# Outline for Retrieving API Documentation Files

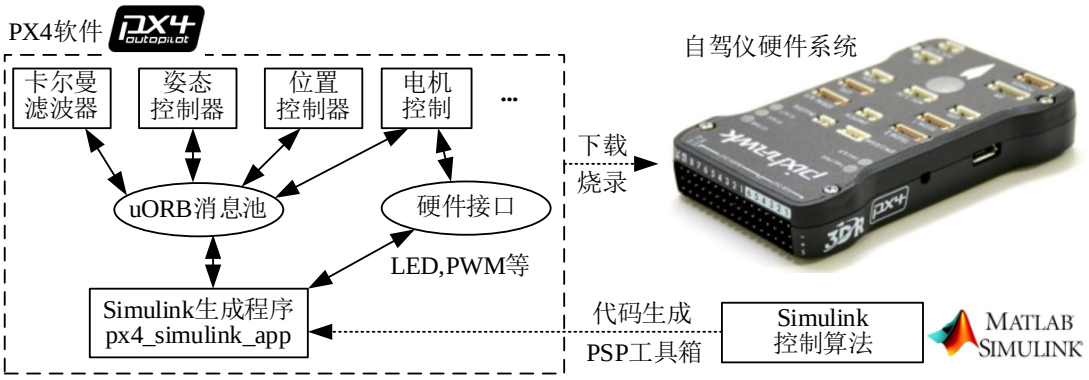
# 1. RflySim Automatic Code Generation Simulink Configuration Interface

## 1.1. The Basic Architecture of RflySim's Automatic Code Generation System

The Pixhawk Pilot Support Package (PSP), also known as “Autopilot Support Package” in Chinese, is an official toolbox released by MathWorks for Pixhawk. This toolbox allows the automatic compilation and deployment of Simulink models containing autopilot algorithms to Pixhawk hardware systems using the Embedded Coder in Simulink. Its main functionalities include:

- Capable of simulating and testing various aircraft models and autopilot algorithms within Simulink, with automatic deployment of algorithms to the autopilot.
- The toolbox provides practical examples including light control, remote controller data processing, and attitude controllers.
- Numerous interface modules are provided in the toolbox for accessing both the software and hardware components of the autopilot.
- Automatic logging of flight data from sensors, actuators, and deployed controllers.
- Ability to subscribe to and publish uORB topic messages. All data within the PX4 autopilot software is temporarily stored in a uORB message pool. Subscribing to uORB allows reading topics of interest from this pool, while publishing to uORB allows specific topics to be published into the pool for use by other modules.

The PX4 software system can be divided into several small modules, each running independently (in parallel threads). These modules transmit and interact with data through the subscription and publication functions of the uORB messaging module. After deploying the code generated by Simulink to the PX4 autopilot software, it does not affect the operation of the native PX4 autopilot software. Instead, it adds an independent module called "px4_simulink_app" (running in a separate thread) parallel to the other modules. Since the native PX4 control algorithms may need to access the same hardware output resources as "px4_simulink_app," this can lead to read-write conflicts. Therefore, the platform's one-click deployment script provides an option to automatically block the PX4 native firmware from accessing the actuators, ensuring that only the "px4_simulink_app" module can output motor controls, as shown in the diagram below.




The PSP toolbox generates C code from control algorithms designed in Simulink. This code is then imported into the source code of the PX4 software system to create a standalone program called "px4_simulink_app" that runs independently. The toolbox invokes the compilation tool to compile all the code into a PX4 software firmware file with the extension ".px4". This firmware file is downloaded and flashed into the flight controller, enabling the flight controller to execute the PX4 software with the generated algorithm code.

## 1.2. Internal Communication in the PX4 Software System

PX4 consists of two main components: the flight control stack, which primarily includes state estimation and flight control systems, and the middleware, a universal robotic application layer capable of supporting any type of autonomous robot. The middleware is responsible for internal/external communication and hardware integration.

All supported PX4 drone models, including other platforms like unmanned boats, cars, underwater vehicles, etc., share a common codebase. The entire system employs a reactive design, meaning:

- All functionalities can be divided into several replaceable and reusable components.
- Communication is done through asynchronous message passing.
- The system can handle different workloads.

  The PX4 system achieves inter-module communication through a publish-subscribe message bus called uORB. Utilizing this publish-subscribe message bus implies:
- The system is reactive; it operates asynchronously, updating immediately upon the arrival of new data.
- All activities and communications within the system are fully parallelized.
- System components can access data from anywhere while ensuring thread safety.

uORB (Micro Object Request Broker, Micro Object Request Broker) is a crucial module in the PX4/Pixhawk system, tasked with the entire system's data transmission. All sensor data, GPS, PPM signals, etc., are obtained from the chip and transmitted through uORB to various modules for computation and processing. In reality, uORB is a set of inter-process IPC communication modules.

In Pixhawk, all functionalities are independently implemented and operate as process modules. The inter-process data exchange is crucial and must meet the real-time and ordered characteristics.

Internally, the flight controller utilizes the NuttX real-time ARM system. For NuttX, uORB is just a regular file device object that supports Open, Close, Read, Write, Ioctl, and Poll mechanisms. Through the implementation of these interfaces, uORB provides a "point-to-multipoint" inter-process broadcast communication mechanism. Here, "point" refers to the communication message's "source," and "multipoint" means a source can have multiple users to receive and process data. The relationship between the "source" and "user" is such that the source does not need to consider whether a user can receive a broadcast message or when they receive it. It simply pushes the data to the uORB message "bus." For the user, the number of times the source pushes messages is not important; what matters is retrieving the latest message. In the communication process, the sender is only responsible for sending data without caring about who receives it or whether the receiver can receive all the data. Similarly, the receiver does not care who sent the data or whether all data was received during the process.

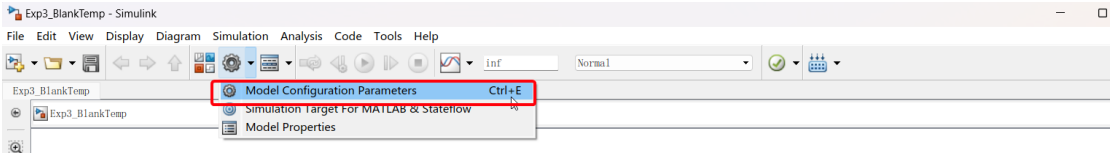
uORB does not guarantee that all sender data can be received by the receiver during data publishing and reception; it only ensures that the receiver can receive the latest data when desired. The separation of sending and receiving allows modules during flight to operate independently without interference. In practice, a uORB can be published by multiple senders and received by multiple receivers, forming a many-to-many relationship. Publishers update and publish data to the uORB platform at a certain frequency without concern for who is receiving. Subscribers can retrieve data at any time. For further learning resources, please refer to: [https://docs.px4.io/main/zh/middleware/uorb.html](https://docs.px4.io/main/zh/middleware/uorb.html)


## 1.3. Simulink Configuration Interface

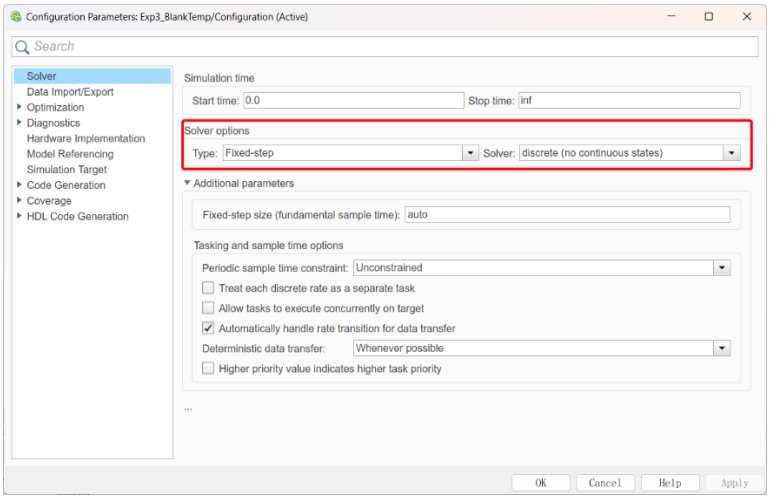
The Embedded Coder module of MATLAB/Simulink can generate readable, compact, and fast C and C++ code for use with embedded processors in large-scale production. It extends the capabilities of MATLAB Coder and Simulink Coder, providing precise control over generated functions, files, and data through advanced optimizations. These optimizations enhance code efficiency and f

acilitate integration with existing code, data types, and calibration parameters. Third-party develop ment tools can be integrated to build executable files for deployment on embedded systems or rapi d prototype boards.

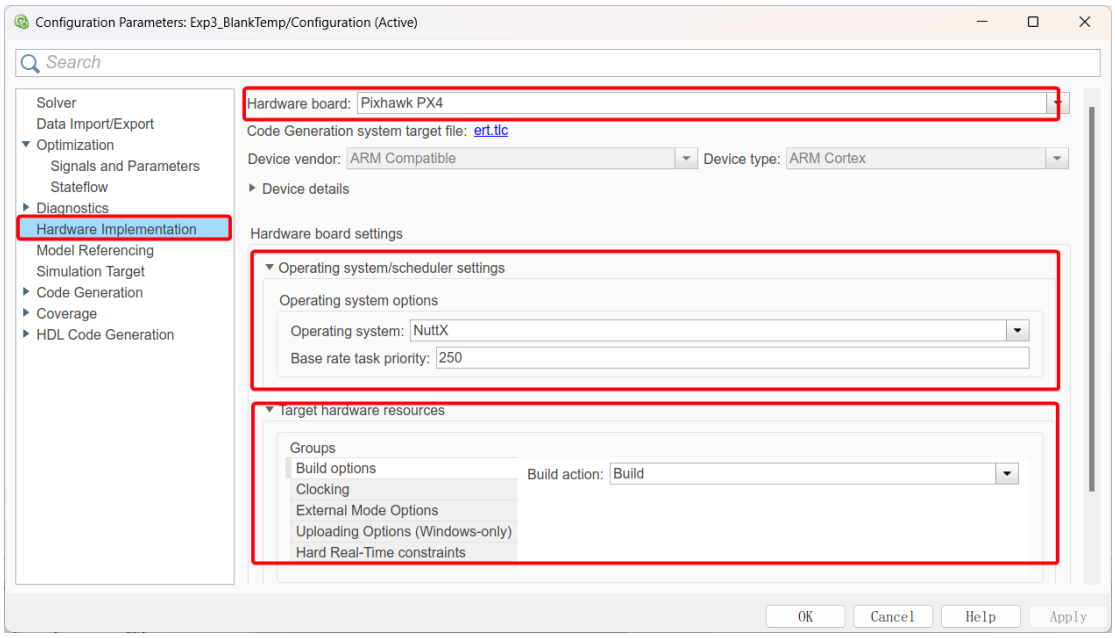
To generate code for Pixhawk series flight controller hardware after completing the model in Simulink, follow these settings in the Model Configuration Parameters:



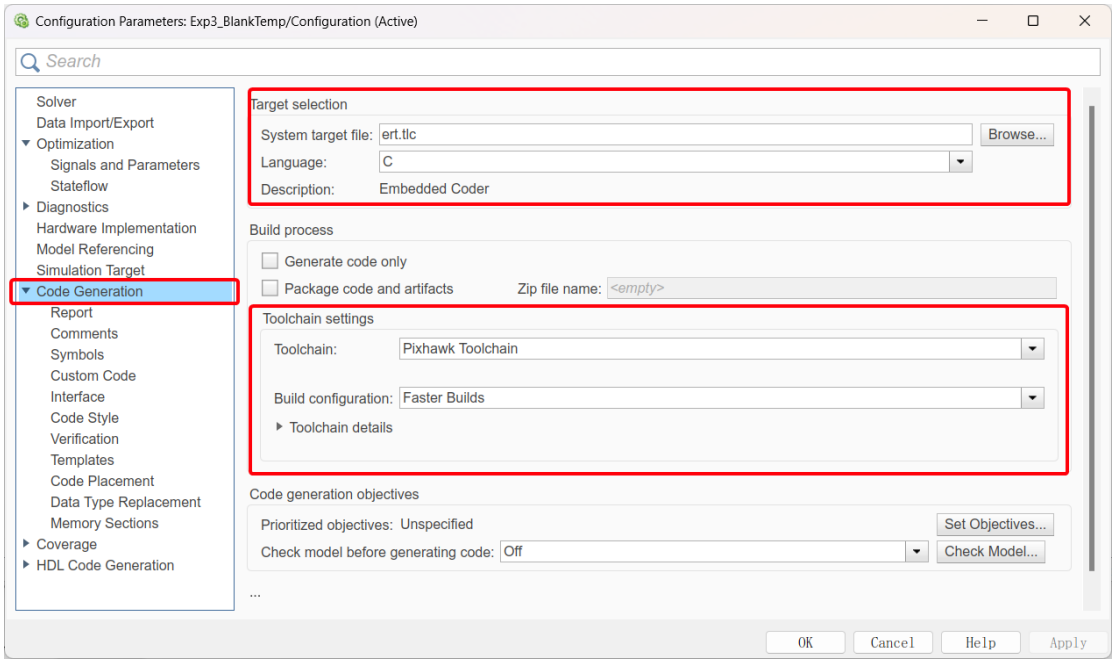

1) Set the solver type to: Fixed-step, Solver selection: discrete. That is, select a discrete fix ed-step solver.



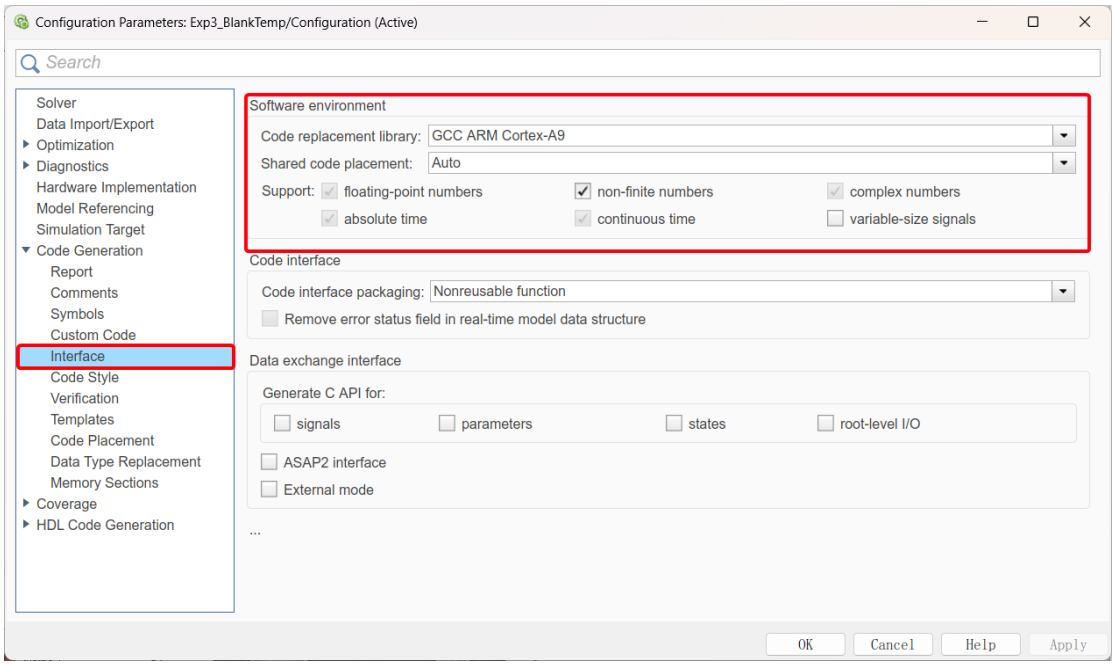

2) Go to Hardware Implementation, set the Hardware board to Pixhawk PX4, choose the o perating system to be Nuttx, which is supported by the PX4 software system. Set the clo ck frequency to 250, and for Target hardware resource, select "build only."



3) Under Code Generation, set the System target file to ert.tlc from Embedded Coder, and c
hoose the C language. Select the Pixhawk Toolchain as the toolchain, and configure it fo
r Faster Builds.




4) In the Code Generation interface settings, set the code's dependency library to GCC AR
M Cortex-A9, and configure the support settings as shown in the figure below.




For more options for automatic code generation settings, please refer to the document： [0.Ap
iExps\3.DesignExps\Readme.pdf]

## 1.4. Code Modification Interface

The one-click installation script for the RflySim platform makes some modifications to the official PX4 source code for better compatibility with the platform. The core modifications include:

1) In *\PX4PSP\Firmware\boards, locate the corresponding compilation command cmake file, such as px4_fmu-v6x_default in *\PX4PSP\Firmware\boards\px4\fmu-v6x\default.cmake, and add the module compilation registration for px4_simulink_app.

2) In *\PX4PSP\Firmware\ROMFS\px4fmu_common\init.d\rcS startup script, add the automatic startup for px4_simulink_app.

3) In *\PX4PSP\Firmware\src\modules directory, create a px4_simulink_app folder and generate source code that meets the compilation requirements.

4) Modify the source code under Firmware to disable PX4's own motor output, allowing px4_simulink_app to gain control over the motors.

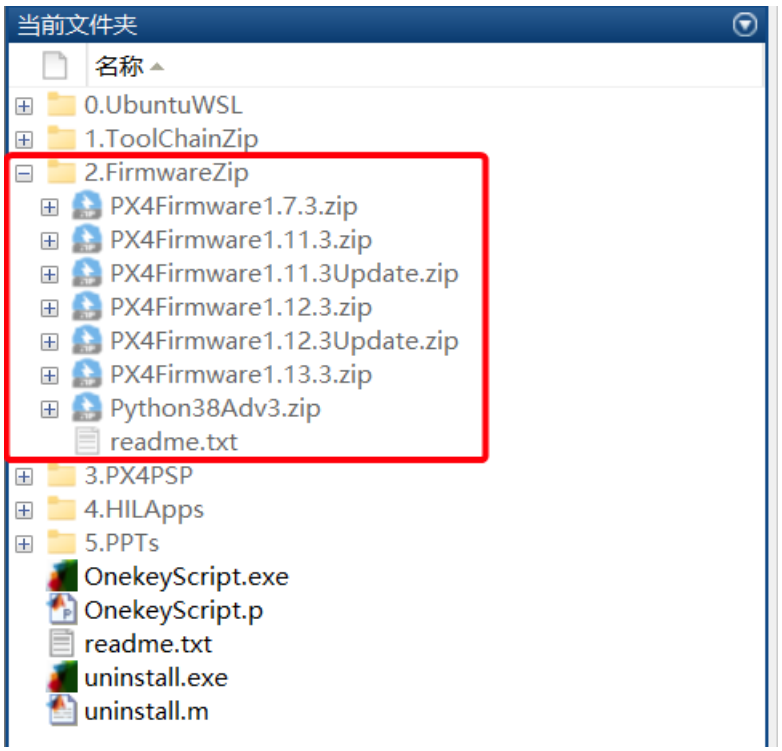
## 1.5. Custom Source Code Import Interface

The 2.FirmwareZip directory in the RflySim platform's installation package contains various PX4 source code firmware, and it supports importing custom-developed PX4 source code. When redeploying firmware using the one-click installation script, it first deletes the *\PX4PSP\Firmware folder. Then, based on the selected option, it unpacks "2.FirmwareZip\PX4Firmware***.zip" to the *\PX4PSP directory. Finally, it extracts the contents of PX4Firmware***Update.zip and forcefully overrides them into the Firmware directory.

In PX4Firmware***.zip, the official source code is stored, downloaded from GitHub without any modifications. PX4Firmware***Update.zip contains the files we modified, which will overwrite the Firmware directory. Therefore, there are two ways to deploy custom source code:

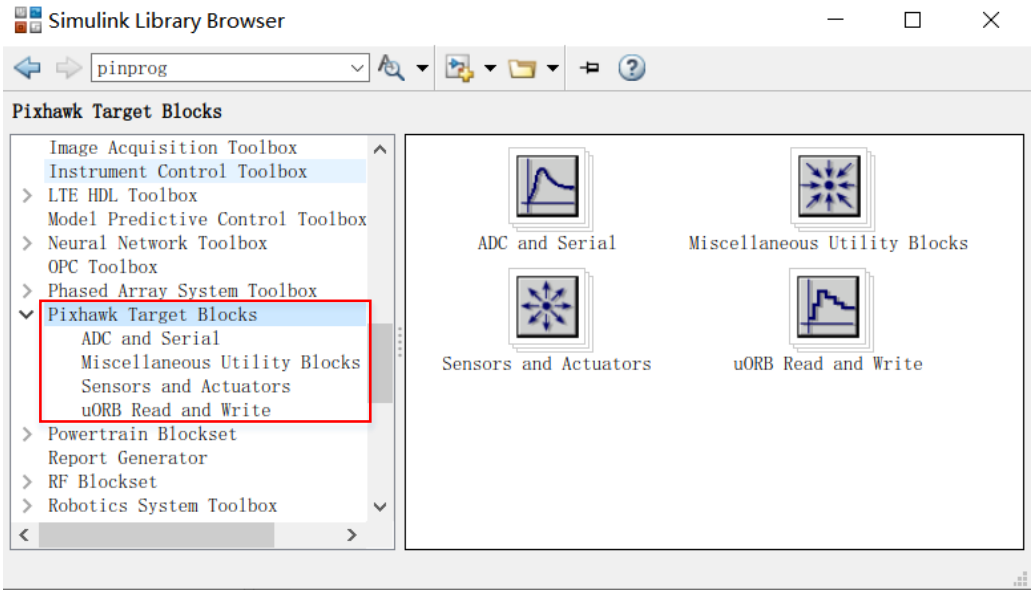
1) Directly package the modified Firmware directory, rename it according to your version as PX4Firmware*****.zip (see naming rules in 2.FirmwareZip\readme.txt), and delete the PX4Firmware****Update.zip file. This way, the one-click installation script will use your own script for deployment.
2) Alternatively, you can directly place your modified source code parts, structured by File Contents, inside PX4Firmware***Update.zip. During deployment, they will be copied in and forcibly replace the original code.

The RflySim platform also supports other PX4 firmware versions, such as 1.9.2, 1.10.2, etc., as mentioned in the 2.FirmwareZip\readme.txt file, as shown in the following figure.



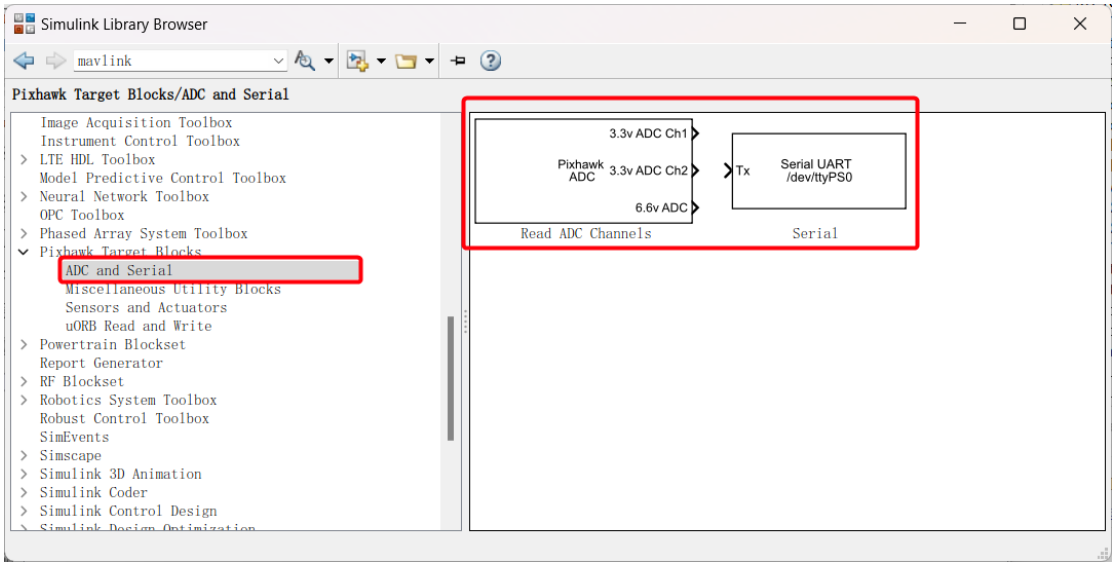

## 2. Simulink/PSP Toolbox Module Interface

PSP3.04 provides some Simulink modules as the hardware interface to Pixhawk. These modules are only responsible for generating the corresponding interface code and do not include the modeling of the peripheral hardware. It is best to organize your own control system into a Simulink module at the time of simulation, leaving the necessary interfaces in order to connect with these hardware interface modules, so that it is also easy to reuse. These hardware modules can be viewed in Simulink's toolbox Pixhawk Target Blocks, as shown in the figure below, which consists of four sublibraries: ADC and Serial, Miscellaneous Utility Blocks, Sensors and Actuators, and uORB Read and Write.



## 2.1. ADC and Serial — ADC and Serial Communication Library

As shown in the figure below, the ADC read module can obtain the data of 3 external ADC ch annels, and the serial port module can read and write the specified serial port.



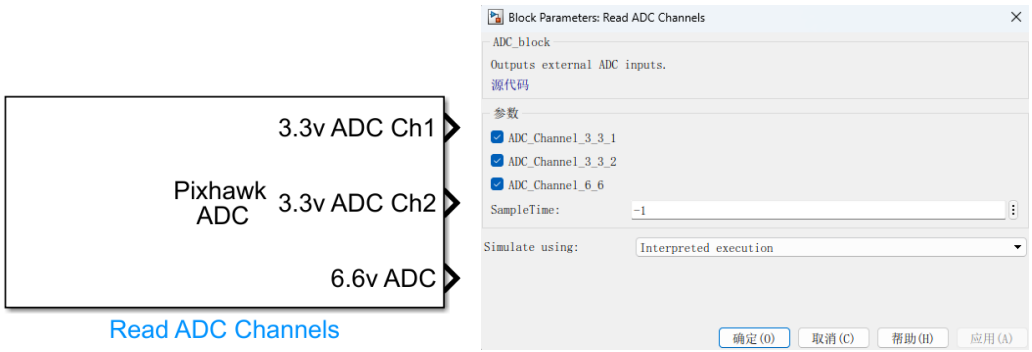

### 2.1.1. Read ADC Channels—Outputting the input of an external ADC

Read ADC Channels: Select which ADC channel from the 3.3V and 6.6V analog inputs to be the output of this module, as shown in the figure below. Note that channels 1 and 2 correspond to p ins 14 and 15 of the 3.3V ADC, respectively.

Signal definition:

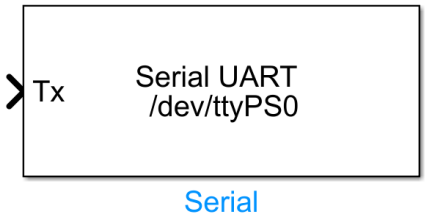
- 3.3V analog-to-digital conversion (int32)
- 3.3V Analog-to-Digital conversion (int32)
- 6.6V Analog-to-Digital (int32)

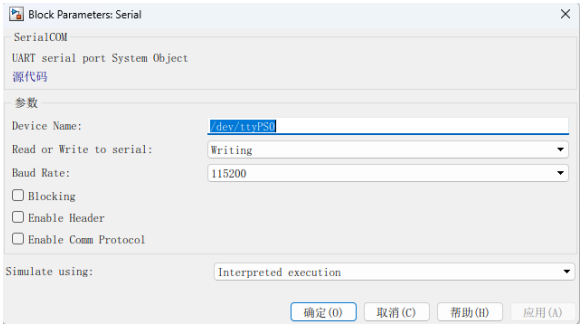
More information is available by clicking the "Help" button in the module dialog box.



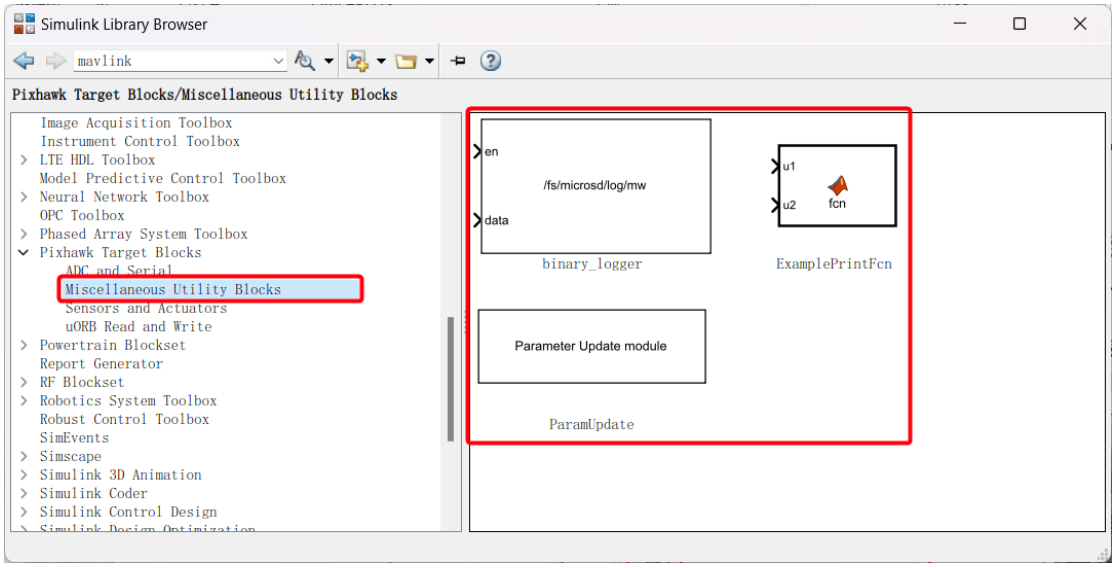

### 2.1.2. Serial—Serial Communication Module

Serial: As shown in the image below, serial communication block that allows reading and writ ing devices, specifying device name and read or write options. The specific usage can be checked by clicking the "Help" button of the module dialog box.



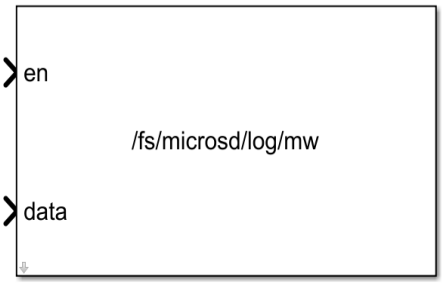




## 2.2. Miscellaneous Utility Blocks—Other libraries

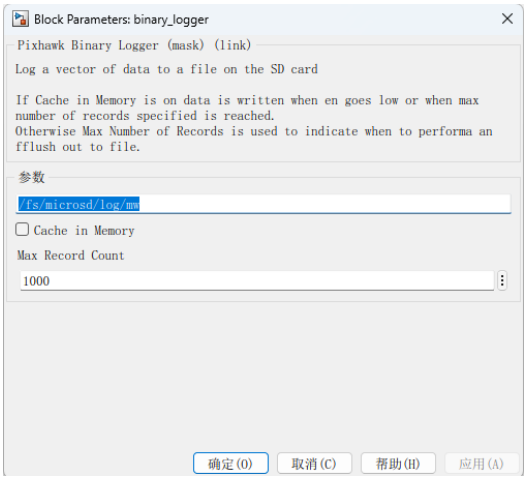



### 2.2.1. binary_logger—Data Logging Module

This module records the data as a double and stores it in the SD card, and if the cache is turned on in memory, writes the data when en gets low or reaches the specified maximum number of records. The first enable input signal during program execution must first trigger the high level and then drop to the low level in order to record the data successfully. If the low level is not dropped before the code finishes running, then the record file is considered to be open and inaccessible. In addition, the log must be stored under the directory "/fs/microsd/". As shown in the following figure, you can set the path for storing logs and the maximum number of records. The specific usage can be clicked on the "Help" button of the module dialog box to view, and the specific routine experiment can be seen in the file: 0.ApiExps\5.Log-Write-Read\Readme.pdf

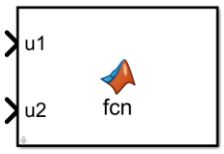



binary_logger




## 2.2.2. ExamplePrintFcn—Print Function Example

As shown below, this module prints the signal data content to the PX4 Nuttx console terminal. This block should be considered an "example" block, and the print message can be constructed in any way the user sees fit. Coder.ceval() is a MATLABCoder function that evaluates printf() statem ents to pass two values. Note that there is a bug in Nuttx that sometimes floating-point numbers do not display correctly. Use warnx() instead of printf() as a workaround.

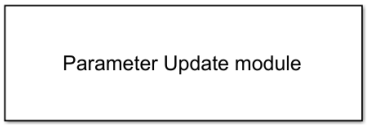



ExamplePrintFcn

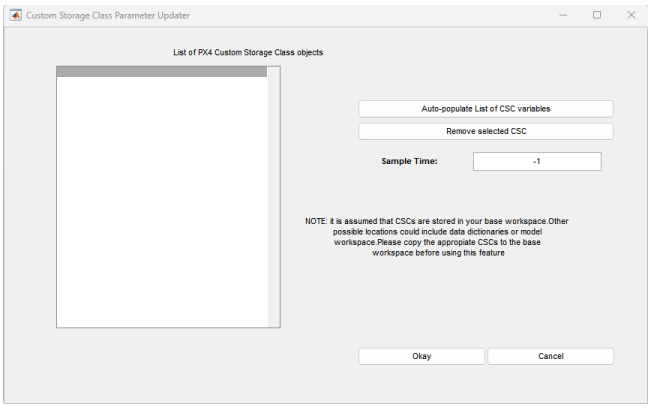
```
function fcn(u1, u2)
%#codegen

if strcmp(coder.target, 'rtw') == true
    coder.ceval('printf','%d %d %c', int32(u1), int32(u2), int8(10))
end
```

## 2.2.3. ParamUpdate—Custom Storage Class Parameter Update Module

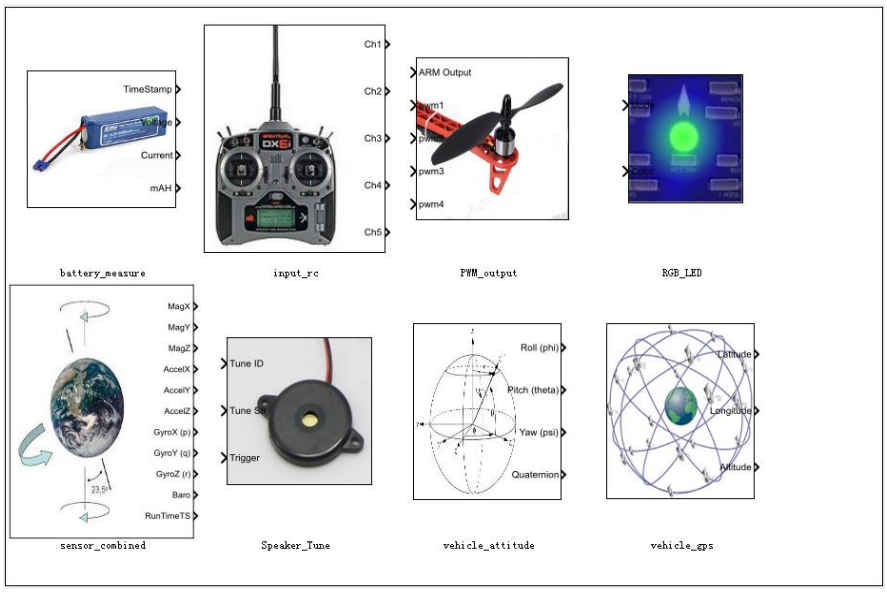
As shown in the image below, this module updates custom PX4 software parameters. Note wh en using: Assume that the csc is stored in your base workspace, other possible locations include a d ata dictionary or a model workspace. You will need to copy the appropriate csc into your base wor kspace before using this feature.




ParamUpdate



## 2.3. Sensors and Actuators—Sensor and Actuator Interface Library



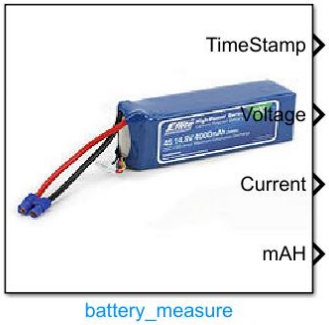

### 2.3.1. Battery_measure—Battery Data Module

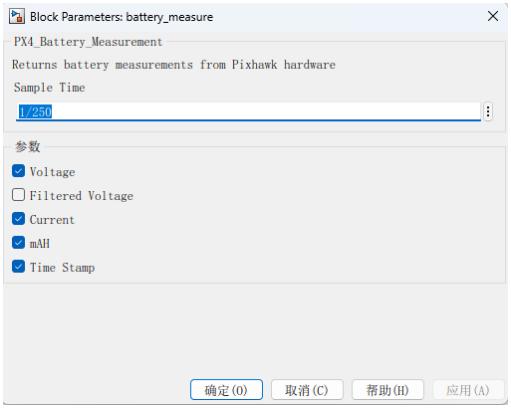
This module allows you to obtain the battery health status by subscribing to messages posted by the uORB topic publisher battery_status, so you need to ensure that the power module is plugge d into Pixhawk in order to obtain the correct data.

The signal definition:

- Voltage: (double) The battery voltage, in volts.
- Filtered Voltage: (single) Filtered voltage of the battery, in volts.
- Current: (single) Current of the battery, in amperes.
- mAH: (single) The amount of discharge in mAH.
- Timestamp: (int32) Timestamp of the measurement.

As shown in the image below, the module provides a setting box for the sample rate, as well a s selectable battery status options. More information can be viewed by clicking the "Help" button o f the dialog box.



### 2.3.2. Input_rc—Remote Control Input Module

This module allows the user to access the signal from the RC transmitter. Through this module, the output signal can be selected, including the value of multiple remote control channels, and some other information. As shown in the following picture, these include:

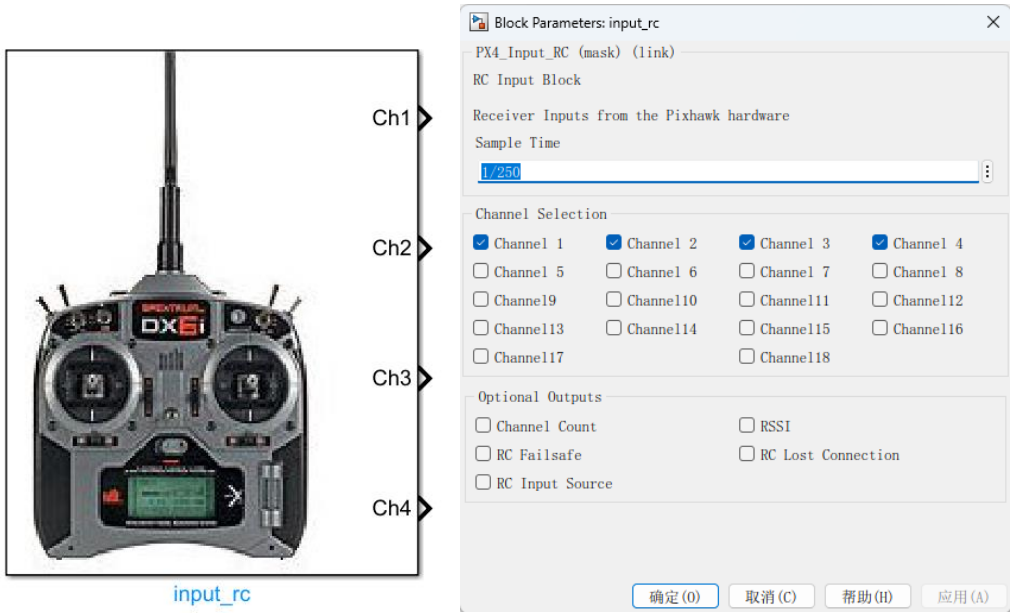
1. Channel Selection - Channel selection
   a) uint16 data type, which represents the PWM(in use) value from the controller.
   b) Measure the pulse width of each support channel.
2. Channel Count - Number of channels
   a) Uint32 Indicates the number of channels detected by the PX4 detector.
3. RC Failsafe - Remote control signal failsafe
   a) Boolean data type, indicating that RC Tx is sending FailSafe signal (if set correctly)
   b) Displays the failsafe flag: true if Tx fails or if Tx is out of range, false otherwise.
   c) Only the true state is reliable, as there are some (PPM) receivers on the market that go into failsafe without explicitly telling us.
4. RC Input Source - Remote control signal input source
   a) Enumerate the data type, indicating which source the RC input comes from.
   b) Find valid values in the ENUM file:
   RC_INPUT_SOURCE_ENUM.m

```
RCINPUT_SOURCE_UNKNOWN         (0)
RCINPUT_SOURCE_PX4FMU_PPM      (1)
RCINPUT_SOURCE_PX4IO_PPM       (2)
RCINPUT_SOURCE_PX4IO_SPEKTRUM  (3)
RCINPUT_SOURCE_PX4IO_SBUS      (4)
```

5. RSSI - received signal strength indicator
   a) Received Signal Strength Indicator (RSSI) : <0: undefined; 0: No signal; 255: full reception.
6. RC Lost Connection - Remote control signal lost connection
   a) Boolean data type indicating RC receiver connection status.
   b) True if no frame arrived in the expected time, false otherwise.
   c) True usually means that the receiver is disconnected, but can also mean that the radio link is lost on a "dumb" system.
   d) If an RX with the failsafe option continues to transmit frames after the link is lost, it remains false.

More information can be viewed by clicking the "Help" button in the dialog box or by viewing the official PDF document. See the file for specific routine experiment: 0.ApiExps\2.PSPOfficial

Exps\Readme.pdf




### 2.3.3. PWM_output—Motor PWM Module

Through this module, PWM signal can be sent to the output port of PX4IO to control the mot or rotation, you can select the PWM update rate and input channel.

In order for the flight control arm (enabled) to output from the software side, the ARM output input must be kept high (Boolean TRUE). Only then will the PWM value be sent to the PX4 hard ware port. This is usually a feature of the RC Tx combined with other flight modes programmed in the Simulink model by the user.

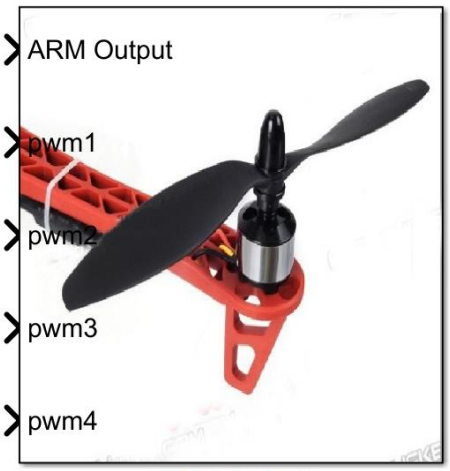
The block has 8 available ports (data type uint16) that can be selected selectively. These corre spond to the 8 PWM output ports on the px4fmu hardware.

The unit value of PWM is microsecond (usec), which corresponds to the pulse width (1500 is 1500usec or 1.5 ms).

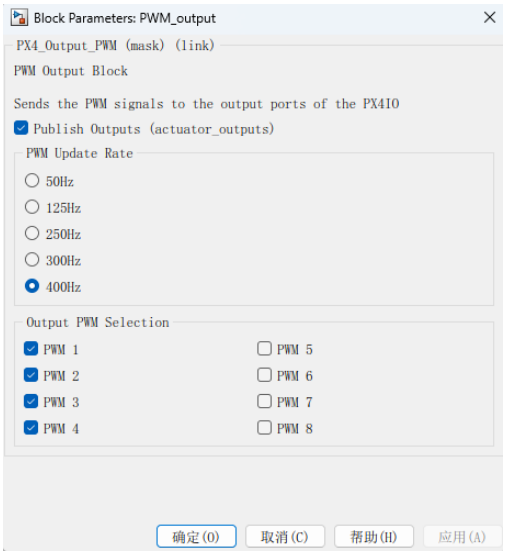
The PWM update rate is set to 400Hz when the px4simulinkapp is started (or whatever the us er sets in the block dialog). The available PWM update rates are :50, 125, 250, 300, and 400Hz.

px4_simulink_app will also set an idle value of 900usec at startup and when the ARM port is set to low (Boolean FALSE) so that the ESC controller does not time out.

As shown below, for more information click the "Help" button of the dialog box to see.




PWM_output




### 2.3.4. RGB_LED— LED Light

The mode and color of the LED light flashing can be controlled through this module. As show n below, the module receives two inputs, one is Mode, the other is Color, and these two inputs are enumerated data types. You can find a valid value in the MATLAB command window by entering the following command:

>>:enumeration ("RGBLED COLOR ENUM")

Enumeration members of the 'RGBLEDCOLORENUM' class:

COLOR_OFF

COLOR_RED

COLOR_YELLOW

COLOR_PURPLE

COLOR_GREEN

COLOR_BLUE

COLOR_WHITE

COLOR_AMBER

COLOR_DIM_RED

COLOR_DIM_YELLOW

COLOR_DIM_PURPLE

COLOR_DIM_GREEN

COLOR_DIM_BLUE

COLOR_DIM_WHITE

COLOR_DIM_AMBER

Enumeration member of the 'RGBLED_MODE_ENUM' class:

MODE_OFF

MODE_ON

MODE_BLINK_SLOW

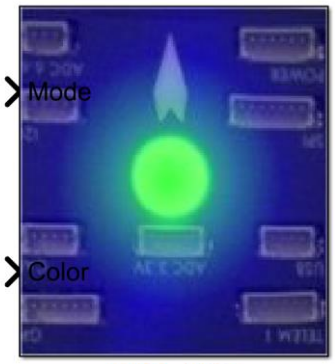
MODE_BLINK_NORMAL

MODE_BLINK_FAST

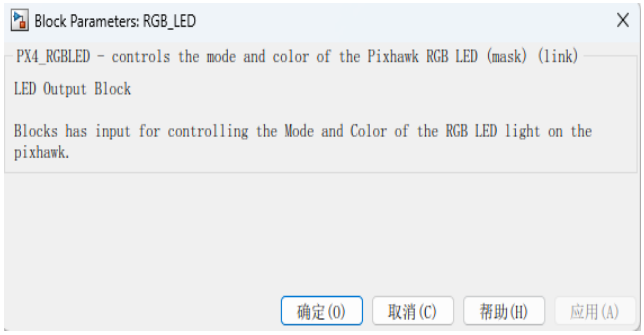
MODE_BREATHE

MODE_PATTERN

More information can be viewed by clicking on the "Help" button of the dialog box. See file f or specific routine experiment: 0.ApiExps\2.PSPOfficialExps\Readme.pdf




RGB_LED




### 2.3.5. sensor_combined—Sensor Fusion Module

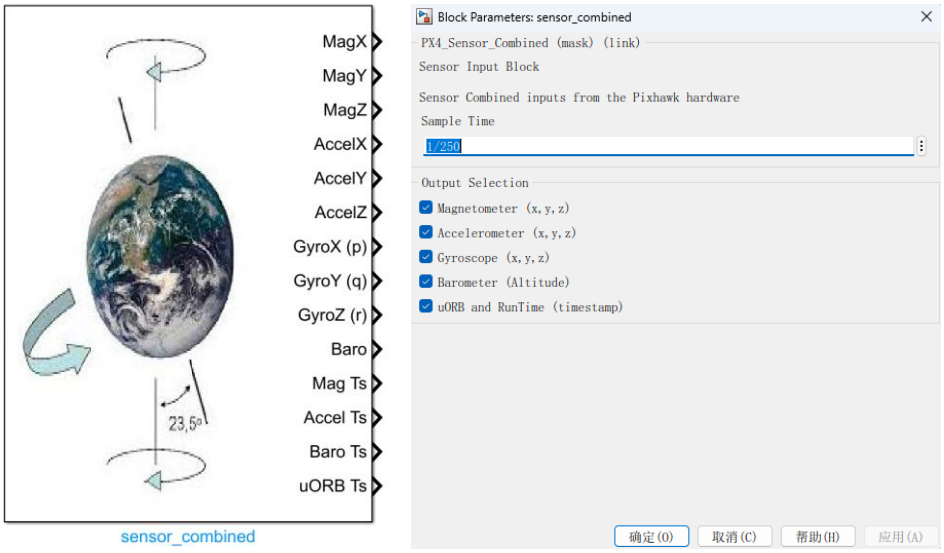
It is through this module that the sensor data available in Pixhawk can be obtained, which can then be used to control the design of the model. The data available includes magnetometers, accel erometers, gyroscopes, barometers, and time stamps.

Signal definition:

- Magnetometer(x,y,z):(single) Magnetometer(x,y,z), magnetic field under NED, Gauss un it.
- Accelerometer(x,y,z): (single) accelerometer (x,y,z), the frame of acceleration under NE D, in m/s^2.
- Gyroscope(x,y,z): (single) gyroscope (p,q,r), angular velocity in radians per second.
- Barometer(Altitude): (single) barometer (altitude), air pressure with temperature compen sated (mbar).
- uORB and RunTime(timestamp):(double) Time stamp in microseconds since the gyrosc ope was activated.

The sensor_combined block needs to run the px4io service on the PX4 hardware in order to g et a valid signal value.

As shown below, more information can be viewed by clicking the "Help" button of the dialog box. See file for specific routine experiment: 0.ApiExps\11.SenorDataGet\Readme.pdf




### 2.3.6. Speaker_Tune—Buzzer Module

Through this module, you can control the buzzer to emit a specific tone at a specific event. As shown below, this module accepts 3 input signals.

- Tune ID: This is the enumerated data type of predefined "tunes" that can be played. Enumeration members of class 'PX4 TUNE ENUM' :

  STOP TUNE

  STARTUP TUNE

  ERROR TUNE

  NOTIFY_POSITIVE_TUNE

  NOTIFY_NEUTRAL_TUNE

  NOTIFY_NEGATIVE_TUNE

  ARMING_WARNING_TUNE

  BATTERY_WARNING_SLOW_TUNE

  BATTERY_WARNING_FAST_TUNE

  GPS_WARNING_TUNE

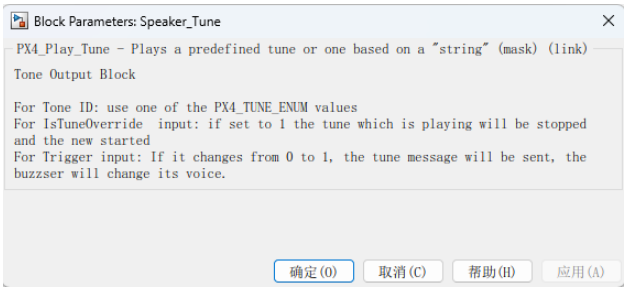
  ARMING_FAILURE_TUNE

  PARACHUTE_RELEASE_TUNE

- IsTuneOverride: Used to define a custom tune to play. The user can use a constant block to define the string to be played.
- Trigger: This is a trigger signal that indicates when a predefined tune (if the trigger value changes to 1) or a custom tune (if the trigger value changes to 2) will be played. A change in the tune will only be triggered by a change in that value.

More information is available by clicking on the "Help" button of the dialog box.


Speaker_Tune




### 2.3.7. vehicle attitude—Attitude Data Module

This module provides filtered attitude data (Euler Angle and quaternion) and provides access to a running service that calculates the attitude of the UAV. A uORB theme (vehicle attitude(Attitude Measurement)) publisher must be run in order for the block to provide valid signal values.

One of these must be run on px4fmu in order for the block to return a valid value. For example:

px4fmu-v2 ekf2: ekf Extended Kalman filter for attitude estimation

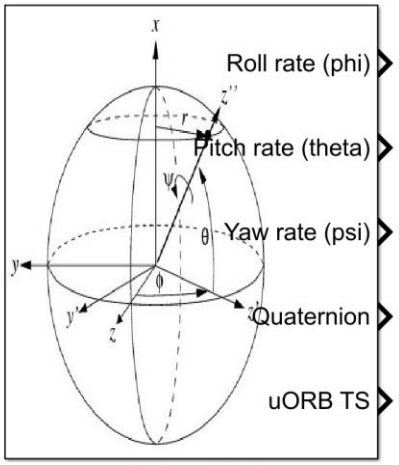
px4fmu-v2 default: SO(3)- Attitude estimation using accelerometers, gyroscopes and magnetometers

As shown in the image below, the module provides a setting box for the sampling rate, as well as selectable attitude options.

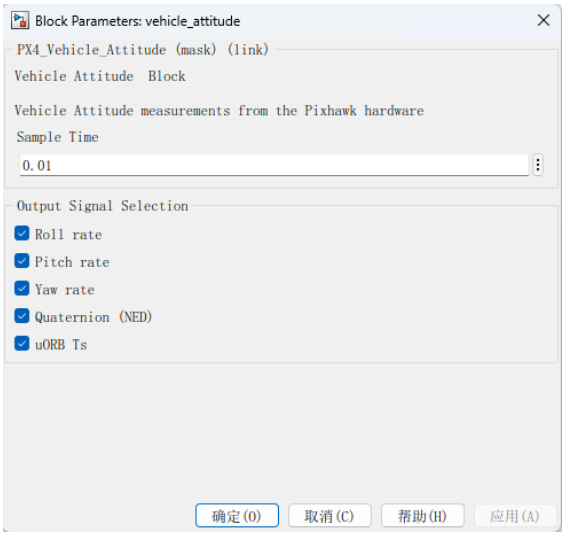
Signal definition:

- Roll rate:(single) Roll rate in degrees per second or radians per second (NED).
- Pitch rate: (single) Pitch rate in degrees per second or radians per second (NED).
- Yaw rate: (single) yaw rate in degrees/second or radians/second (NED).
- Quaternion(NED): (single) Optional according to uORB publisher (NED).
- uORB Ts: Used for efficient data exchange and time synchronization.

For more information, click the "Help" button in the dialog box.




vehicle_attitude

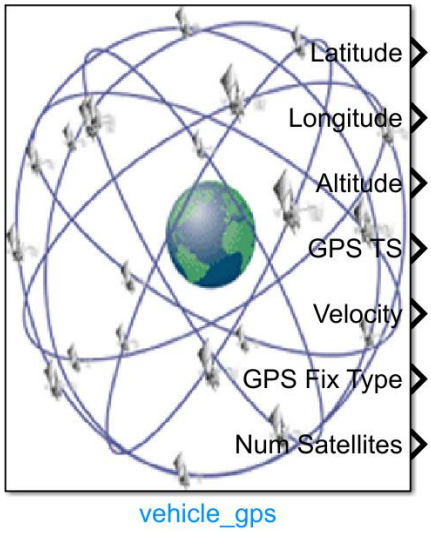


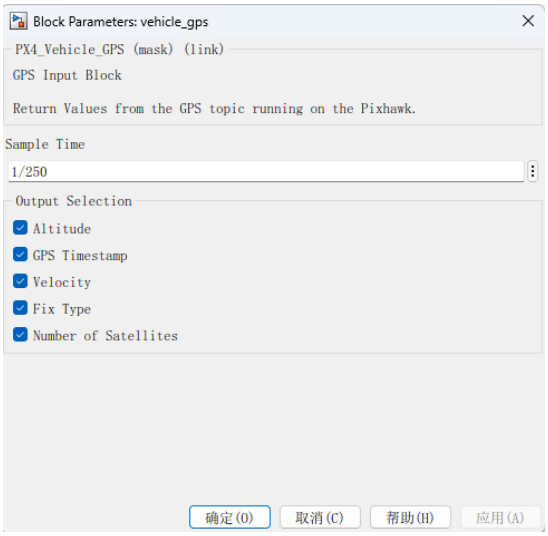
### 2.3.8. vehicle gps— GPS Data Module

The Pixhawk GPS data can be accessed through this module, which is achieved by subscribing to the uORB hashtag "vehicle_gps", so you will need to ensure that the GPS module is plugged into Pixhawk to get the correct data during the actual run. As shown in the image below, the module provides a setting box for the sample rate, as well as selectable attitude options. The meaning of each option is as follows:

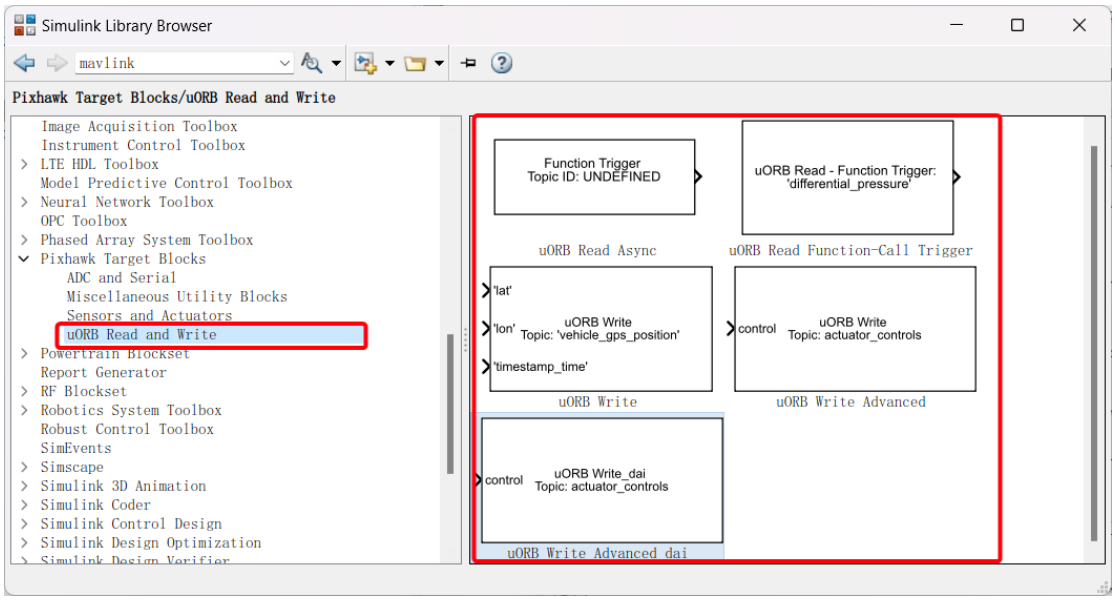
- Latitude: (int32) Global coordinates are given at 1e-7 degrees.
- Longitude: (int32) Longitude is given in 1e-7 degrees.
- Altitude: (int32) is 1e-3 m (mm) above MSL (mean sea level).
- GPS TS: (double) Timestamp (microseconds in GPS format), this is the timestamp from the GPS module.
- Velocity: (single)GPS ground speed, in meters per second.
- GPS Fix Type: (uint8)0-1=NO fix,2=2D fix,3=3D fix.
- Num Satellites: (uint8) Number of satellites used to calculate.

More information is available by clicking the "Help" button in the dialog box.



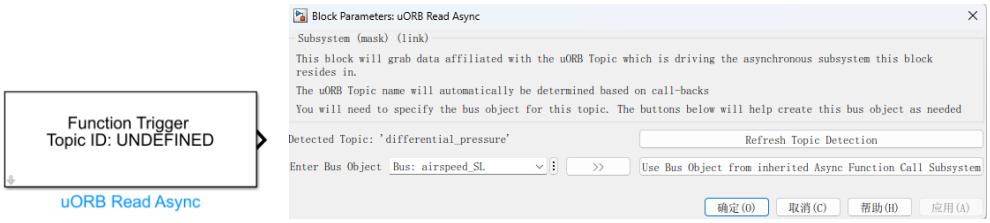



## 2.4. uORB Read and Write— uORB Message Read and Write Library



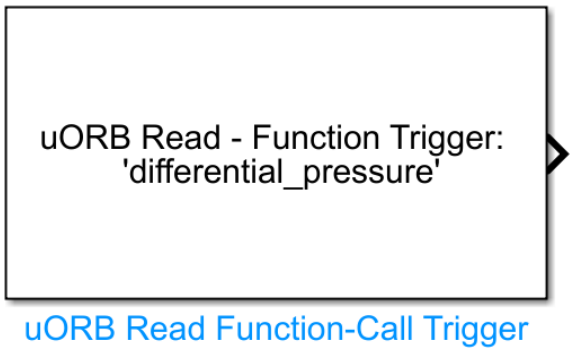

### 2.4.1. uORB Read Async—Retrieve Data Related to uORB Topic

As shown in the image below, the block will get data related to the uORB topic that drives the asynchronous subsystem where the block is located. The topic name will be determined automatically based on the callback, and the bus object will need to be specified for the topic. More information can be found by clicking the "Help" button of the dialog box.

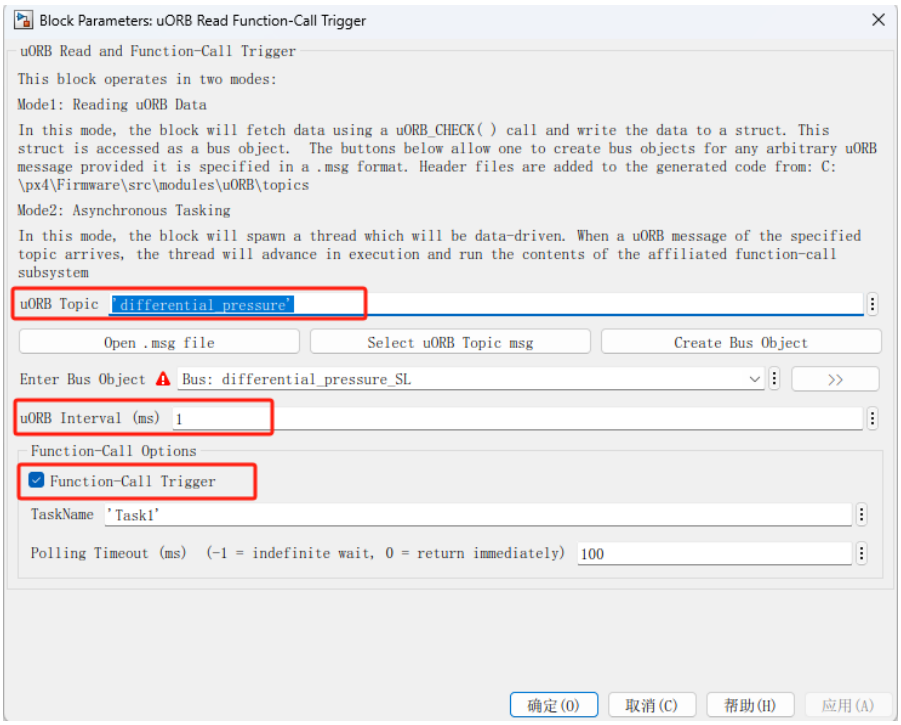



### 2.4.2. uORB Read Function-Call Trigger— uORB Message Read Callback Trigger Module

This module provides two functions, the first of which is to subscribe to messages from a uORB topic. The second is to subscribe to message data on a topic by triggering a function call signal for asynchronous events.




uORB Read Function-Call Trigger




As shown in the figure above, the steps of using the first function are:

- Choose a defined topic

  Click the button "Select uORB Topic msg" to open the list of topics for selection, only to pics that are not C++ objects are supported.

- Create the bus (bus) object

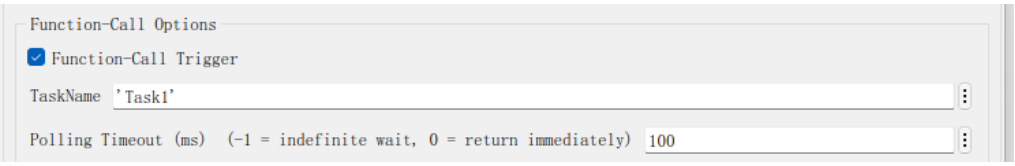
  Simulink's Bus object is used to receive uORB messages, click the button "Create Bus O nject" and Simulink will find the corresponding message file from the.msg folder and map it t o the MATLAB workspace to generate the bus object.

- Set the uORB read interval

  In non-asynchronous mode, you need to set the query frequency in milliseconds. For som e topics, the maximum data update rate is set. Do not set the frequency to exceed this maximu m.

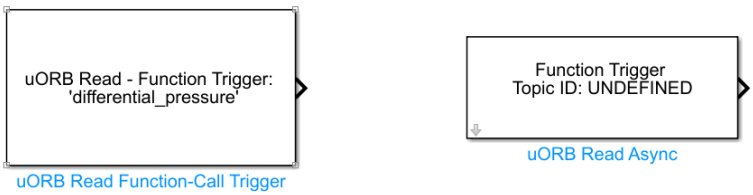
The steps to use the second function are as follows:

- Select the function call trigger




- Set the query timeout and task name

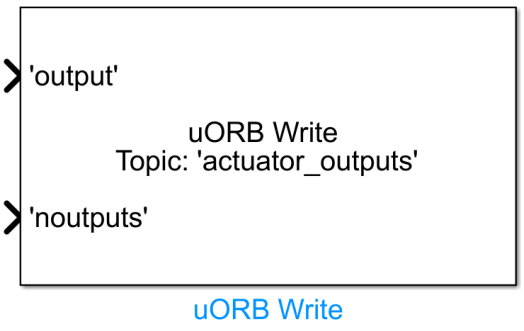
When asynchronous is selected, the sampling time setting box disappears, and you need to set the query timeout parameter and task name. The asynchronous function spawns a new t hread that runs the code associated with the function trigger signal and waits for new data on t he topic by querying it. At this point, another module is needed to read the data on the topic, n amely the "Read uORB function Trigger Data Module", as shown in the figure below.

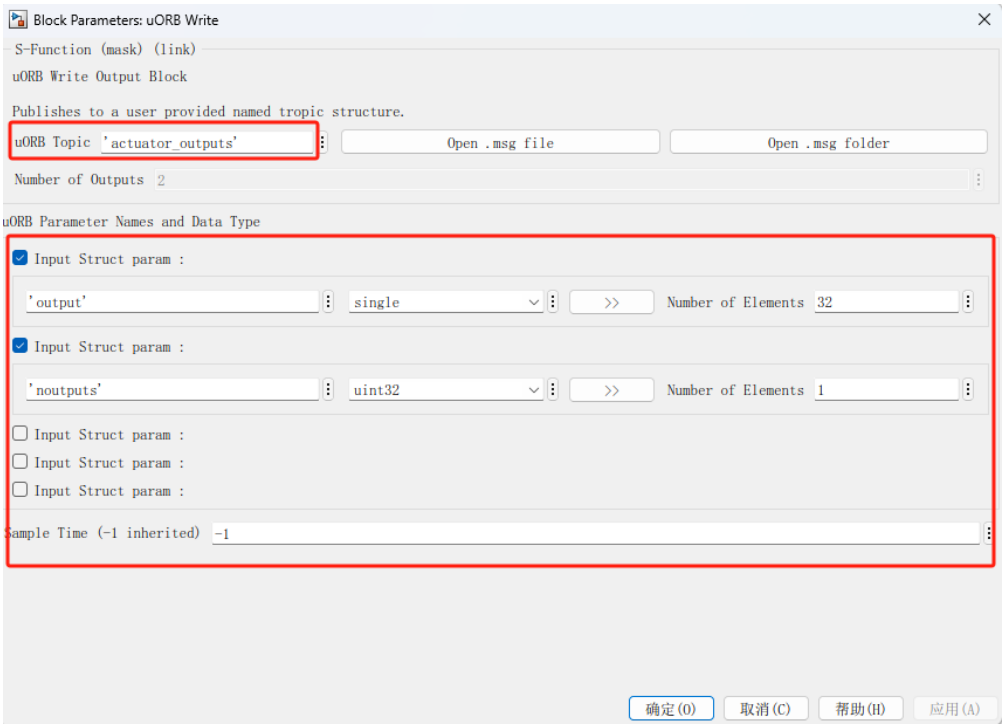


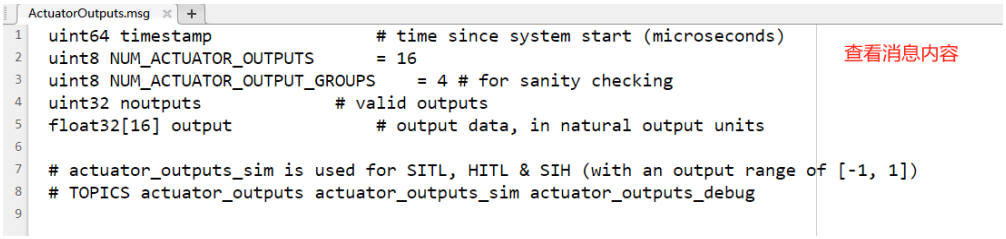

### 2.4.3. uORB Write— uORB Message Data Publishing Interface Module

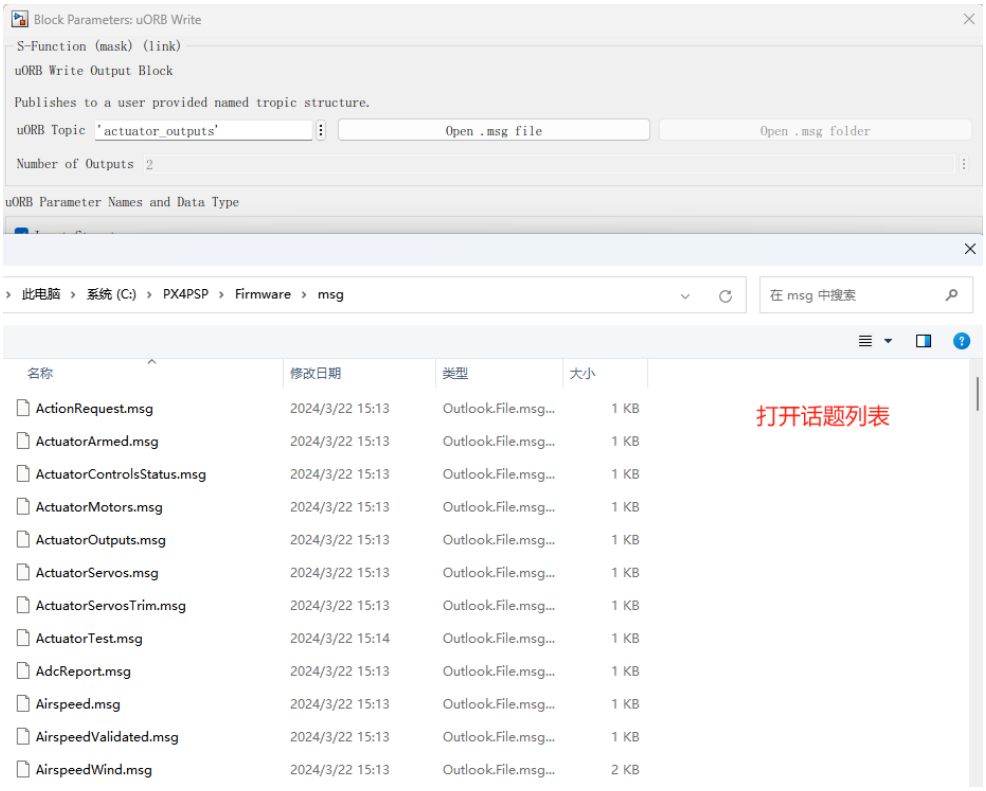
This interface allows users to publish specified values or structs to uORB topics. This module allows users to publish messages to a uORB topic. Topics must be properly defined, and some def ined topics are placed in the C:\PX4PSP\Firmware\msg directory, which automatically contains th e topic definition file in the generated code.

As shown in the following figure, you can enter the topic name, click the button "Open. msg f ile" to Open the corresponding message content, click the button "open. msg folder" to Open the to pic list, and set the input port name and data type to correspond to the topic message.



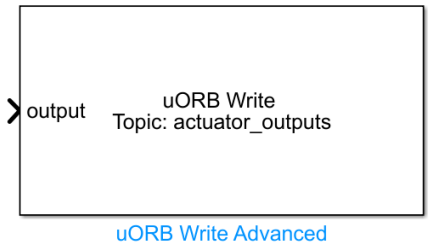

Block Parameters: uORB Write
S-Function (mask) (link)
uORB Write Output Block
Publishes to a user provided named tropic structure.
uORB Topic 'actuator_outputs'
Open .msg file
Open .msg folder
Number of Outputs 2
uORB Parameter Names and Data Type
Input Struct param :
'output'
single
>>
Number of Elements 32
Input Struct param :
'noutputs'
uint32
>>
Number of Elements 1
Input Struct param :
Input Struct param :
Input Struct param :
Sample Time (-1 inherited) -1
确定(O)
取消(C)
帮助(H)
应用(A)


ActuatorOutputs.msg
uint64 timestamp # time since system start (microseconds)
uint8 NUM_ACTUATOR_OUTPUTS = 16
uint8 NUM_ACTUATOR_OUTPUT_GROUPS = 4 # for sanity checking
uint32 noutputs # valid outputs
float32[16] output # output data, in natural output units
# actuator_outputs_sim is used for SITL, HITL & SIH (with an output range of [-1, 1])
# TOPICS actuator_outputs actuator_outputs_sim actuator_outputs_debug
查看消息内容

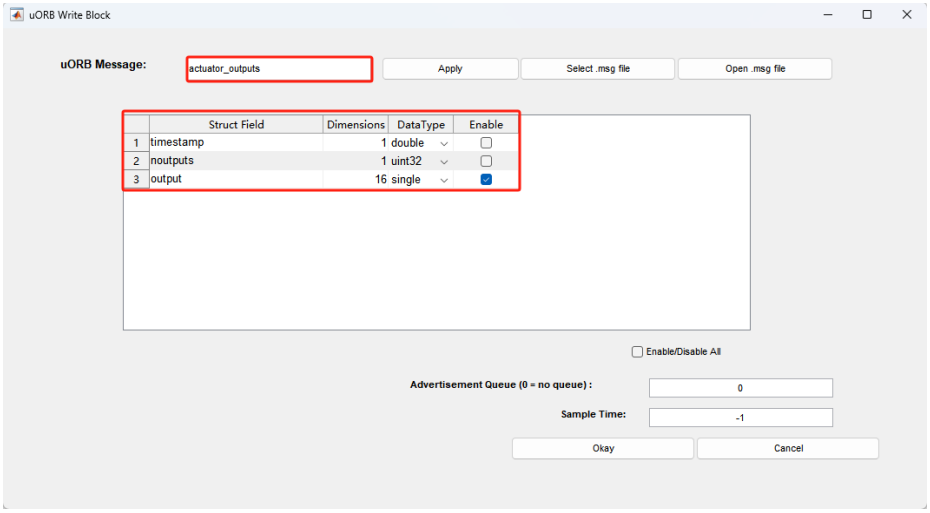

Block Parameters: uORB Write
S-Function (mask) (link)
uORB Write Output Block
Publishes to a user provided named tropic structure.
uORB Topic 'actuator_outputs'
Open .msg file
Open .msg folder
Number of Outputs 2
uORB Parameter Names and Data Type
此电脑 › 系统 (C:) › PX4PSP › Firmware › msg
在 msg 中搜索
名称
修改日期
类型
大小
ActionRequest.msg 2024/3/22 15:13 Outlook.File.msg... 1 KB
ActuatorArmed.msg 2024/3/22 15:13 Outlook.File.msg... 1 KB
ActuatorControlsStatus.msg 2024/3/22 15:13 Outlook.File.msg... 1 KB
ActuatorMotors.msg 2024/3/22 15:13 Outlook.File.msg... 1 KB
ActuatorOutputs.msg 2024/3/22 15:13 Outlook.File.msg... 1 KB
ActuatorServos.msg 2024/3/22 15:13 Outlook.File.msg... 1 KB
ActuatorServosTrim.msg 2024/3/22 15:13 Outlook.File.msg... 1 KB
ActuatorTest.msg 2024/3/22 15:14 Outlook.File.msg... 1 KB
AdcReport.msg 2024/3/22 15:13 Outlook.File.msg... 1 KB
Airspeed.msg 2024/3/22 15:13 Outlook.File.msg... 1 KB
AirspeedValidated.msg 2024/3/22 15:13 Outlook.File.msg... 1 KB
AirspeedWind.msg 2024/3/22 15:13 Outlook.File.msg... 2 KB
打开话题列表

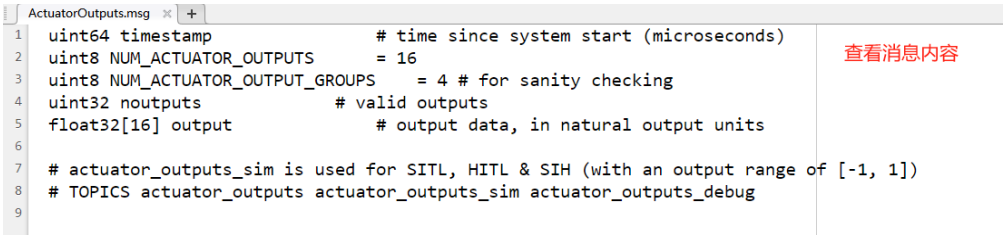
### 2.4.4. uORB Write Advanced— Advanced Module for uORB Message Data Publishing Interface

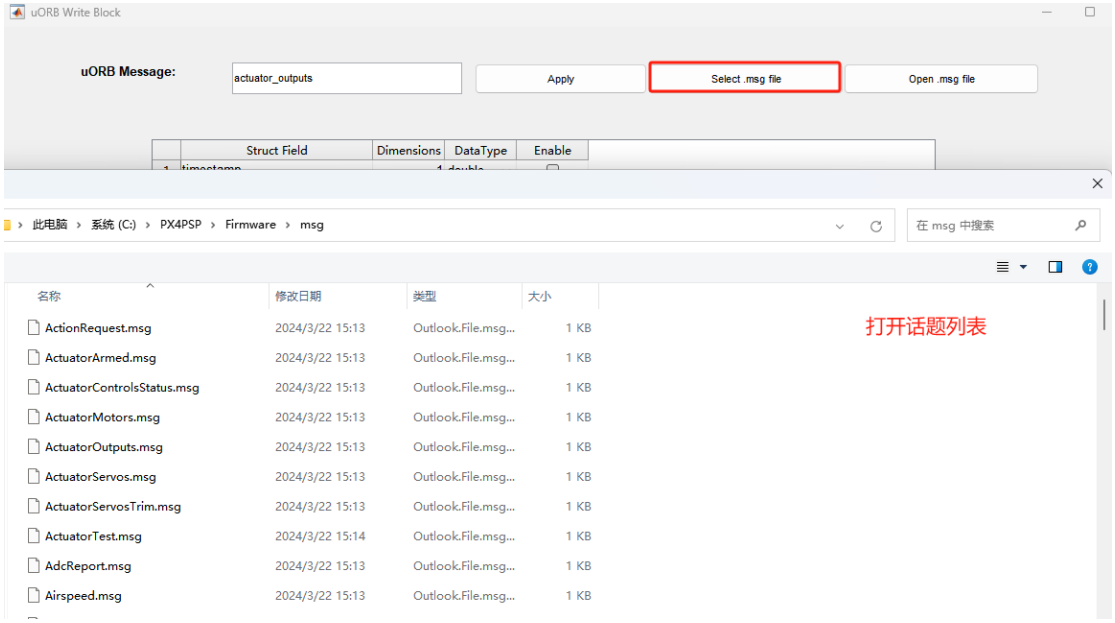
This interface allows users to have more flexible control over the data they publish. Using the uORB Write Advanced interface in Simulink, a more complex and precise way of publishing messages can be achieved. You can select the message file and a message ID to write to. In addition, you can also set the priority of the message, queue size and other advanced options.




As shown in the following picture, you can see the name of the current topic, click the button "Open. msg file" to Open the corresponding message content, and click the button "Select. msg file" to open the topic list, and you can set the input port name and data type to correspond to the topic message.

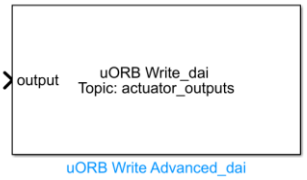


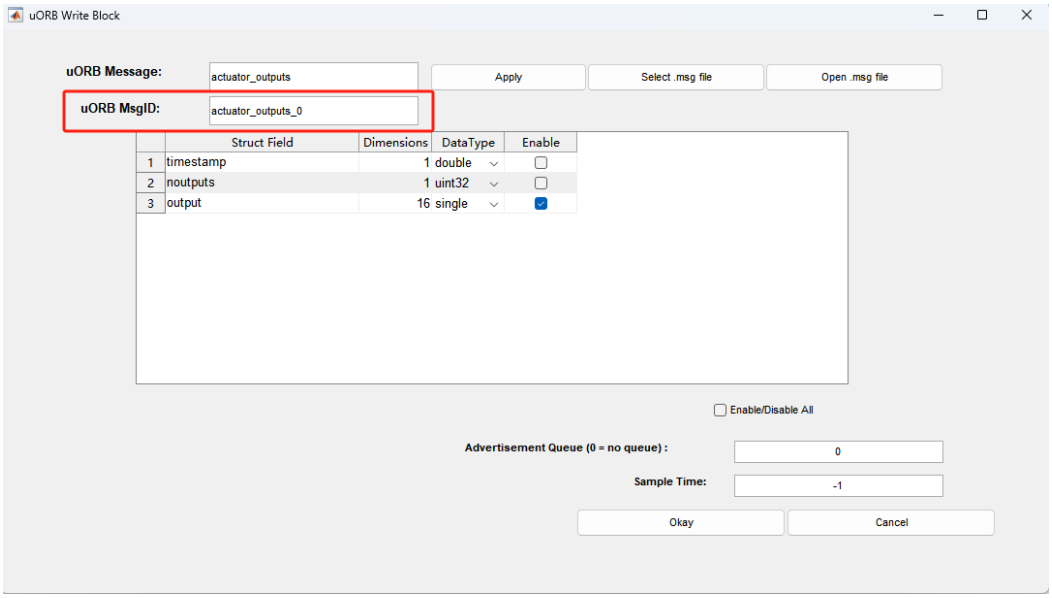



### 2.4.5. uORB Write Advanced_dai— Advanced Module for uORB Message Data Publishing Interface

Compared with uORB Write Advanced, uORB Write Advanced_dai adds the function of cust omizing uORB MsgID.




uORB Write Advanced_dai




See the file for the specific routine experiment:

- 0.ApiExps\5.Log-Write-Read\Readme.pdf

- 0.ApiExps\6.uORB-Read-Write\Readme.pdf

# 3. MATLAB Command Line Interface

The RflySim platform also supports running relevant commands through the MATLAB comm and line window, including:

## 3.1. PX4Upload

You can upload the PX4 firmware to the flight controller with a single click. The firmware up loaded at this time is located at: *\PX4PSP\Firmware\build\[build command]\[build command].px 4 (For example, [build command] could be px4_fmu-v6x_default).

```
PX4Upload
```

After executing the above command, a black window will pop up, prompting the user to plug and unplug the flight controller, and displaying the upload progress bar.

## 3.2. PX4CMD

To switch to the compilation environment for the Pixhawk 6C flight controller, you can perfor m the following steps to replace the firmware compilation options:

```
PX4CMD('px4_fmu-v6c_default')
或
PX4CMD 'px4_fmu-v6c_default'
```

## 3.3. PX4Build

Firmware compilation is possible.

## 3.4. PX4AppName

Rename the PX4 software's APP to support multiple automatic code generation programs. For detailed usage, please refer to:

```
PX4AppName('rfly_simulink_app')
%  或
PX4AppName 'rfly_simulink_app'
```

**Related examples can be found at: 2.AdvExps\e0_AdvApiExps\1.CusMaskPX4Code\Re adme.pdf、 2.AdvExps\e0_AdvApiExps\2.RenamePX4App\Readme.pdf**

## 3.5. PX4AppLoad

Load the renamed PX4 software's App to import previously developed App programs. The us age is as follows:

```
PX4AppLoad('C:\PX4PSP\rfly_simulink_app')
或
PX4AppLoad 'C:\PX4PSP\rfly_simulink_app'
```

**Related examples can be found at: 2.AdvExps\e0_AdvApiExps\1.CusMaskPX4Code\Re adme.pdf、 2.AdvExps\e0_AdvApiExps\3.LoadPX4App\Readme.pdf**

## 3.6. PX4ModiFile

To replace parts of the code in PX4 software through Excel, follow these steps:

```
PX4ModiFile('C;\Users\dream\Desktop\自定义屏蔽 UORB 消息的例子 px4Block.xlsx')
```

**Related examples can be found at: 2.AdvExps\e0_AdvApiExps\1.CusMaskPX4Code\Readme.pdf、 2.AdvExps\e0_AdvApiExps\2.RenamePX4App\Readme.pdf**

## 3.7. PX4Official

By executing the command, you can directly generate official firmware (without output masking), which can be used to restore the flight controller for HITL external control or to repair problematic flight controllers. The usage is as follows:

```
PX4Official
执行完上述命令后，再输入下面指令，可以将官方固件上传到飞控中：
PX4Upload
```

## 3.8. PX4SitlSet

To enable the current automatically generated controller, px4_simulink_app, to support SITL simulation, follow these steps: In the Simulink program, click "Build" to generate the hardware-in-the-loop (.px4) file. Run PX4SitlSet directly after generating the .px4 file.Execute SITLRun (for standard quadcopter) or other SITL simulation scripts driven by the DLL model to simulate the hardware-software interaction of the automatically generated algorithm.

Command format:

```
PX4SitlSet
```

**Related examples can be found at: 0.ApiExps\14.SITLVeriGenCodeFirm\Readme.pdf**

## 3.9. PX4SitlRec

In the SITL simulation code, remove the automatically generated controller, px4_simulink_app, to revert to the normal software-in-the-loop simulation mode, which supports QGC control and Offboard external control. Note: After testing the Simulink controller by running PX4SitlSet, if you want to run the platform's official vision or external control routines again, please use PX4SitlRec to restore the environment first.

Command format:

```
PX4SitlRec
```

**Related examples can be found at: 0.ApiExps\14.SITLVeriGenCodeFirm\Readme.pdf**

# 4. Automatically generated external communication interface

When running the one-click installation script for RflySim, the platform modifies the source code in the Firmware directory by adding four uORB messages. These messages are registered in *\

PX4PSP\Firmware\msg\CMakeLists.txt*. Detailed information and usage guidelines are provided below:

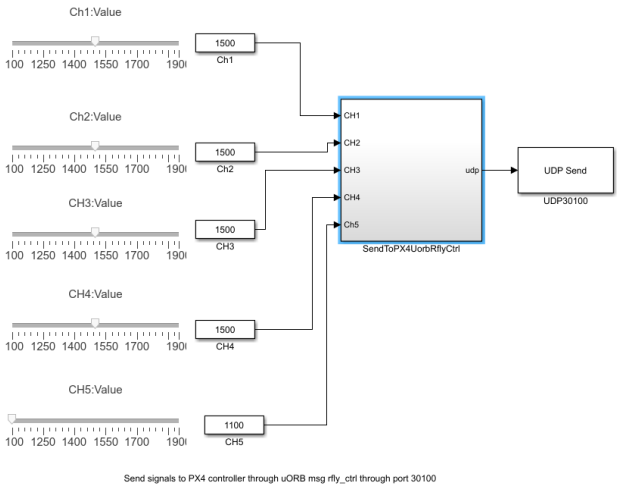
## 4.1. rfly_ctrl.msg

Format of messages transmitted from external sources into PX4 via UDP or MAVLink protocol:

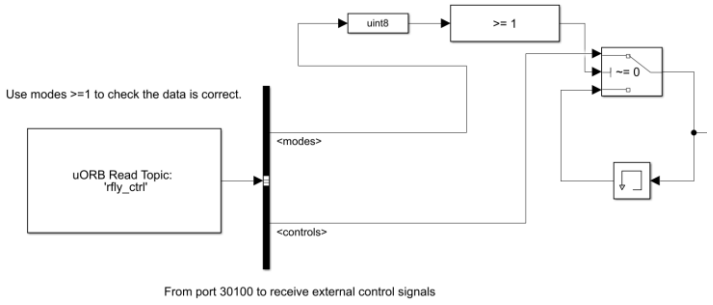
```
uint64 timestamp            # time since system start (microseconds)
uint32 flags                # control flag
uint8 modes                 # mode flag
float32[16] controls            # 16D control signals
```

**External data transmission from Simulink to internal uORB messages in PX4:**

Transmitter: Refer to the module "SendToPX4UorbRflyCtrl" in the example "0.ApiExps\9.PX4CtrlExternalTune\Readme.pdf". This module can send messages through the 30100 series port to CopterSim, and then forward them to the internal uORB message "rfly_ctrl" in PX4 through the MAVLink protocol.




Receiver: When designing low-level controllers in Simulink, simply subscribe to the "rfly_ctrl" message to receive the data.



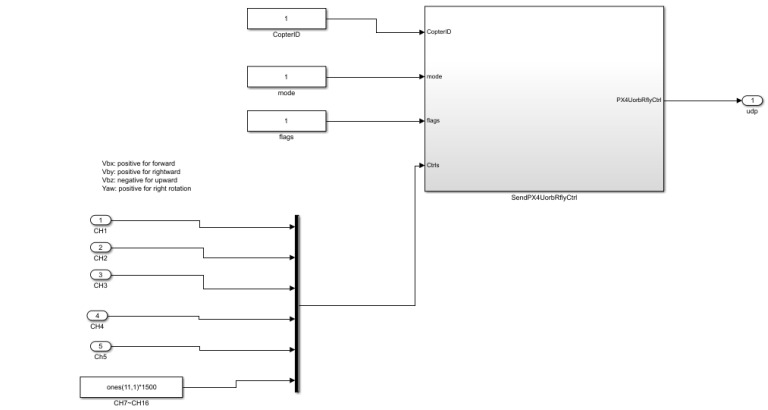

Experimental Principle: After receiving the "rfly_ctrl" message, PX4ExtMsgReceiver.slx simulates the first five elements of "controls" as inputs from the remote controller, which are then fed into the attitude controller of the previous example to perform attitude control. Therefore, PX4ExtMsgSender.slx can simulate sending inputs from the remote controller to test the attitude algorithm.

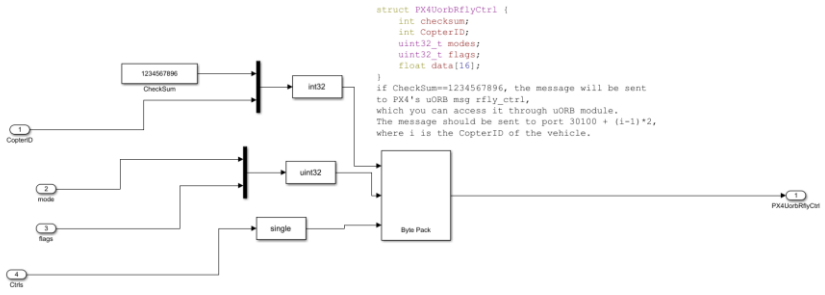
Experimental Procedure: PX4ExtMsgReceiver.slx is burned into the flight controller through automatic code generation and launched for hardware-in-the-loop simulation. Then, PX4ExtMsgSender.slx is opened in Simulink. By adjusting the sliders on the left side of the "SendToPX4UorbRflyCtrl" block, various channel data from the remote controller can be simulated to control the drone.

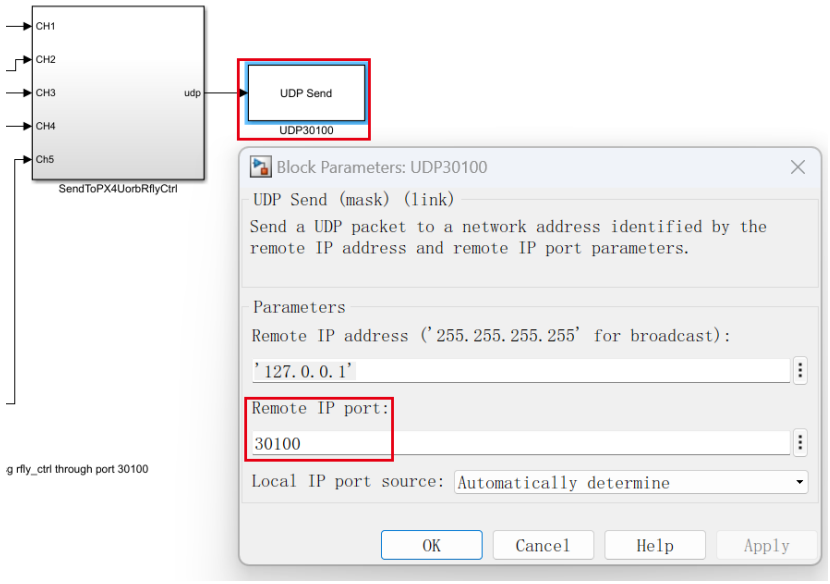
**Communication Principle from Simulink to CopterSim:** Analyzing the "SendToPX4UorbRflyCtrl" module reveals that this Simulink module sends the following structure via UDP to the 127.0.0.1:30100 port on the local machine.

```
struct PX4UorbRflyCtrl {
int checksum;
int CopterID;
uint32_t modes;
uint32_t flags;
float data[16];
}
```

Note: The checksum here must be set to 1234567896 in order to pass the verification by CopterSim.






**Communication Principle from CopterSim to PX4:** After receiving the PX4UorbRflyCtrl message, CopterSim will further relay it as the "hil_actuator_controls" MAVLink message (https://mavlink.io/en/messages/common.html#HIL_ACTUATOR_CONTROLS), and then forward it to the PX4 flight controller. The definition of this message is as shown in the following figure.

**HIL_ACTUATOR_CONTROLS ( #93 )**

[Message] Sent from autopilot to simulation. Hardware in the loop control outputs (replacement for HIL_CONTROLS)

| Field Name | Type | Units | Values | Description |
|---|---|---|---|---|
| time_usec | uint64_t | us | | Timestamp (UNIX Epoch time or time since system boot). The receiving end can infer timestamp format (since 1.1.1970 or since system boot) by checking for the magnitude of the number. |
| controls | float[16] | | | Control outputs -1 .. 1. Channel assignment depends on the simulated hardware. |
| mode | uint8_t | | MAV_MODE_FLAG | System mode. Includes arming state. |
| flags | uint64_t | | | Flags as bitfield, 1: indicate simulation using lockstep. |

Note: In reality, during hardware-in-the-loop simulation, this message is used by PX4 to transmit motor control commands to CopterSim. Here, it is borrowed to transmit data back to PX4.

The RflySim platform has modified the source code at C:\PX4PSP\Firmware\src\modules\mavlink\mavlink_receiver.cpp to add support for the hil_actuator_controls message.

**Internal parsing of the rfly_ctrl message by PX4:** As seen in the source code below, both rfly_ctrl and rfly_ext borrow the hil_actuator_controls message for data transmission. When the mode is 123, data is sent to the Simulink controller via rfly_ext, while in other cases, it is sent to the controller via rfly_ctrl.

C: > PX4PSP > Firmware > src > modules > mavlink > mavlink_receiver.cpp

```
109  MavlinkReceiver::handle_message(mavlink_message_t *msg)
110  {
111      switch (msg->msgid) {
112      case MAVLINK_MSG_ID_HIL_ACTUATOR_CONTROLS:{
113          mavlink_hil_actuator_controls_t hil_actuator_control;
114          mavlink_msg_hil_actuator_controls_decode(msg, &hil_actuator_control);
115          if(hil_actuator_control.mode==123){
116              rfly_ext_s re{};
117              re.timestamp = hrt_absolute_time();
118              re.modes = 1;
119              for(int i=0;i<16;i++){
120                  re.controls[i]=hil_actuator_control.controls[i];
121              }
122              _rfly_ext_pub.publish(re);
123          }else{
124              rfly_ctrl_s rc{};
125              rc.timestamp = hrt_absolute_time();
126              rc.modes = hil_actuator_control.mode;
127              rc.flags = hil_actuator_control.flags;
128              for(int i=0;i<16;i++){
129                  rc.controls[i]=hil_actuator_control.controls[i];
130              }
131              _rfly_ctrl_pub.publish(rc);
132          }
133          break;
134      }
```

**Python Interface Usage:** According to the forwarding principle of CopterSim, we can send the PX4UorbRflyCtrl structure to port 30100 or directly send MAVLink messages to the PX4 flight controller. There are two methods to input the rfly_ctrl message.

First Method: Utilize the sendPX4UorbRflyCtrl function from the PX4MavCtrlV4.py interface for data transmission.

```
23  ctrls=[1500,1500,1900,1500,1900,1100,1
24  mav.sendPX4UorbRflyCtrl(ctrls)
25  # 发送SendToPX4UorbRflyCtrl数据，在PX4内
26  # 本消息主要设置油门通道（3通道）置于最高，
```

The sendPX4UorbRflyCtrl function internally sends the PX4UorbRflyCtrl structure to the 30100 series port, which is then forwarded to the PX4 flight controller.

```
1258      # }2i2I16f
1259      def sendPX4UorbRflyCtrl(self,data=[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],modes=1,flags=1):
1260          checksum=1234567896
1261          buf = struct.pack("2i2I16f",checksum,self.CopterID,modes,flags,*data)
1262          self.udp_socket.sendto(buf, (self.ip, self.port+10000))
```

Since we know that both rfly_ctrl and rfly_ext are Mavlink messages originating from hil_actuator_controls to transmit data, we can also directly use Python to send Mavlink messages for data transmission.

```
22
23  ctrls=[1500,1500,1900,1500,1900,1100,1100,1100,1100,1100,1100,1100,1100,1100,1100,1100]
24  mav.SendHILCtrlMsg(ctrls)
25  # 发送SendToPX4UorbRflyCtrl数据，在PX4内部产生rfly_ctrl的uORB消息
26  # 本消息主要设置油门通道（3通道）置于最高，飞机起飞。同时5通道置于最高，控制器解锁。
```

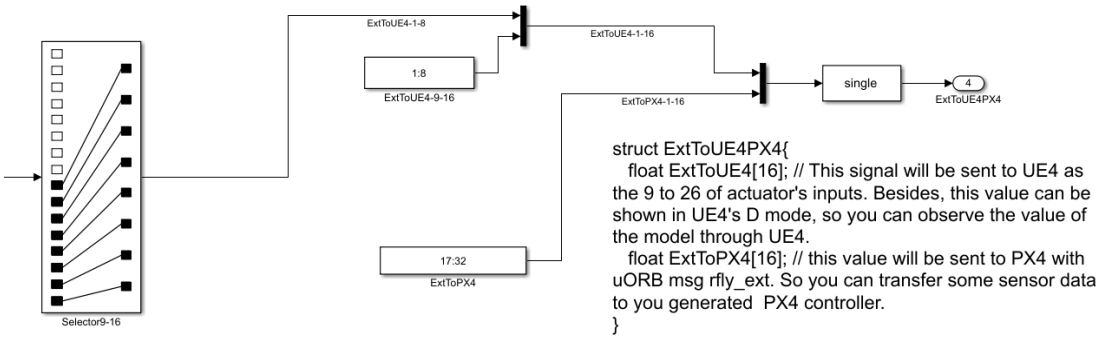
```
2213    # send hil_actuator_controls message to Pixhawk (for rfly_ctrl uORB message)
2214    def SendHILCtrlMsg(self,ctrls):
2215        """ Send hil_actuator_controls command to PX4, which will be transferred to uORB message rfly_ctrl
2216        https://mavlink.io/en/messages/common.html#HIL_ACTUATOR_CONTROLS
2217        """
2218        time_boot_ms = int((time.time()-self.startTime)*1000)
2219        controls = [1500,1500,1100,1500,1500,1500,1500,1500,1500,1500,1500,1500,1500,1500,1500,1500]
2220        for i in range(len(ctrls)):
2221            if i<len(controls):
2222                controls[i]=ctrls[i]
2223        if self.isCom or self.isRealFly:
2224            self.the_connection.mav.hil_actuator_controls_send(time_boot_ms,controls,1,1)
2225        else:
2226            buf = self.mav0.hil_actuator_controls_encode(time_boot_ms,controls,1,1).pack(self.mav0)
2227            self.udp_socket.sendto(buf, (self.ip, self.port))
2228        #print("Msg Send.")
2229
```

## 4.2. rfly_ext.msg

Transferring directly from the DLL model into the PX4 internals, the message definition form at is:

```
uint64 timestamp            # time since system start (microseconds)
uint8 modes                 # mode flag
float32[16] controls            # 16D control signals
```

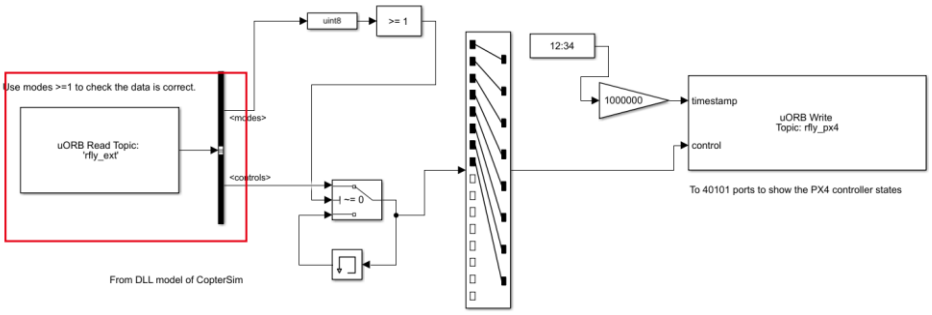
1）During DLL model development, the output port ExtToUE4PX4 is introduced. The first 16 dimensions of data from this interface are directly sent to UE4 for actuator control execution, while the subsequent 16 dimensions are directly sent to PX4. These can be received in the low-level controller via the rfly_ext message.




2) When loading the DLL model in CopterSim, the 32-dimensional data from ExtToUE4PX4 is split into a 16-dimensional ExtToUE4 structure and a 16-dimensional ExtToPX4 structure. The latter is then passed into the PX4 flight controller via the MAVLink message hil_actuator_controls. It's important to note that a passphrase "mode=123" is forcibly set during this transmission. Therefore, as mentioned earlier, mavlink_receiver.cpp will parse it as rfly_ext before forwarding it.

3) In the Simulink low-level controller, subscribing to "rfly_ext" allows for receiving data from the DLL model. This interface can be used to transmit sensor data that CopterSim currently does

not support to PX4, thus accelerating the debugging speed of hardware-in-the-loop simulation env ironments.



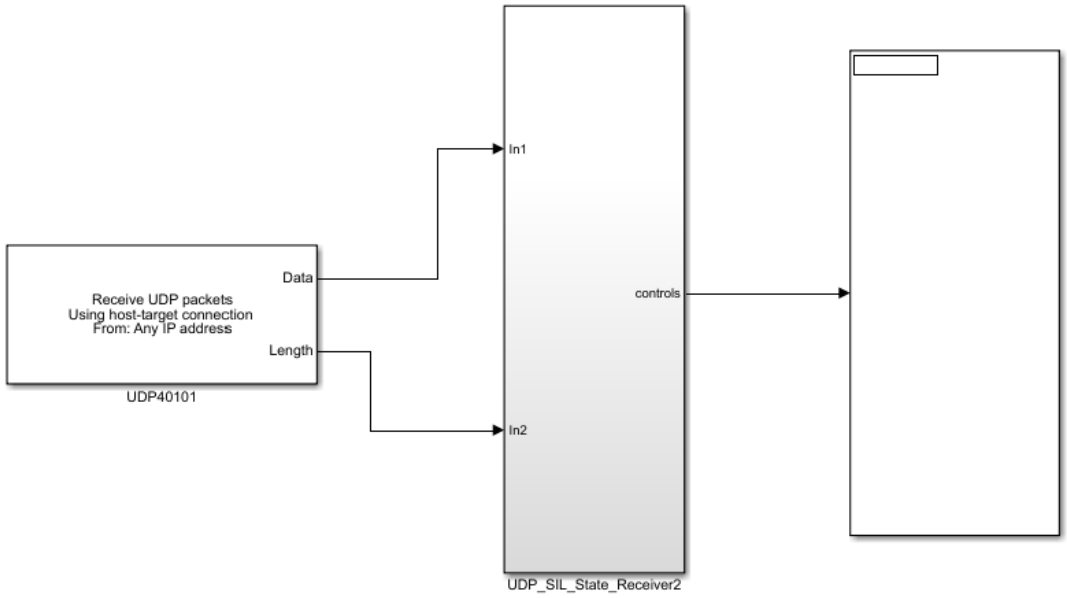

4）Testing Method: PX4ExtMsgReceiver.slx automatically generates code and burns it into t he flight controller. Exp2_MaxModelTemp.slx generates the DLL model and enables hardware-in-the-loop simulation. In the DLL model, numbers 17 through 32 are sent to rlfy_ext, and then the lo w-level controller forwards them as rfly_px4 UORB messages. Therefore, the method described la ter can be used to view the data in QGroundControl (QGC) to verify if the messages from the DLL model are correctly transmitted.

## 4.3. rfly_px4.msg

To transmit PX4 internal data externally through UDP or MAVLink protocol, the message def inition format is as follows:

```
uint64 timestamp            # time since system start (microseconds)
float32[8] control          # 8D control signals
```

1）Publish the rfly_px4 message in the low-level controller, then subscribe to the data throug h port 40100 in Simulink. Refer to the example in the UDP_SIL_State_Receiver2 module located i n 0.ApiExps\9.PX4CtrlExternalTune\Readme.pdf.




Get desired signals from PX4 through rfly_px4 uORB API with port 40101

2）The RflySim platform has modified the "Firmware\src\modules\mavlink\streams\ACTUATOR_CONTROL_TARGET.hpp" file to subscribe to rfly_px4 and forward it as ACTUATOR_CONTROL_TARGET MAVLink messages, which are then received by CopterSim.

```
109         actuator_controls_s act_ctrl;
110         if(N==0){
111             rfly_px4_s rf_px;
112             if (_rfly_px4_sub && _rfly_px4_sub->update(&rf_px)) {
113                 mavlink_actuator_control_target_t msg{};
114 
115                 msg.time_usec = rf_px.timestamp;
116                 msg.group_mlx = 123;
117 
118                 for (unsigned i = 0; i < sizeof(msg.controls) / sizeof(msg.controls[0]); i++) {
119                     msg.controls[i] = rf_px.control[i];
120                 }
121 
122                 mavlink_msg_actuator_control_target_send_struct(_mavlink->get_channel(), &msg);
123 
124                 return true;
125             }
```

Please refer to the detailed definition of the message：https://mavlink.io/en/messages/common.html#ACTUATOR_CONTROL_TARGET

## ACTUATOR_CONTROL_TARGET ( #140 )

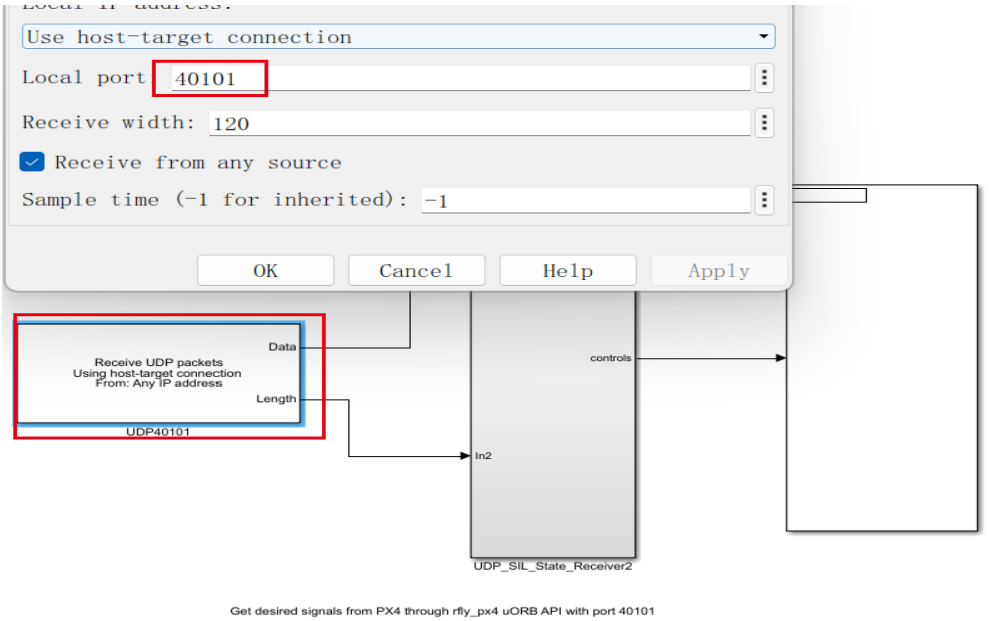
[Message] Set the vehicle attitude and body angular rates.

| Field Name | Type | Units | Description |
|---|---|---|---|
| time_usec | uint64_t | us | Timestamp (UNIX Epoch time or time since system boot). The receiving end can infer timestamp format (since 1.1.1970 or since system boot) by checking for the magnitude of the number. |
| group_mlx | uint8_t | | Actuator group. The "_mlx" indicates this is a multi-instance message and a MAVLink parser should use this field to difference between instances. |
| controls | float[8] | | Actuator controls. Normed to -1..+1 where 0 is neutral position. Throttle for single rotation direction motors is 0..1, negative range for reverse direction. Standard mapping for attitude controls (group 0): (index 0-7): roll, pitch, yaw, throttle, flaps, spoilers, airbrakes, landing gear. Load a pass-through mixer to repurpose them as generic outputs. |

3）After receiving the ACTUATOR_CONTROL_TARGET message, CopterSim will check if group_mlx equals the code 123. If it does, it will forward the following structure to the 40100 series port.

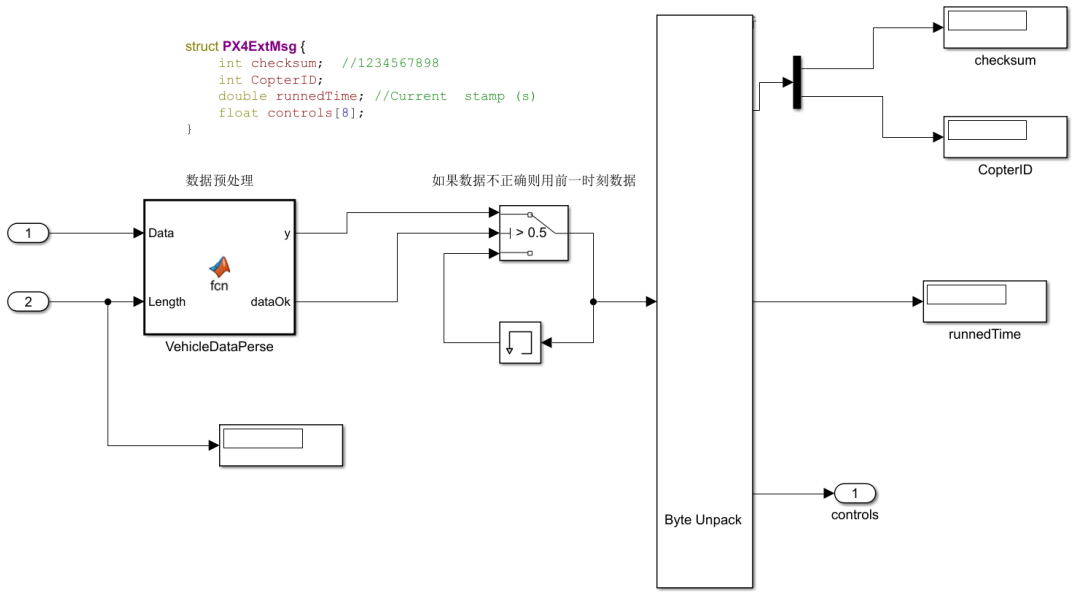
```
struct PX4ExtMsg {
    int checksum;  //1234567898
    int CopterID;
    double runnedTime; //Current  stamp (s)
float controls[8];
}
```

Note: CopterSim uses odd-numbered ports for outgoing communication and even-numbered ports for incoming communication. Therefore, for the first aircraft, when publishing the PX4ExtMsg message to the corresponding 40100 series port, it is actually using port 40101.




Get desired signals from PX4 through rfly_px4 uORB API with port 40101

After listening to port 40101 through the UDP module, the data is parsed and outputted.




In the parsing function, checks are performed on the length and checksum flags to prevent reading incorrect data. Additionally, if the correct data is not received, the aforementioned data will be retained.

编辑器 - Block: PX4ExtMsgSender/UDP_SIL_State_Receiver2/V

UDP_SIL_State_Receiver2/VehicleDataPerse

```
1   function [y,dataOk] = fcn(Data,Length)
2   y=uint8(zeros(48,1));
3   dataOk = 0;
4   if Length ~= 48
5       return;
6   end
7   checksum=typecast(Data(1:4), 'int32');
8   if checksum ~= 1234567898
9       return;
10  end
11
12  y = Data(1:48);
13  dataOk = 1;
```

Using Python or C language, one can similarly listen for MAVLink's ACTUATOR_CONTROL_TARGET message to obtain the "rfly_ext" data. Readers are encouraged to try this out themselves.

## 4.4. rfly_insils.msg

Message Definition Format for Internal Data Transmission in PX4:

```
uint64 timestamp            # time since system start (microseconds)
uint32 checksum                 # checksum/flag
int32[8] in_sil_ints            # 8D int signals
float32[20] in_sil_floats       # 20D float signals
```

The routine for this message is continuously being improved and can be broadly utilized in four aspects:

1) Directly modifying the PX4 source code by adding subscription code for "rfly_insils", and then publishing this message through Simulink to establish communication between the Simulink controller and the modified code interface.
2) Directly modifying the PX4 source code by adding publishing code for "rfly_insils", and then subscribing to this message through Simulink.
3) Generating two automatic code applications for subscribing and publishing communication between the two.
4) Modifying the PX4 source code in two locations to enable bidirectional communication and connect the two sets of code.

# 5. Flight Log Recording and External Data Communication Interface

## 5.1. Simulation Ground Truth Data Analysis

### 5.1.1. Offline Acquisition Method

For the i-th aircraft, you only need to create a new file under PX4PSP\CopterSim called CopterSim+i+.csv (for example, CopterSim1.csv). After each simulation, it will record the simulation ground truth data (similar to RflySim3D received data, including position, velocity, motor RPM, etc.). For detailed operational steps, please refer to the following:*\PX4PSP\RflySimAPIs\2.RflySimUsage\1.BasicExps\e14_Log-Get\Readme.pdf。

### 5.1.2. Online Acquisition Method

RflySim provides two methods for simulation data retrieval and analysis:

- Method One: Detailed operational steps for the MATLAB/Simulink version are as follows:*\PX4PSP\RflySimAPIs\10.RflySimSwarm\0.ApiExps\7.DataAnalysis_Mat\Readme.pdf。
- Method Two: Detailed operational steps for the Python version are as follows:*\PX4PSP\RflySimAPIs\10.RflySimSwarm\0.ApiExps\8.DataAnalysis_Py\Readme.pdf。

## 5.2. Flight Data Analysis

### 5.2.1. Offline Log Analysis

Detailed operational steps for offline log analysis can be found at:*\PX4PSP\RflySimAPIs\2.RflySimUsage\1.BasicExps\e4_Log-Reads-Python38Env\Readme.pdf。

### 5.2.2. Online Log Analysis

Visit https://logs.px4.io/ and upload the ulog file for analysis.

## 5.3. Controller and External Data Communication Interface

### 5.3.1. Actuator_output Message - HIL Simulation

Contents of the "actuator_output.msg" file:

```
uint64 timestamp                # time since system start (microseconds)
uint8 NUM_ACTUATOR_OUTPUTS      = 16
uint8 NUM_ACTUATOR_OUTPUT_GROUPS = 4  # for sanity checking
uint32 noutputs                 # valid outputs
float32[16] output              # output data, in natural output units
```

This message is only used for hardware-in-the-loop (HIL) simulation.

### 5.3.2. pwm_output Message - HIL & Actual Flight

Contents of the "pwm_output.msg" message file:

```
uint64 timestamp   # Time since system start (microseconds)
uint64 error_count # Timer overcapture error flag (AUX5 or MAIN5)
uint32 pulse_width # Pulse width, timer counts
uint32 period      # Period, timer counts
```

This message is only used for HIL and actual flight, and the flight controller's output must support px4io.

### 5.3.3. actuator_control_0—HIL & Actual Flight

Contents of the "actuator_controls.msg" message file:

```
uint64 timestamp            # time since system start (microseconds)
uint8 NUM_ACTUATOR_CONTROLS = 8
uint8 NUM_ACTUATOR_CONTROL_GROUPS = 6
uint8 INDEX_ROLL = 0
uint8 INDEX_PITCH = 1
uint8 INDEX_YAW = 2
uint8 INDEX_THROTTLE = 3
uint8 INDEX_FLAPS = 4
uint8 INDEX_SPOILERS = 5
uint8 INDEX_AIRBRAKES = 6
uint8 INDEX_LANDING_GEAR = 7
uint8 INDEX_GIMBAL_SHUTTER = 3
uint8 INDEX_CAMERA_ZOOM = 4

uint8 GROUP_INDEX_ATTITUDE = 0
uint8 GROUP_INDEX_ATTITUDE_ALTERNATE = 1
uint8 GROUP_INDEX_GIMBAL = 2
```

```
    uint8 GROUP_INDEX_MANUAL_PASSTHROUGH = 3
    uint8 GROUP_INDEX_ALLOCATED_PART1 = 4
    uint8 GROUP_INDEX_ALLOCATED_PART2 = 5
    uint8 GROUP_INDEX_PAYLOAD = 6

    uint64 timestamp_sample     # the timestamp the data this control response is based on was
 sampled
    float32[8] control

    # TOPICS actuator_controls actuator_controls_0 actuator_controls_1 actuator_controls_2 ac
tuator_controls_3
    # TOPICS actuator_controls_4 actuator_controls_5
    # TOPICS actuator_controls_virtual_fw actuator_controls_virtual_mc
```

This message is primarily used to convey actuator control commands, with different message IDs corresponding to different control groups. Please refer to section 8.1 in the PX4 control group definition, such as the actuator_control_0 series messages. It corresponds to the control group of multirotors, supporting both hardware-in-the-loop simulation for multirotors and actual flight on real vehicles. Note: When using this message, it is necessary to select the option to suppress the actuator_control_0 message in the installation script of RflySim (a simulator). Additionally, it cannot be directly mapped to motors; instead, the output needs to be configured using PX4's mixer rules.

## 5.4. Flight Controller Communication Interface with External Data

Note: Please refer to the corresponding example code for this section:*\PX4PSP\RflySimAPIs\5.RflySimFlyCtrl\0.ApiExps\9.PX4CtrlExternalTune。

### 5.4.1. Port 20100 Series—Receiving Internal State Estimation Values from PX4

The 20100 series port primarily receives internal state estimation values from PX4. The received data includes:

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

The naming format for the receiving port of the i-th aircraft is: 20100 + (2*i-1). For example,

for the first aircraft, the port is 20101, for the second aircraft, it is 20103, and so on.

### 5.4.2. Port 30100 Sesries—Receiving CopterSim Flight Simulation Values and Sending rfly_ctrl Messages to the Flight Controller

The 30100 series port receives flight simulation values from CopterSim and sends rfly_ctrl messages to the flight controller. The received data includes:

```
struct SOut2Simulator
{
    int copterID;  //飞机 ID
    int vehicleType;  //飞机构型
    double runnedTime; //仿真时间
    float VelE[3];   //NED 地球系速度
    float PosE[3];   //NED 地球系位置
    float AngEuler[3];  //欧拉角
    float AngQuatern[4]; //四元数
    float MotorRPMS[8];  //电机转速 RPM
    float AccB[3];       //机体轴加速度
    float RateB[3];      //机体轴角速度
    double PosGPS[3];    //地球 GPS 经纬高
}

//SOut2Simulator 的定义与 Simulink 里面新加的 MavVehileInfo 结构体完全一致
//可以直接对新的 simulink 模型的 MavVehile3DInfo 输出口数据进行打包
//两个结构体的长度都是 152
sizeof(SOut2Simulator) = sizeof(MavVehileInfo) = 152

打包封装的结构体为
typedef struct _netDataShort
{
    TargetType tg;   //这里目前随便填，写 1 即可  uint32
    int        len;  //这个长度为传输结构体长度，目前是 152
    char       payload[192];  //这里面前 152 位存放了 SOut2Simulator 结构体数据，后面的 40 位保留
}netDataShort;

//UDP 包的总长度为 200
sizeof(netDataShort) = 200


//发送的 UDP 端口号为：20010
```

The naming format for the receiving port of the i-th aircraft is: 30100 + (2*i-1). For example, for the first aircraft, the port is 30101, for the second aircraft, it is 30103, and so on.

The format of the uORB message data it sends is:

```
uint64 timestamp            # time since system start (microseconds)
uint32 flags                # control flag
uint8 modes                 # mode flag
float32[16] controls             # 16D control signals
```

The naming format for the sending port of the i-th aircraft is: 30100 + (2*i-2). For example, for the first aircraft, the port is 30100, for the second aircraft, it is 30102, and so on.

### 5.4.3. 40100 System Port—Receiving Internal rfly_px4 Messages from Flight Controller

The 40100 system port receives internal rfly_px4 messages from the flight controller. The received data includes:

```
struct PX4ExtMsg {
    int checksum;  //1234567898
    int CopterID;
    double runnedTime; //Current  stamp (s)
    float controls[8];
}
```

The naming format for the receiving port of the i-th aircraft is: 40100 + (2*i-1). For example, for the first aircraft, the port is 40101, for the second aircraft, it is 40103, and so on.

# 6. Code masking and replacement interface

When developing based on the underlying control algorithms of RflySim, to validate the developed control algorithms, we need to mask the output of the PX4 software. In most cases, we only need to directly mask the motor output in the PX4 software system. However, certain specific development tasks require masking a certain intermediate quantity of a module in the PX4 software system to meet development needs. For example, if we need to mask the module of the attitude angular rate loop controller in the PX4 software system (this location is for PX4 version 1.12.3, for other versions, please refer to the PX4 official documentation) at: *\PX4PSP\Firmware\src\modules\mc_rate_control. Open the "MulticopterRateControl.cpp" file in this folder. Based on the PX4 source code architecture, we can know that the output uORB message of the attitude angular rate loop is "actuator_controls_0" (detailed definition of this message can be found at https://docs.px4.io/v1.12/en/concept/mixing.html). By examining the code, we can find that the code for publishing the "actuator_controls_0" message is as follows (you can also find it by searching for "_actuators_0_pub.publish(actuators);"):

```
253                 }
254             }
255
256             actuators.timestamp = hrt_absolute_time();
257             _actuators_0_pub.publish(actuators);
258
259         } else if (_v_control_mode.flag_control_termination_enabled) {
260             if (!_vehicle_status.is_vtol) {
261                 // publish actuator controls
262                 actuator_controls_s actuators{};
263                 actuators.timestamp = hrt_absolute_time();
264                 _actuators_0_pub.publish(actuators);
265             }
266         }
267     }
268
```

To mask the above two lines of code, there are two methods that can be used.

**Method 1:** We only need to delete them, comment them out, or replace them with other invalid code. Since PX4's compilation checks are very strict, directly commenting out the above two line

s of code may lead to compilation errors due to the actuators being defined but not used. Therefore, here we need to use the UNUSED macro to achieve masking. The masking of the code can follow one of the following rules depending on the situation.

a) Replace "_actuators_0_pub.publish(actuators);" with "" (empty string, equivalent to deletion. Note that this method is only applicable when there will be no error due to unused variable.)

b) Replace "_actuators_0_pub.publish(actuators);" with "//_actuators_0_pub.publish(actuators);" (this is equivalent to commenting out the line. Note the precautions mentioned above.)

c) Replace "_actuators_0_pub.publish(actuators);" with "UNUSED(actuators);" (this is equivalent to replacing it with an invalid statement. Note that this method is applicable when directly deleting the actuators variable would result in an error, and it is suitable for PX4 1.12 and earlier versions, as the UNUSED macro was removed starting from firmware version 1.13.)

d) Replace "_actuators_0_pub.publish(actuators);" with "(void) (actuators);" (this is equivalent to replacing it with an invalid statement. Note that this method is applicable when directly deleting the actuators variable would result in an error, and it is suitable for all versions including firmware version 1.13.)

**Method 2:** Another approach is to directly replace the file to be modified with a pre-modified file. The platform's provided interfaces can meet the requirements mentioned above. The one-click installation script of the RflySim platform offers a file replacement feature. The core idea is to describe the files and their contents to be replaced according to a given Excel file template. When running the installation script, you input the address of the Excel file, enabling automatic file replacement or modification of file contents during platform installation. For specific masking methods, please refer to the example files for details: 2.AdvExps\e0_AdvApiExps\1.CusMaskPX4Code\Readme.pdf

# 7. Multi-module parallel development interface

The latest version of the RflySim platform supports the rapid creation of multiple modules for parallel development. Based on the multi-process running state in the PX4 software system, the PX4 application generated automatically by MATLAB's code generation is named "px4_simulink_app." You can rename the PX4 application using the "MATLAB Command Line Interface." By renaming px4_simulink_app, you can continue to build models through Simulink to generate another application with the name px4_simulink_app. If you need to add another application, you can continue to rename it, and so on. In theory, this approach can facilitate the creation of numerous PX4 applications to meet development needs. For detailed usage of the interface, please refer to the experi

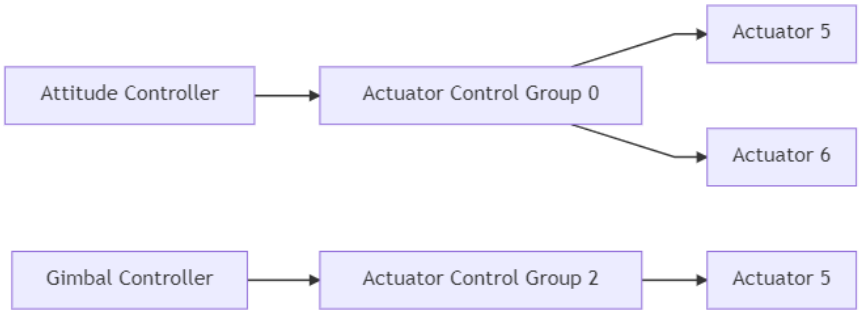
ment file: *\RflySimAPIs\5.RflySimFlyCtrl\2.AdvExps\e0_Basic-Interface\4.MultPX4App\Readme.pdf.

# 8. Different aircraft model development interface

## 8.1. Introduction to PX4 Flight Controller

The PX4 architecture ensures that no special handling for airframe layout is required within the core controller. Mixer refers to the distribution of input commands (e.g., right turn from the remote controller) to the actuators of motors and servos (such as electronic speed controllers or servo PWM commands). For fixed-wing aircraft, for instance, where each aileron is controlled by a servo, mixing involves controlling one aileron to lift while the other drops. Similarly, for multirotors, pitch operation requires changing the speed of all motors. Separating the mixing logic from the actual attitude controller significantly improves reusability.

A specific controller sends a normalized force or torque command (scaled to -1..+1) to the mixer, which then sets each individual actuator accordingly. Control output drivers (such as UART, UAVCAN, or PWM) then translate the mixer's output into the native units for the actuators' operation, such as outputting a PWM command with a value of 1300.




The control channel outputs of PX4 mainly consist of 4 control groups, which are:

- actuator_controls_0:The primary control channels of the flight controller are used to output control signals for various channels such as pitch, roll, yaw, throttle, etc. Their specific definitions are 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:Backup control channels, used in VTOL for outputting control signals in fixed-wing mode. Their specific definitions are 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: Gimbal control channel. Their specific definitions are 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. Their specific definitions are as follo ws:

```
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
```

## 8.2. PX4 Mixer Definition

One of the functions of the mixer in PX4 is to connect the outputs of upper-level application modules (such as the output from the attitude controller algorithm) with the actuators of the underl ying hardware (corresponding to PWM values for motors or servos). The mixer module effectively isolates the upper-level application modules from the hardware interface, so developers don't need to worry about which motor receives which control signal when writing upper-level application m odules. Additionally, by changing different mixer configuration files, PX4 can adapt to different ai rframe types without needing to modify the code of upper-level application modules. The specific code can be found in *\PX4PSP\Firmware\ROMFS\px4fmu_common\mixers.

The default settings for mixer files in the PX4 software system can be found in the *\PX4PSP \Firmware\ROMFS\px4fmu_common\init.d\airframes folder. Alternatively, these settings can be o verridden by placing a mixer file with the same name in the */etc/mixers/ directory on the SD card. Mixer files with the prefix "xxxx.main.mix" are mapped to the main (MAIN) output, while files n amed "yyyy.aux.mix" are mapped to the auxiliary (AUX) output. The prefixes (xxxx/yyyy) depend on the airframe and its configuration. Typically, MAIN and AUX outputs correspond to MAIN an d AUX PWM outputs, but when enabled, these may be loaded onto the UAVCAN (or other) bus.

The main mixer file name (prefix xxxx) is used in the airframe configuration with the command "set mixer xxxx," for example, calling "set mixer quad_x" in the airframes/4011_dji_f450 configuration loads the main mixer file quad_x.MAIN.mix). The AUX mixer file (prefix yyyy) depends on the airframe setting or defaults, such as:

- `MIXER_AUX` can be used to set which yyyy.aux.mix file is loaded. For example, in the file *\PX4PSP\Firmware\ROMFS\px4fmu_common\init.d\airframes\13006_vtol_standard_delta*, after `set MIXER_AUX vtol_delta`, the loaded mixer file will be *\PX4PSP\Firmware\ROMFS\px4fmu_common\mixers\vtol_delta.aux.mix.
- For multirotor and fixed-wing aircraft models, if the `MIXER_AUX` option is not set, the default loaded mixer file will be *\PX4PSP\Firmware\ROMFS\px4fmu_common\mixers\pass.aux.mix. (Note: `pass.aux.mix` is the pass-through mixer file for the remote control, which passes the values of four user-defined remote control channels (set using `RC_MAP_AUXx`/`RC_MAP_FLAPS` parameters) to the first four outputs of the AUX output.)
- Vertical take-off and landing (VTOL) drones will load the `yyyy.aux.mix` set if relevant settings are applied; otherwise, only the mixer file set by `MIXER` will be loaded.
- Enabling the general control mode (with the output mode set to AUX) will override the `MIXER_AUX` setting in the airframe and load the AUX output on *\PX4PSP\Firmware\ROMFS\px4fmu_common\mixers\mount.aux.mix*.

Note: The loading of mixer files as described above is implemented through the file *\PX4PSP\Firmware\ROMFS\px4fmu_common\init.d/rc.interface.

## 8.3. The syntax of PX4 mixer files

Mixer files are text files that define one or more mixer definitions: a mapping between one or more inputs and one or more outputs. There are mainly four types of definitions: multirotor mixer, helicopter mixer, summing mixer, and null mixer.

- multirotor mixer: defines the outputs of a + or x-shaped rotorcraft with 4, 6, or 8 outputs.
- helicopter mixer: defines the outputs of helicopter swashplate servos and main motor ESCs (the tail rotor is a separate mixer).
- summing mixer: 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.
- null mixer: produces an output of zero for the actuator output (when not in fail-safe mode).

The output produced by each mixer depends on the type and configuration of the mixer. For example, a multirotor mixer can have 4, 6, or 8 outputs depending on the aircraft type, while a summing mixer or null mixer only produces one output. Multiple mixers can be specified in each file. The output order (assigning mixers to actuators) is specific to the device that reads the mixer definitions, and for PWM, the output order matches the declaration order.

The first statement defined in each mixer file is:

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

The tag here represents the selected mixer type, as follows:

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

## 8.4. Summing Mixer—Additive Mixer

The Summing 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 mixing 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 rotation 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, then the sum is effectively zero, and the mixer outputs a fixed value constrained by `<lower limit>` and `<upper limit>`.

The second line defines the output scalar using the scalar parameters mentioned above. Although calculations are performed as floating-point operations, values stored in the definition file are scaled by a factor of 10,000; for instance, an offset of -0.5 is encoded as -5000. The `<traversal time>` on the output scale (optional) is used for actuators, and if the value is too large, it may damage the aircraft—e.g., a tilted actuator on tilt-rotor VTOL aircraft. It can be used to limit the rate of change of actuators (if not specified, rate limiting should not be applied). For example, a `<traversal time>` value of 20,000 will limit the rate of change of the actuator, requiring at least 2 seconds to go from `<lower limit>` to `<upper limit>` and vice versa.

Note 1: `<traversal time>` should only be used when required by hardware!

Note 2: Do not impose any restrictions on actuators controlling vehicle attitude (such as servos for aerodynamic surfaces), as this can easily lead to controller instability.

Proceed with defining the inputs and their scaling for `<control count>`, in the following form:

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

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

Note 4: Mixer outputs (where `<group>=0` and `<index>=3`) with throttle input will not operate in the armed or pre-armed state. For example, a servo with four inputs (roll, pitch, yaw, and throttle) will not move even with roll/pitch/yaw signals in the armed state.

The `<group>`value identifies the control group from which the scalar mixer reads, while the `<index>` value indicates the offset within that group. These values are specific to the devices defined in the mixer. When used for mixing vehicle controls, mixer group 0 is for vehicle attitude control, with 0 to 3 typically representing roll, pitch, yaw, and thrust respectively. The remaining fields utilize the parameter configurations for controlling scalers as discussed above. Values stored in the defini

tion file are scaled by a factor of 10,000 when computed as floating-point operations; for example, an offset of -0.5 is encoded as -5000. Below is an explanation of a typical mixer file example. For detailed example analysis, please refer to:https://docs.px4.io/v1.13/en/dev_airframes/adding_a_new_frame.html#mixer-file。

## 8.5. Null Mixer—Flight Controller

This mixer does not consume any control channels and generates a single actuator output with a constant value of zero. Typically, the Null Mixer is used as a placeholder in mixer collections to achieve a specific pattern of actuator outputs. It can also be utilized to control the output values for fault safety mechanisms (output is 0 during normal operation; during fault safety, the mixer is ignored, and the fault safety values are used instead). The definition is as follows:

```
Z:
```

## 8.6. Multirotor Mixer—Multirotor Mixer

The Multirotor Mixer combines four control inputs (roll, pitch, yaw, thrust) into a set of actuator outputs used for driving motor speed controllers. The definition is as follows:

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

Supported aircraft types include:

- 4x - Quadcopter X Configuration
- 4+ - Quadcopter Plus Configuration
- 6x - Hexacopter X Configuration
- 6+ - Hexacopter Plus Configuration
- 8x - Octocopter X Configuration
- 8+ - Octocopter Plus Configuration

The proportional values for roll, pitch, and yaw determine the proportion of roll, pitch, and yaw control relative to thrust control. When computed as floating-point operations, values stored in the definition file are scaled by a factor of 10,000; for example, 0.5 is encoded as 5000. The range of inputs for roll, pitch, and yaw is from -1.0 to 1.0, while the range for thrust input is from 0.0 to 1.0. The output range for each actuator is from -1.0 to 1.0.

The idle speed ranges from 0.0 to 1.0. Idle 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 for all actuators are readjusted to constrain the saturated actuators to 1.0.

## 8.7. Helicopter Mixer—Helicopter Mixer

The Helicopter Mixer combines three control inputs (roll, pitch, thrust) into four outputs (collective and main motor ESC settings). The first output of the helicopter mixer is the throttle setting

for the main motor. The subsequent outputs are for the servo of the swashplate. Tail rotor control can be achieved by adding a simple mixer. The thrust control input is used for the main motor setting and collective pitch of the swashplate. It employs a throttle curve and a pitch curve, each composed of five points

Note: The throttle and pitch curves map the positions of the "throttle" stick input to throttle values and pitch values (respectively). This allows for 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:` Defines points for the throttle curve. `P:` Defines points for the pitch curve. Both curves consist of 5 points ranging from 0 to 10000. For a simple linear variation, the five values of the curve should be 0, 2500, 5000, 7500, 10000.

The definition for each servo of the swashplate (3 or 4) is as follows:

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

`<angle>` is in degrees, with 0 degrees representing the direction of the nose. Positive angles are clockwise. `<arm length>` is a normalized length, where 10000 equals 1. If all servo arms are of the same length, the value should be 10000 for each. A larger arm length will decrease 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. Within the full servo range, `<lower limit>` and `<upper limit>` should be -10000 and +10000, respectively.

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

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

By directly mapping the tail rotor to the yaw command. This applies to both tail rotors controlled by two servos and tail rotors with a dedicated motor.

The 130-blade helicopter mixer file is as follows:

```
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 steep, reaching 6000 (0.6).
- At 100% thrust, it reaches 10000 (1.0) with a smaller slope.

- The pitch curve is linear, but its entire range isn't used.
- At 0% throttle, the collective control stick setting is already at 500 (0.05).
- At maximum throttle, the collective control stick is only at 4500 (0.45).
- Using higher values for this type of helicopter will cause the blades to stall.
- The swashplate system of this helicopter is set at angles of 0, 140, and 220 degrees.
- Servo arm lengths are not equal.
- Compared to the first servo system, the arm lengths of the second and third servo system s are 1.3054.
- The servos are limited to -8000 and 8000 because of mechanical constraints.

## 8.8. VTOL Mixer—Vertical Takeoff and Landing (VTOL) Drone Mixer

The vertical take-off and landing system utilize a multirotor mixer as the output in multirotor mode and a sum mixer as the output in fixed-wing mode. The mixer system of the vertical take-off and landing UAV can be either combined into a single mixer, where all actuators are connected to IO or FMU ports, or divided into separate mixer files for IO and AUX.

# 9. PX4 Native Interfaces

PX4 is highly portable and serves as an operating system-independent solution for unmanned aerial vehicles (UAVs). Developed by world-class developers from both industry and academia, it benefits from active global community support, powering a wide range of vehicles from racing and cargo drones to ground vehicles and underwater vehicles. For more information, visit the official website: https://docs.px4.io/main/en/. Below are some commonly used uORB messages, modules, and parameters in the PX4 software system.

## 9.1. Getting Started with the PX4 Software System

Drones are unmanned "robotic" vehicles that can be operated remotely or autonomously, and they are widely used in many consumer, industrial, governmental, and military applications. These applications include but are not limited to aerial photography/videography, cargo transportation, racing, search and surveying, among others. Different types of drones are used in aerial, ground, marine, and underwater environments. These drones are referred to as Unmanned Aerial Vehicles (UAVs), Unmanned Aerial Systems (UAS), Unmanned Ground Vehicles (UGV), Unmanned Surface Vehicles (USV), and Unmanned Underwater Vehicles (UUV).

The "brain" of a drone is known as the autopilot system. It runs flight stack software on the vehicle controller ("flight controller") hardware. Some drones are also equipped with a separate onboard computer, providing a powerful platform for networking, computer vision, and many other task

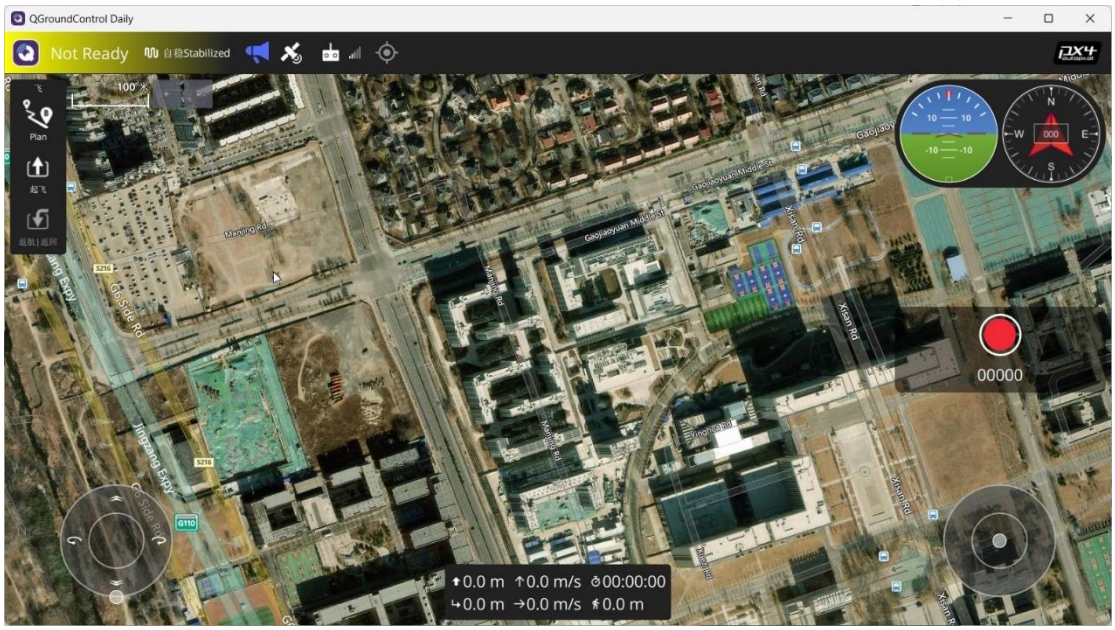
s.

PX4 is a powerful open-source drone flight control stack, with some key features including:

- Control a variety of vehicle frames/types, including drones (multirotors, fixed-wing aircraft, a nd VTOL aircraft), ground vehicles, and underwater vehicles.
- Offer a robust selection of vehicle controllers, sensors, and other peripheral devices.
- Provide flexibility and robustness in flight modes and safety features.
- Deeply integrate with onboard computers and robot APIs (such as ROS 2, MAVSDK).

PX4 serves as a core component of a broader drone platform, which includes QGroundContro l-based ground stations, Pixhawk hardware, and MAVSDK, integrating with companion com puters, cameras, and other hardware using the MAVLink protocol.

The ground control station for Dronecode is called QGroundControl. You can use QGroundC ontrol to flash PX4 onto vehicle control hardware, configure vehicles, change various parame ters, get real-time flight information, and create and execute fully autonomous missions. QGr oundControl runs on Windows, Android, MacOS, or Linux. Download and install it from her e.




PX4 supports aerial, ground, and underwater vehicles. You can view all types and variants ("f rames") of vehicles for PX4 testing/tuning here: Airframe Reference. Choose different vehicles bas ed on the type you need:

**Multirotors:** Offer precise hovering and vertical takeoff but have shorter range and typically fly slower. They have modes in PX4 that make them easy to fly and are the most popular type of fl ying vehicle.

**Helicopters:** Similar to multirotors but mechanically more complex and efficient.

**Fixed-wing aircraft:** Provide longer flight times and faster speeds, making them better for ap plications like ground surveys. However, they are more challenging to fly and land than multirotor s. They are not suitable if you need to hover or fly at very slow speeds (e.g., when surveying vertic al structures).

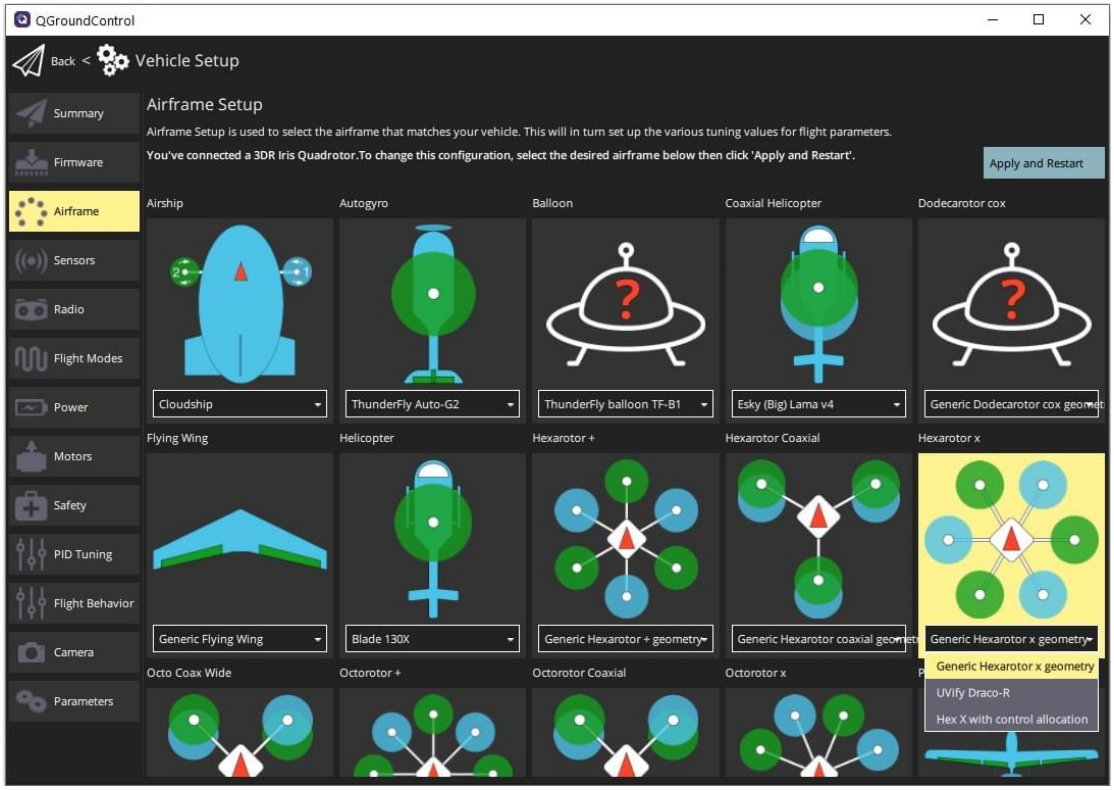
**Vertical Takeoff and Landing (VTOL) drones:** Come in various types such as tiltrotors, tail sitters, quadplanes, etc. They offer the dual advantage of vertical takeoff like multirotors and then t ransitioning to forward flight like airplanes. They are typically more expensive and harder to manu facture and tune than multirotors and fixed-wing aircraft.

**Balloons/Airships:** Are lighter-than-air flying vehicles that typically offer high-altitude, long -duration flights, usually with limitations (or no limitations) on range and speed control.

Rovers are ground vehicles similar to cars. They are easy to control and often fun.

**Unmanned boats:** Are surface vehicles.

**Unmanned underwater vehicles:** Are underwater vehicles.

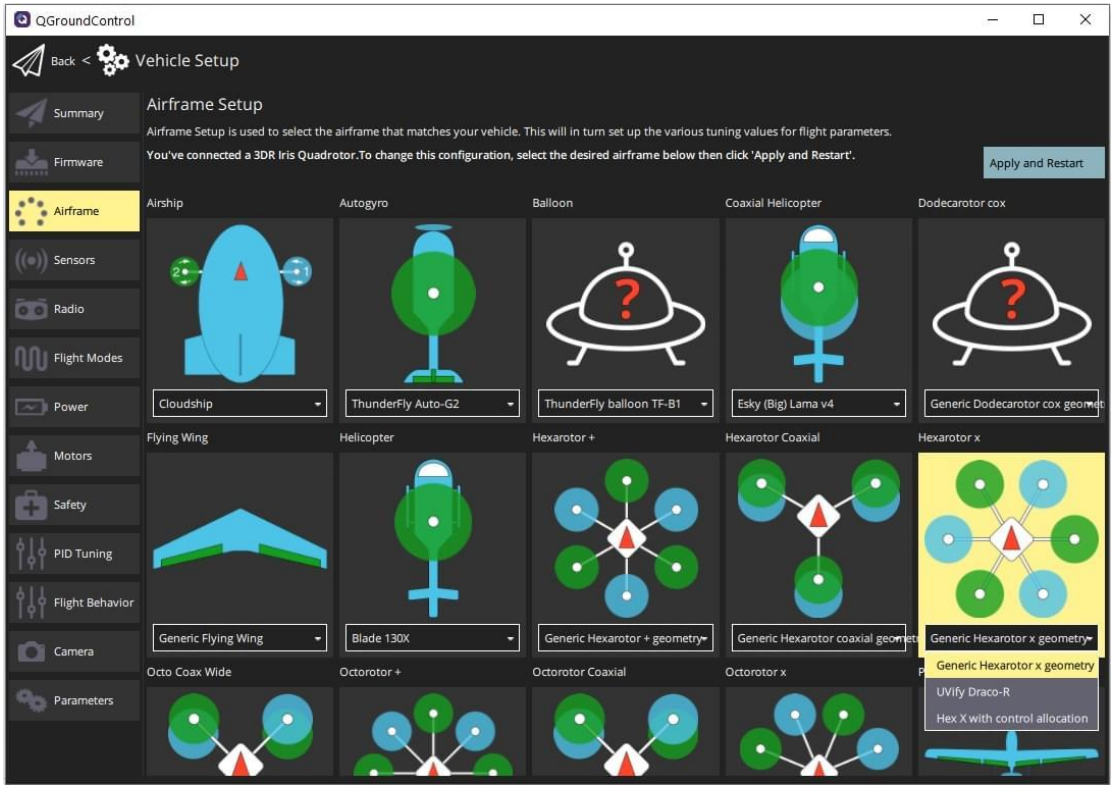



## 9.1.1. Firmware Download

After installing the RflySim platform, you can find the complete PX4 source code at the path: *\PX4PSP\Firmware. Open the WSL subsystem on the desktop (*\Desktop\RflyTools\Win10WSL. lnk) for compilation. After the compilation is complete, open the QGC software for firmware down load. Note: You can also directly download the firmware through QGC if there is an internet conne ction.

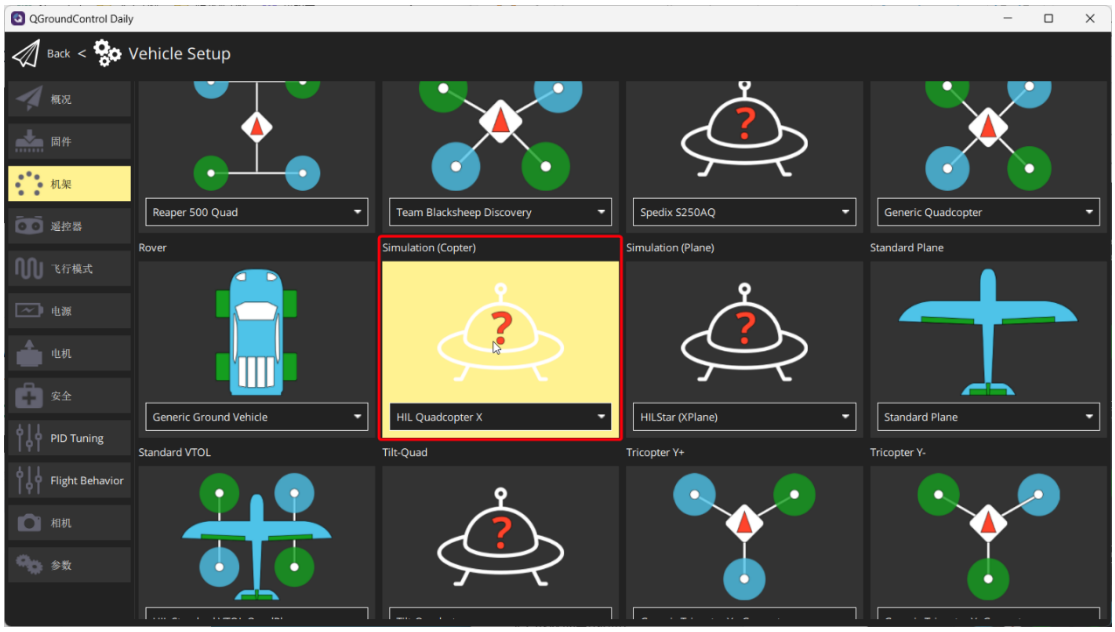
### 9.1.2. Model Configuration

After firmware flashing is complete, launch the QGroundControl software. Once connected to the flight controller, select the "Q" icon > Vehicle Setup > Airframe (sidebar) to open the airframe configuration. Choose the vehicle group/type that matches your frame, and then within the group, use the drop-down menu to select the frame most suitable for your vehicle. For example, if you are simulating quadcopter hardware in a loop, you can select the frame as follows: Simulation -> HIL Quadcopter X.

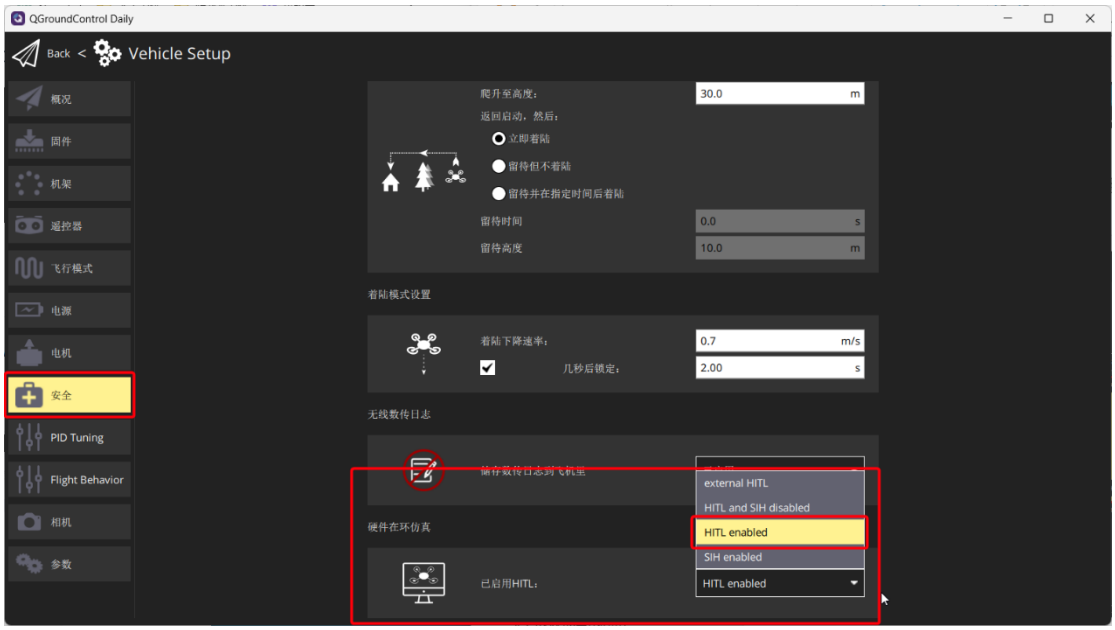


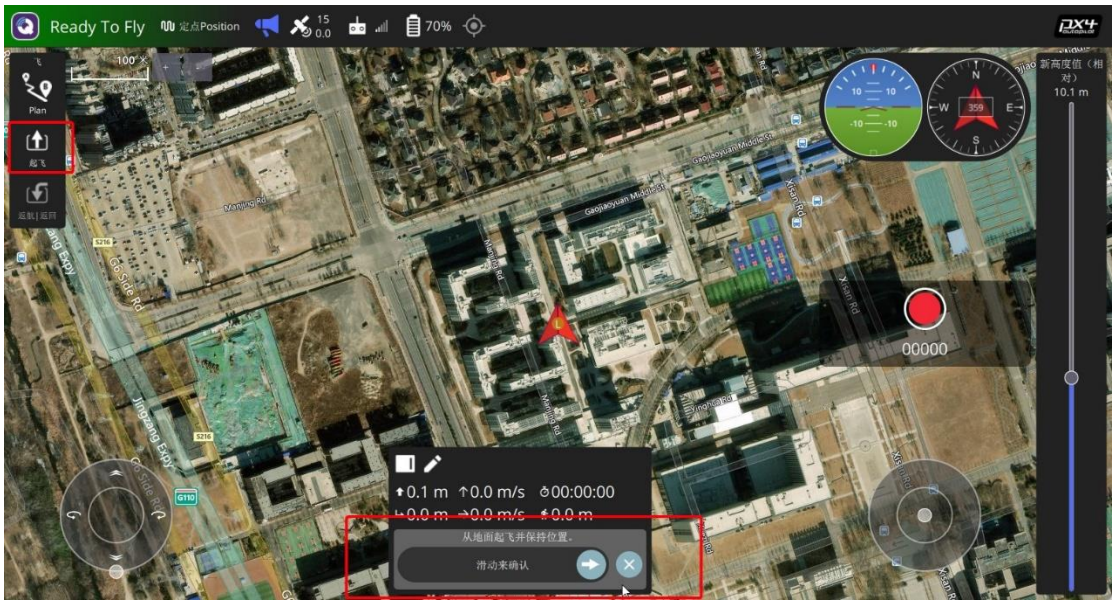

### 9.1.3. Hardware-in-the-loop (HIL) simulation

Connect the flight controller to the computer, open the QGroundControl ground station, and s elect the airframe as follows: Simulation -> HIL Quadcopter X. In the safety options, set: Safety -> Hardware-in-the-loop Simulation -> HITL enabled.




Open the "*\Desktop\RflyTools\HITLRun.lnk" to start the hardware-in-the-loop simulation script with a single click. Input the port number of the flight controller and wait until the message bar at the bottom left of CopterSim displays: "PX4: GPS 3D fixed & EKF initialization finished". Then, you can unlock and take off the aircraft in QGC.







### 9.1.4. Aircraft actual flight

Choose the airframe for your specific vehicle, for example: for a quadcopter, you can select th e airframe as follows: Quadrotor X -> DJI F450 w/ DJI ESCs. In the safety options, set: Safety -> Hardware-in-the-loop Simulation -> external HITL. For more detailed steps on actual flight, please refer to the experiment: 1.BasicExps\e5-AttitudeCtrl\e5.4\Readme.pdf。

## 9.2. PX4 Official Flight Controller Support Introduction

The flight controllers officially supported by PX4 are maintained by the PX4 Autopilot maint ainers and the Dronecode team. They comply with the Pixhawk standard. For more up-to-date info rmation, please visit: https://docs.px4.io/main/en/flight_controller/. The following table lists some commonly used flight controller hardware and compilation commands:

| Serial Number | Onboard Information | Hardware Name | Compilation Command |
|---|---|---|---|
| 1 | FMUv6X and FMUv6C | CUAV Pixahwk V6X (FMUv6X) | px4_fmu-v6x_default |
| 2 | | Holybro Pixhawk 6X (FMUv6X) | |
| 3 | | Holybro Pixhawk 6C (FMUv6C) | px4_fmu-v6c_default |
| 4 | | Holybro Pix32 v6 (FMUv6C) | |
| 5 | FMUv5 and FMUv5X (STM32F7, 2019/20) | Pixhawk 4 (FMUv5) | px4_fmu-v5_default |
| 6 | | Pixhawk 4 mini (FMUv5) | |
| 7 | | CUAV V5+ (FMUv5) | |
| 8 | | CUAV V5 nano (FMUv5) | |
| 9 | FMUv4 (STM32F4, 2015) | Pixracer | px4_fmu-v4_default |
| 10 | | Pixhawk 3 Pro | px4_fmu-v4pro_default |
| 11 | FMUv3 (STM32F4, 2014) | Pixhawk 2 | px4_fmu-v3_default |
| 12 | | Pixhawk Mini | |
| 13 | | CUAV Pixhack v3 | |
| 14 | FMUv2 (STM32F4, 2013) | Pixhawk | px4_fmu-v2_default |

## 9.3. Introduction to Commonly Used uORB Messages in PX4

| Serial Number | Name | Description | File Address |
|---|---|---|---|
| 1 | actuator_controls.msg | Driver control signal | *\PX4PSP\Firmware\msg\actuator_controls.msg |
| 2 | actuator_outputs.msg | Driver output signal | *\PX4PSP\Firmware\msg\actuator_outputs.msg |
| 3 | input_rc.msg | The remote control inputs a message | *\PX4PSP\Firmware\msg\input_rc.msg |
| 4 | led_control.msg | Controls single or multiple external LEDs | *\PX4PSP\Firmware\msg\led_control.msg |
| 5 | vehicle_status.msg | Vehicle status message | *\PX4PSP\Firmware\msg\vehicle_status.msg |
| 6 | vehicle_imu.msg | Message output from vehicle IMU | *\PX4PSP\Firmware\msg\vehicle_imu.msg |

For more uORB message descriptions, please refer to:https://docs.px4.io/main/en/msg_docs/

## 9.4. Introduction to Commonly Used Modules in PX4

| Serial Number | Name | Description | File Address |
|---|---|---|---|
| 1 | mc_rate_control | Multi-rotor speed control module | *\PX4PSP\Firmware\src\modules\mc_rate_control |
| 2 | mc_pos_control | Multi-rotor position control module | *\PX4PSP\Firmware\src\modules\mc_pos_control |

| | | | |
|---|---|---|---|
| 3 | mc_att_control | Multi-rotor attitude control module | *\PX4PSP\Firmware\src\modules \mc_att_control |
| 4 | navigator | Navigation control module | *\PX4PSP\Firmware\src\modules \ navigator |

For more uORB message descriptions, please refer to:https://docs.px4.io/main/en/

## 9.5. Introduction to Commonly Used Parameters in PX4

| 序号 | Name | Description |
|---|---|---|
| 1 | MPC_THR_MIN | Minimum thrust in automatic thrust control |
| 2 | MPC_THR_HOVER | Hover thrust |
| 3 | MPC_USE_HTE | Hover Thrust Source Selector |
| 4 | MC_ROLLRATE_P | Proportional gain of roll rate |
| 5 | MC_ROLLRATE_I | Integral gain of roll rate |
| 6 | MC_ROLLRATE_D | Differential gain of roll rate |
| 7 | MC_ROLLRATE_FF | Feed forward of roll acceleration |
| 8 | MC_ROLLRATE_K | Roll Rate Controller Gain, Global Gain of Controller |

For more uORB message descriptions, please refer to:https://docs.px4.io/main/en/

# 10. Reference Materials


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

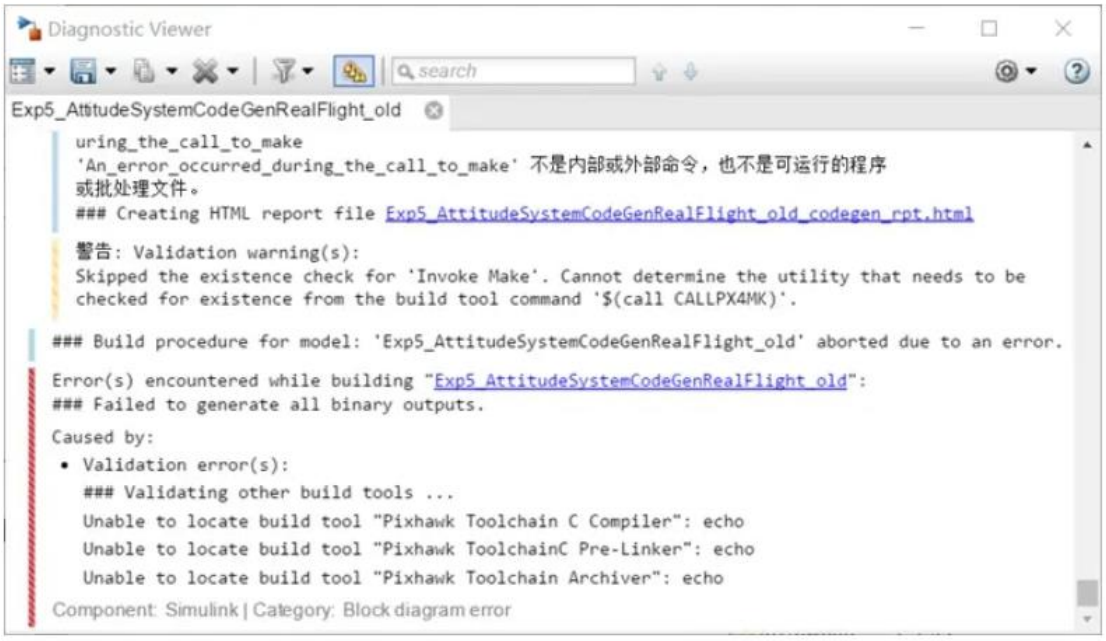
Quan Quan, Du Guangxun, Zhao Zhiyao, Dai Xunhua, Ren Jinrui, translated by Deng Heng, "Design and Control of Multirotor Aircraft" [M], Published by Electronic Industry Press, 2018.

[2]. 全权,戴训华,王帅.多旋翼飞行器设计与控制实践[M],电子工业出版社,2020.

Quan Quan, Dai Xunhua, Wang Shuai, "Design and Control of Multirotor Aircraft: Practice" [M], Published by Electronic Industry Press, 2020.

# 11. Common Questions and Answers

## 11.1. MATLAB/Simulink During automatic code generation, the following error is sometimes reported.

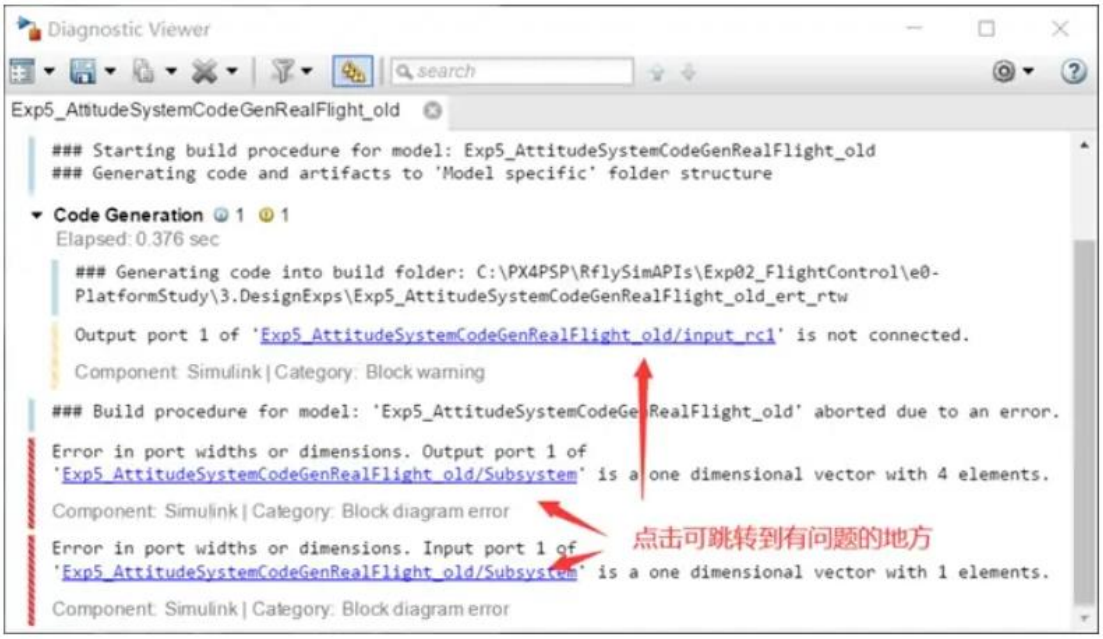



```
Caused by:
Validation error(s):
### Validating other build tools ...
Unable to locate build tool "Pixhawk Toolchain C Compiler": echo
Unable to locate build tool "Pixhawk ToolchainC Pre-Linker": echo
Unable to locate build tool "Pixhawk Toolchain Archiver": echo
```

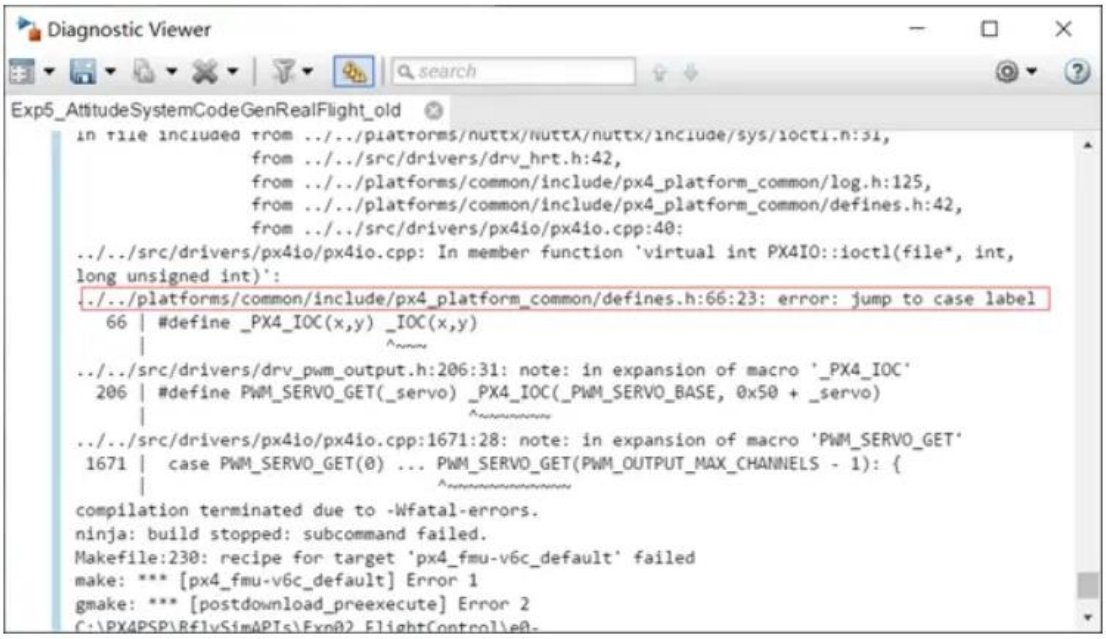
In case of compilation errors, possible compilation problems can be classified as: MATLAB model problems, PX4 firmware problems, MATLAB model and PX4 firmware linking problems.

**Questions and answers:**

Dealing with MATLAB model problems. MATLAB automatic code generation checks the model at the initial stage of compilation, so these kinds of problems are often displayed within seconds of clicking the compile button. The most common MATLAB problem is that the data of each interface does not match. Click the module that prompts the error to jump to the problem.

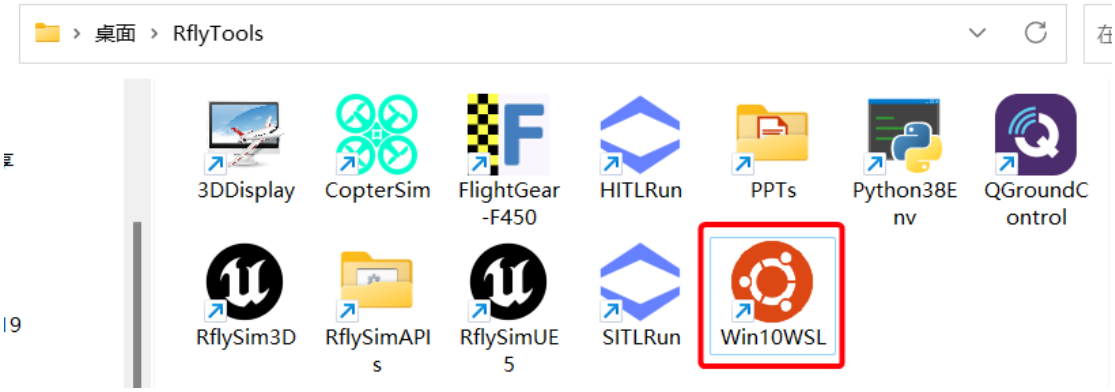



Address the PX4 firmware issue. If the PX4 source code has compilation problems, it will ge nerally be displayed in the compilation log prompt of MATLAB. The following figure shows the l ocation of the problem code. Modify it according to the prompt. The firmware for the PX4 in the p latform is located at PX4PSP \ Firmware.




Handles linking of MATLAB models to PX4 firmware. This kind of problem is often caused by the change of some interfaces due to the upgrade of PX4 firmware version, and the interfaces g enerated by MATLAB automatic code may not match, so errors will occur in the final link stage. T his kind of problem cannot see the specific error in MATLAB. You need to open Win10WSL (refer to other tools if other compilation tools are selected) and re-execute the following compilation co

mmands, such as `make px4_fmu-v6c_default` (change other versions to their corresponding co mmands) to see the specific problem.



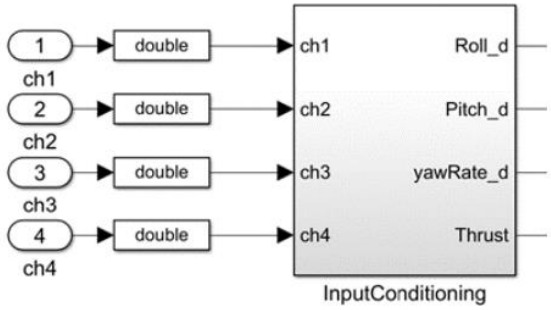

注意：在定位飞控编译问题时，应该保证平台是正确安装的，代码版本和编译命令能够相匹配。

## 11.2.In the automatic code generation controller, the delay module is used to directly generate control instructions, which causes the aircraft to fly around.

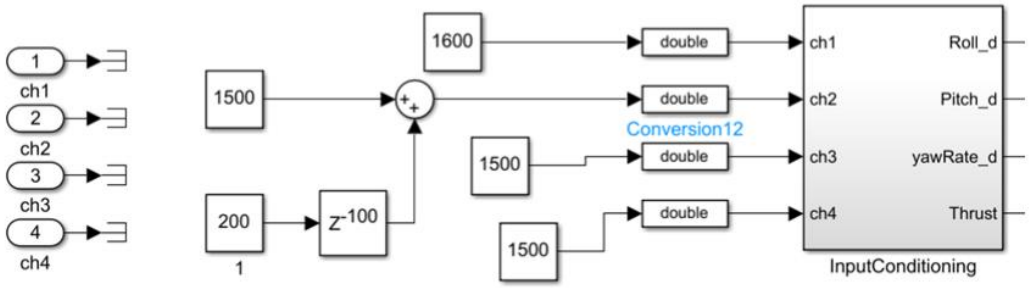
**Description of the problem:**

Routine: 1. BasicExps\e5-AttitudeCtrl\Readme.pdf disconnect the remote controller inputs CH1-CH4 from the inputConditioning module, and change the input signals of roll (CH1), pitch (CH2), throttle (CH3) and yaw (CH4) to the desired fixed value inputs. Only the AttitudeControl _ HIL. Slx is modified in this place, and then the hardware-in-the-loop simulation is carried out with the same operation steps as the original AttitudeControl _ HIL. Slx routine. After the hardware-in-the-loop is started, the aircraft itself flies on the ground, and an error will be reported after a while. What is the reason? How to solve it?

修改前：

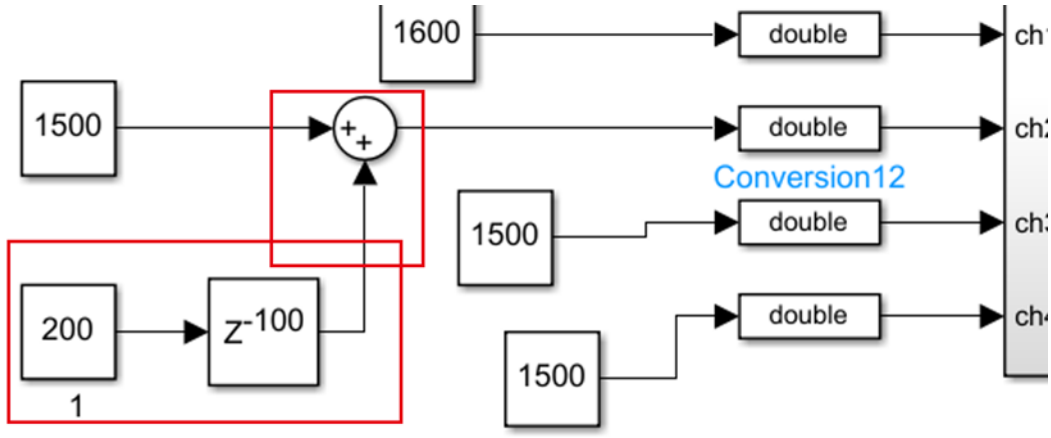



修改后：




Simulation phenomenon: After the change, the UAV flies on the ground during the hardware-in-the -loop simulation.

This modification is not feasible because the total delay of 200 steps is only 200 * 0.001 = 0.2 seco nds. Because the px4 _ simulink _ app program starts to run when it is powered on, it is equivalent to inputting control instructions after powering on, and it becomes a full throttle state. In this case, the flight control has not been initialized, the filter triggers the divergent state, and the accurate po se information is not obtained.




Perfect solution: directly read the uORB message of the EKF state, judge that the filter is initialize d, and then delay a period of time to give control instructions. Simple solution: extend the delay ti me a little more, or directly use MATLAB function to write a trigger mechanism, and then give co ntrol instructions after the simulation time reaches 60s.

## 11.3.SIL Or HIL When simulating, RflySim3D Appear Fatal error: [File:D://Build/++UE4....]... Report an error

The specific error reporting interface is as follows:


**Questions and answers:**

The above RflySim3D error may be due to the compatibility problem caused by the graphics card driver of the computer. It is recommended to upgrade the graphics card driver to the latest ver sion and see if it can be solved. If it cannot be solved, please contact the relevant after-sales person nel of the RflySim platform.

## 11.4.How to do UAV attitude autonomous control?

**Description of the problem:**

The detailed description is: we need to let the UAV fly to a certain height and fix it, and then l et the UAV move autonomously according to the input of the expected attitude angle given by ours elves, so that the three attitude angles can move to the given expected value.

**Questions and answers:**

The channel of the remote controller can be used to transmit trigger information. For example, when CH5 is dialed to the full position, the program starts to execute automatically. Write a state machine in Simulink to let the aircraft take off first, and then control the attitude after reaching the altitude. The specific design method of the controller belongs to the category of Simulink program ming, which can be understood by referring to relevant literature. If you want to input the informat ion such as the expected attitude angle from the outside after switching the fixed height, please use 0. ApiExps\9.PX4CtrlExternalTune\Readme.pdf the external control interface in this directory to s ubscribe to the rfly _ ctrls message (transmitted from the external program) in the flight control as the expected attitude angle.

## 11.5.How to get the result data of attitude control hardware in the loo p? Do I know how to download the flight log to get what I want?

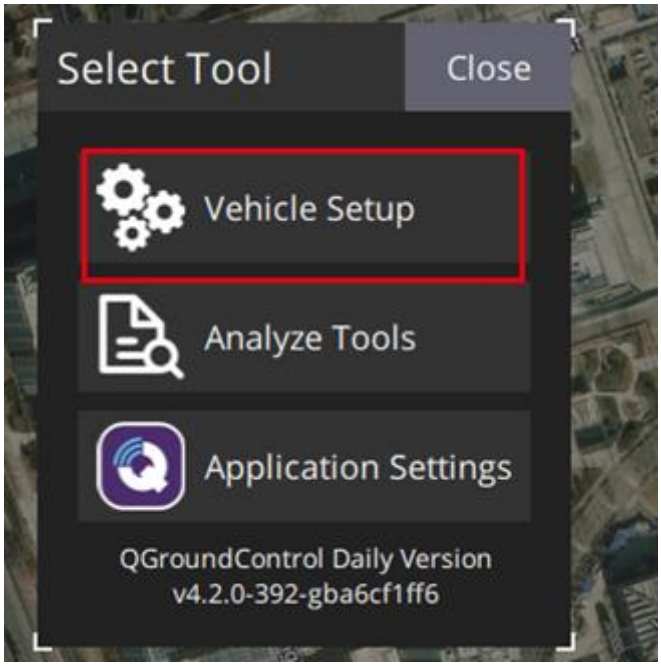
**Questions and answers:**

Please use 0. ApiExps\9.PX4CtrlExternalTune\Readme.pdf the interface of the routine to obta in the real attitude (simulation value) of the model in Simulink in real time and the data from the fl ight control (fill in the flight control attitude). Similar data can also be obtained from the flight log.

## 11.6.QGC Yes Analyze Tools- Flight log, after refreshing when down loading, I can't find the log of the time corresponding to the hard ware in the ring.

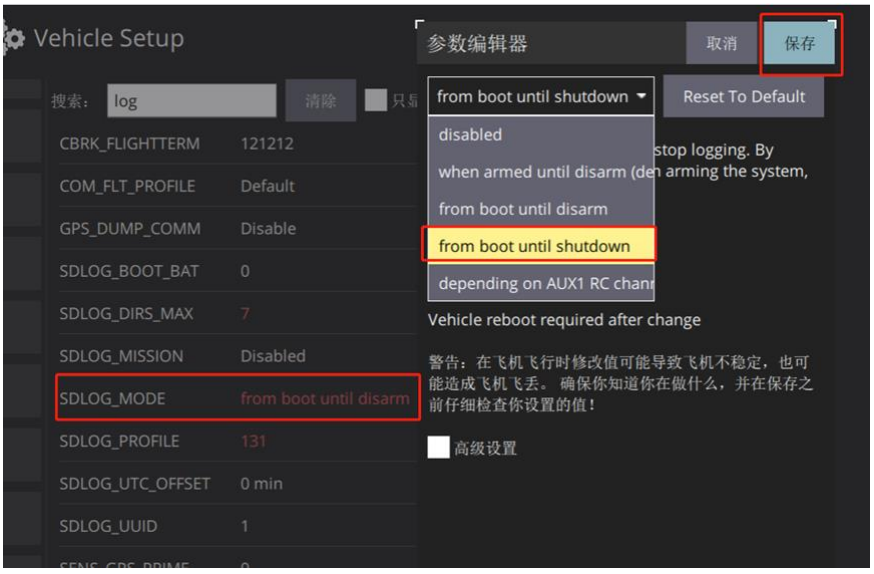
**Questions and answers:**

Open the QGC and enter the vehicle setup



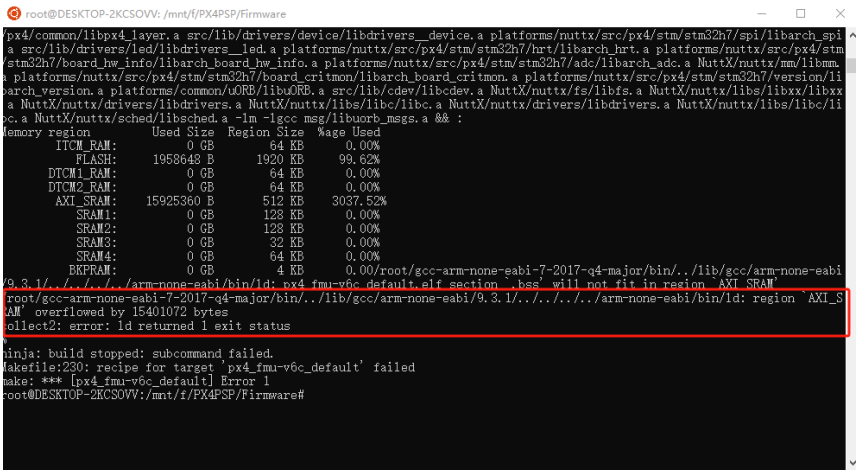


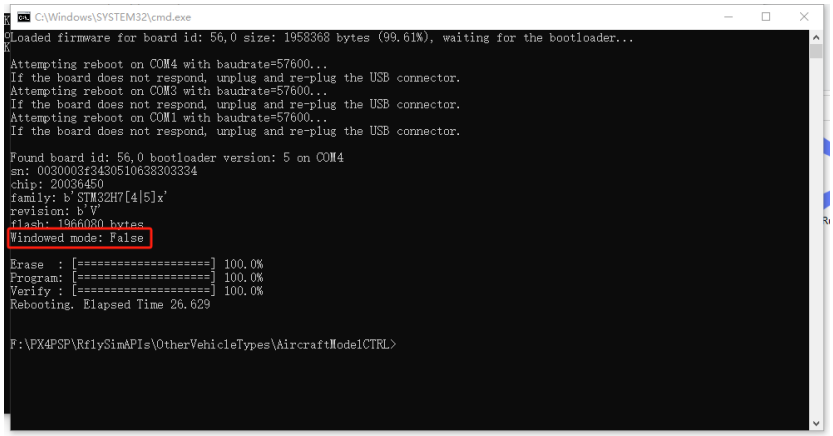
Go to the parameter label at the bottom, search for "log", find the parameter of the SDLOG _ MODE, change it to the following figure, record the log from power-on to power-off, and then you can see the log.



## 11.7. Win10WSL When compiling the firmware, displays: region `AXI_SRAM' overflowed by 15401072 bytes

**Description of the problem:**




**Questions and answers:**

The problem is that the compiled firmware is larger than the content of the flight control, causing the firmware to overflow and report an error. You can enter the Cmake file of PX4 to comment out some unused modules. The specific address of Pixhawk series flight control is: * PX4PSP \ Firmware \ boards \ px4. For example, the experiment conducted is a quadrotor related low-level control algorithm development experiment, the flight control used is Pixhawk 6C, and the unused modules in C: \ PX4PSP \ Firmware \ boards \ px4 \ fmu-v6c \ default. Cmake can be annotated (as follows). And then compile.

```
……
        MODULES
                airspeed_selector
                px4_simulink_app
                attitude_estimator_q
                camera_feedback
                commander
                dataman
```

```
        ekf2
        esc_battery
        events
        flight_mode_manager
        # fw_att_control       注释掉固定翼姿态控制模块
        fw_pos_control_l1
        gyro_calibration
        gyro_fft
        land_detector
        landing_target_estimator
        load_mon
        local_position_estimator
        logger
        mavlink
        mc_att_control
        mc_hover_thrust_estimator
        mc_pos_control
        mc_rate_control
        #micrortps_bridge
        navigator
        rc_update
        rover_pos_control
        sensors
        sih
        temperature_compensation
        # uuv_att_control      注释掉 UUV 姿态控制模块
        # uuv_pos_control      注释掉 UUV 位置控制模块
        vmount
        # vtol_att_control      注释掉 VTOL 姿态控制模块
……
```