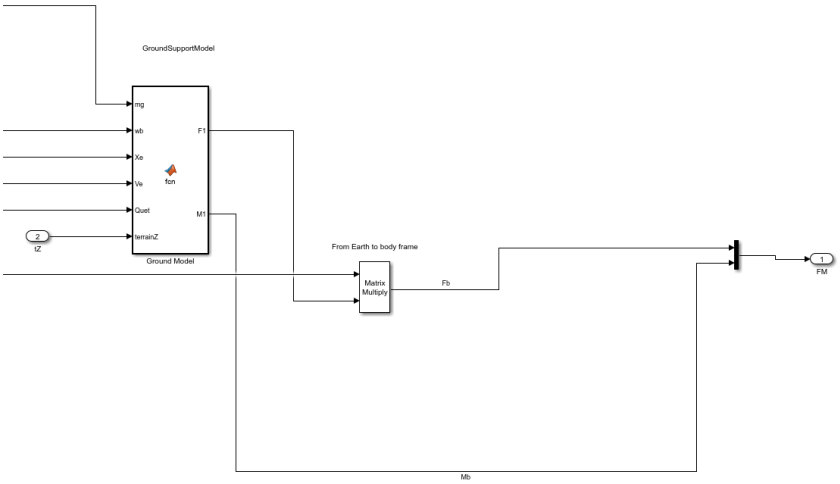
The collision force of the body mainly comes from the supporting force and friction force of the ground, as well as the physical collision force when there are obstacles and obstacles. F_zDue to the complexity of the shape of the object, it is very difficult to solve the physical contact point and calculate the collision force for the complex shape, and in most cases, such a high-precision collision force is not necessary. At present, most physics engines use a simplified method to solve the physical force, that is, all objects are simplified into a relatively simple basic geometry (such as a cylinder or a

cuboid) to calculate the physical contact force between them and the ground or other objects. Each object has a spring-cushioning model under each surface to simulate the contact, collision and cushioning of the actual object. By adjusting the stiffness of the spring, the surface softness of different objects can be simulated, or the buffering effect of the ground can be simulated. The cushioning contact of a typical ground support force used in this model can be expressed as follows:

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

Where, represents the displacement of the body from the ground surface in the z direction; $\Delta z \Delta z > 0$ Indicates that the body is located above the ground, and there is no contact, so the force is 0; $\Delta z < 0$ It means that the object is under the ground (falling into the ground). At this time, the ground generates a feedback controller and generates a support force to try to control the displacement to 0, so as to realize the simulation of ground contact and buffer. Δz In the above equation, and is the buffer coefficient of the spring. The larger the value is, the stronger the action force to restore deformation is, the greater the hardness of the object surface is, and the greater the instantaneous reaction force when the collision occurs. $k_1 > 0 k_2 > 0$





6.5 6DOF rigid body moduleS_{body}

The main role of the fuselage subsystem is to calculate the motion state of the aircraft according to the total force and torque on the fuselage. $\boldsymbol{S}_{\text{body}}\boldsymbol{y}_{\text{fm}}\boldsymbol{y}_{\text{body}}$ In the actual aircraft modeling, two basic assumptions are usually used. The first assumption is the flat earth assumption, that is, the aircraft movement range is small, and the curvature of the earth surface can be ignored. Secondly, rigid body assumption, that is, the body of the aircraft is assumed to be rigid and will not be deformed randomly. The above assumption applies to most unmanned vehicle systems. On this basis, the motion of the vehicle body can usually be expressed as a quaternion based dynamic equation with six degrees of freedom as follows

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

${}^{e}\boldsymbol{p} \in \mathbb{R}^3$Is the position vector of the aircraft defined in the earth coordinate system;${}^{b}\boldsymbol{v} \in \mathbb{R}^3$ Is the velocity vector of the aircraft defined in the body coordinate system;${}^{b}\boldsymbol{\omega} \in \mathbb{R}^3$ Is the angular velocity vector of the aircraft defined in the body coordinate system;$\boldsymbol{R}_b^e \in \mathbb{R}^{3\times3}$ Is the rotation matrix that transforms the vector from the body coordinate system to the earth coordinate system;$\boldsymbol{J} \in \mathbb{R}^{3\times3}$ And$m \in \mathbb{R}^{+}$ are the moment of inertia matrix and mass of the aircraft, and denote a cross-product matrix operator. For example, let, then the cross-product matrix operator is defined as follows$[\ ]_\times {}^{b}\boldsymbol{\omega} \triangleq [w_1, w_2, w_3]^T$

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

To define the quaternion, let$\boldsymbol{q}_e^b \in \mathbb{R}^4 \boldsymbol{q}_e^b = [q_0 \ q_1 \ q_2 \ q_3]^T = [q_0 \quad {q_v}^T]^T$

Then we have

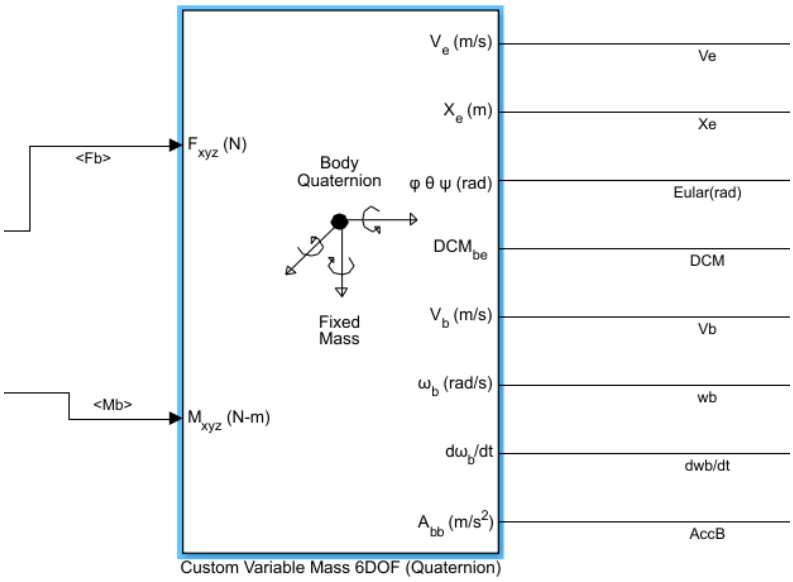
$$\dot{\boldsymbol{q}}_e^b = \frac{1}{2}\begin{bmatrix} 0 & -{}^b\boldsymbol{\omega}^{\boldsymbol{T}} \\ {}^b\boldsymbol{\omega} & -[{}^b\boldsymbol{\omega}]_\times \end{bmatrix} \boldsymbol{q}_e^b \xrightarrow{q_e^b=[q_0 \quad {q_v}^T]^T} \begin{cases} \dot{q}_0 = -\frac{1}{2} q_v^T \cdot {}^b\boldsymbol{\omega}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

In practice, the angular velocity can be approximately measured by a three-axis gyroscope, and the above differential equation is linear$^b\boldsymbol{\omega}$

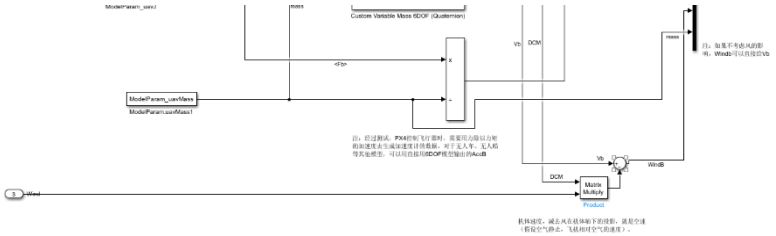
$$R_b^e = C(\boldsymbol{q}_e^b) = \begin{bmatrix} 1-2(q_2^2+q_3^2) & 2(q_1q_2-q_0q_3) & 2(q_1q_3+q_0q_2) \\ 2(q_1q_2+q_0q_3) & 1-2(q_1^2+q_3^2) & 2(q_2q_3-q_0q_1) \\ 2(q_1q_3-q_0q_2) & 2(q_2q_3+q_0q_1) & 1-2(q_1^2+q_2^2) \end{bmatrix}$$

The 6-DOF module of the UAV is used to describe the attitude and position changes of the UAV when it moves in the air. This model is based on the principle of rigid body dynamics, which considers the UAV as a rigid body and takes into account the rotational motion of the UAV in three coordinate axes (pitch, roll and yaw) and the translational motion between the body and the earth coordinate system (front-back, left-right and up-down).

The inputs of the model are the force and torque of the body, and the outputs are the velocity and acceleration in the body coordinate system, the velocity in the earth coordinate system, the position, the Euler Angle, the direction cosine matrix (rotation matrix), the angular velocity and angular acceleration.



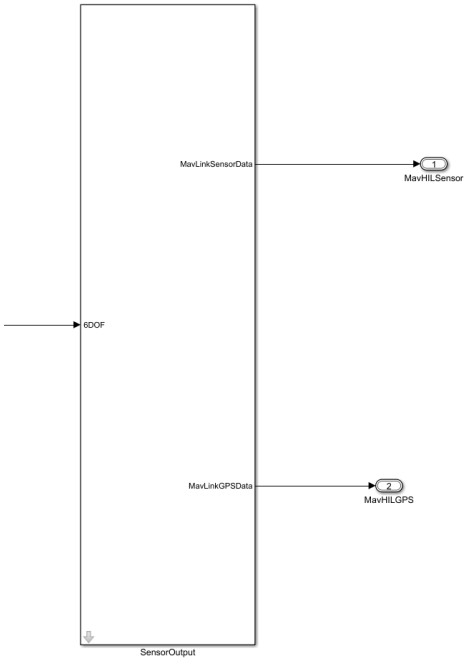
Through the mathematical modeling of these degrees of freedom, the module can derive the dynamic equation of the UAV when it moves in the air, which can be used for control system design, path planning and flight simulation. In addition, the model can be extended according to the actual needs to consider more factors, such as the nonlinear characteristics of the aircraft, aerodynamic force and moment of inertia.



6.6 SensorOutput Sensor output moduleS_{sens}

The module includes the environment model, sensor model and GPS model. The environment model simulates the impact of gravity and atmospheric pressure on the flight of the unmanned system. In the sensor model, the magnetometer and inertial navigation are modeled, and the noise simulation is added. The GPS model is used to calculate the GPS data, which is fed back to the PX4 controller during simulation.

The input of the module is a 6DOF Bus structure, and the output is MavHILSensor and MavHILGPS. The platform is mask encapsulated. Users only need to copy the module to the unmanned vehicle model and input data to the module according to the 6DOF Bus structure to complete the sensor and GPS modeling.



6.7 3DOutput 3D display moduleS_{3d}

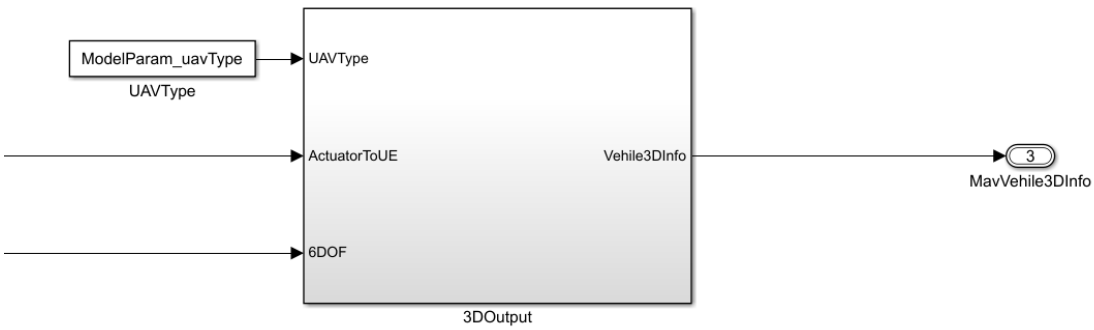
The input of this module is:

1) UAVType: 3D display ID, from ModelParam_uavType in ***_init.m, determined by XML file in 3D model file, for example: UAVType is 3 for regular quadrotor and 100 for small fixed wing.

2) ActuatorToUE: from the motor model, it determines the rotation of the motor/servo of the unmanned vehicle system in the 3D engine.

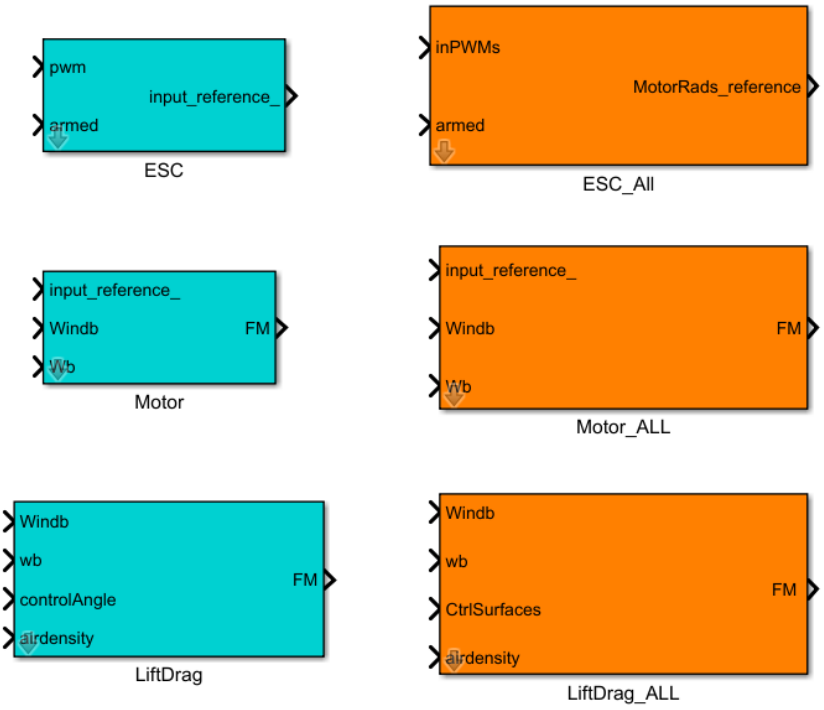
3) 6DOF: The 6DOF Bus input from the 6DOF model receives information such as position, velocity, attitude, and acceleration of the unmanned vehicle system.

The output is MavVehile3DInfo: in the 3D display module, the input information will be packaged according to the protocol, and the data will be sent to the 3D engine through the interface to realize visualization processing.



6.8 Gazebo Model Module

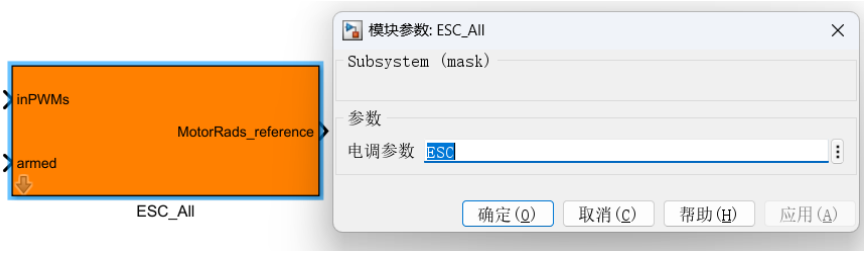
At present, there are six types of Gazebo model modules that have been completed in the RflySim platform, namely ESC module, Motor module, LiftDrag module, ESC_All module, Motor_ALL module and LiftDrag_ALL module. Among them, ESC_All module is a composite module composed of 8 ESC modules, similarly, Motor_ALL module and LiftDrag_ALL module can be obtained.



Users can use the Gazebo model module to realize the modeling of rack types supported by Gazebo platform on RflySim platform, such as rotor, fixed wing, car, etc.

6.8.1ESC_ALL module

1) Feature introduction



Module input

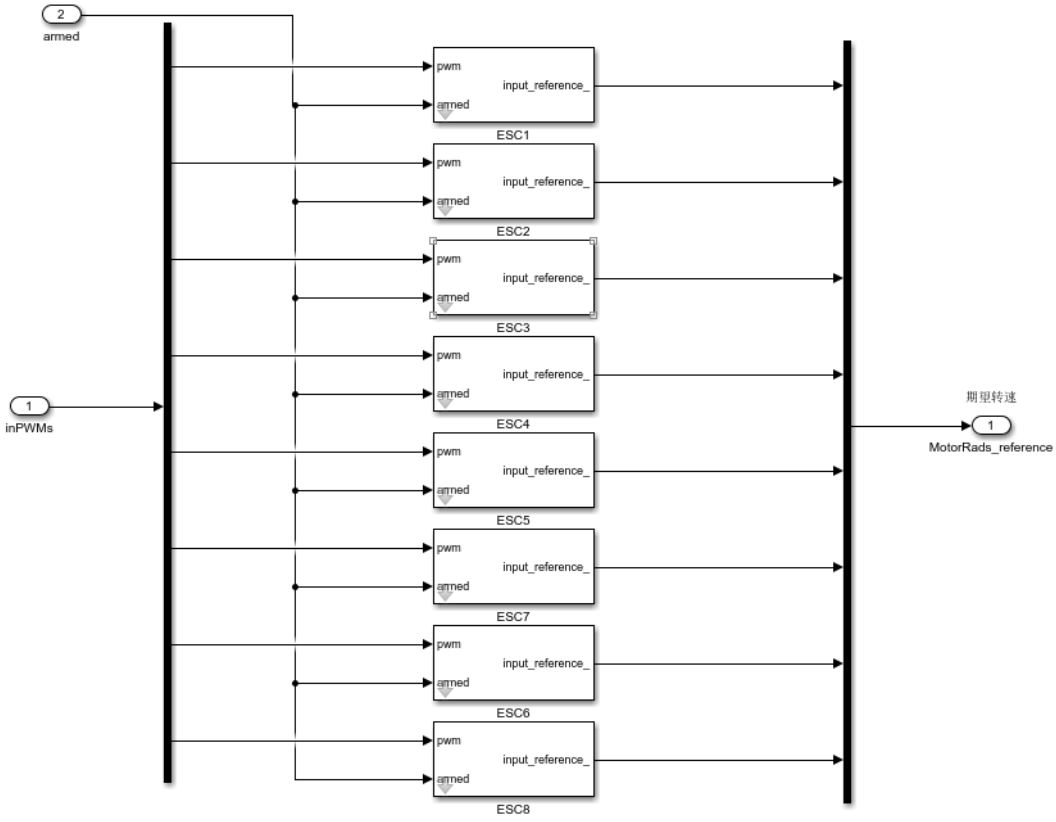
(1) inPWMs: PWM signal (8 channels, data from the controls of the motor control mavlink_hil_actuator_controls_t message returned by the PX4 flight control, and the same order as the channel definition at the end of the corresponding "xxx.sdf" file in Gazebo).

(2) armed: the Boolean logic variable of external input. False means that the output of

ESC_ALL module is blocked, and the output is 0. Otherwise, the output of ESC_ALL module is enabled. The value here generally comes from the first dimension data of the inCopterData interface, which is the unlocking flag bit from CopterSim, so as to ensure that the aircraft motor should not turn and maintain the initial value when it is not unlocked in the simulation process. Only after unlocking can the inPWMs input be turned on, so as to avoid that in some cases, When the simulation starts but the aircraft is not unlocked, it is the case that the aircraft is disorderly.

Module output

MotorRads_reference: expected speed, the expected value of 8 motor control signals, including two control types: motor speed and elevator and other parts rotation. The MotorRads_reference is an 8-dimensional vector.



2) Module parameters

The ESC_ALL module is masked and contains eight ESC submodules. The input parameters are the "ESC" structure vector in the "ModelName_init.m" file, ESC(1), ESC(2), ESC(3)... Are the parameter structures for each channel, in the same order as the channel definition at the end of the "xxx.sdf" file corresponding to the Gazebo model (file path is *\PX4PSP\Firmware\Tools\sitl_gazebo\models).

The "ESC" struct vector in the "ModelName_init.m" file is as follows.

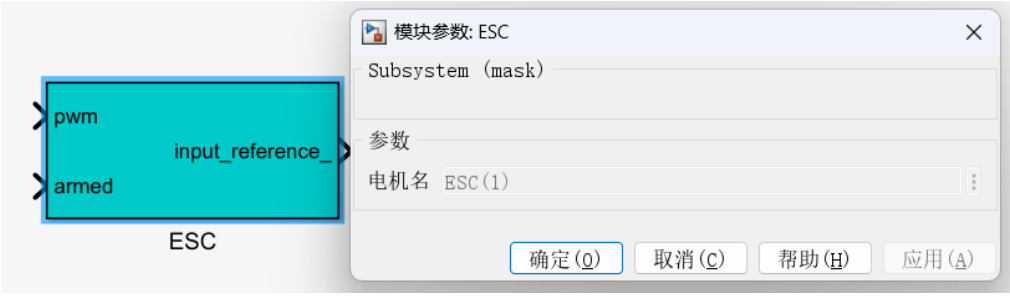
```
% Electrical tuning parameter template
ESCTmp.isEnable= false;     % Whether this switch is enabled or not
ESCTmp.input_offset = 0;    % input offset
```

```
    ESCTmp.input_scaling = 1;   % input scaling factor
    ESCTmp.zero_position_disarmed = 0; % zero position unlock value
    ESCTmp.zero_position_armed = 0; % zero position lock value
    ESCTmp.joint_control_type=1; % joint control type, 0: velocity, 1: the position,...

    % Define the order of ESC channels and ESCTmp parameter values according to
control_channels at the end of xxx.sdf.jinja file.
    ESC = [ESCTmp, ...
           ESCTmp, ...
           ESCTmp,...
           ESCTmp, ...
           ESCTmp, ...
           ESCTmp, ...
           ESCTmp,...
           ESCTmp]; % electrically adjusted data structure, expanded to 8 dimensions
```

6.8.2 ESC module

Below is the ESC module of Gazebo model module. This module is also masked. The input parameter is the "ESC(1)" structure in the "ModelName_init.m" file.



Input:

(1) pwm: PWM signal, the data in the simulation comes from one of the channels of the inPWMs interface of the model, and the size is between -1 and 1.

(2) armed: unlock flag bit, external input Boolean logic variable, same as armed of ESC_ALL module.

Output:

input_reference_ : The desired speed corresponding to the motor/servo in RPM.

6.8.3 Motor_ALL module

1) Function introduction

Enter:

(1) input_reference_ : The control signal processed by the ESC module expects the motor speed control type signal to be input to the interface of the motor module.

(2) Windb: Input from the external Environment Model Bus, representing the relative wind speed of the body, as a 1x3 matrix variable.

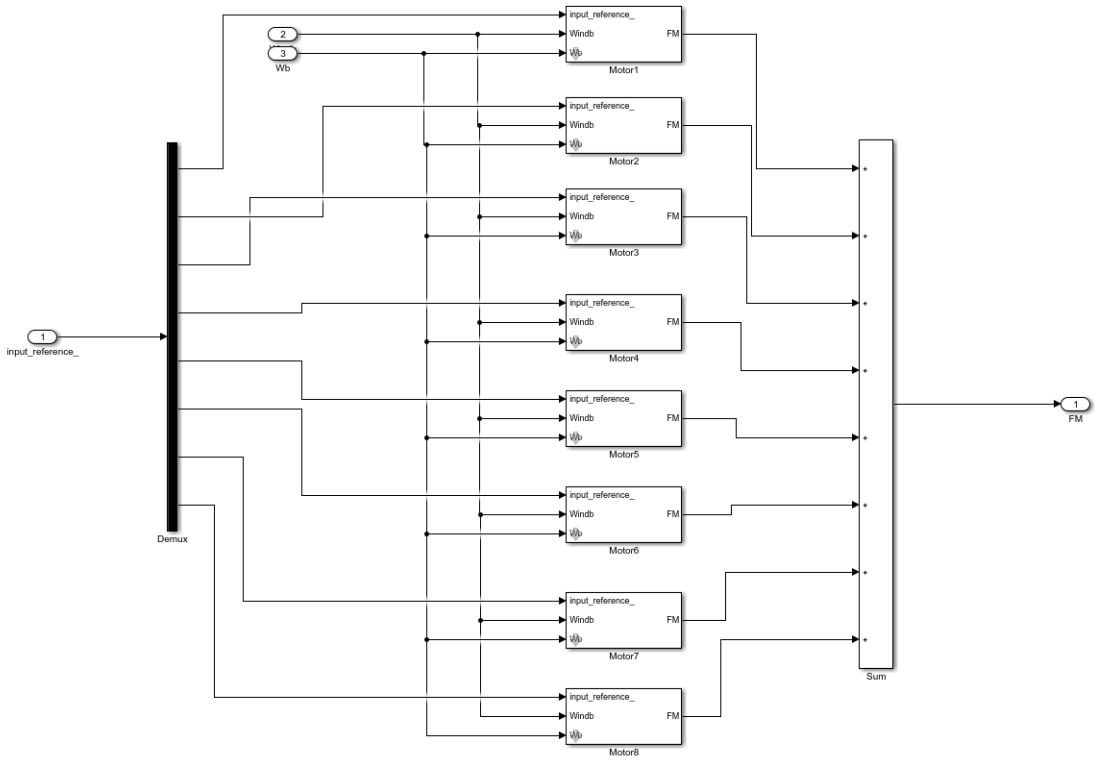
(3) wb: Input from the external 6DOF Bus, representing the rotational angular velocity of the body.

Output:

FM: The force and torque generated by the motor in the body coordinate system, 6-dimensional vector, the first 3 dimensions are the resultant force generated by the motor module, and the last 3 dimensions are the resultant torque generated by the motor module.

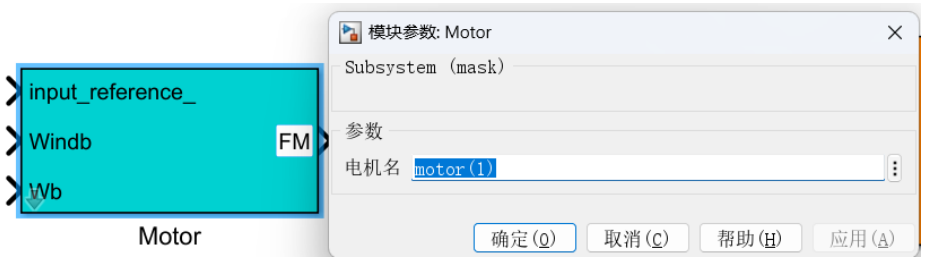
2) Parameters of the module

Motor_ALL is masked, the input parameter is the "motor" structure vector in the "ModelName_init.m" file, and the parameter values refer to the part of the parameters of the motor_model plug-in in the xxx.sdf.jinja file. It contains 8 Motor sub-modules, which process one of the channel control signals of input_reference_ respectively, and can support the control of up to 8 motors of UAV.



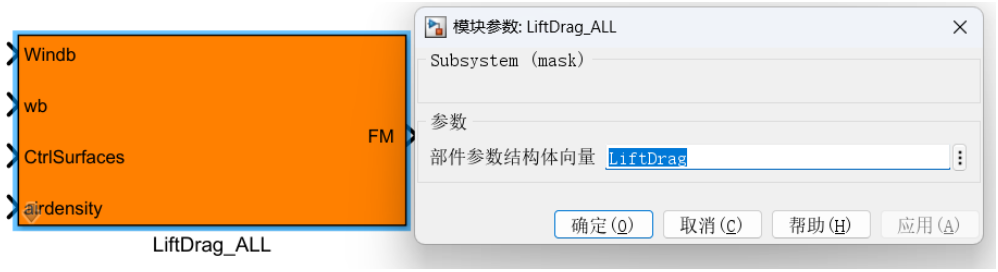
6.8.4 Motor module

The motor sub-module is also masked. The input parameter is "motor(1)" in the "ModelName_init.m" file, which is defined and initialized in "ModelName_init.m". The final output of the module is the force and torque of a single motor module, with force in the first three dimensions and torque in the second three dimensions.



6.8.5 LiftDrag_ALL module

1) Feature introduction



Enter:

(1) Windb: Input from the external Environment Model Bus, representing the relative wind speed of the body, as a 1x3 matrix variable.

(2) wb: Input from outside the 6DOF Bus, which represents the rotational angular velocity of the body.

(3) CtrlSurfaces: 8-dimensional vector, output signal from ESC module (rudder surface control signal only).

(4) airdensity: The density of the air, used to calculate the dynamic pressure,.

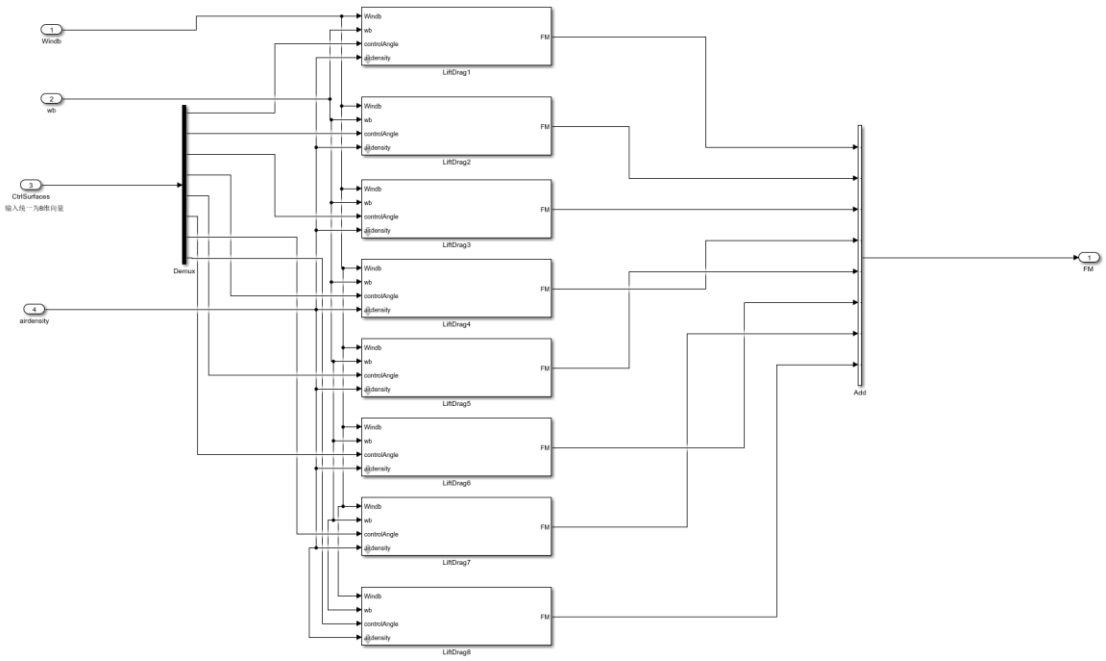
$$q = \frac{1}{2} rho * V^2$$

Output:

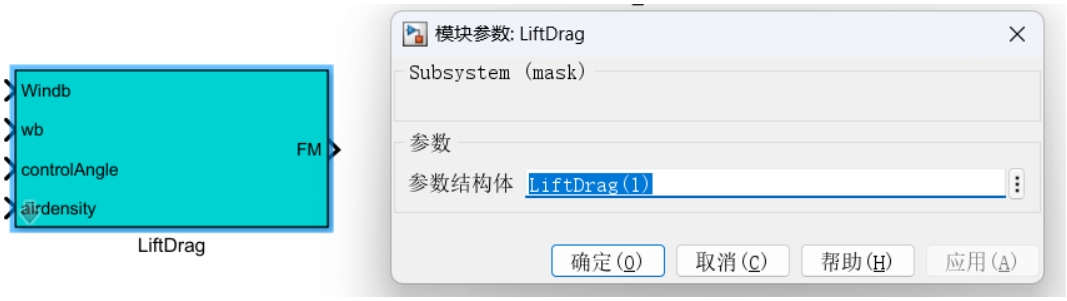
FM: The force and torque in the body coordinate system generated by the rudder surface such as elevator and rudder, 6-dimensional vector, the first 3 dimensions are the resultant force, and the last 3 dimensions are the resultant torque.

2) Module parameters

The "LiftDrag_ALL" module is masked, and the input parameters are the LiftDrag structure variables in the "ModelName_init.m" file, LiftDrag (1), LiftDrag (2), LiftDrag (3)... Are the parameter structures for each rudder surface, or elevator, rudder and other different parts, and the order is the same as the channel definition order at the end of Gazebo "xxx.sdf" file.



6.8.6 LiftDrag module



It is used to calculate the lift, drag and torque of the components responsible for the elevator, rudder, aileron and so on. The input parameters of the LiftDrag module are from the LiftDrag (1) structure in the "ModelName_init.m" file, which is defined and initialized in "ModelName_init.m".

7. RflySim has supported the introduction of vehicle simulation operation

7.1 Multi-rotor model

The modeling and simulation case of any multi-rotor model is based on the platform unified Simulink modeling template proposed above. The only difference between them is the control efficiency model that maps the force and torque of all rotors (projection and summation of magnitude and direction) to the resultant force of the fuselage. This is achieved in the platform modeling template through the ModelParam_uavType parameter to the computer frame and torque distribution. At the same time, to realize different multi-rotor models, the parameters in the table below should be adapted

Table 1 Comparison table of model constant parameters1

Parameter names	**The name of the parameter in the formula**	**.m file parameter name**
Total mass	m	ModelParam.uavMass
Gravitational acceleration	g	ModelParam.envGravityAcc
Matrix of moment of inertia	J	ModelParam.uavJ
Multirotor fuselage radius	$\frac{d}{2}$	ModelParam.uavR
Propeller pull factor	c_T	ModelParam.rotorCt
Propeller torque coefficient	c_M	ModelParam.rotorCm
Throttle to motor steady-state speed curve slope	C_R	ModelParam.motorCr
Throttle to motor steady-state speed curve zero	ω_b	ModelParam.motorWb
Motor propeller moment of inertia	J_{RP}	ModelParam.motorJm
Motor response time constant	T_m	ModelParam.motorT
Drag coefficient	C_d	ModelParam.uavCd
Damping moment coefficient	C_{dm}	ModelParam.uavCCm

7.1.1 Quadrotor

7.1.1.1 Modeling principle

1) Control efficiency model (force and torque)

Map the force and torque of all rotors (projection and sum of magnitude and direction) to the body net force. Regarding the control efficiency model of the multi-rotor, we mainly divide it into

two aspects, the single propeller tension and inverse torque model and the overall machine tension and torque model.

① Single propeller pull and reverse torque model

When the multi-rotor is hovering in the absence of wind, is the speed of the propeller, is the rate of change of the angular velocity of the propeller (theoretically, it is no change of the speed when hovering in the absence of wind, but the change of battery power and the wear of the propeller may make a small change of the angular velocity of the propeller), and is a constant and can be determined by experiments$\varpi_i i\dot{\varpi}_i i c_T c_M$

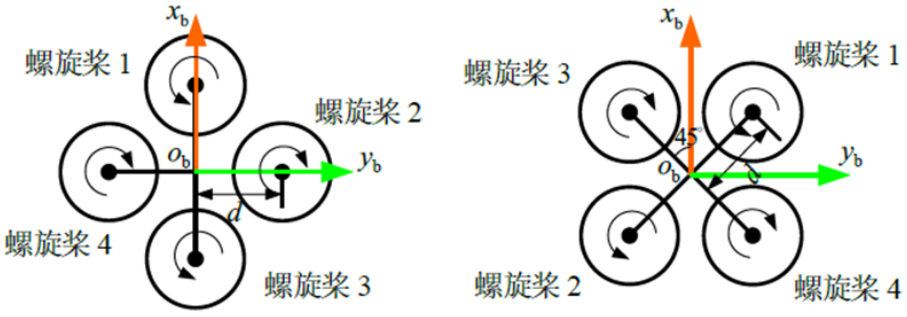
The propeller pulling force can be expressed as follows. $T_i = c_T \varpi_i^2$

The static antitorsional torque can be expressed as:$M_i = c_M \varpi_i^2$

The dynamic antitorsion moment can be expressed as follows.$M_i = c_M \varpi_i^2 + J_{RP}\dot{\varpi}_i$

② Complete machine tension and torque model

For the multi-rotor, in order to achieve control allocation, it is first necessary to determine the position of all motors in the body coordinate system.



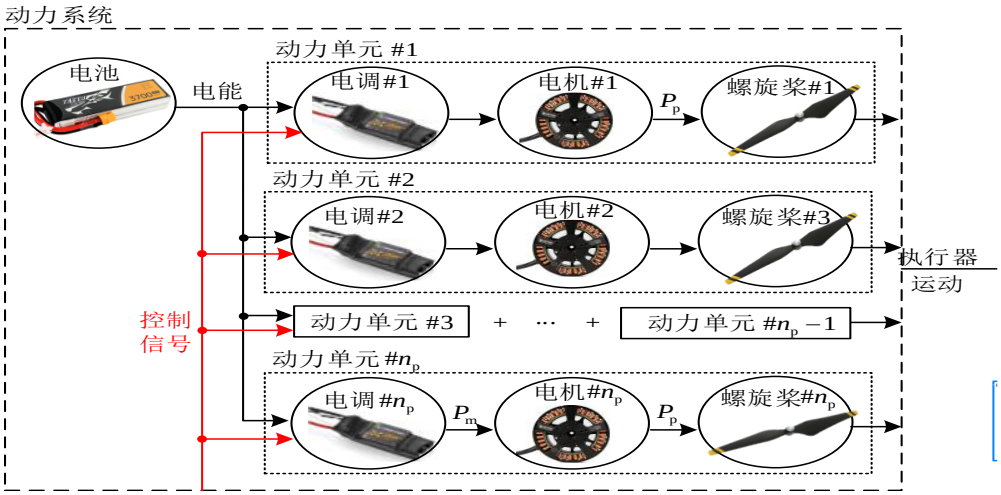
The Angle between the body shaft and the support arm where each motor is located is, and the distance between the center of the body and the motor is denoted as.$o_b x_b \varphi_i \in \mathbb{R}_+ \cup \{0\} d \in \mathbb{R}_+ \cup \{0\}$

For the X-shaped quadrotor, the pull and torque generated by the propeller can be expressed as follows.

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

Where is the total pulling force, and, are the combined torques in the x, y and z directions, respectively.$f \tau_x \tau_y \tau_z$

2) Power unit model



The power unit model is the whole power mechanism which is a group of brushless DC motor, electric and propeller. The input is the motor throttle command of 0 ~ 1, and the output is the propeller speed.

After receiving the throttle command and the battery output voltage, the electric regulator generates the equivalent average voltage as. $\sigma U_b U_m = \sigma U_b$Firstly, a voltage signal is input, and the motor can rotate to a steady speed. ϖ_{ss}This relationship is usually linear and is denoted by

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

Where, and are constant parameters. $C_R = C_b U_b C_b \varpi_b$Secondly, when a throttle command is given, it takes some time for the motor to reach the steady-state speed, which determines the dynamic response of the motor and is denoted as. $\varpi_{ss} T_m$In general, the dynamic process of BLDC motor can be simplified as a first-order low-pass filter, and its transfer function can be written as follows.

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

In other words, when a desired steady-state speed is given, the motor speed cannot be reached immediately, but needs to be adjusted over a period of time. $\varpi_{ss} \varpi_{ss}$Combining the above two equations, the complete power unit model can be obtained as follows:

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

7.1.1.2 Modeling and simulation case

1) Quadrotor model based on maximum template

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

2) Quadrotor model with collision detection

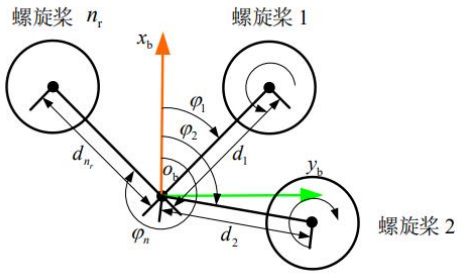
*:\PX4PSP\RflySimAPIs\4.RflySimModel\2.AdvExps\e2_MultiModelCtrl\2.MultiModelCtrl

Coll\Readme.pdf

7.1.2 hexacopter

7.1.2.1 Modeling principle

The only difference from the quadrotor is the control efficiency model, for the multirotor, in order to realize the control allocation, it is first necessary to determine the position of all the motors in the body coordinate system.



For a multirotor with one propeller, label the propellers clockwise from to, as shown in the image above. $n_r i = 1 i = n_r \geq 5$The Angle between the body shaft and the support arm where each motor is located is denoted by, and the distance between the center of the body and the first motor is denoted by.$o_b x_b \varphi_i \in \mathbb{R}_+ \cup \{0\} i d_i \in \mathbb{R}_+ \cup \{0\}, i = 1,2,\dots,n_r$

The pull force and torque generated by the propeller can be expressed as follows.

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

7.1.2.2 Modeling and simulation cases

1) Six rotor model based on maximum template

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

7.1.3 Quad-axis ocrotor

7.1.3.1 Modeling principle

Refer to the modeling principle of the six rotor and the four rotor

7.1.3.2 Modeling and simulation case

1) Quad-axis ocrotor model based on maximum template

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

7.1.4 Octocoxrotor

7.1.4.1 Modeling principle

Refer to the modeling principle of the six rotor and the four rotor

7.1.4.2 Modeling and simulation case

1) Eight-rotor model based on maximum template

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

7.2 Small fixed wing

7.2.1.1 Modeling principle

1) Aerodynamic forces and torques

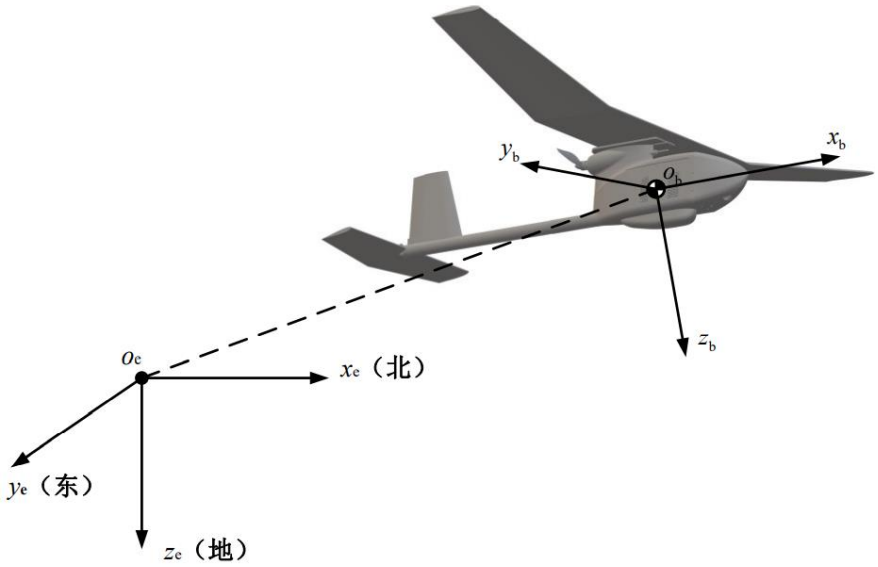
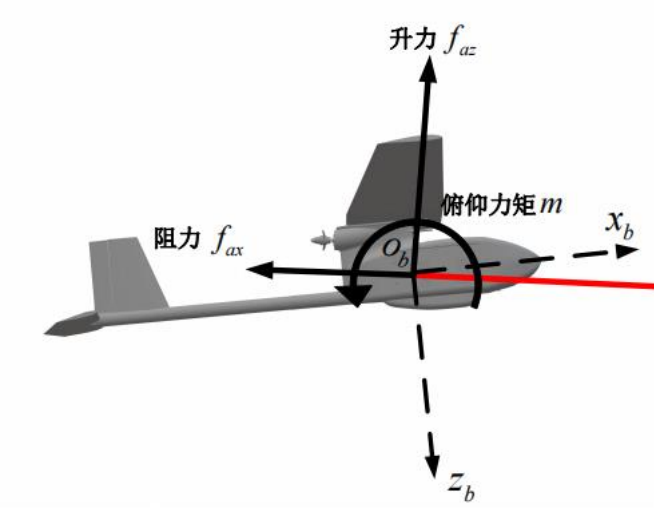


Figure 0.1 Inertial coordinate system and body coordinate system01

① Longitudinal aerodynamics

Longitudinal aerodynamic forces and moments include lift, drag and pitch moments, under which the body will move in the plane (inertial coordinate system: North East ground), which is also known as the pitch plane$o_e x_e z_e$



Lift, drag and pitch moments are all mainly affected by the change in Angle of attack. αIt is also affected by the pitch rate of the body and the elevator. $\omega_{y_b}\delta_e$Therefore, the formulas for lift, drag and pitching moment can be written as

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

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

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

Model simplification: Perform a first-order Taylor expansion on the above equation, and then dimensionless the partial derivative after the approximate linearization process

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

Finally, the longitudinal aerodynamic forces and torques are expressed as

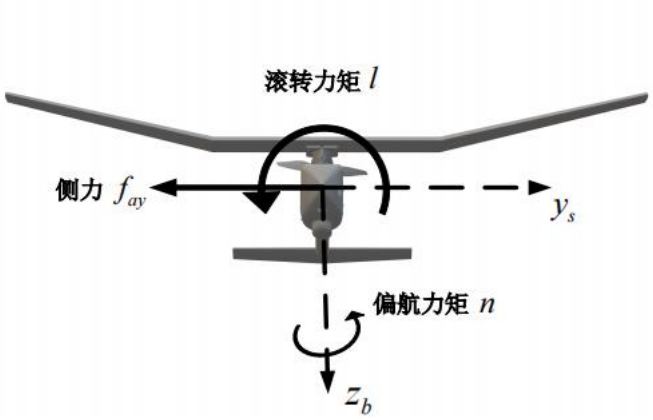
$$f_{az} = \frac{1}{2}\rho {V_a}^2 S\left(C_{L_0} + C_{L_\alpha}\alpha + C_{L_q}\frac{c}{2V_a}\omega_{y_b} + C_{L_{\delta_e}}\delta_e\right)$$

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

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

② Lateral aerodynamics

The aerodynamic forces and moments in the transverse direction cause the fixed wing UAV to move along the axis direction (the body coordinate system), and also cause the roll and yaw motion$o_b y_b$



The aerodynamic force in the transverse direction is mainly affected by the sideslip Angle, but also by the roll rate, yaw rate, ailerons and rudder, which is denoted by

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

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

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

Model simplification: By applying the first-order Taylor expansion to the above equation and then dimensionless the partial derivative after the approximate linearization process, the linear representation of the aerodynamic force and moment in the transverse direction can finally be obtained as

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

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

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

2) The thrust of the dynamical system

In the modeling process, it is considered that the power system of the fixed-wing UAV is installed along the body shaft, and the thrust generated by the propeller power system at zero airspeed is$o_b x_b$

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

Therefore, when the airspeed is, the thrust of the power system is expressed as in the body coordinate systemV_a

$$^{b}T = \begin{bmatrix} T - K\frac{V_a}{\sqrt{T}} \\ 0 \\ 0 \end{bmatrix}$$

7.2.1.2 Modeling and simulation case

1) Fixed-wing model based on minimum input and output interface

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

2) Fixed wing model with collision detection

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

3) Fixed wing position control

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

4) Fixed wing attitude control

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

5) Fixed wing speed altitude yaw control

*:\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 Unmanned vehicles

7.3.1 Refine the unmanned vehicle model

7.3.1.1 Modeling principle

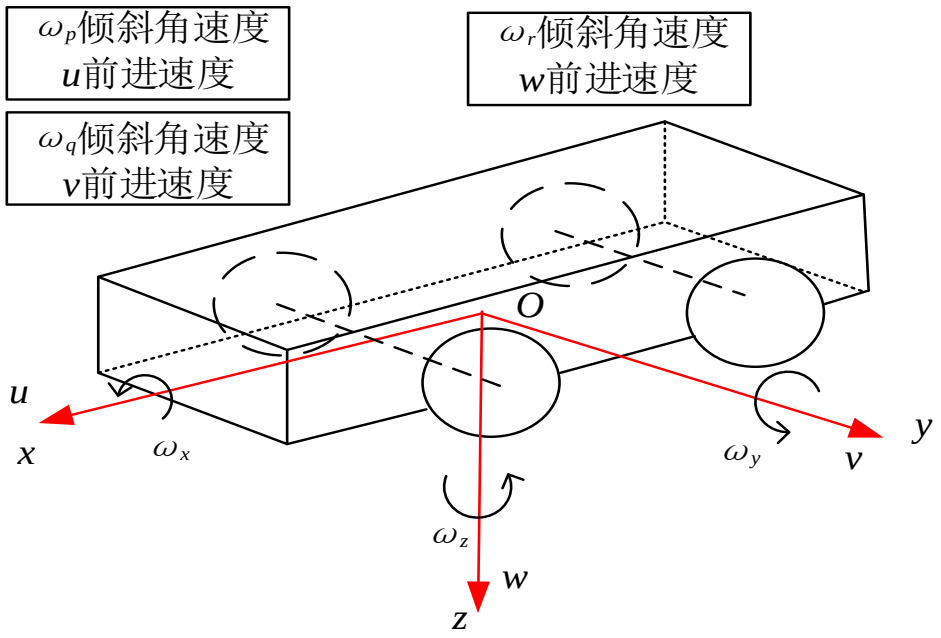


Figure 0.2 Diagram of the vehicle coordinate system02

1) Force and torque model of the tire

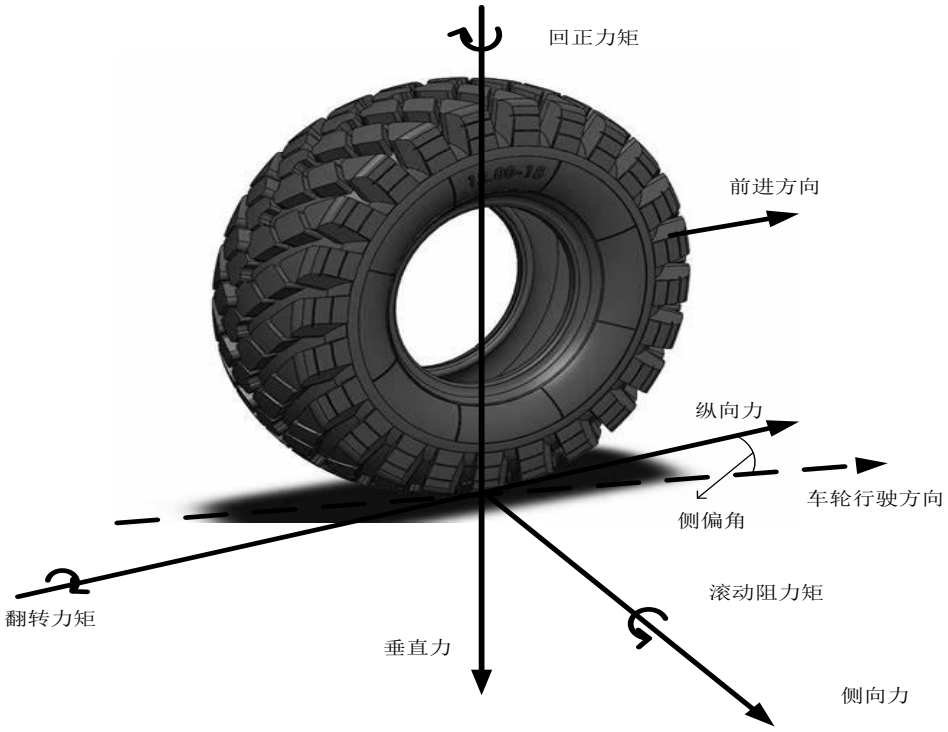


Figure 0.3 Tire model diagram03

Here, the mechanical characteristics of the vehicle tire are discussed based on the magic formula tire model. The magic formula condition of the tire model is shown in the figure. The input is the longitudinal slip rate, the lateral Angle, the inclination Angle, and the vertical load of the wheel. The output is longitudinal force, lateral force, overturning moment, rolling resistance moment and reversing moment.

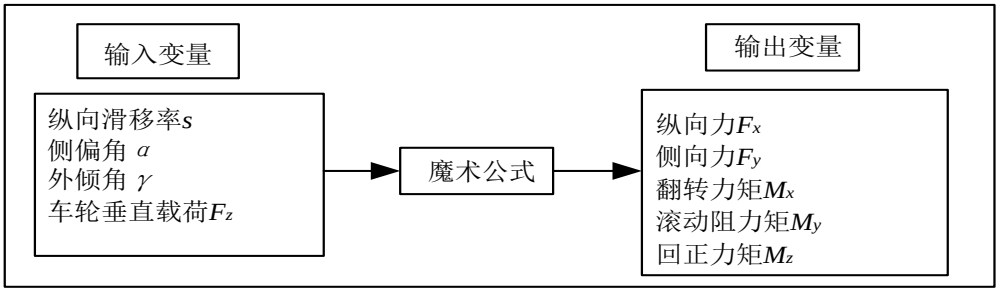


Figure 0.4 Magic formula schematic diagram04

① Longitudinal slip rate *s*

Longitudinal tire slip rate refers to the degree of tire sliding in the longitudinal direction during vehicle driving. It is calculated by comparing the difference between the actual sliding speed of the tire and the theoretical sliding speed. The larger the value of the longitudinal slip rate of the tire, the greater the sliding degree of the tire in the longitudinal direction, and its expression is as follows:

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

Here, V_x is the longitudinal velocity of the vehicle (in m/s), R *is* the radius of the tire (in m), and ω is the angular velocity of the tire (in rad/s).

② The sideslip Angle α

Figure 0.4 shows that the side-slip Angle refers to the Angle between the tire and the vertical direction when the vehicle is moving. Figure 0.4 Magic formula schematic diagram04It describes the degree of deviation of the tire from the direction of travel of the vehicle when it is driving in a turn or curve. The sidestep Angle of the tire has an important impact on the handling performance and stability of the vehicle.

③ Camber Angle γ

Tire camber is the Angle at which the wheel tilts with respect to the vertical direction. It is an important parameter in the vehicle suspension system, which can affect the handling, driving stability and tire wear of the vehicle.

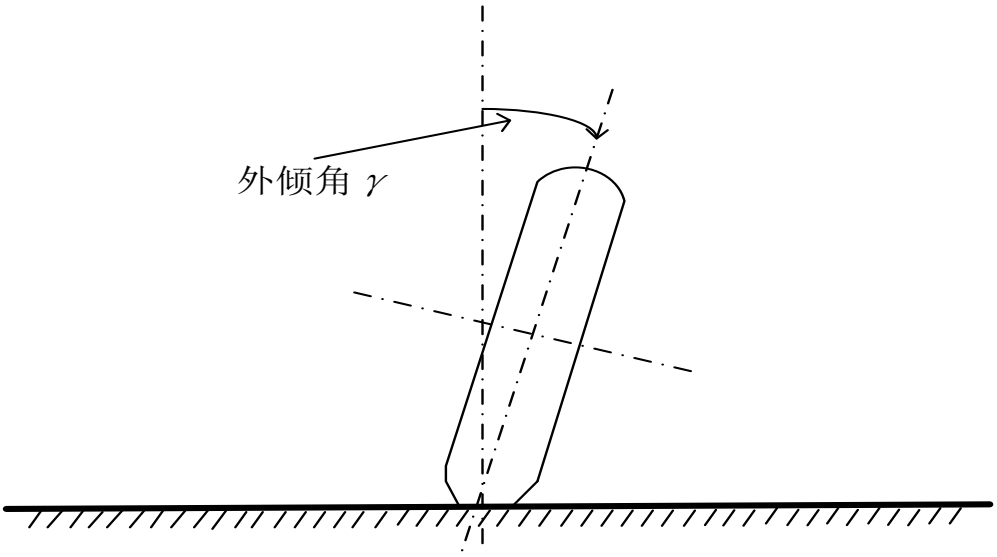


Figure 0.5 Diagram of the camber Angle05

④ Wheel vertical load F

Wheel vertical load is the weight of the vehicle and the weight of the load distributed to a single tire. Generally, the vehicle is running smoothly or stationary on the plane, and the vertical load on each wheel is the same.

⑤ Longitudinal force *F*

Longitudinal tire force refers to the force generated by the tire in the process of vehicle driving and related to the longitudinal movement of the vehicle, it mainly includes traction and braking force, traction is the force of the tire to push the vehicle forward, so that the vehicle can accelerate or maintain constant speed. When the driving force is applied to the tire, the friction between the tire and the ground will be generated, and this friction is the source of traction. Braking force refers to the resistance generated by the tire when braking, slowing down or stopping the vehicle.

$$\begin{cases} F_x = \left(D\ sin\left(C \times arctan\left(BX_1 - E(BX_1 - arctan(BX_1))\right)\right)\right) + s_v \\ X_1 = s + s_h \\ C = B_0 \\ D = B_1 F_z^2 + B_2 F_Z \\ B = (B_3 F_z^2 + B_4 F_Z) \times e^{-B_5 F_z}/(C \times D) \\ s_h = B_9 F_z + B_{10} \\ s_v = 0 \\ E = B_6 F_z^2 + B_7 F_z + B_8 \end{cases}$$

Where, is the horizontal drift of the curve and is the vertical drift of the curve. $s_h s_v$C is the shape factor of the curve, D is the peak factor of the curve, B is the stiffness factor, and E is the curvature factor of the curve.

⑥ Lateral force *F*

When the car is driving, due to the lateral tilt of the road surface, the centrifugal force when the curve is driving, etc., the wheel center generates a lateral force along the axle direction.

Because the wheel is elastic, so when the lateral force does not reach the maximum friction between the wheel and the ground, the lateral force causes the tire to deform, so that the wheel tilts, resulting in the wheel driving direction away from the predetermined driving route.

$$\begin{cases} F_y = \left(D\,sin\left(Carctan\left(BX_1 - E(BX_1 - arctan(BX_1))\right)\right)\right) + s_v \\ X_1 = \alpha + s_h \\ C = A_0 \\ D = A_1 F_z^2 + A_2 F_z \\ B = A_3\,sin\left(2\,arctan\frac{F_z}{A_4}\right) \times (1 - A_5|\gamma|)/(C \times D) \\ s_h = A_9 F_z + A_{10} + A_8\gamma \\ s_v = A_{11} F_z \gamma + A_{12} F_z + A_{13} \\ E = A_6 F_z + A_7 \end{cases}$$

⑦ Return to the positive moment *M*

The reverting torque is the torque that acts on the tire around the OZ axis when the tire is sidetracked. In circular driving, the reverting torque is one of the main torques that bring the wheel back to a straight driving position.

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

⑧ The reversing torque M

The overturning torque is the torque exerted on the tire around the OX *shaft* when the tire is sidetracked.

$$M_x = -F_z \cdot D_e$$

Where, is the lateral deformation and $D_e = F_y/L_s L_s$ is the lateral stiffness of the tire, which is often assumed to be a constant in the tire model.

⑨ Rolling resistance moment *M*

The torque opposite to the rolling direction of the tire is the tire rolling resistance.

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

Here, R_e is the rolling radius of the tire; R is_p the rolling resistance coefficient.

Table 2 Magic Formula parameter value table2

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) Force and torque model for the full vehicle

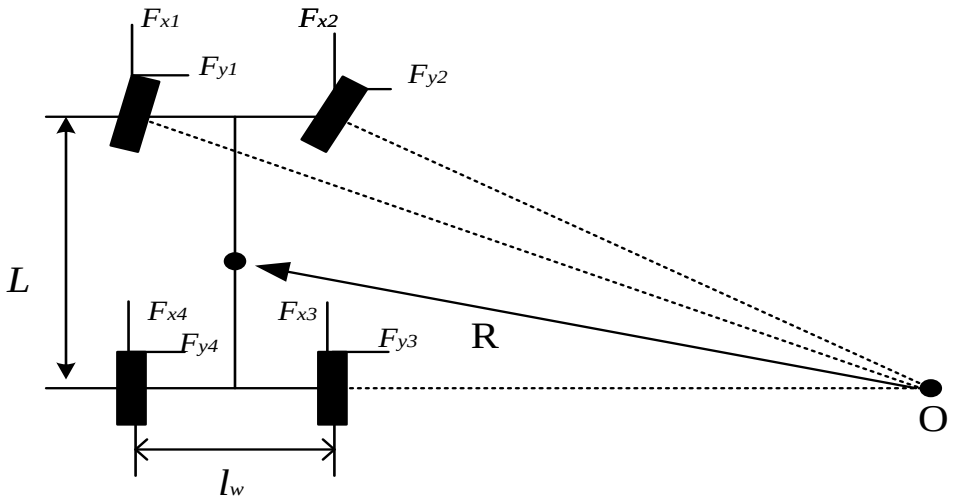


Figure 0.6 Force diagram of the vehicle06

The force and torque model of the vehicle can be expressed as follows:

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

Where F_{YX} *and* F are the longitudinal force and lateral force of the vehicle respectively,M_X,M_Y,M_Z are the torques of the vehicle in three directions; F_{xi}(i=1,2,3,4), F_{yi}(i=1,2,3,4) are the longitudinal and lateral forces of each tire, which have been obtained by the magic formula of the tire model.

7.3.1.2 Modeling and simulation case

3.CustExps\e4_TrailerModelCtrl\Readme.pdf

7.3.2 Akaman Chassis unmanned vehicle

7.3.2.1 Modeling and simulation case

1) Akaman chassis unmanned vehicle model based on minimum input and output interface

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

2) Position control for unmanned vehicles on Akaman chassis

*:\PX4PSP\RflySimAPIs\4.RflySimModel\2.AdvExps\e5_CarAckermanCtrl\1.CarAckerman

PosCtrl_Py\Readme.pdf

3) Akaman chassis unmanned vehicle speed control

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

7.3.3 Differential autonomous vehicles

7.3.3.1 Modeling and simulation case

1) Differential unmanned vehicle model based on minimum input and output interface

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

For the rest: 2.AdvExps\e6_CarR1DiffCtrl\Readme.pdf

7.4 VTOL UAV

7.4.1 4+1 droop

7.4.1.1 Modeling and simulation case

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

7.4.2 Quadrotor tail-mount draping

7.4.2.1 Modeling and simulation case

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

7.5 Unmanned Vessel

Replenishment required

7.6 Helicopters

7.6.1.1 Modeling and simulation case

2.AdvExps\e8_Helicopter\Readme.pdf

7.7 Unmanned Underwater craft

7.7.1.1 Modeling and simulation case

2.AdvExps\e9_UUV\Readme.pdf

8. External control interface

8.1 QGC

QGC (QGroundControl) external control interfaces are mainly as follows:

MAVLink message interface: through the MAVLink message protocol, it communicates with the flight control to realize the control and monitoring of the aircraft.

UDP command interface: Send and receive control instructions through UDP protocol to realize the control of the aircraft.

TCP command interface: send and receive control instructions through TCP protocol to realize the control of the aircraft.

WebSocket interface: data exchange and communication through WebSocket protocol to realize the control and monitoring of the aircraft.

8.2 Simulink control interface

Simulink provides a variety of ways to implement the external control interface of the UAV. The commonly used methods are:

Simulink Coder is used to generate the Simulink model into C code, and then the C code is used to interact with the external control program.

UDP or TCP/IP protocol is used to communicate between Simulink and external control program.

Use Simulink Real-Time Workshop and hardware connection boards for real-time control.

8.3 Python Control interface

Python external control interfaces generally control the speed, position, attitude and other states of the vehicle. The following common interfaces are:

Fixed wing takeoff control interface: `"sendMavTakeOff"`, controls the fixed wing to take off at a `specified location`.

```
      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)
```

Unlocking interface: `"SendMavArm"`, vehicle unlocking `command`.

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

Target position control interface: `"SendPosNED" to` send the target position as well as the yaw Angle in the North East ground coordinate system.

```
    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
```

Fixed wing cruise radius interface: `"SendCruiseRadius"`, changes the cruise radius of the

fixed wing.

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

Vehicle attitude sending interface: `"SendAttPX4"`, send the target attitude in the right front and lower coordinate system.

```
    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
```

Fixed wing cruise speed interface: `"SendCruiseSpeed"`, changes the cruise speed of the fixed wing.

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

Target position control interface: `"SendPosNEDNoYaw"`, send the target position in the North east ground coordinate system.

```
     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
```

Ground speed control interface: `"SendGroundSpeed"`, controls the fixed wing ground speed.

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

Target speed control interface: `"SendVelNEDNoYaw"`, send command to change the ground speed (m/s) in the north east ground coordinate system.

```
    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
```

Target speed control interface: `"SendVelNED"`, sends the target speed as well as yaw angular velocity in the North East ground coordinate system.

```
      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
```

Target speed control interface: `"SendVelFRD"`, sends the target speed as well as yaw angular speed in the right front and bottom coordinate system.

```
    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
```

Target speed control interface: `"SendVelNoYaw"`, sends the target speed in the right front and lower coordinate system.

```
    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
```

Target position control interface: `"SendPosFRD"`, sends the target position and yaw Angle in

the right front and bottom coordinate system.

```
     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
```

Target position control interface: `"SendPosFRDNoYaw"`, sends the target position in the right front and lower coordinate system.

```
     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
```

Maximum speed control interface: `"SendCopterSpeed", which` sets the maximum flight speed of the rotor.

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

Fixed wing landing control interface: `"sendMavLand" to` set the position where the fixed wing is expected to land.

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

9 Synthesize the model

9.1 Synthesis Model Protocol

9.1.1 Background

The controller is implemented on the basis of the original dynamic model to form a comprehensive model. The controller uses MATLAB Simulink to realize basic attitude control and fixed-point function. The controller directly takes the real state of the model as input. The key of synthesizing model protocol is to define the input and output interface. The overall interface design only considers the full mode, while the simplified mode is considered in CopterSim.

Model parameters: Contains model parameters and controller parameters. Consider adding an interface for setting controller parameters in CflightModel.

Input interface: Consider the messages that the synthesis model sends to or receives from CopterSim.

Command input: Used to control basic processes such as unlocking, taking off, and landing of the UAV. Command input goes inside InSIL.

Commands	Support or not
Unlock	Support
Take-off	Support
Landing	Support
Return flight	Support

Hover	Suitable for fixed wing
Set the current desired position	Simply support the command to complete the trajectory control

Collision input: Given by the collision detection scheme.

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

OffBoard control: The PX4 can be controlled by an independent helper computer via cable or wifi. The partner computers typically communicate via the MAVLink API. The vehicle executes the position, speed, and attitude commands set by the offboard computer through MVLink. Offboard mode is mainly used to perform complex air maneuvers, such as automatic path tracking, target tracking, etc. For missions such as takeoff, landing, and return, specialized flight modes such as AUTO.TAKEOFF, AUTO.LAND, and AUTO.RTL are often used

Rotorcraft Mavlink Offboard message

SET_POSITION_TARGET_LOCAL_NED Desired position (x, y, z), desired velocity (v_x, v_y, v_z), desired acceleration (a_{fx}, a_{fy}, a_{fz}). The desired velocity is added to the output of the position controller as the input to the velocity controller. The desired acceleration is added to the output of the velocity controller and is used to calculate the thrust vector. Supported coordinate systems include MAV_FRAME_LOCAL_NED, MAV_FRAME_BODY_NED. SET_POSITION_TARGET_GLOBAL_INT Desired position (lat_int, lon_int, alt), desired velocity (v_x, v_y, v_z), desired acceleration (a_{fx}, a_{fy}, a_{fz}). The desired velocity is added to the output of the position controller as the input to the velocity controller. The incorporation of the desired acceleration is not yet perfect. Supported coordinate system MAV_FRAME_GLOBAL. SET_ATTITUDE_TARGET 1 Attitude SET_ATTITUDE_TARGET.q+ Throttle SET_ATTITUDE_TARGET.thrust 2 Angular rate SET_ATTITUDE_TARGET body_roll_rate, body_pitch_rate, body_yaw_rate+ throttle set_attitude_target.thrust.

The above is the Offboard control message officially supported by PX4. The coordinate system and various combination modes are more complex. The messages controlled by this Offboard are mainly divided into two types. One is the position type, which is often used in formation flight. One is the attitude class, which is often used for stunts.

```
    # Set desired position, velocity, acceleration message, yaw Angle, yaw Angle rate, integer for position when using latitude and longitude high
    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)
    # Set desired pose, angular rate, throttle
    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 Synthesis model Offboard control design

The Offboard control mode supported by the integrated model.

Terminal	Supported or not
python Offboard	Support
Matlab Offboard	Support

9.1.3 Complete typing with inSIL

inSILInts protocol

The zeroth digit of the inSILInts is used to characterize and modify the state, and a corresponding bit of 1 indicates that the system is in the corresponding state. For example, the first bit indicates simulation mode, and when the first bit of the received inSILInts[0] is 1, it indicates that the system enters simulation mode.

Only when 0:hasCMD is 1, the state should be set once, otherwise the integrated model will continue to use the original state. The original state can come from the external setting value, or it can be an internal state automatic transition. For example, after receiving the take-off command, it first switches to the take-off mode, and automatically switches to the fixed-point mode after the take-off is completed.

inSILInts[0] Vehicle Command Bitmap

0:hasCMD	1: SIL	2: Armed	3:	4:	5:	6:	7:
Have new orders	Emulation	Unlock					
8:Takeoff	9: Position	10: Land	11: Return	12:Lotier	13:Height	14:Hor	15:
Take-off	Fixed /waypoint	Landing	Turning back	Hover (fixed wing)	Fixed height mode	Horizontal position control	
16:OffboardPos	17:OffboardAtt	18:	19:	20:	21:	22:	23:
Offboad Position Control series	Offboad Position Pose Series						
24:	25:	26:	27:	28:	29:	30:	31:

Note: When enabling position control, horizontal position and vertical position are enabled at the same time

Bits 0-7 of inSILInts[1] are position class flags, and bits 8-15 are posture class flags.

inSILInts[1] Offboard control flag

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

s				te			
Location	Speed	Acceleration	Yaw Angle	Yaw Angle rate			
8:hasAtt	9:hasRollRate	10:hasPitchRate	11: hasThrust	12:	13:	14:	15:
Posture	Roll Angle rate	Pitch Angle rate	Yaw angular rate	Throttle			
16:NED	17:Global	18:	19:	20:	21:	22:	23:
Position and speed NED	Position and speed Global						
24:	25:	26:	27:	28:	29:	30:	31:
Integer type latitude	Integer Type Precision						

inSILInts[6] represents the latitude of the integer type, and inSILInts[7] represents the longitude of the integer type.

Remark: The corresponding value is 1 for position control only, 2 for speed control only, 8 for yaw Angle control only, and 16 for angular rate control only. If the combination of multiple controls is required, the value of the separate control is added.

The inSILFloats protocol

inSILFloats are used to store the actual data, and the meaning can change depending on what you set up in inSILInts. The first three represent locations, but inSILInts controls which coordinate system they are in.

```
    inSILFloats[0-2] =pos;
    inSILFloats[3-5]=vel; / / speed | | pitch, roll and yaw of the remote control signal inSILFloats [3] can be used as a rate
    InSILFloats [6-8] = acc;
    inSILFloats[9-11]=att; // Attitude control uses Euler angles, which are more intuitive for the user.
    inSILFloats[12-14]=attRate;
    inSILFloats[15]=thrust; // throttle
```

9.1.4 The output interface

The integrated model output interface remains the same as the original output interface.

```
    struct outHILStateData{
        uint32_t time_boot_ms; // message timestamp ms
        uint32_t copterID;     // aircraft ID
        int32_t GpsPos[3];      // Filtered GPS latitude and longitude, where latitude and longitude (degrees *1e7), height upward positive (in m*1e3->mm)
        int32_t GpsVel[3];     // Filtered GPS speed, NE ground, (in m/s*1e2->cm/s)
        int32_t gpsHome[3];      //GPS raw data, where latitude and longitude (degrees *1e7), height upward positive (in m*1e3->mm)
```

```
        int32_t relative_alt;  // Filtered GPS relative height, NE, (in m*1e3->mm)
        int32_t hdg;           // filtered GPS heading Angle, NE ground, (in m*1e3->mm)
        int32_t satellites_visible; //GPS raw data, number of satellites
        int32_t fix_type;     //GPS raw data, positioning accuracy level, 3 means fixed
        int32_t resrveInit;       // reserve bit Int
        float AngEular[3];    // Filtered aircraft Euler Angle in rad
        float localPos[3];    // filtered local location in m
        float localVel[3];    // filtered local location in m/s
        float pos_horiz_accuracy;   // filter state, horizontal positioning error, in m
        float pos_vert_accuracy; // filter status, vertical positioning error in m
        float resrveFloat;      // reserve bit Float
    }
```

Simulink receives state structures

```
       struct outHILStateShort{
           int checksum; // checksum bit 1234567890
           int32_t gpsHome[3];      //Home GPS position, lat&long: deg*1e7, alt: m*1e3 and up
is positive
           float AngEular[3];    //Estimated Euler angle, unit: rad
           float localPos[3];     //Estimated locoal position, NED, unit: m Consider changing
it to double
           float localVel[3];    //Estimated locoal velocity, NED, unit: m/s
    }
```

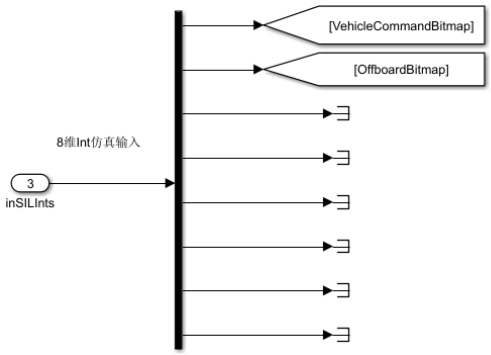
9.2.Synthesis model implementation

The synthesis model is implemented using MATLAB automatic code generation DLL model, which encapsulates the necessary interfaces for CopterSim to load and call.

9.2.1 Implementation of rotorcraft synthesis model

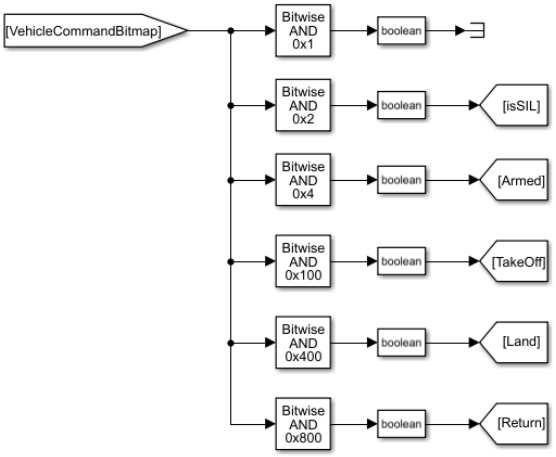
9.2.1.1 Protocol analysis

Protocol parsing is the parsing of network packets received by CopterSim into the instructions of Section 3.2. inSILInts is an 8-dimensional input that currently only uses the zeroth number instruction and the first number Offboard mode. Subsequent 6th and 7th numbers will be used as integer representations of latitude and longitude in the Global coordinate system. In the figure below, the inSILInts vector is broken down into eight individual numbers.

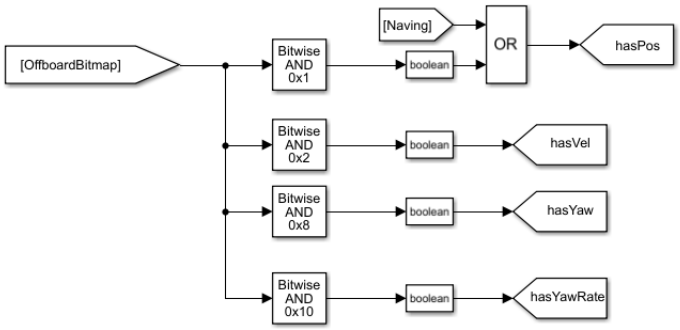


Each bit of the insilints has a meaning, so we need to parse each bit further. The following figure uses the Bitwise module to parse the bit.Bit 0 is not used yet. The first bit identifies whether

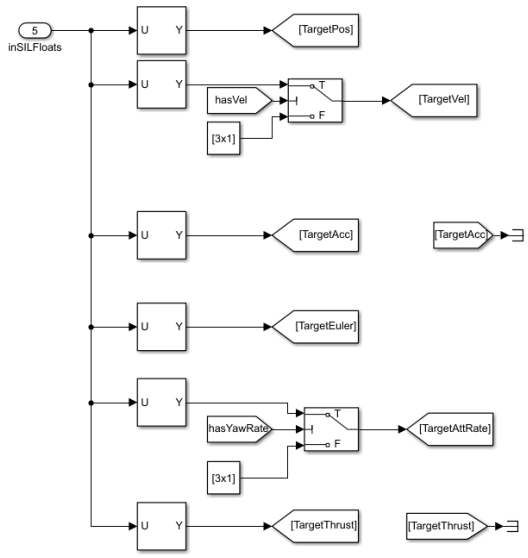
the simulation mode is used. When the bit is 1, it means that the simulation mode is used. When the bit is 0, it means that the hardware is in the loop mode. The second bit indicates whether to unlock, other bits also include the take-off, landing, and return functions, the specific meaning of which is referred to Section 3.2.2.



The following is a sign that indicates which offboard control information is available: The protocol in Section 3.2.2 supports full PX4 offboard control, which supports the most urgent requirements for the project at hand, including position, speed, yaw, and yaw Angle rate.

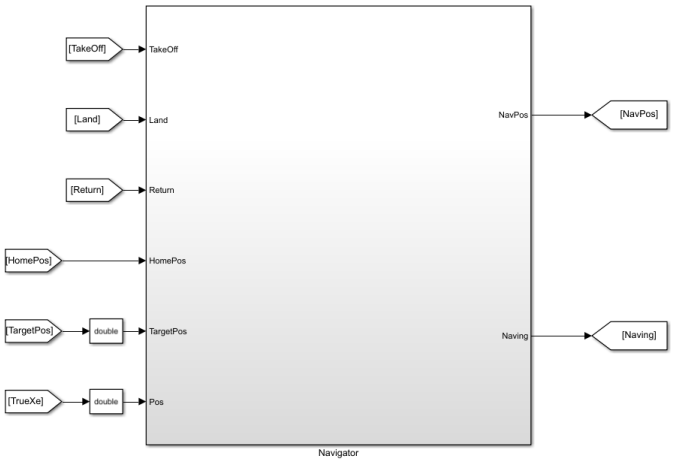


Further, parse the actual position, velocity values, etc. sent by the offboard as shown in the figure below. In the top image, it is used to indicate whether the corresponding value exists or not, while the bottom image shows the specific value. Currently only position, velocity, yaw, and yaw Angle rate are used. In the parse of the figure below, the hasVel flag and the hasYawRate flag are used. When these two flags are false, it means that there is no input for speed and yaw Angle rate, at which point it will switch to remote control mode. In remote control mode, 1500 represents a desired speed or desired yaw Angle rate of 0.



9.2.1.2 Navigator

In order for the integrated model to be able to receive upper level commands, such as take-off, landing, return and other functions, these upper level commands need to be converted into a series of desired positions. This work is done by the Navigator module. The Navigator first needs to receive these commands, and then it needs to rely on the current location information and the desired location information set by the user to get the desired location. For example, the user can set the altitude for take-off, return, and landing. In addition, the user only needs to send the take-off command once by design, so the Navigator needs to determine whether the take-off is complete by itself, so it needs to know both the desired position and the actual position.

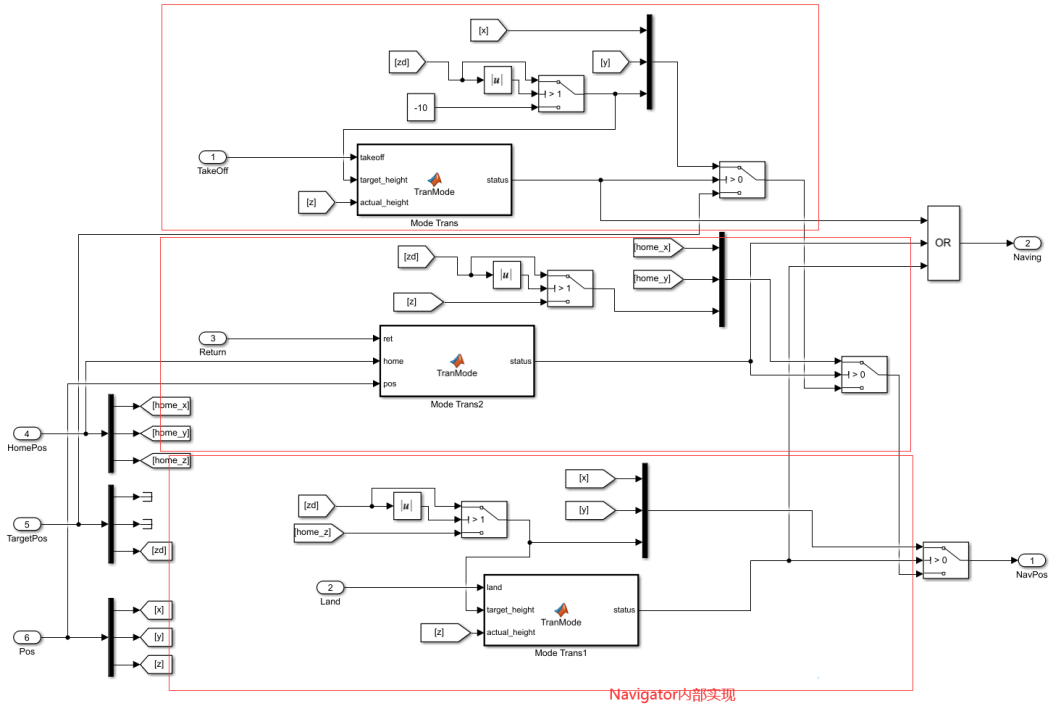


Below is the internal implementation of Navigator. At the top is the take-off module, which has the lowest priority. That is, the priority relation of receiving takeoff, return and landing at the same time is: takeoff < return < landing. When the user does not specify the height, its default height is -10m under NED coordinates. Consider that by design, after a successful take-off command is received, the Navigator will continue to generate the desired position until the

specified altitude is taken off. So the Mode Trans module is designed. The function of the module is to switch to the takeoff mode when the takeoff command is received, and it will switch to the position mode when the aircraft reaches the specified altitude. In the process of takeoff, the aircraft will not respond to the position set by the user.

On the return flight, the aircraft will return to the home point position from its current position with the same altitude. The return altitude can also be set. When no altitude is specified, the altitude at the previous moment is taken as the desired altitude, that is, altitude maintenance. Because by default, when the height is not set, the height value is 0. Considering that the GPS positioning accuracy is generally not more than 1m, 1m is used as the criterion for whether the height is set. The home point is the initial position recorded during the first run to the location acquisition when the model is running.

Landing is the highest priority command, and you can also specify the height when you land. This is taking into account that landing does not necessarily land at the home point position. Other external modules can detect the current distance from the ground to determine the landing altitude, so a setting altitude interface is reserved for landing.

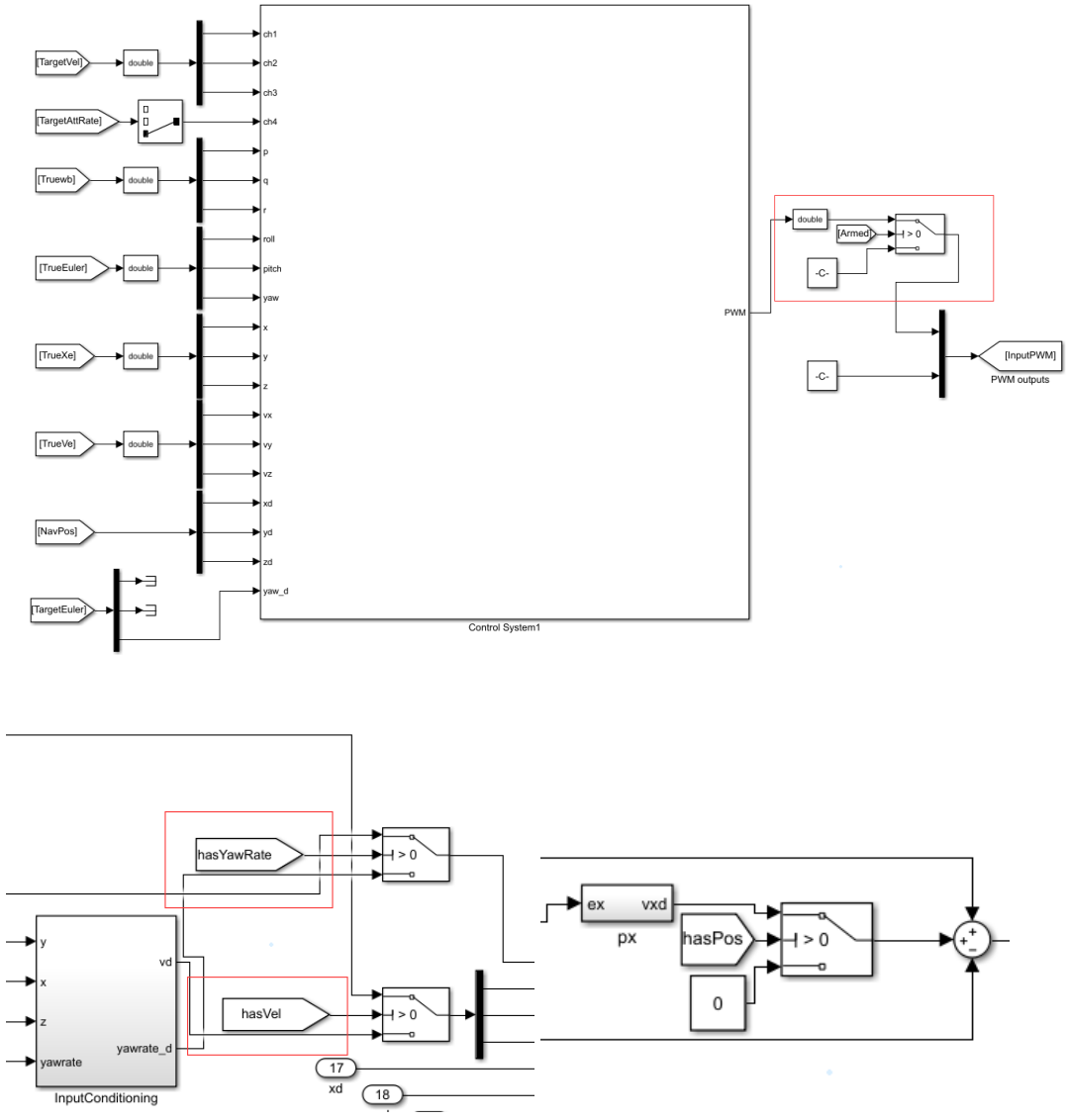


The output of the Navigator is NavPos, which refers to the desired position information. Naving means that the Navigator is in the process of executing an instruction, and when Naving is True, it sets hasPos to true.

9.2.1.3 Controllers

The rotorcraft integrated model is a four-stage PID controller, including a position loop, a

velocity loop, an attitude loop and an angular velocity loop. Currently, the controller supports the control of position, velocity, yaw Angle and yaw Angle rate. When no position control is performed, the hasPos flag is false and the output of the position controller is 0, which can be observed in the figure below. When no speed signal or yaw angular velocity signal is received, the remote control mode will be entered. At present, only the remote control interface is reserved, the value of the remote control is set to 1500, and the remote control is not really enabled.

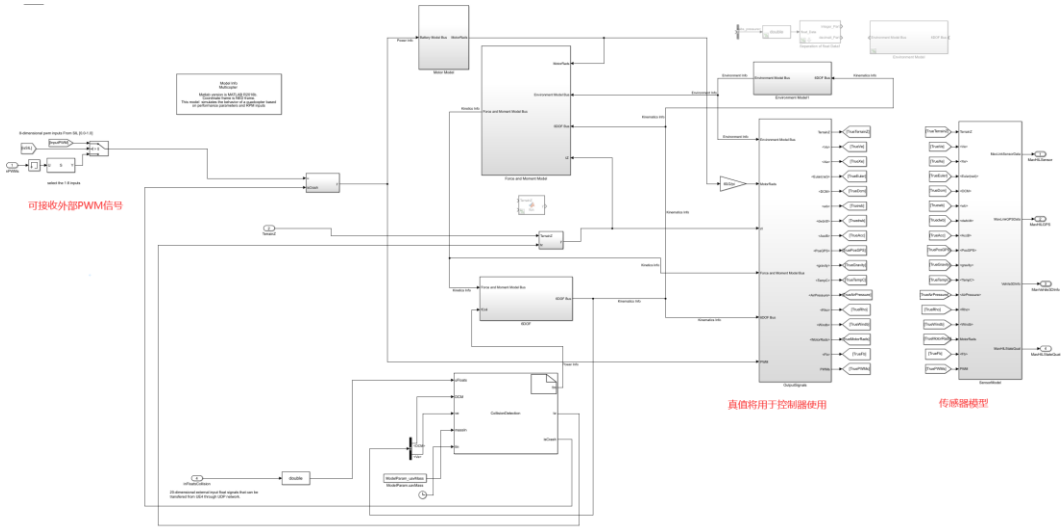


9.2.1.4 Models

Shown below is the model of the quadrotor. The model can receive input from the internal controller as well as input from the external controller. These two modes are controlled through inSIL, and when this flag is true, it indicates the simulation mode and the PWM wave pulse width is obtained from the internal controller. When the flag is false, the PWM wave will be obtained

from the outside.

Main modules included in the model: motor model, 6DOF dynamics model, 6DOF kinematics model, environment model, collision model, sensor model. For the internal control, because no filter is designed, the true value is used directly. However, the sensor data output to CopterSim is noisy, and the influence of the environment model will be reflected in the sensor.



9.2.2 Implementation of fixed-wing integrated model

9.2.2.1 Protocol analysis

Fixed wings and rotors follow mutually compatible protocols, but the fixed wing mode is different from the rotor because the fixed wing cannot hover like the quadrotor. Shown below is a breakdown of the protocol within the fixed-wing integrated model. Position, Height, Hor are added, and the behavior of take-off, return, and landing is also different from that of rotorcraft.

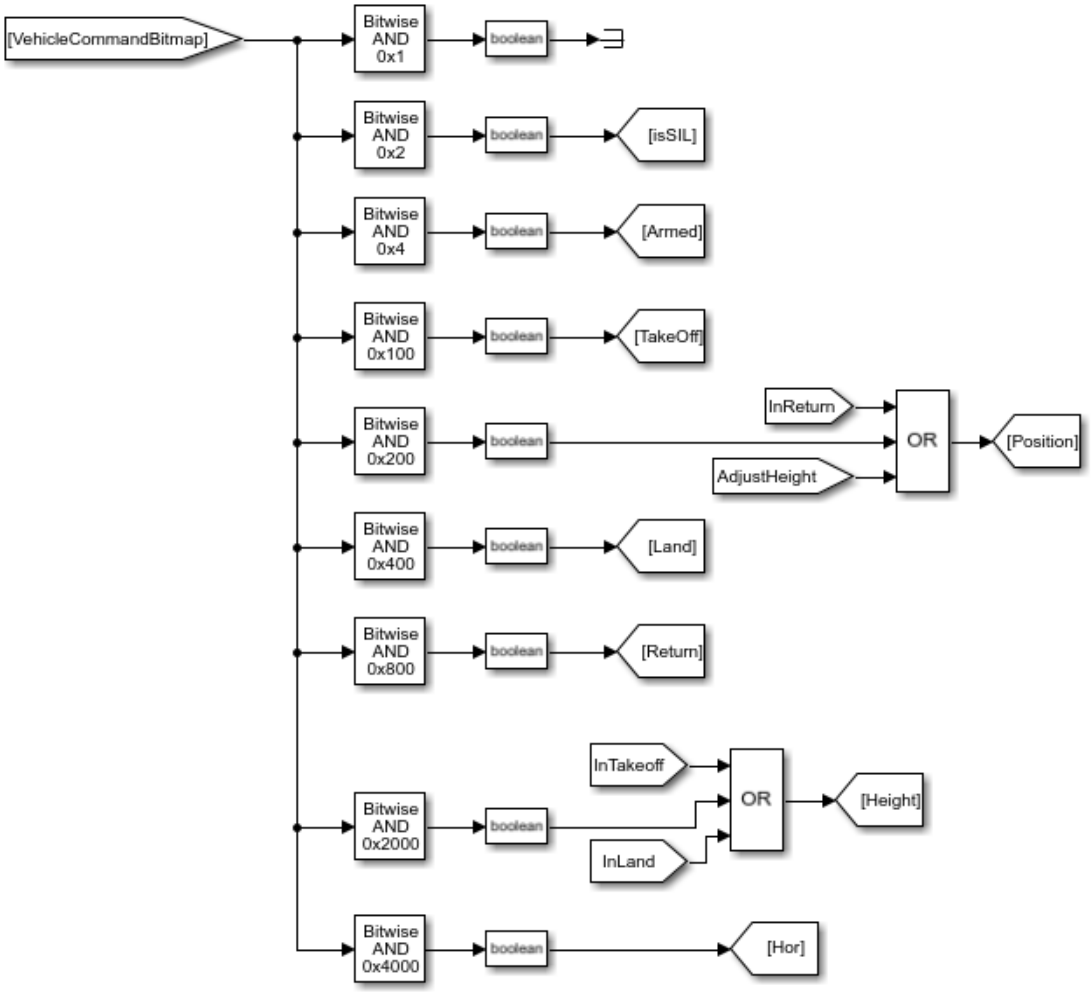
Take off, control altitude and speed. Take off at a fixed pitch Angle when the fixed wing receives the command to take off, default 15°. The user can set the take-off height, and when the take-off height is reached, the model will make a judgment internally and exit the take-off mode. When the take-off is successful, if no further operation instructions are received, it will continue to fly forward and no adjustments of throttle, pitch and roll will be made.

Position, while controlling horizontal position and altitude. The user can specify the position control mode directly through the inSILInts, and the system can also automatically trigger the position mode when returning or landing. In the position mode, if the corresponding position is set, fly to the corresponding position first. After reaching the designated position, if the next position is not specified, it will automatically hover. In the return mode, essentially the horizontal Position is back to the starting point and supports the specified return height, so this function can be implemented using the Position mode. On landing, the altitude of the aircraft is first adjusted, as shown in the AdjustHeight diagram. The adjustment of the altitude of the aircraft is completed by

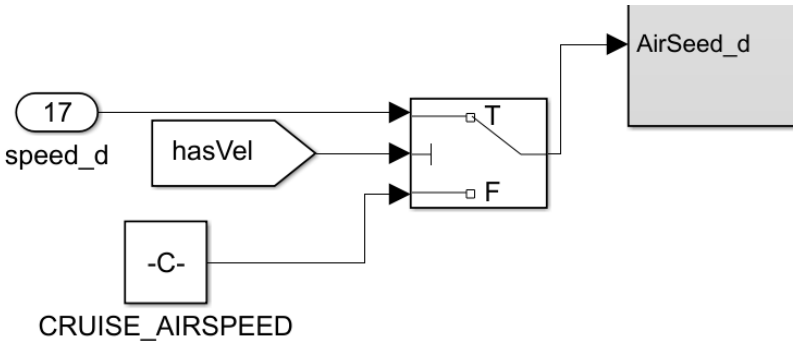
circling, and the horizontal position will also change during circling, so the position mode is also used for control.

Height mode controls the altitude and airspeed, but does not control the horizontal position, that is, the aircraft will only fly forward. Height mode is used during take-off and at the end of the landing phase (when the aircraft has reached its altitude).

Hor mode is to control the horizontal position alone. This mode is supported by current models, but is generally less used.



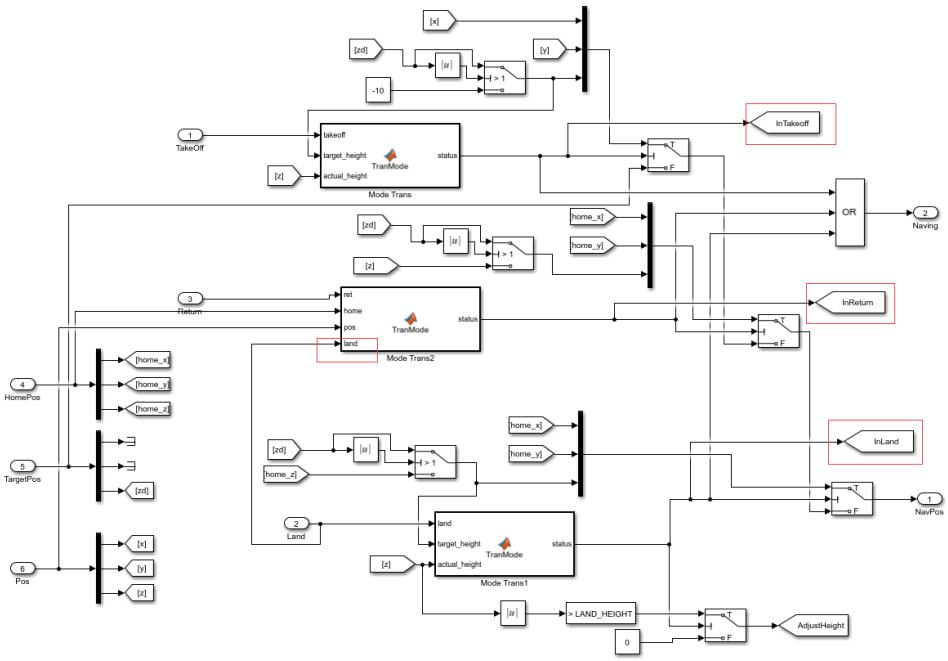
Unlike rotorcraft, fixed wings do not have free control of speed in all directions. It is possible to set the speed in the horizontal direction. In the full protocol, the desired speed has 3 components and only use the first component as the rate in the horizontal direction in the fixed wing.



9.2.2.2 Navigator

The implementation of the fixed-wing Navigator is similar to that of the rotorcraft, but with the addition of the InTakeoff, InReturn, InLand, AdjustHeight flags. Because the fixed wing has different operations during take-off, return, and landing, these signs are used to make a distinction. During take-off, the output of TECS controller pitch is masked and set directly to a fixed Angle of 15°. While during the return flight, the complete position control needs to be triggered automatically. For landing, there are two phases: an altitude adjustment phase, marked by AdjustHeight, which triggers full position control; The other is the landing phase, which only controls the altitude. Because the ground friction is not modeled, the speed of the landing is not controlled for the time being.

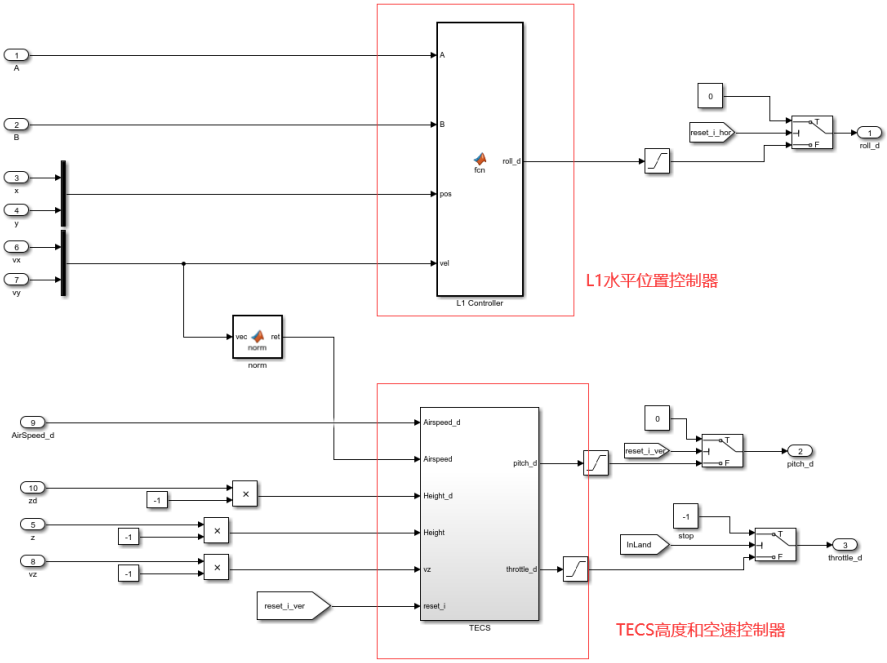
In addition, after the return mode is executed, only the landing mode can be executed. In the following mode switch, you will exit from the return mode only if you receive the landing command.



9.2.2.3 Controller

The position controller for a fixed wing is very different from the position controller for a rotorcraft because the fixed wing needs to maintain a certain speed once it takes off.

In the following figure, the L1 controller is used to control the horizontal position. The internal implementation of the controller will not be explained here, only focusing on the input and output of the controller. The L1 controller needs to calculate the desired roll Angle based on the current desired horizontal position, the last desired horizontal position, the current horizontal position, and the current horizontal speed jointly. It can be seen that the L1 controller has many inputs, but only one roll Angle as its output. The L1 controller does not operate for the entire flight, so using switch simplifies the internal implementation of the L1 controller.



The TECS controller is used for altitude and speed control. For the fixed wing, only the airspeed is controlled, and the speed in all directions cannot be arbitrarily controlled. TECS provides a solution by reflecting the problem in terms of energy rather than the initial set point. The total energy of a vehicle is the sum of the kinetic and potential energy of the vehicle. Thrust (controlled through the throttle) can add to the total energy of the aircraft. A given total energy state can be achieved by any combination of potential and kinetic energy. In other words, the total energy of the vehicle flying at a low airspeed at a high altitude is equivalent to flying at a high airspeed at a low altitude. We call this situation the specific energy balance, which is calculated based on the current altitude and the true airspeed set point. The specific energy balance of the vehicle can be controlled by controlling the pitch Angle. An increase in the pitch Angle converts

kinetic energy to potential energy, and a decrease in the pitch Angle does the opposite. In this way, the control problem is decoupled by converting the initial airspeed and altitude set points into energy values (airspeed and altitude are coupled, but energy values can be controlled independently). We use the throttle to adjust the specific total energy of the vehicle, and the pitch Angle to maintain a specific balance between potential energy (altitude) and kinetic energy (true airspeed).

The TECS controller inputs the desired rate and altitude values, feeds in the current height and rate values, and finally outputs the desired pitch Angle and throttle values.

10 Reference materials

[1]. Translated by Quan Quan, Du Guangxun, Zhao Shiyao, Dai Xunhua, Ren Jinrui, Deng Heng. Design and Control of Multi-rotor Aircraft [M], Publishing House of Electronics Industry,2018.

[2]. Quan Quan, DAI Xunhua, WANG Shuai. Design and Control Practice of Multi-rotor Aircraft [M], Publishing House of Electronics Industry,2020.