API description file retrieval outline

Catalogue

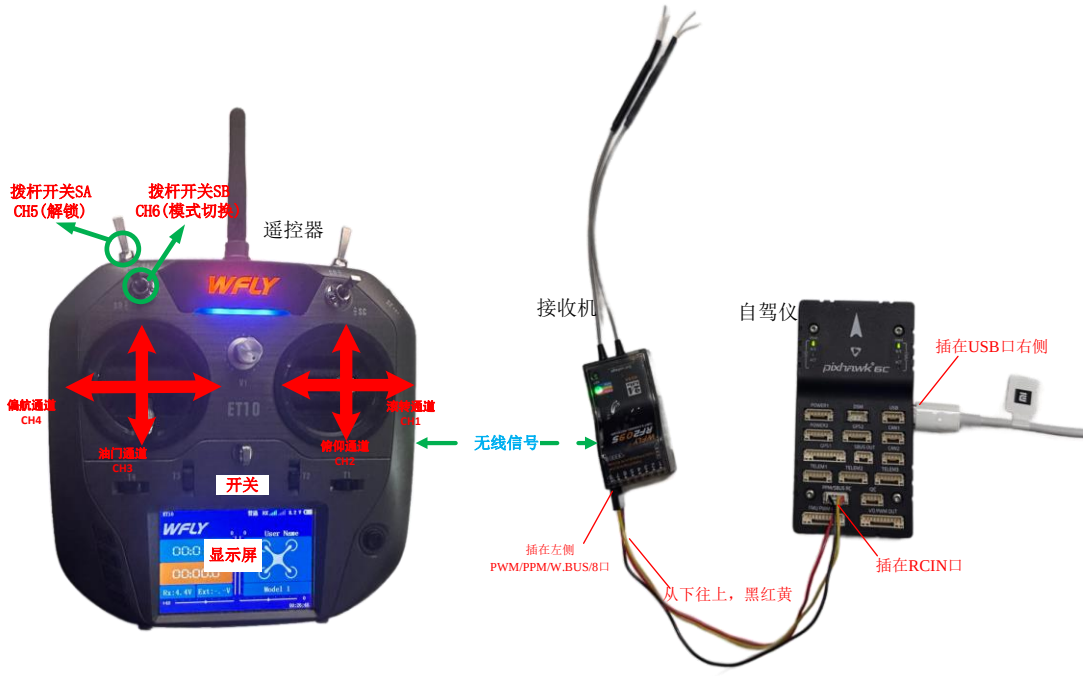
1. RflySim platform control hardware interface

1.1. Remote control

1.1.1. Real remote control

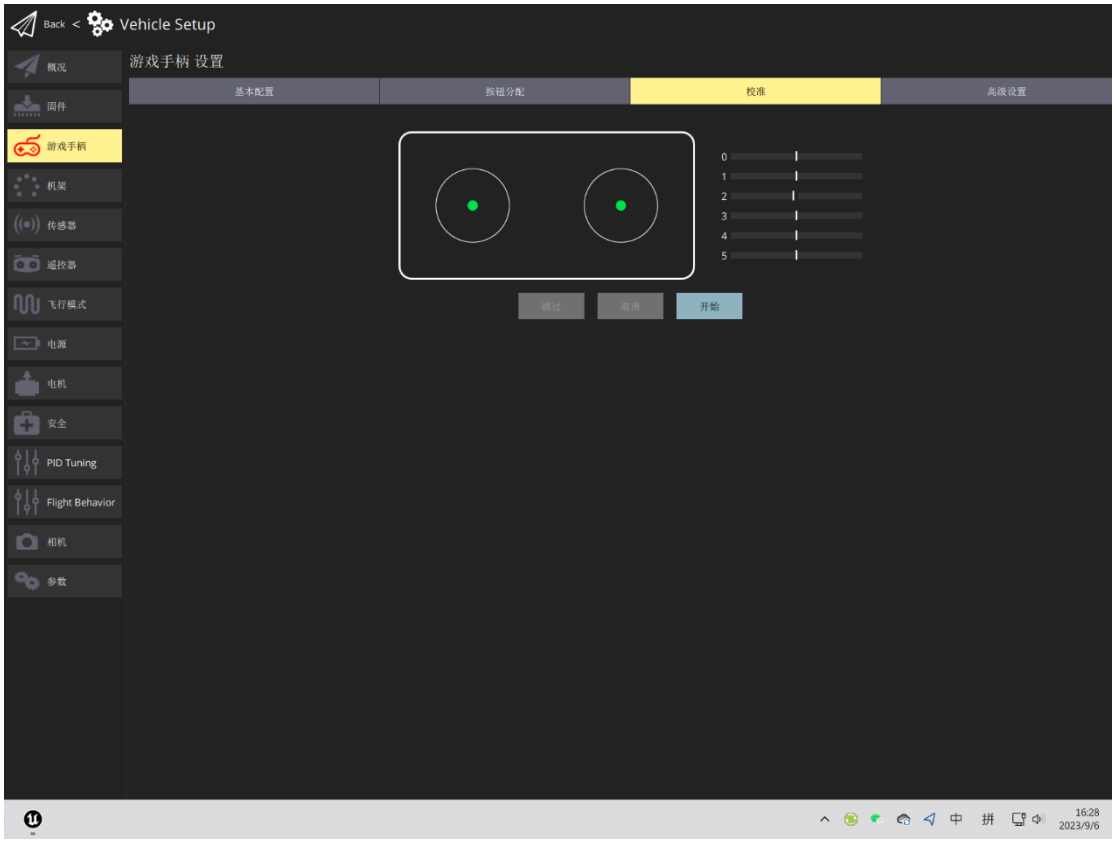
The remote control used in this platform is recommended to use the "American hand" control mode, that is, the left rocker corresponds to the throttle and yaw control amount, while the right rocker corresponds to the roll and pitch. The roll, pitch, throttle and yaw in the remote control correspond to the CH1~CH4 channel of the receiver respectively, and the left and right upper side lever corresponds to the CH5/CH6 channel to trigger the flight mode switch.

The throttle rod (CH3 channel) corresponds to the PWM signal fluctuation from 1100 to 1900 from the bottom end and the top end respectively (different channels or different remote controls will be different, so need to be calibrated); Roll (CH1 channel) and yaw (CH4 channel) rockers from leftmost end to rightmost end correspond to PWM signals from 1100 to 1900; The pitch (CH2 channel) rocker corresponds to the PWM signal from 1900 to 1100 from the bottom to the top; The CH5 is a two-stage switch with the corresponding PWM signals of 1100 and 1900. The CH6 is a three-segment switch with the corresponding PWM signals of 1100, 1500 and 1900. More remote control related configuration please see: *\PX4PSP\RflySimAPIs\2.RflySimUsage\1.BasicExps\e11_RC-Config\Readme.pdf



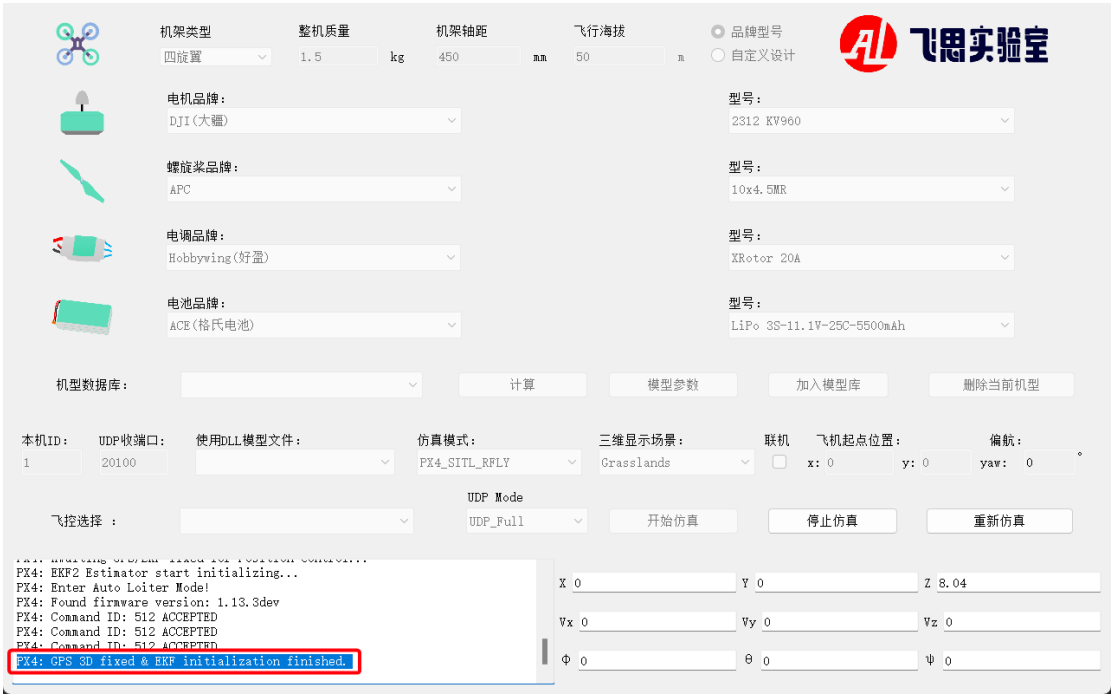
1.1.2. QGC+ remote control USB control

The remote control is generally through the receiver + flight control and then linked to the computer, you can then conduct relevant experiments on the simulation computer, but the remote control that supports the game controller can be linked to the simulation computer and the remote control through the USB cable, such as: The FS-i6s can directly connect the QGC of the simulation computer through USB cable. Before the first simulation, the remote control needs to be calibrated. Connect the simulation computer through USB cable, open the QGC software in the left gamypad, click "Calibration", and calibrate according to the figure.

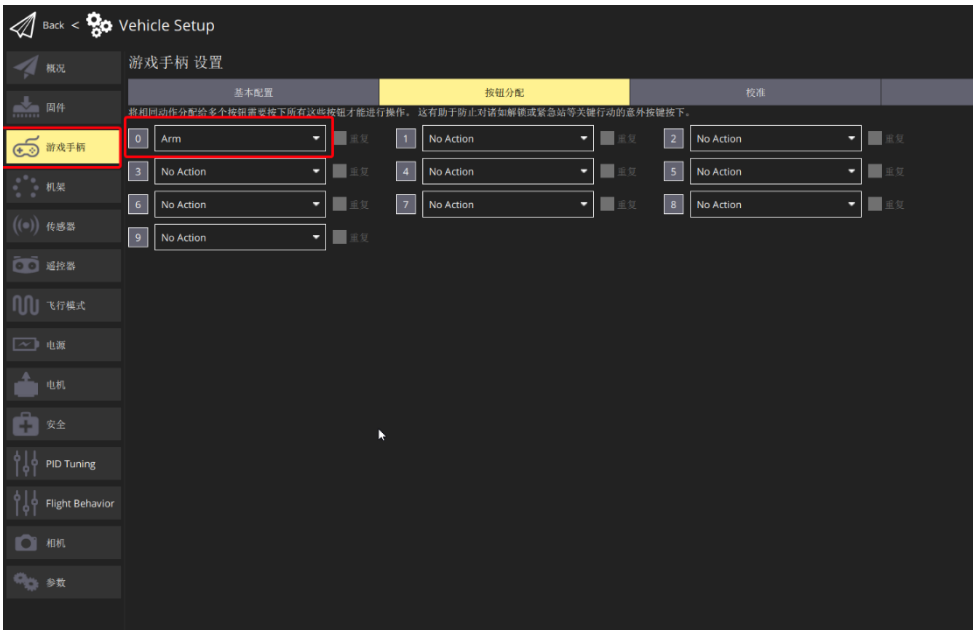


The detailed steps are as follows:

1、Double-click the SITLRun file, start the one-click software in the ring simulation script, Enter "1" in the pop-up CMD dialog box, press the Enter key, wait for the RflySim platform to start CopterSim, RflySim3D and QGC software, wait for the CopterSim message box to display: PX4: GPS 3D fixed & EKF initialization finished.



2、Connect the computer via USB cable, open the button allocation of the gamepad, and perform the following Settings:

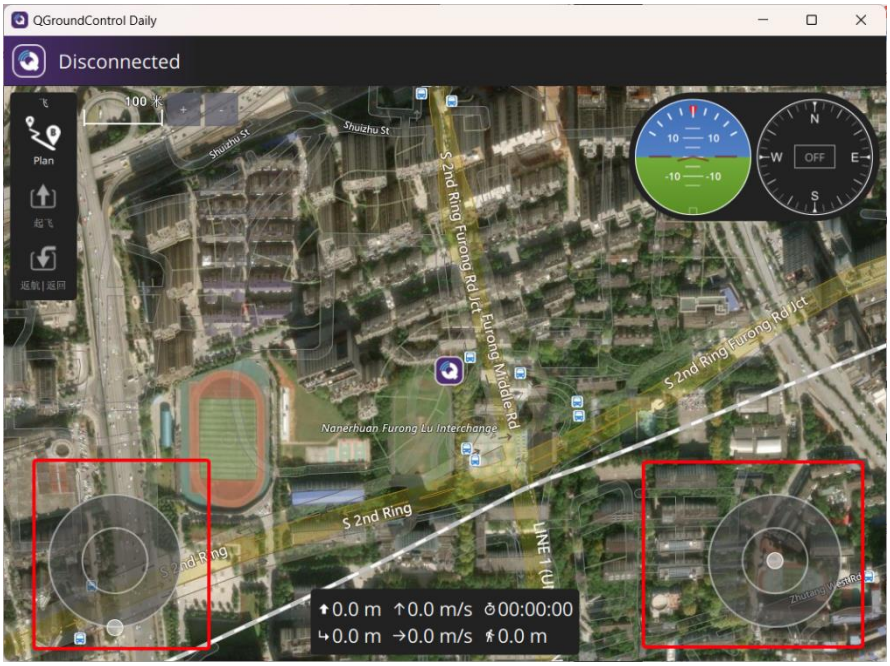


3、Unlock via SWB/CH5 channel and watch the aircraft take off in RflySim3D.

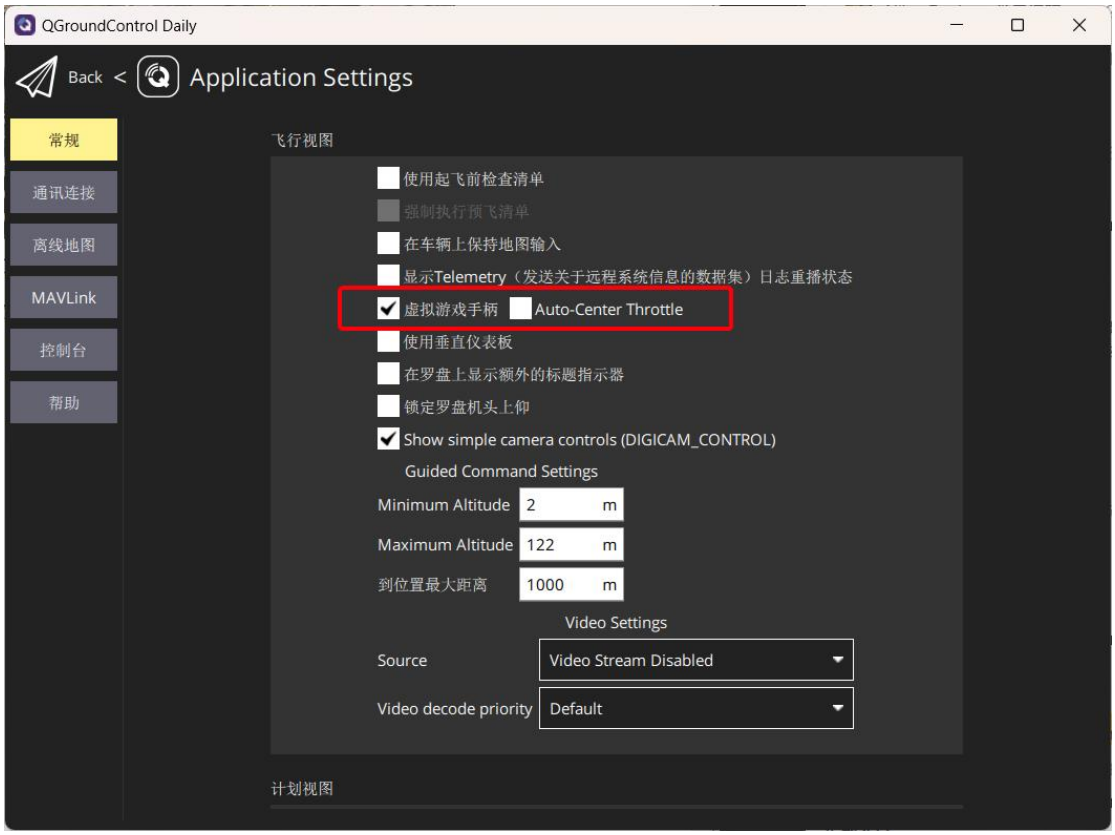
Note: Set COM_RC_IN_MODE=1 - Joystick/No RC Checks.

1.1.3. QGC virtual remote control

QGroundControl allows you to control the vehicle using a virtual thumb on the scree n. They appear as follows in flight view:



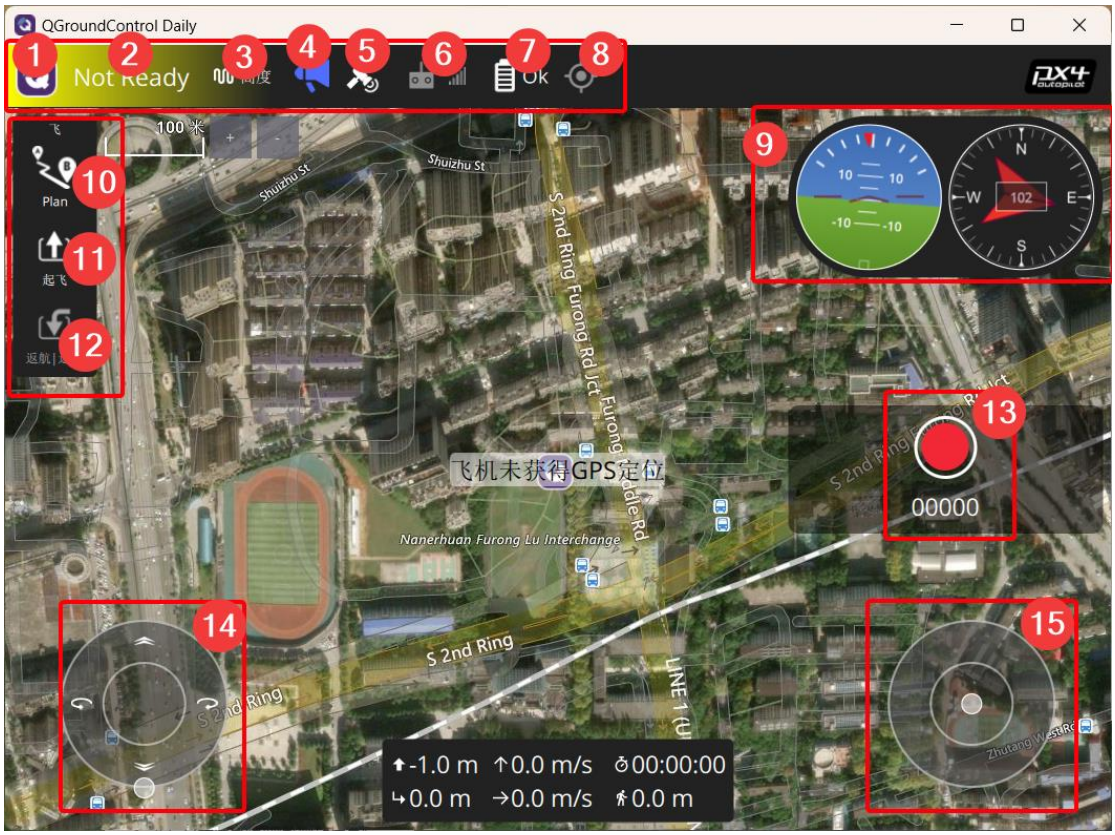
The response of the virtual joystick control is not as good as using the RC transmitte r (because the information is sent via MAVLink). Another option is to use a USB joystic k/game pad. Enable the virtual Joystick: 1. Select the Q icon ->Application Settings from the top toolbar, then select General from the sidebar. 2. Select the virtual gamepad check box.



1.2. Ground station control

1.2.1. Parameter Settings

The overall interface of QGroundContrl is shown in the following figure, and the rele vant explanations of each button in the interface are shown as follows:



① Start button: This button displays a shortcut menu for vehicle initialization Settings, a nalysis tool usage, and related software property Settings.

② Vehicle status display: generally, the overall state of the vehicle can be quickly viewe d from here.

③ Control mode selection: The button can switch different control modes, such as: man ual, self-stabilizing, stunt and so on.

④ Notification: Here you can view the information when the vehicle is running, such as: warning information, error information, etc.

⑤ GPS status: shows the number of satellites that the current vehicle can search.

⑥ Handle link status display.

⑦ Battery power display.

⑧ ROI region identification.

⑨ IMU Status real-time dashboard.

⑩ Route planning.

⑪ Take-off button.

⑫ Return button.

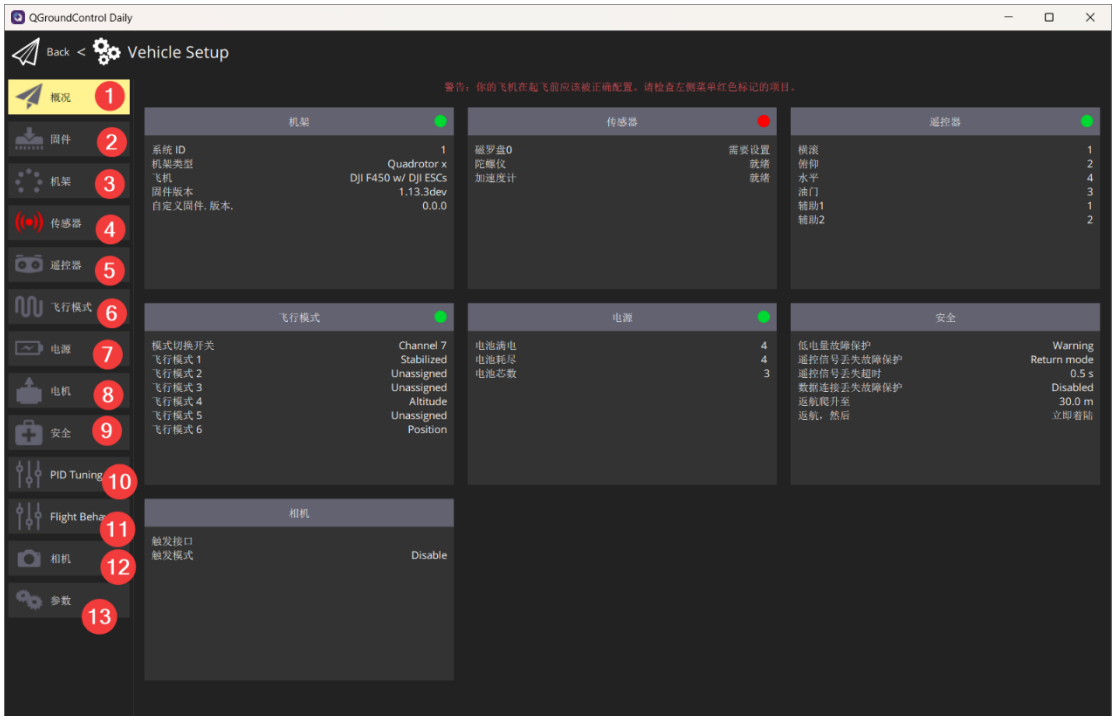
⑬ Record button: can record QGC interface video.

⑭ Virtual Handle CH3/CH4 channels.

⑮ Virtual handle CH1/CH2 channel.

1.2.2. Key commands (take-off, landing, return) (Vehicle Setup)

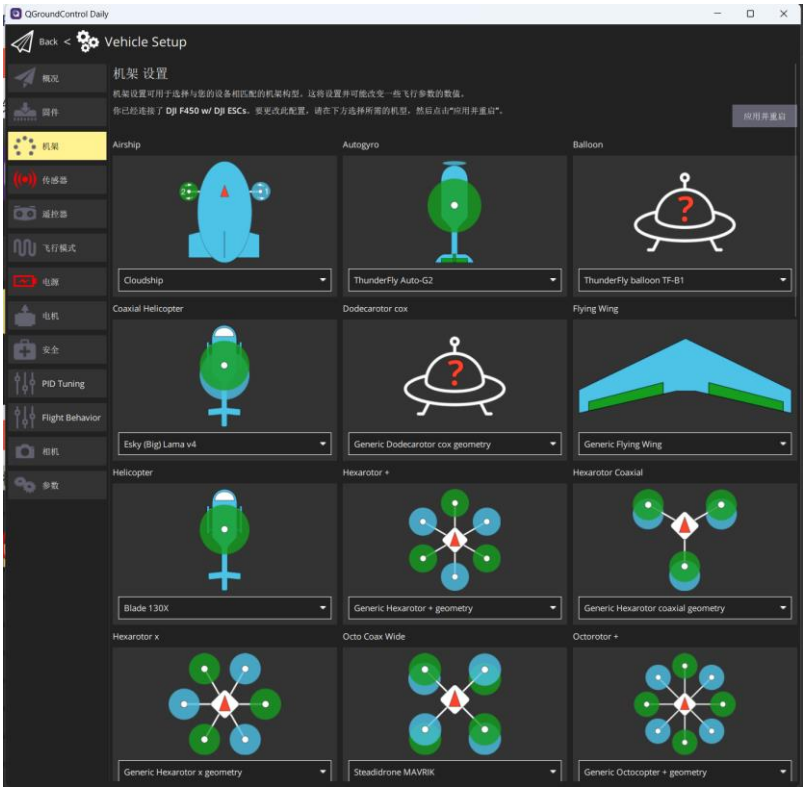
The following key commands are displayed on the initial interface of the QGC ground station https://docs.qgroundcontrol.com/master/en/qgc-user-guide/setup_view/setup_view.html



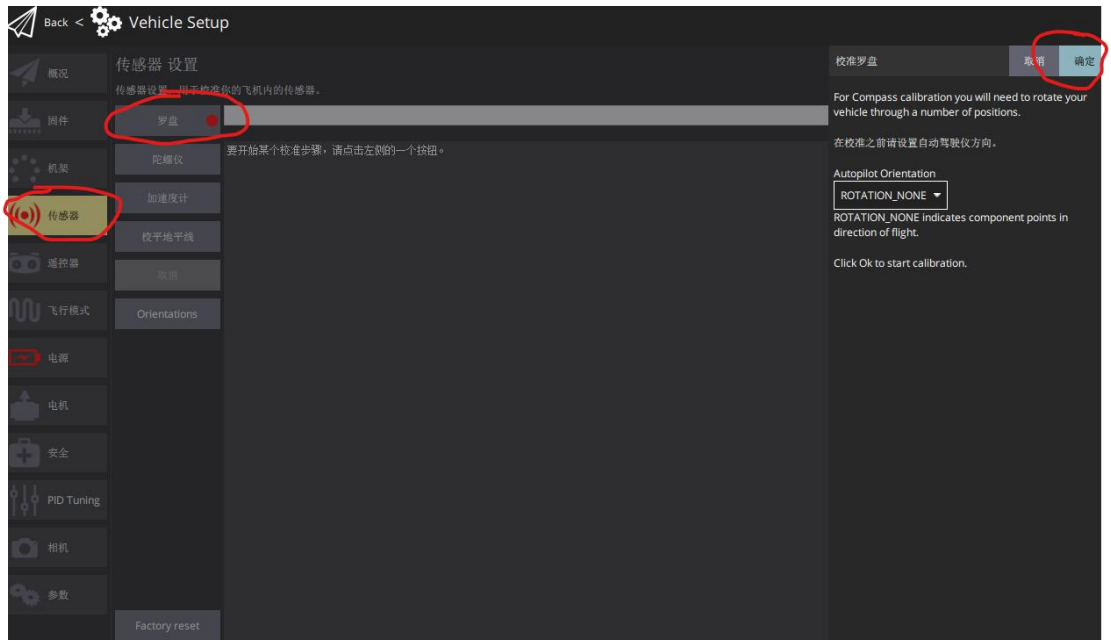
① Vehicle overview: Displays the overall status of the currently connected vehicle, such as: rack, sensor, remote control, flight mode, etc.

② Firmware: Do not connect to the flight control, click the following page, and then use USB to connect to the computer, pay attention to the flight control do not use batteries or other devices other than USB power supply.

③ Rack: Connect the flight control to the ground station, set the rack to the model you want to set, after setting, please "apply and restart" in the upper right corner to take effect.

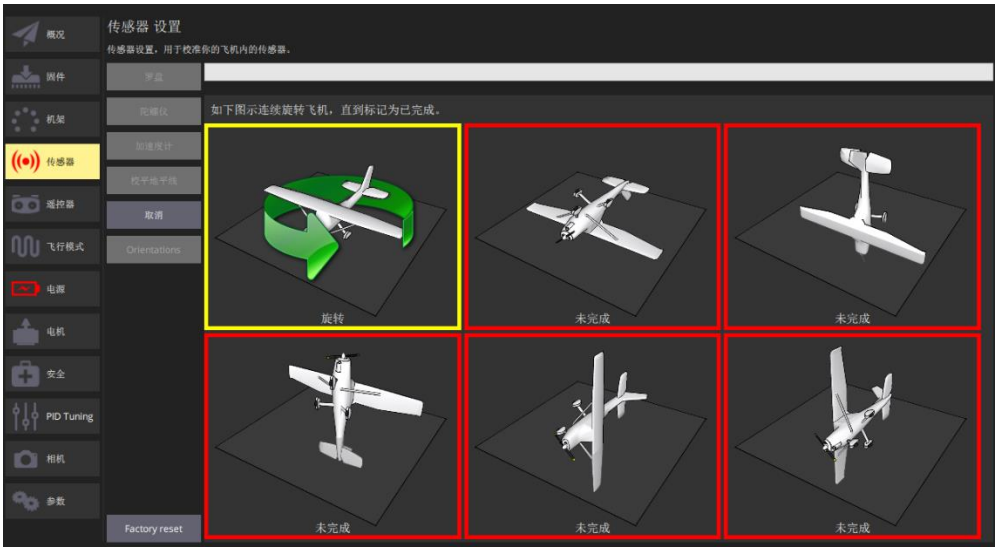


④ Sensor: The sensor mainly includes the sensor involved in the IMU, when calibrating, generally calibrate the compass first, the steps are as follows:



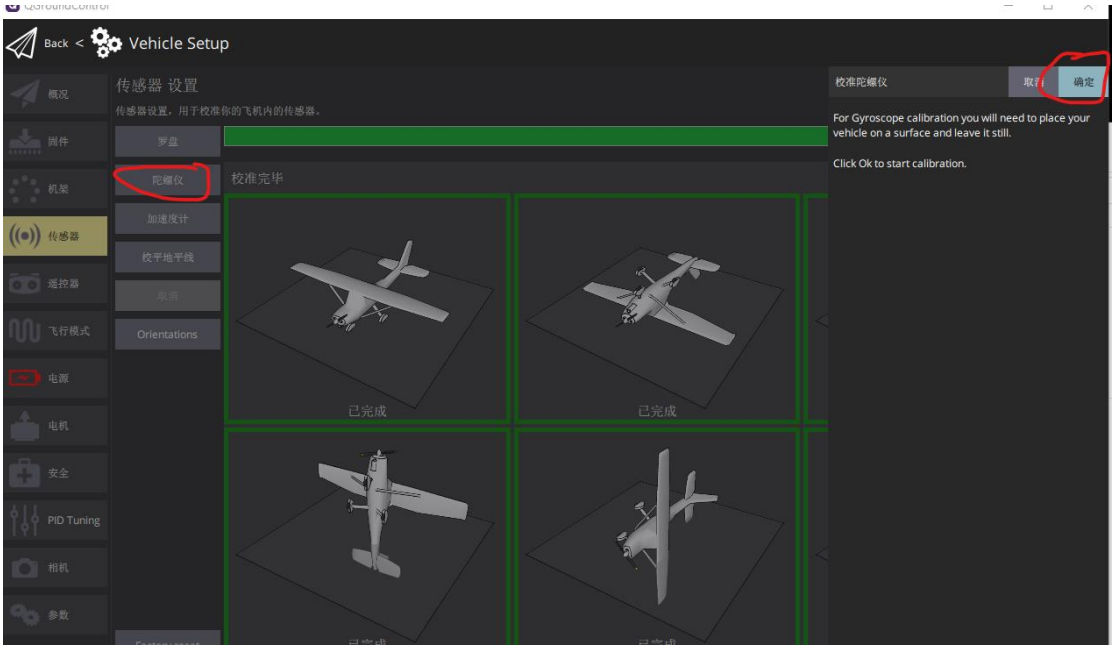
Position the drone in any direction shown in red and remain stationary. When prompt ed (the directional image turns yellow), rotate the vehicle around the specified axis in

either/both directions. When the current orientation calibration is complete, the relevant image on the screen will turn green.



Repeat the calibration process for all directions. When all orientations are calibrated, QGroundControl will display Calibration complete (all directional images will appear green and the progress bar will be fully filled). You can then move on to the next sensor.

Calibrating the gyroscope: Click the gyroscope sensor button to place the drone horizontally on the ground and remain stationary. Click OK to start the calibration. A full bar at the top indicates successful calibration.

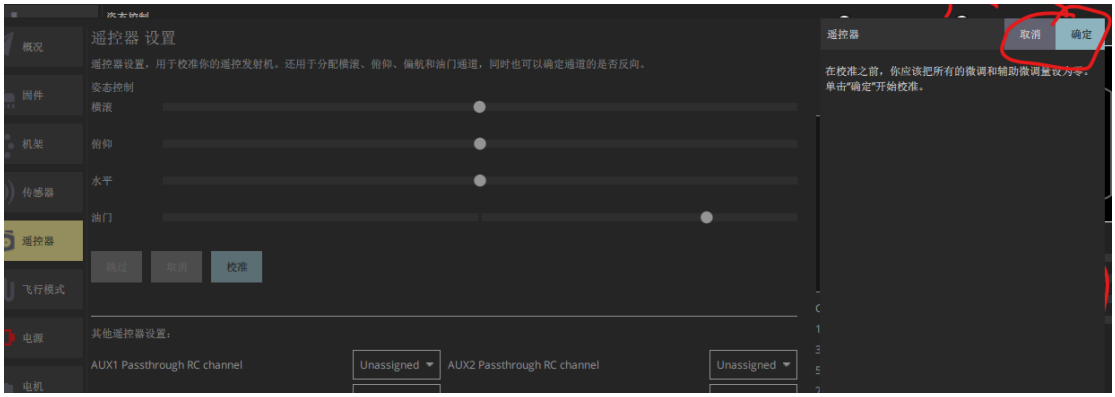


⑤ Remote control

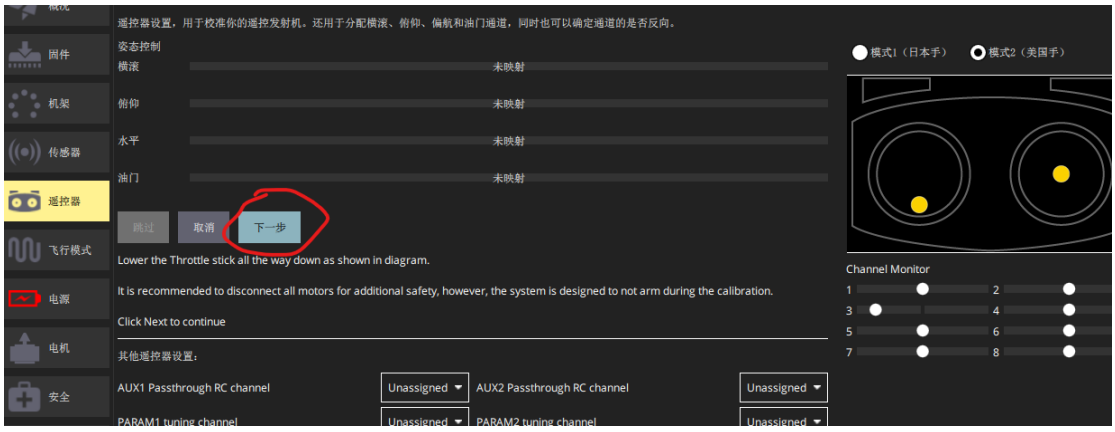
Open the remote control, switch to the remote control page, check whether the chann

el can be identified in the lower right corner, if it can be identified, you can calibrate, select the operation mode in the upper right corner, and then click Calibration

And then click "OK"

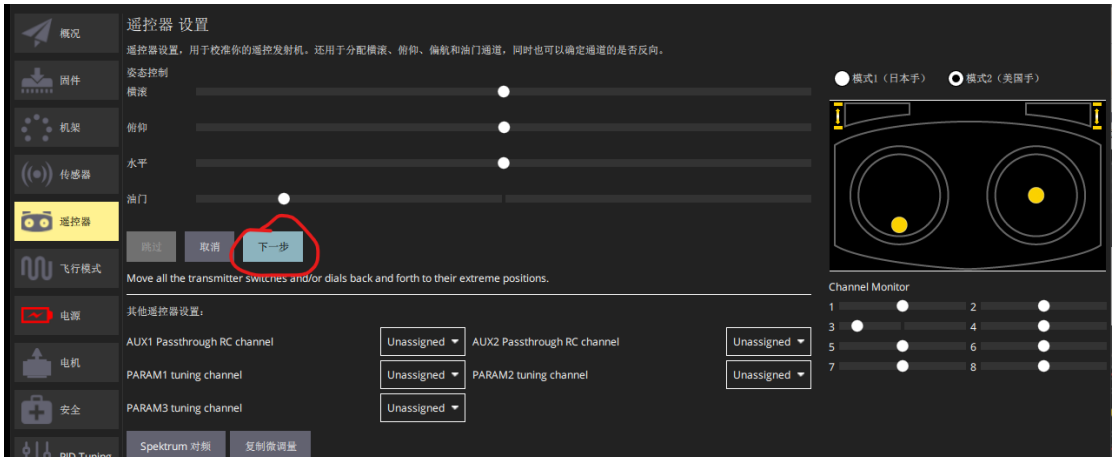


Then click "Next"



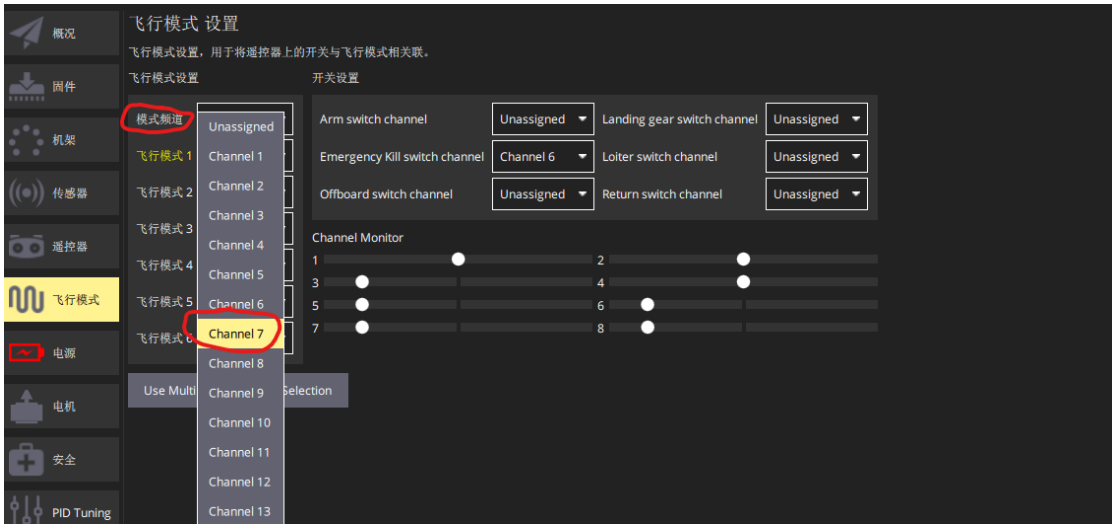
Move the remote control stick to the position indicated in the image below.

When the rod is in place, the ground station will prompt the next position to be dialed, and after dialing all positions, press "Next" twice to save the Settings.



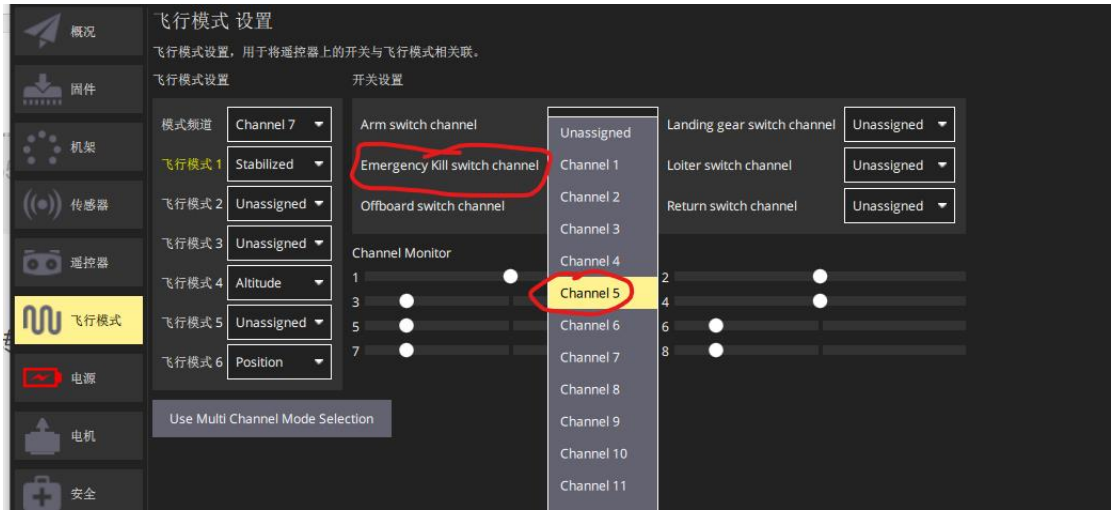
⑥ Flight mode switch

Click the check box to the right of "Mode channel" to set the corresponding remote control dip switch channel.



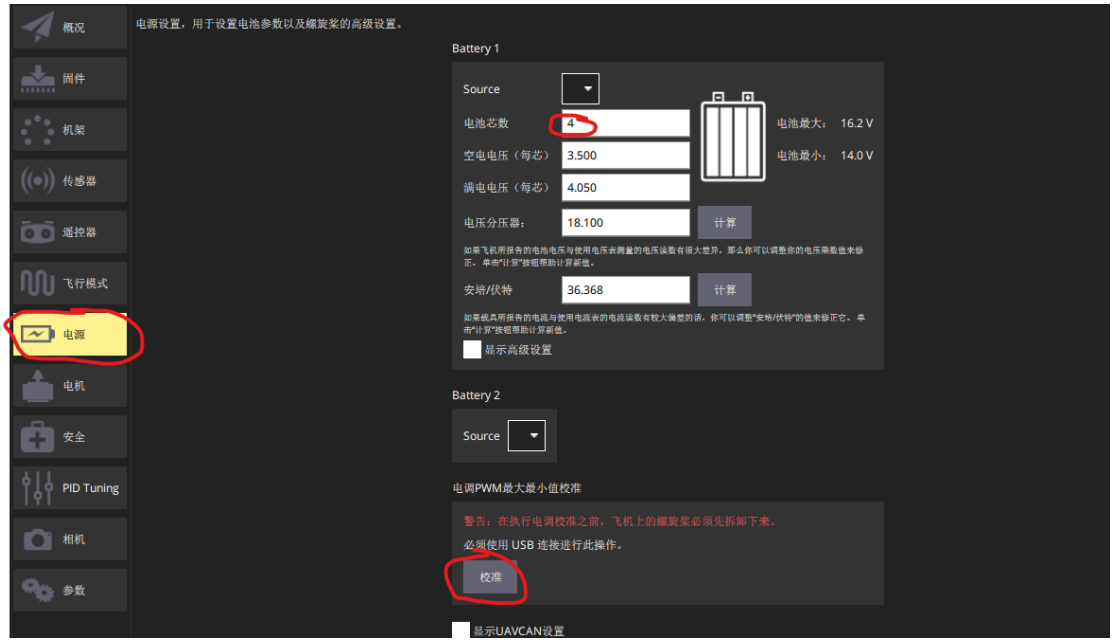
Then set the flight mode for each of the three gears.

Other switch channels are on the right side of flight mode, as follows. If you need t o set which one, you can set the remote control channel on the right side of this switch. I have set a Kill switch here, and the channel is the fifth channel of the remote control. The function of the brake is to stop the motor directly, which can be set as required



⑦ Power supply

When calibrating the electrical control, use USB to connect the flight control to the g round station, no battery, no pulp blade, and the signal line of the electrical control is co nnected to the flight control. Switch to the "Power" page, enter the number of battery cor es and press enter, click "Calibrate", and then plug in the battery to calibrate.



⑧ Motor: Display PWM of the motor

⑨ Safety: Under this menu, you can set the vehicle's low power fault protection trigger, object detection, remote control signal loss fault.

⑩ PID tuning: the PID control parameters of the vehicle can be adjusted.

⑪ Flight Behavior.

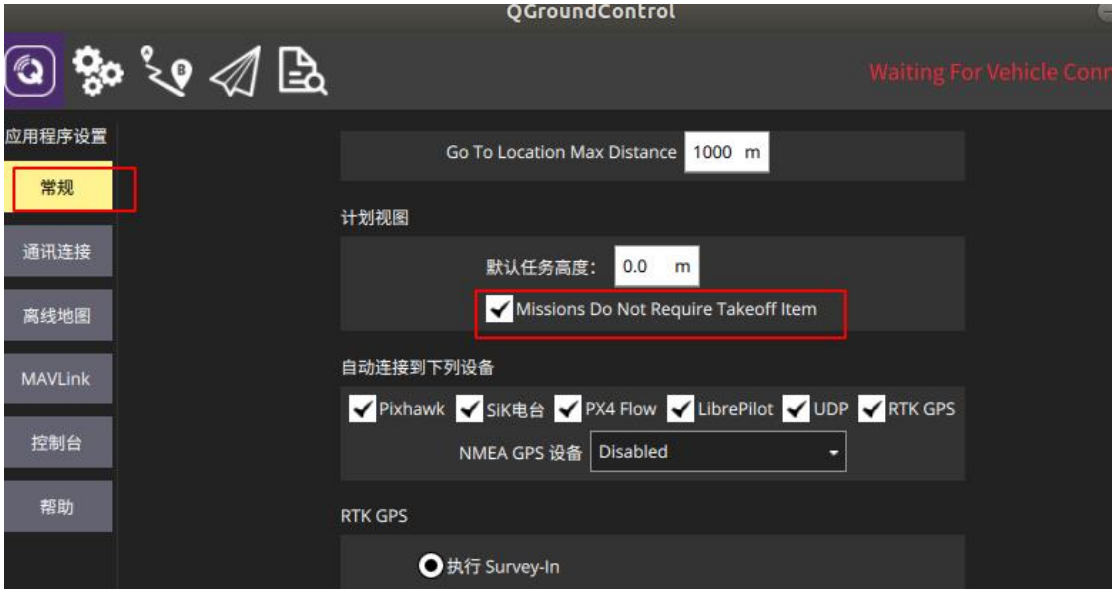
⑫ Camera settings

⑬ Any parameter defined in the PX4 software can be modified here, and the modificati on takes effect only after the restart.

1.2.3. Route Planning (Plan)

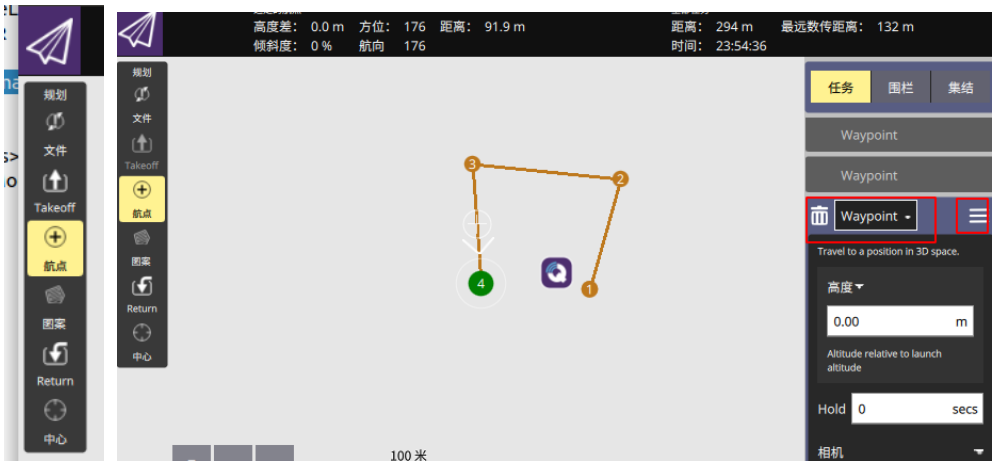
Route planning supports manual doting function. Click File -> Blank. By default, take off point must be set before other takeoff points can be set

If you want to directly set the takeoff point (such as unmanned ships and other vehic les) without setting the takeoff point, you can check the option below in the general Setti ngs of the ground station:

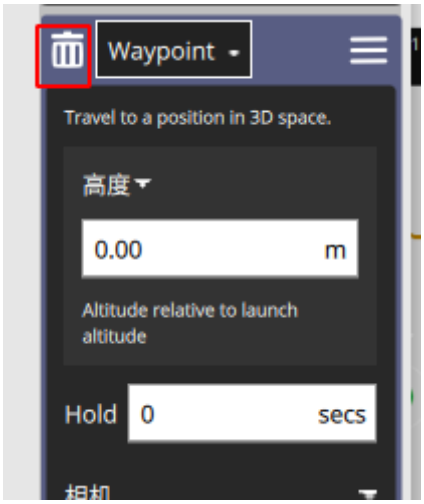


Then you can set the departure point without setting the departure point. After clickin g "takeoff", the takeoff point will be generated at the current position of the UAV. Then click the "Waypoint" button, and click on the map to set the waypoint position. The set waypoint can also be selected by the left mouse button and dragged to change its positio n. Here, after clicking the take-off point, three waypoints are set, with a total of four poi nts, as shown below. On the right side of the map, there is a setting page for each line

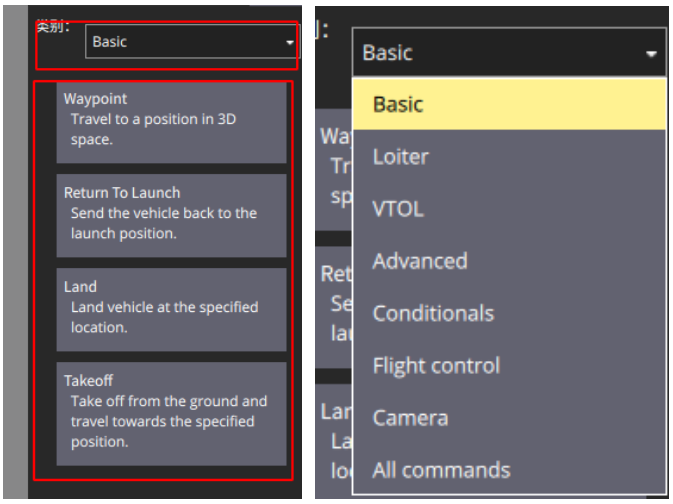
point.



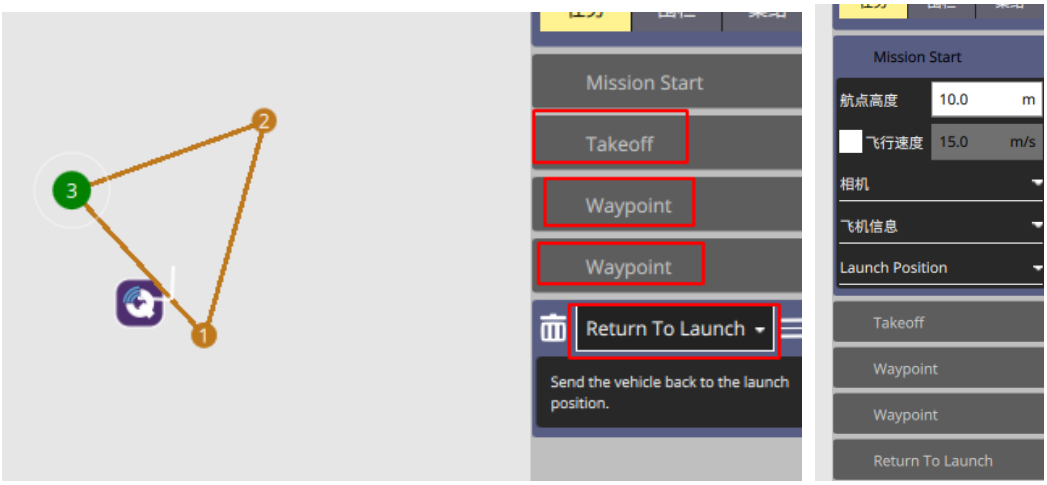
Click the small trash can icon to the left of the corresponding waypoint to delete the way point.



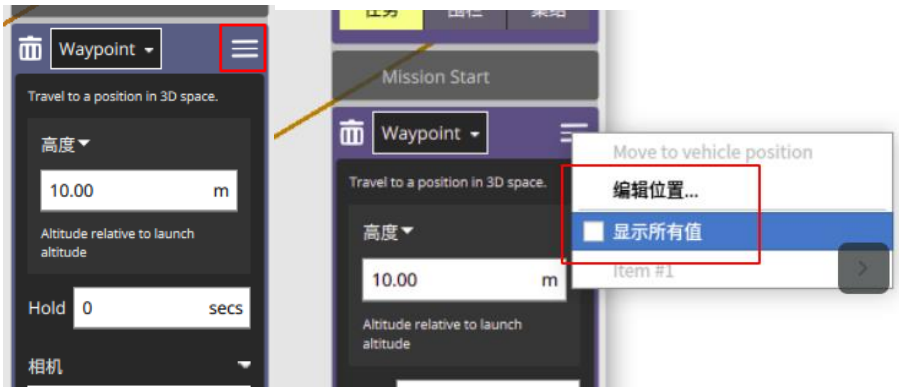
Click Waypoint To the right of the small trash can icon to set the type of the point, the default is Basic category, there are four Waypoint, Return To Launch, Land, Takeoff, r espectively, corresponding to the action to be performed after reaching the waypoint. The above four points are sufficient for normal quadrotor use, but if you need to perform som e more advanced actions (such as hanging mode switching), you need to look for the oth er categories below.



Here, the first point is set as the departure point, the second and third points are set as the departure point, and the fourth point is set as the return point, so that the unmann ed aircraft will take off and fly first to the second point, then fly to the third point, and return to the departure point after reaching the third point. In the Mission Start column, y ou can set the altitude of the landing point to be equal to the flying speed. This setting will be valid for all mission points.



In each task point, you can also set the height and residence time equivalent, but the setting here is only valid for the task point, there is a three-bar icon on the right side o f each task point, you can click the icon to set more information about the task point

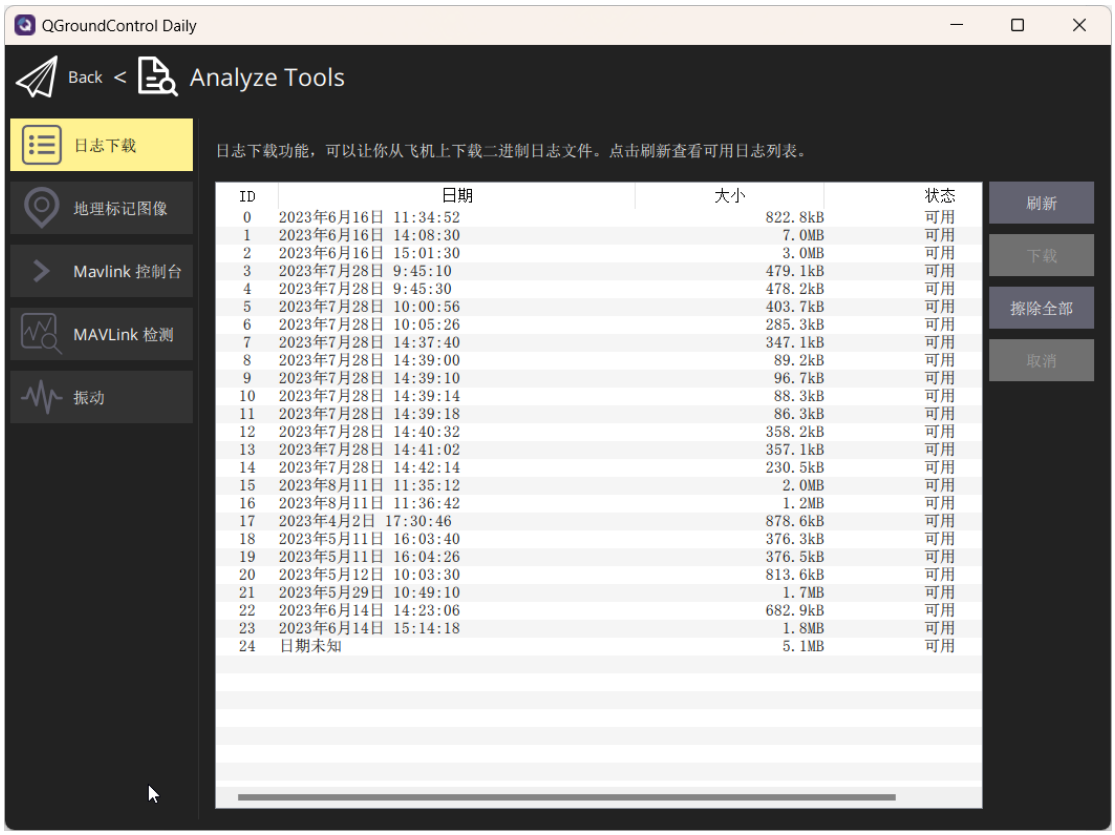


Edit Location allows you to edit the location in detail. Show All Values allows you t o edit all parameters of a task point

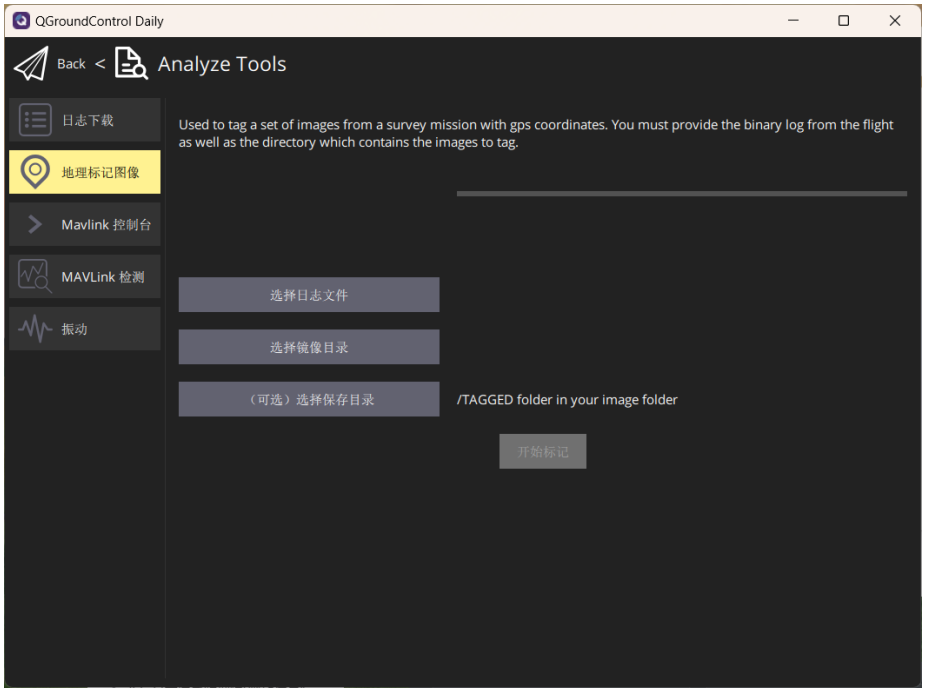
1.2.4. Analyze Tools

QGC provides a wealth of data analysis tools, including log downloads, geotagged im ages, MAVlink console, MAVLink detection, and vibration.

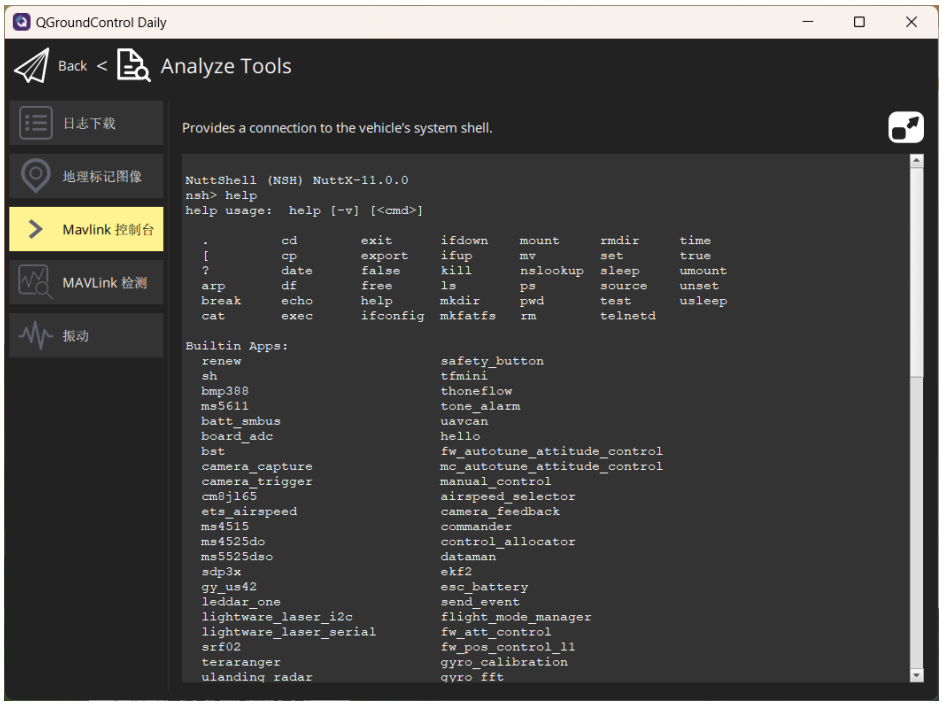
Log download: In the case of linking to the flight control, you can select any log in formation stored in the current flight control memory card to download the.ULG format fi le. The file can be downloaded through the website: https://docs.px4.io/main/zh/log/flight_log_analysis.html log for analysis.



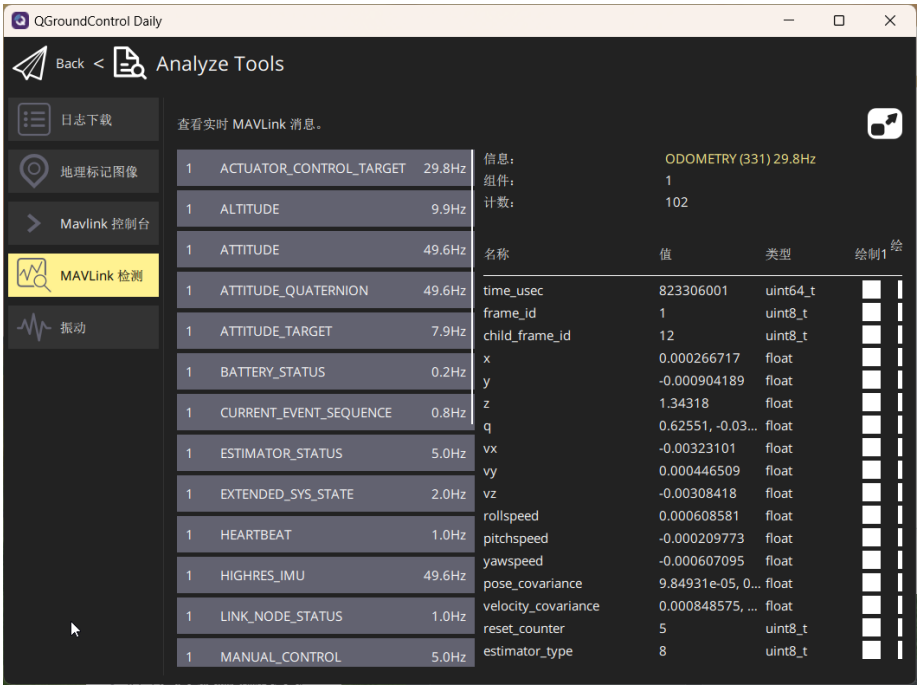
Geotagged images: Used to tag a set of survey mission images with gps, but a binary log of waypoints must be provided along with a directory containing the images to tag.



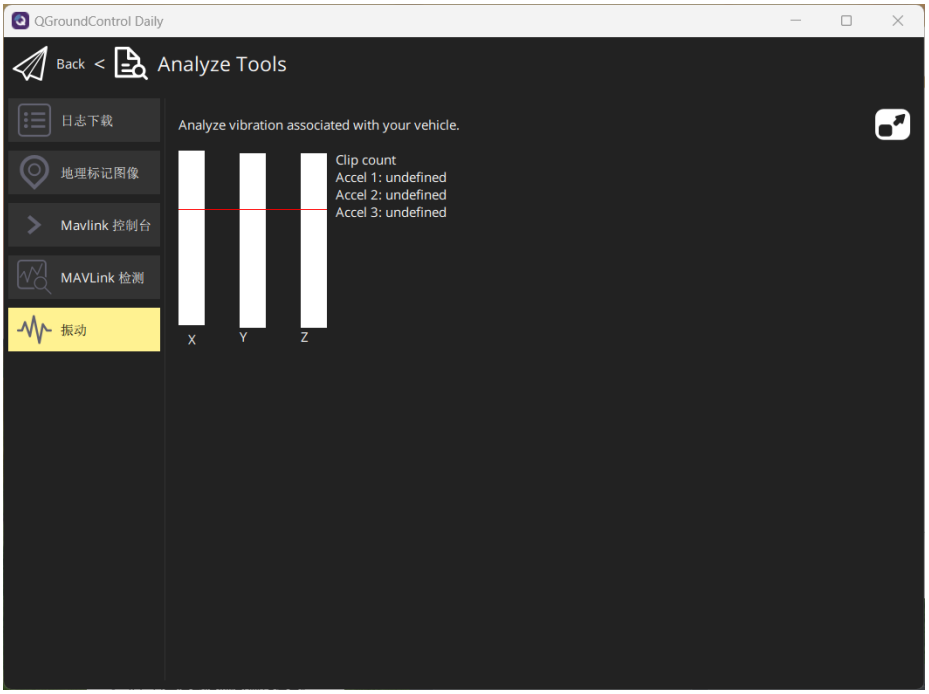
MAVLink Console: It provides a Shell for data communication with Nuttx, the on-board flight control operation system.



MAVLink detection:

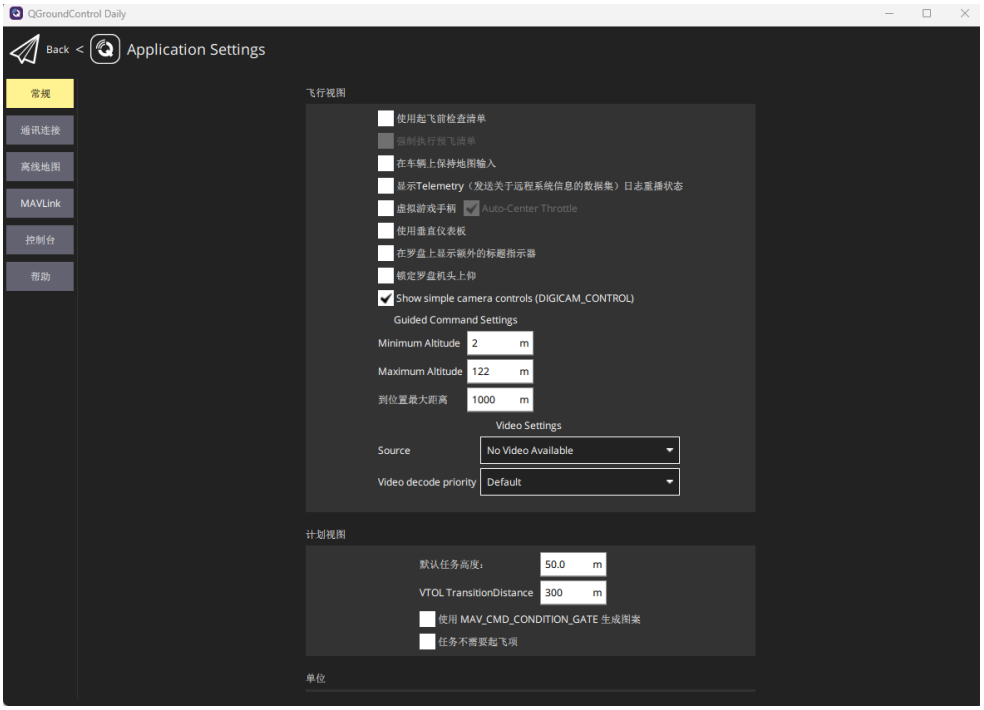


Vibration: Analyze vibrations associated with vehicles

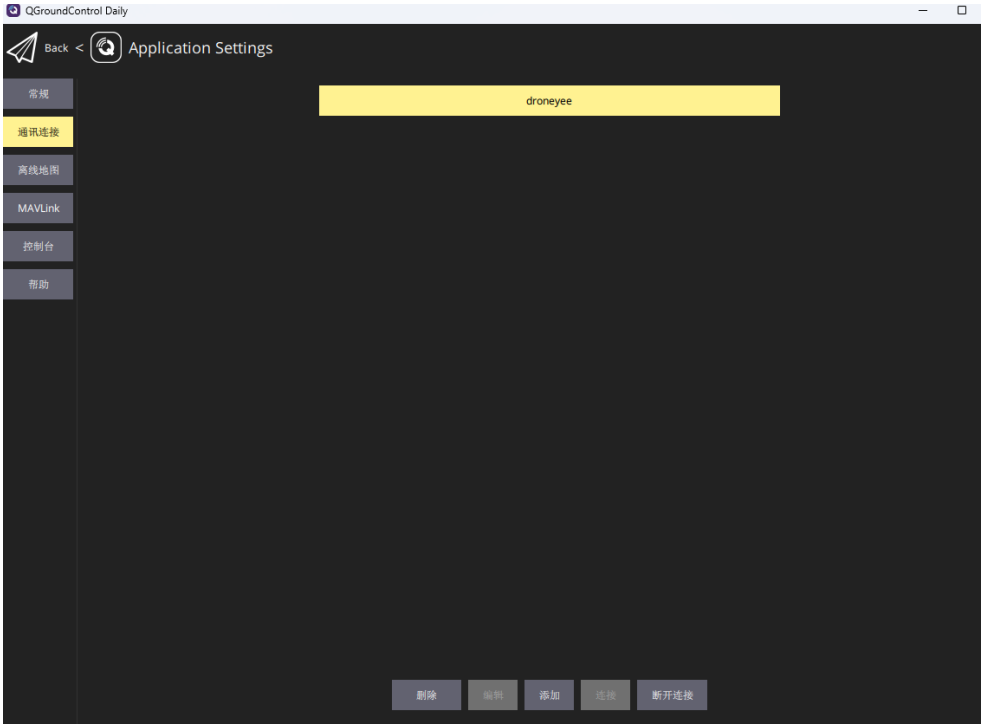


1.2.5. Ground Station Application settings

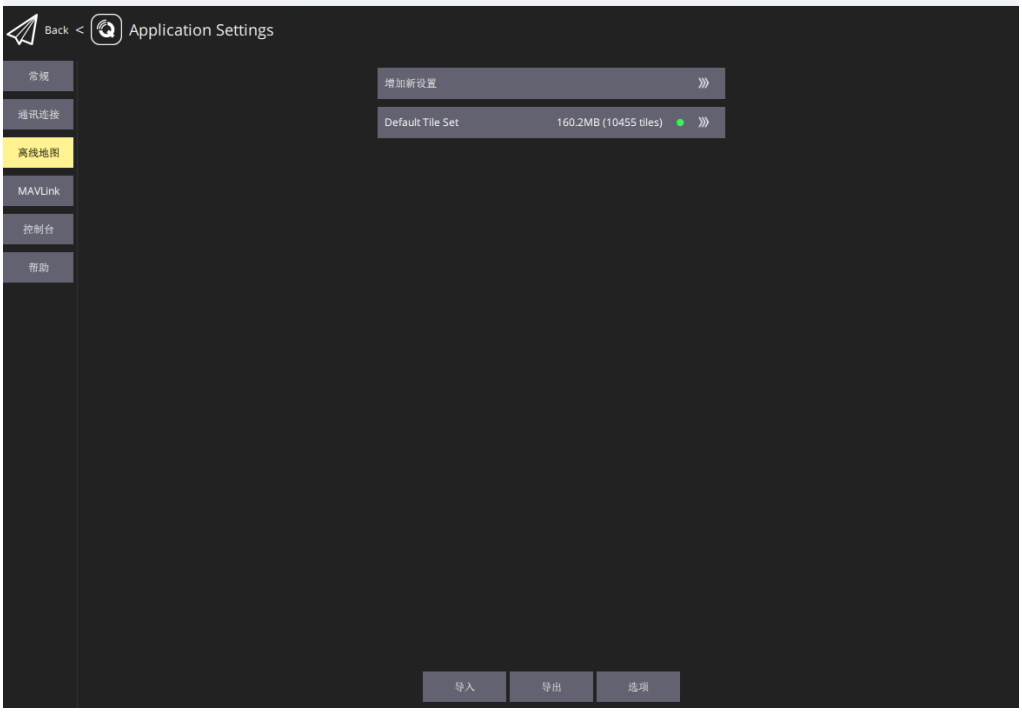
General options: Mainly for application Settings, these are used to specify: display unit, auto connect device, video display and storage, RTK GPS, etc.



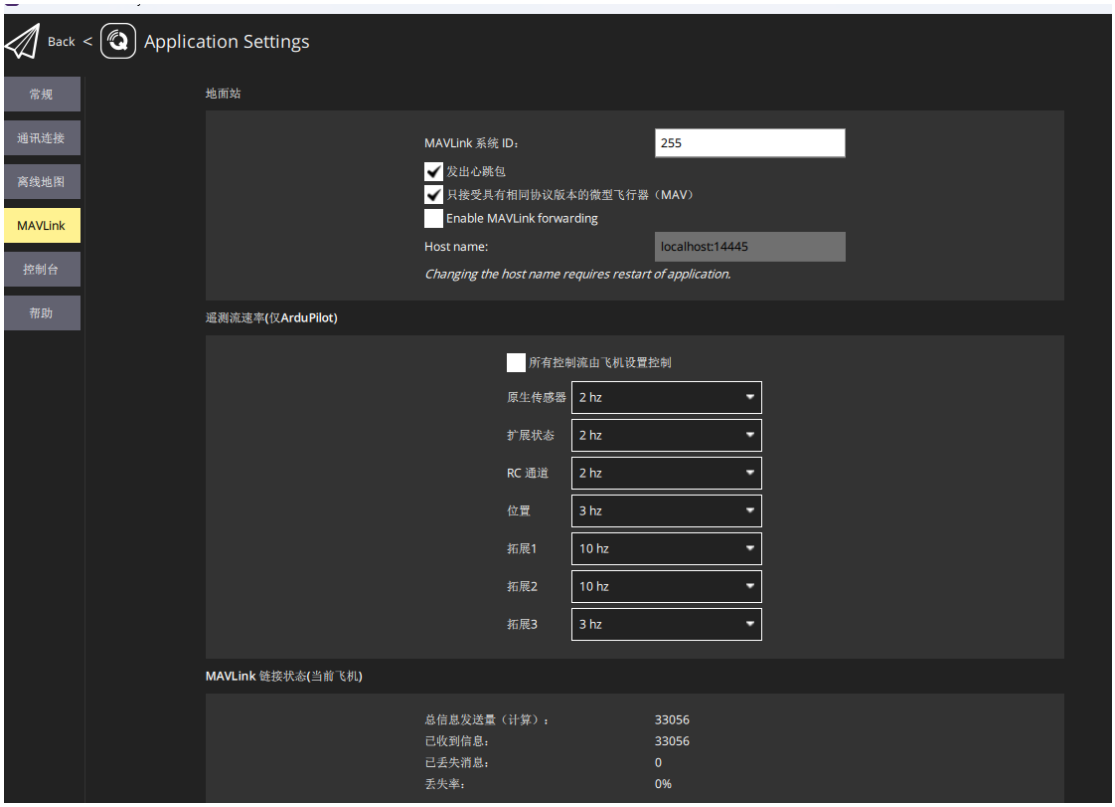
Communication connection: Allows you to manually create communication connections



Offline map: Allows you to cache maps for use when there is no Internet connectio n



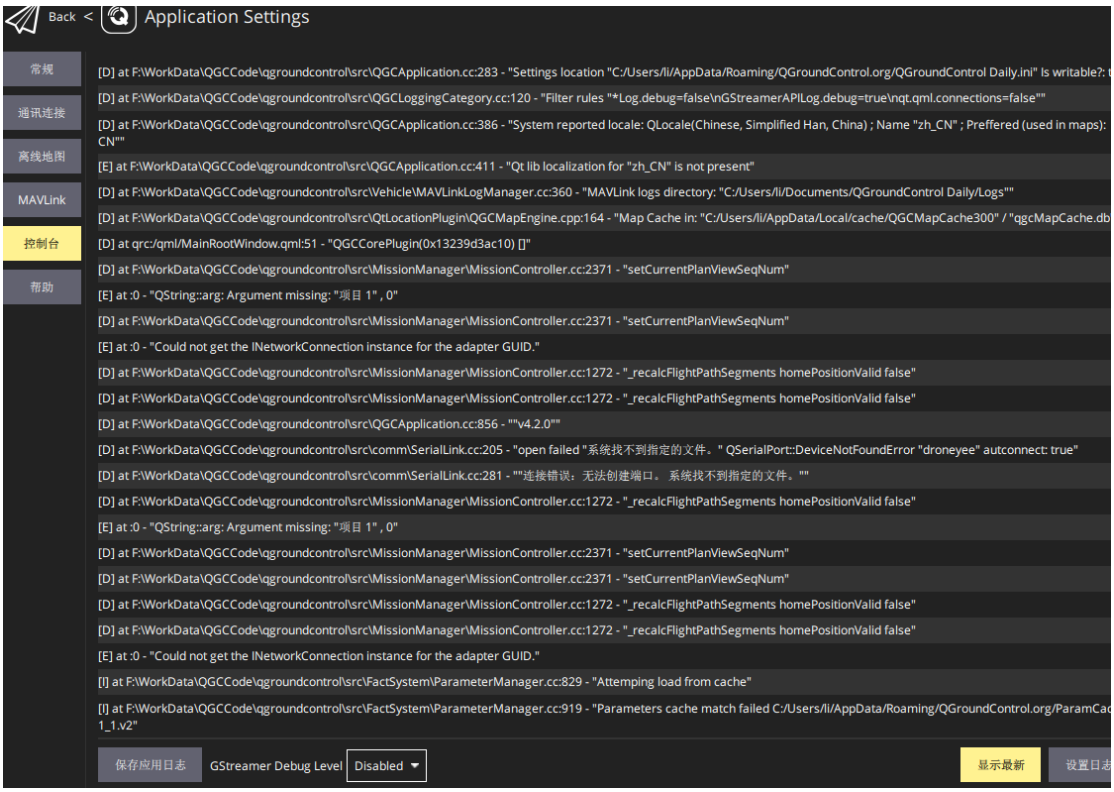
MAVlink： Allows you to configure options and view messages specific to MAVLink communication. This includes setting the MAVlink system ID for the QGC and looking at the connection quality.



Control console: Used to capture application logs to help with application problems

Common logging options

Otion	Description
LinkManagerLog,MultiVehicleManagerLog	Debug connection problem.
LinkManagerVerboseLog	Debug did not detect serial port. The continuous output of the available serial port is very noisy.
FirmwareUpgradeLog	Debugging firmware flash problems.
ParameterManagerLog	Debug parameter loading problems.
ParameterManagerDebugCacheFailureLog	The debug parameter cache crc did not match.
PlanManagerLog,MissionManagerLogGeoFenceManagerLogRallyPointManagerLog	Debug scheduled upload/download issues.
RadioComponentControllerLog	Debug radio calibration problems.



1.3. Control equipment and communication media

1.3.1. Wireless data transmission (PC connects to PX4 through data transmission and controls it)

Wireless data transmission (optional) can be used to establish a wireless MAVLink connection between the QGroundControl ground station and the PX4 flight control. This section covers two topics: wireless data transmission that is already supported and integrating new data transmission into PX4 systems.

http://docs.px4.io/main/en/telemetry/index.html lists the PX4 has support for wireless data transmission system. Includes data transmission using SiK Radio firmware and 3DR Wi Fi wireless data transmission.

1.3.2. Wired serial port module (NX connects PX4 through wired serial port and controls it)

Port preconfiguration: The following functions are usually mapped to the same specific serial port on all boards, so are mapped by default:

- MAVLink is mapped to pull 1 port with baud rate 57600(for telemetry modules).
- GPS 1 (GPS driver) is mapped to GPS 1 port with automatic baud rate (with th

is setting, GPS will automatically detect baud rate - except Trimble MB-Two, w hich requires 115200 baud rate).

- On Pixhawk devices with an Ethernet port, map the MAVLink to the Ethernet p ort using MAV_2_CONFIG.
- Other ports are not specified by default (disabled).

Configure ports:

- Restart the vehicle to make the additional configuration parameters visible.
- Sets the baud rate for the selected port to the desired value.
- Configure module-specific parameters (i.e. MAVLink flow and data rate configura tions).

The GPS/Compass > Secondary GPS section provides a practical example of how to configure ports in QGroundControl (it shows how to run secondary GPS on TELEM 2 po rts using GPS_2_CONFIG)

1.3.3. WIFI module (with onboard board for message fo rwarding)

WiFi telemetry enables MAVLink communication between WiFi radios on the vehicle and GCS. WiFi typically offers shorter distances than regular telemetry radios, but support s higher data rates and can more easily support FPV/ video feeds. Usually the vehicle onl y needs one radio unit (assuming the ground station already has WiFi).

The PX4 supports telemetry over UDP and Wifi. It broadcasts the heartbeat to port 1 4550(255.255.255.255) until it receives the first heartbeat from the ground control station, at which point it sends data only to this ground control station.

Compatible WiFi data transmission modules are:

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

2. RflySim platform control mode interface

2.1. Conventional flight control mode

2.1.1. Take-off Mode:

Take-off flight mode allows the vehicle to take off to a specified altitude and await f urther input. The model requires a good location estimate (e.g., from GPS). You must unl ock this mode before using it. This mode is automatic and does not require user intervent

ion to control the vehicle. The remote switch can change the flight mode on any drone. Moving the remote controller joystick in multi-rotor mode (or VTOL in multi-rotor mode) will switch the drone to position mode by default, unless dealing with battery fail-safe. If there is a problem during takeoff, the fault detector will automatically stop the engine.

Multiple rotors (MCS) rise to the height defined in MIS_TAKEOFF_ALT and hold po sition. The remote control joystick moves the drone into position mode (default). Take-off is affected by the following parameters:

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

More aircraft take-off model explanation, please see: http://docs.px4.io/v1.13/zh/flight_modes/takeoff.html

2.1.2. Landing mode:

Land flight mode lands the vehicle in the location where the mode is being used, an d after landing, the drone will lock up after a short period of time (by default). This mod e requires a valid position estimate, unless the mode is entered due to fail-safe, in which case only an altitude estimate is required (usually the flight control has a barometer built in). This mode is automatic and does not require user intervention to control the vehicle. The remote control switch can be used to change the flight mode of any drone. Moving t he remote controller joystick in multi-rotor mode (or VTOL in multi-rotor mode) will swit ch the drone to position mode by default, unless dealing with battery fail-safe.

The drone will land at the location specified by the pattern. The drone descends at t he speed specified by MPC_LAND_SPEED and locks up after landing (default). The remo te control joystick moves the drone into position mode (default). Landing is affected by th e following parameters:

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

For more models, please see:http://docs.px4.io/v1.13/zh/flight_modes/land.html

2.1.3. Fixed point/hover mode:

The spot/hover mode (aka "wait"/" hover ") stops the vehicle and maintains its curren t GPS position and altitude (the multi-rotor will hover at the GPS position, while the fixe d wing will rotate around it). This mode can be used to pause tasks or help you regain c ontrol of the vehicle in an emergency. It is usually activated by a pre-programmed switch. The multi-rotor UAV hovers at its current position and altitude. Remote control stick mo vement will switch the drone to position mode (default). You can configure this action wit h the following parameters.

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

More models which explain please see: http://docs.px4.io/v1.13/zh/flight_modes/hold.html

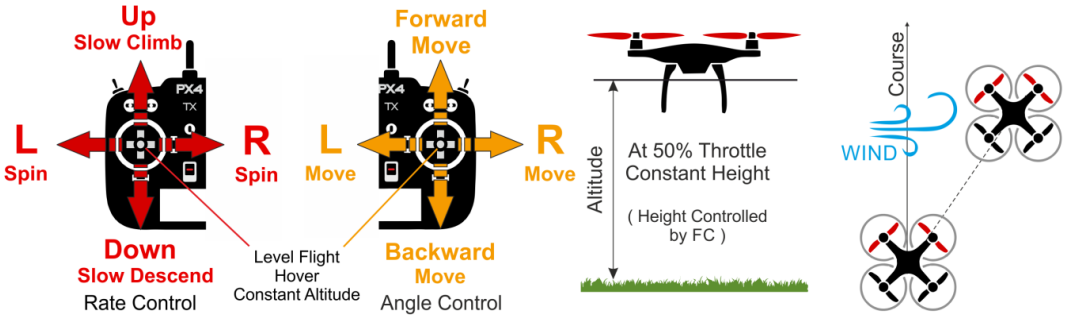
2.1.4. Task mode:

Mission mode enables the vehicle to perform predefined autonomous tasks (flight plan s) that have been uploaded to the flight controller. Ground station (GCS) applications such as QGroundControl(Open New Window)(QGC) are commonly used to create and upload t asks. Missions are usually created in a ground control station (e.g., QGroundControl (ope

ns new window)）and uploaded before launch. They can also be created by the developer API, and/or uploaded on the fly. Individual mission commands are handled in a manner appropriate to the flight characteristics of each vehicle (e.g., helicopter hovers and fixed-wing hovers). VTOL aircraft follow fixed-wing behavior and parameters in fixed-wing mode and rotorcraft behavior and parameters in multi-rotor mode. More explanation, please see: http://docs.px4.io/v1.13/zh/flight_modes/mission.html

2.1.5. Set height mode:

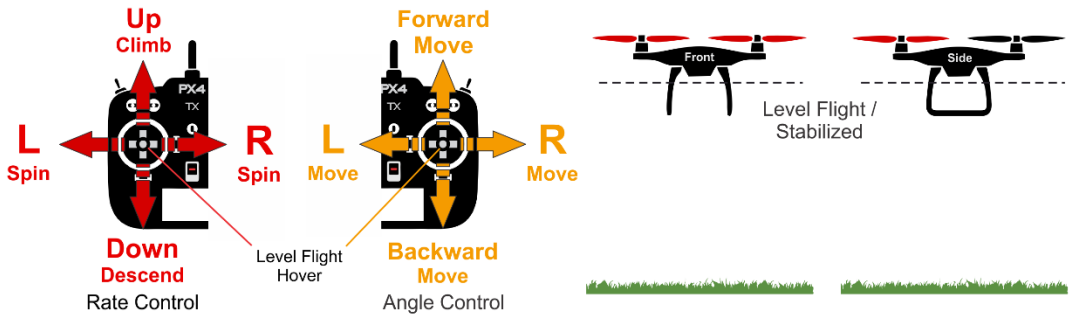
High altitude mode is a relatively easy to fly RC mode, in which the roll and pitch rods control the movement of the vehicle in the left, right and forward directions (relative to the "front" of the vehicle), the yaw rod controls the rate of rotation in the horizontal plane, and the throttle controls the rise-descent speed. When the joystick is released/centered, the vehicle will level and remain at its current height. The following diagram shows this mode visually (using the remote control of the American hand as an example). More explanations can be found at: http://docs.px4.io/v1.13/zh/flight_modes/altitude_mc.html



2.1.6. Self-stabilization/manual control (attitude Angle control mode) mode:

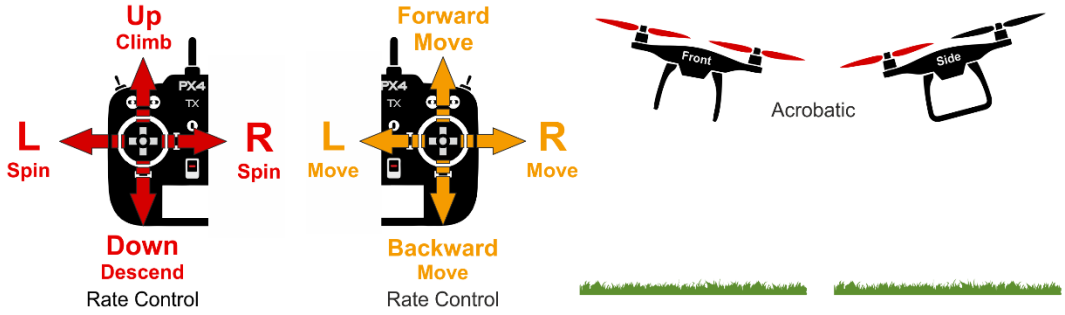
Manual/stable mode when stabilizing multiple helicopters, RC control rods are concentrated. To manually make the body move/fly, you can move the joystick so that it is off center. If you set manual or stable mode, multi-rotor mode is enabled. Under manual control, the transverse rocker and pitch rods control the Angle (attitude) of the vehicle around their respective axes, the yaw rod controls the rate of rotation on the horizontal plane, and the throttle controls the height/speed. Once the joystick is released, they will return to the central dead zone. Once the roll and pitch rocker are centered, the multi-rotor drone will stabilize and stop moving. The body will then hover in the proper position/maintain altitude - provided the balance is right, the throttle is set properly (view below), and no external forces are applied (e.g. wind). The vehicle will drift in any wind direction and you

must control the throttle to maintain altitude. More explanations can be found at:http://docs.px4.io/v1.13/zh/flight_modes/manual_stabilized_mc.html



2.1.7. Stunt (Angular speed control) mode:

The mode is RC mode and is used to perform acrobatic moves such as flips, rolls, and loops. The roll, pitch, and yaw rods control the angular rate of rotation around the respective axis, and the throttle is passed directly to the output mixer. When the joystick is centered, the aircraft will stop rotating, but keep its current orientation (on its side, upside down or any other direction) and move according to the current momentum. For more explanation please refer to:http://docs.px4.io/v1.13/zh/flight_modes/acro_mc.html



2.1.8. Return mode:

The mode is used to fly the vehicle onto a safe, barrier-free path to a safe destination where it can wait (hover or hover) or land. The PX4 provides several mechanisms for selecting a safe return path, return destination, and landing, including the use of actual location, assembly (" safe ") points, mission path, and mission landing sequence.

More explanations can be found at: http://docs.px4.io/v1.13/zh/flight_modes/return.html

2.2. External control mode

（Refer tohttps://docs.px4.io/main/en/flight_modes/offboard.html）

The PX4 can be controlled by an independent auxiliary computer over wired or wifi,

and the partner computers usually communicate via the MAVLink API, which is called ex ternal control mode or Offboard control. The vehicle executes the position, speed and attit ude instructions set by the auxiliary computer through MAVLink. The Offboard mode is mainly used to do air maneuvers, and the corresponding proprietary mode is more appropr iate for take-off, landing and return.

Offboard control is often used as the basic interface of upper layer algorithm develop ment, such as visual obstacle avoidance, cluster formation needs to use Offboard position and speed interface; For stunts, a lower-level attitude interface is needed. In addition, Offb oard messages can also be used to control PTZ cameras, weapons, and more by setting ta rget_component to the corresponding component ID.

The Offboard control has some key parameters that can be viewed or modified by Q GC. The following table lists several key parameters. The user may encounter signal loss when using Offboard control, and the signal enters a specific mode after the loss. These b ehaviors are controlled by the following parameters.

Offboard Control key parameter table

COM_OF_LOSS_T	The offboard lost failsafe is triggered when the signal is lost for more than this time
COM_OBL_ACT	In the absence of an RC signal, offboard lost failsafe switchover mode. 0: Land, 1: Hold, 2: Return
COM_OBL_RC_A CT	Description offboard lost failsafe Mode to which the RC signal is triggered. 0: Position, 1: Altitude, 2: Manual, 3: Return , 4: Land
COM_RC_OVERR IDE	Controls whether stick movement triggers switching to position mo de. The function is disabled by default.
COM_RC_STICK_ OV	stick movement triggers the number of switches to position mode. (COM_RC_OVERRIDE enabled)

Attention:

- The desired status must be set to a value greater than 2Hz; otherwise, the mode cannot be activated or the vehicle automatically exits the mode.
- This mode requires position or attitude information.
- RC (remote control) control allows only switching modes.
- The vehicle must be unlocked to use this mode.
- Not all coordinate systems and field values supported by MAVLink are supported.

2.2.1. Control messages

Three MAVLINK messages are used

SET_POSITION_TARGET_LOCAL_NED（https://mavlink.io/en/messages/common.html#SET_POSITION_TARGET_LOCAL_NED）

SET_POSITION_TARGET_GLOBAL_INT（https://mavlink.io/en/messages/common.html#SET_POSITION_TARGET_GLOBAL_INT）

SET_ATTITUDE_TARGET（https://mavlink.io/en/messages/common.html#SET_ATTITUDE_TARGET）

MAVLINK has three messages related to Offboard control:SET_POSITION_TARGET_LOCAL_NED、SET_POSITION_TARGET_GLOBAL_INT、SET_ATTITUDE_TARGET, The first interface is the most used of the three messages. The first two are position-controlled messages that contain information about desired position, desired speed, desired acceleration, and so on. The main difference between these two position control messages is that when sending GPS coordinates, because single-precision floating point numbers can not meet the accuracy requirements, the GPS coordinates need to be converted to int type, which makes the data format of the two position messages different.

The following table shows the specific field details of the position control message. There are 16 fields in total. The field filled with color is the difference between the two position control messages. In the following message, target_system is the target system ID, which is usually obtained automatically after the connection is established. In normal flight control, target_component is the default value.

There are two most commonly used coordinate systems, one is LOCAL_NED, where position, velocity, acceleration/force are all located in the NED coordinate system. The other is GLOBAL_INT, where the position is represented by latitude, longitude, and altitude, and the latitude and longitude are multiplied by 107 and converted to integers for transmission, while the velocity, acceleration/force are still represented in the NED coordinate system. If the user wants to use other more special coordinate system, can refer to the official documentation to set at https://mavlink.io/en/messages/common.html.

As you can see, the yaw Angle and yaw Angle rate are also included in the position control message. This is because the Offboard control message is designed from the perspective of the user task and needs to be turned when doing cluster formation or visual tasks, so these two messages are also added to the position control message.

SET_POSITION_TARGET_LOCAL_NED/SET_POSITION_TARGET_GLOBAL_INT

serial n	name	type	units	description

umber				
1	time_boot_ms	uint32_t	ms	system time
2	target_system	uint8_t		System ID, each flight contro ller can be considered to hav e a unique ID
3	target_component	uint8_t		Component ID, the default I D for flight control is 0, if y ou are controlling the PTZ c amera or weapon equipment, you can use other ids
4	coordinate_frame	uint8_t		The most commonly used co ordinate system is MAV FRA ME LOCAL NED = 1
				The most commonly used FR AME is MAV FRAME GLO BAL INT = 5
5	type_mask	uint16_t		The bit flag indicates that co ntrol information should be i gnored
6	x	float	m	x position in NED coordinate s
	lat_int	int32_t	degE7	WGS84 coordinates latitude multiplied by 10^7
7	y	float	m	y position in NED coordinate s
	lon_int	int32_t	degE7	WGS84 coordinates longitude multiplied by 10^7
8	z	float	m	NED coordinate height
	alt			height above sea level
9	vx	float	m/s	NED coordinate system veloc ity in the x direction
10	vy	float	m/s	NED coordinate system y vel ocity
11	vz	float	m/s	NED coordinate system z vel ocity

12	afx	float	m/s^2	NED coordinates acceleration or force in the x direction
13	afy	float	m/s^2	Acceleration or force in the y direction in the NED coordinate system
14	afz	float	m/s^2	NED coordinate system acceleration or force in the z direction
15	yaw	float	rad	yaw angle
16	yaw_rate	float	rad/s	Yaw Angle rate

Note: The padding in blue indicates that SET_POSITION_TARGET_LOCAL_NED is unique, and the padding in green is unique to SET_POSITION_TARGET_GLOBAL_INT.

In the aerobatics and other high maneuver control, need to do a lower level of control, SET_ATTITUDE_TARGET message can support this function. The first four fields are the same as the message above, except that the message supports the attitude, angular rate, and throttle value of the specified quaternion.

As you can see, attitude control messages are simpler than position control messages. It is not enough to control the position by specifying a series of attitude values, you also need to specify the throttle value. So the throttle value appears in the attitude control message.

SET_ATTITUDE_TARGET

serial number	name	type	units	description
1	time_boot_ms	uint32_t	ms	system time
2	target_system	uint8_t		System ID, each flight controller can be considered to have a unique ID
3	target_component	uint8_t		Component ID, the default ID for flight control is 0, if you are controlling the PTZ camera or weapon equipment, you can use other ids
4	type_mask	uint16_t		The bit flag indicates that control information should be ignored

5	q	float[4]		Attitude quaternion
6 6	body_roll_rate	float	rad/s	Roll Angle rate
7	body_pitch_rate	float	rad/s	Pitch Angle rate
8	body_yaw_rate	float	rad/s	Yaw Angle rate
9	thrust	float		accelerograph
10	thrust_body **	float[3]		standing off

2.2.2. Control interface

Location interface

Speed interface

Attitude interface

Acceleration interface

Interface of force

Yaw Angle interface

Yaw angular velocity interface

Offboard control supports position, speed, attitude, acceleration, force, and angular rate control, and is encapsulated in two types of messages according to the user's usage scena rio, one is position control message and the other is attitude control message. Whether usi ng python or matlab to control the aircraft, it is built on the basis of these two types of messages. Control information such as position and speed is not mandatory, but the type_ mask flag bit must be used to specify what information can be ignored. type_mask is a u int16_t type where each binary bit can identify a state information.

SET_POSITION_TARGET_LOCAL_NED and SET_POSITION_TARGET_GLOBAL_IN T have different data formats in the data link. But they will all be converted into uORB messages of type vehicle_local_position_setpoint_s. It can be seen that the NED coordinate s are the reference coordinate system for PX4 position control, while other coordinate syst ems are provided only for the convenience of users to describe specific tasks.

1）Location class interface introduction

Location interface. Contains 3 data, the Offboard location is entered into the controll er as the desired location. The position interface is more complex than other interfaces be cause of the coordinate system. Common coordinate systems include LOCAL_NED and G LOBAL_INT. When LOCAL_NED, all three data are float; When the value is GLOBAL_I

NT, the latitude and longitude are int32_t and the height is float. The location interface corresponds to three flags, which are the 0-2 bits respectively. For example, if the 0 bit of type_mask is 0, the x position is displayed. If all three component positions are specified, bits 0-2 are all 0.

Speed interface. The speed interface belongs to the position control class message and contains three pieces of data. The desired speed specified by Offboard is superimposed on the output of the position controller, that is, the true expected speed is the sum of the desired speed specified by the position controller output and the desired speed specified by Offboard. When the output of the position controller is 0, the true desired speed is the desired speed specified by Offboard. For both LOCAL_NED and GLOBAL_INT, the expected velocity is also in the NED coordinate system. The type_mask corresponding to the speed interface is bits 3-5, which can also be specified independently.

Acceleration/force interface. The acceleration/force interface belongs to the position control class message and contains three pieces of data. The acceleration/force is superimposed on the output of the speed controller, and the desired acceleration is the value specified by Offboard only if the output of the speed controller is 0. Offboard Acceleration or force Indicates force when the 9th bit of type_mask is 1; otherwise, acceleration is specified. The presence or absence of acceleration/force is indicated by the 6-8 digits. Acceleration control is not mature at present, although the interface is supported, it is not necessarily able to achieve better control effect. In both LOCAL_NED and GLOBAL_INT cases, the acceleration and force are described in the NED coordinate system.

Yaw Angle interface. Yaw Angle can belong to either position control message or attitude control message and contains 1 data. Because position control messages and attitude control messages are for different user scenarios, they are generally not specified at the same time. Thus, no matter what type of task, the yaw Angle/yaw Angle rate needs to be specified. The type_mask corresponding to the yaw Angle is the 10th bit, and when the 10th bit is 0, there is a yaw Angle.

Yaw Angle rate interface. Yaw Angle rate can belong to either position control message or attitude control message, and contains 1 data. The type_mask corresponding to the yaw Angle rate is the 11th bit. When the 11th bit is 0, it indicates that there is a yaw Angle rate.

2）Attitude class interface introduction 【 New and old interfaces are not fully supported 】

Angular rate interface. The angular rate interface belongs to the attitude class interface and contains three data, namely, roll, pitch and yaw angular rate. The corresponding ty

pe_mask is bits 0-2. If bits 0-2 are all 0, it means that all yaw Angle rates are specified. The angular rate specified by Offboard will be superimposed on the output of the attitud e controller. Only when the output of the attitude controller is 0, the real expected angula r rate is the expected angular rate specified by Offboard.

Throttle interface. The throttle interface belongs to the attitude interface and contains 1 data. The corresponding type_mask is the 6th bit. When the 6th bit is 0, the throttle i s specified.

Attitude interface. The attitude data is described by quaternions and contains 4 data in total. The corresponding type_mask is the 7th bit. When the 7th bit is 0, there is pose data. Posture does not support independent posture, to specify you must specify 4 data.

2.2.3. Typical combination mode

PX4 Offboard protocol is very flexible, position, speed, attitude, etc., can be freely co mbined. However, in actual use, the most commonly used patterns are often not free com binations but some typical patterns. The following is an example of a typical combination of rotorcraft and fixed wing.

Typical combination mode of rotorcraft. The following table lists typical combinatio ns of control information for rotorcraft, which can be achieved by setting the two types o f messages in the previous section. The RflySim platform provides interfaces for the follo wing modes, which users can call directly. Users can also set up these messages themselv es through MAVLINK, which enables not only the following patterns, but also more adva nced patterns, such as specifying both location and speed.

In addition, the rotorcraft supports speed setting in the carrier coordinate system. The speed in the carrier coordinate system is set by SET_POSITION_TARGET_LOCAL_NED. When coordinate_frame is set to MAV_FRAME_BODY_NED: 8, the position is in NED c oordinate system and the velocity acceleration is in BODY coordinate system.

Typical combination mode of rotorcraft

serial num ber	description
0	Velocity mode in navigation coordinate system [vx,vy,vz, yaw rate]
1	Velocity mode in body coordinate system [vx,vy,vz, yaw rate]
2	Position mode in navigation coordinate system [x,y,z, yaw]
3	Position mode in body coordinate system [x,y,z, yaw]
4	Attitude throttle control command [roll, pitch, yaw (radian), throttle (0~1)]

5	Acceleration control mode [ax,ay,az,yaw]
6	Acceleration control mode [ax,ay,az,yaw rate]

Typical combination of fixed wings. Unlike rotorcraft, fixed wings are not free to control any trajectory of flight, nor do they support independent control of speed in all directions. The main control modes of fixed wing are [posx, posy, posz, speed]. The designation of the horizontal position must meet certain constraints, because the fixed wing cannot stay in a particular position. A fixed wing generally does not support specifying the yaw Angle, as the fixed wing always keeps the nose facing in the forward direction.

3. Control model

RflySim platform provides two categories of high-precision model and particle model, high-precision model is often used for high-fidelity simulation, the number of vehicles is generally small. The particle model can be used for large-scale clusters. High-precision models are divided into two categories, one is the controller using PX4, which is the most realistic form of use, and the other is the controller is also integrated into the model called comprehensive model. The advantage of the integrated model is that it can support large-scale high-fidelity cluster simulation more stably and reliably.

3.1. High precision model +PX4 controller composed of software /hardware simulation model in the loop (rely on CopterSim, copied from the model group)

3.1.1. CopterSim and PX4 communication port

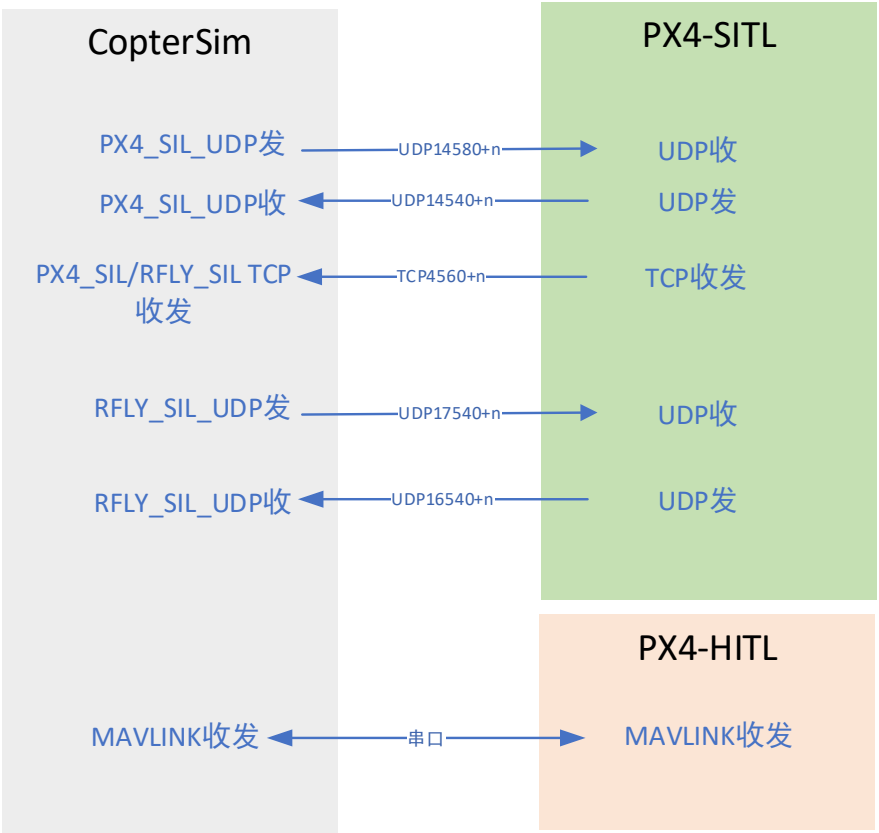
CopterSim can communicate with PX4 in the following way. Whether it is UDP/TCP/ serial port, the packets sent follow the MAVLink protocol. The following figure can be mainly boiled down to three communication modes.

The first is the software in the ring simulation mode of PX4, which corresponds to the PX4_SITL mode of CopterSim. For CopterSim, send messages using UDP14580+n port, send messages using UDP14540+n port, or send and receive messages using TCP4560+n. In PX4 emulation, either UDP or TCP communication is usually chosen based on specific requirements and performance requirements. For example, when transmitting sensor data, UDP can be used to ensure the timeliness of the data, while TCP can be used when it is necessary to control the aircraft and ensure the reliable transmission of instructions. Gener

ally, UDP is more suitable for scenarios that require high real-time performance and can t olerate some data loss, while TCP is more suitable for data transmission scenarios that req uire reliability and integrity. The problem with PX4_SITL mode is that it can support no more than 40 aircraft, otherwise port conflicts will occur.

The second is PX4_SITL_RFLY provided by the RflySim platform. In this mode, Co pterSim uses UDP17540 to send messages and UDP16540 to receive messages. The advan tage of PX4_SITL_RFLY over PX4_SITL mode is that it can support large-scale clusters, up to 1000 vehicles without port conflicts.

The third is hardware-in-the-loop simulation. In this mode, CopterSim will communica te with the PX4 via a serial port.



3.1.2. Messages sent by CopterSim to PX4 over TCP

Messages that CopterSim sends to PX4 over TCP include HIL_GPS, HIL_SENSOR, a nd RC_CHANNELS_OVERRIDE. HIL_GPS is the simulated GPS data, and HIL_SENSOR is the simulated IMU data, including the data of gyroscope, accelerometer, magnetometer and barometer. RC_CHANNELS_OVERRIDE simulates remote control data.

In the process of real flight, the data of the remote control is a radio frequency sign al, which is relatively reliable, so you can also consider using TCP when simulating. The sensor data comes from the flight control board, so TCP simulation is also appropriate. In

practice, some external control instructions are transmitted via data transmission, which is relatively unreliable, so it is reasonable to simulate these data with UDP.

3.1.3. Messages sent by CopterSim to PX4 over UDP

The data sent to PX4 through UDP includes MANUAL_CONTROL, SET_POSITION_TARGET_LOCAL_NED, SET_POSITION_TARGET_GLOBAL_INT, PARAM_SET, and SET_ATTITUDE_TARGET , COMMAND_LONG, HIL_ACTUATOR_CONTROLS, and HEARTBEAT. It also supports the forwarding of QGC data to CopterSim.

The joystick portion of the remote control data is not only sent over TCP, but also converts the ch1-ch4 data into MANUAL_CONTROL messages to be sent to the PX4.

3.2. High-precision model +Simulink controller composed of high-precision integrated model（Rely on CopterSim）

On the basis of the original dynamic model, the controller is realized and the comprehensive model is formed. The controller uses MATLAB Simulink to realize basic attitude control and fixed-point functions. The controller takes the real state of the model directly as input. The most important part of the integrated model protocol is to agree on the input and output interfaces. The overall interface design considers only the full mode, while the simplified mode is considered in CopterSim.

3.2.1. Integrated Model control protocol

OffBoard control. The PX4 can be controlled by an independent auxiliary computer over wired or wifi, and the partner computers usually communicate via the MAVLink API. The vehicle executes the position, speed and attitude instructions set by the auxiliary computer through MAVLink. The Offboard mode is mainly used to do air maneuvers, and the corresponding proprietary mode is more appropriate for take-off, landing and return.

Use the inSIL protocol to complete input and output.

The 0th digit of inSILInts is used to represent and modify the state, and the corresponding bit is 1 to indicate the corresponding state of the system. For example, the first bit indicates emulation mode, and when the first bit of the received inSILInts[0] is 1, it indicates that the system is in emulation mode.

The state is set only once when 0:hasCMD is 1, otherwise the synthesis model will remain in the original state. The original state can come from setting an external setting value, or it can be an internal state automatically converted. For example, after receiving the take-off command, it first switches to the take-off mode, and automatically switches to t

he fixed-point mode after the take-off is completed.

inSILInts[0] Vehicle Command Bitmap

0:hasCMD	1: SIL	2: Armed	3:	4:	5:	6:	7:
Have new orders	scale aeromodelling	unlock					
8:Takeoff	9: Position	10: Land	11: Return	12:Lotier	13:Height	14:Hor	15:
take off	fixed point / waypoint	land	return flight	Hover (fixed wing)	Fixed height mode	Horizontal position control	
16:OffboardPos	17:OffboardAtt	18:	19:	20:	21:	22:	23:
offroad position control series	offroad position attitude series						
24:	25:	26:	27:	28:	29:	30:	31:

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

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

inSILInts[1] Offboard Indicates the control flag

0:hasPos	1:hasVel	2:hasAcc	3:hasYaw	4:hasYawRate	5:	6:	7:
Location	speed	accelerated speed	yaw angle	Yaw Angle rate			
8:hasAtt	9:hasRollRate	10:hasPitchRate	11: hasThrust	12:	13:	14:	15:
attitude	Roll Angle rate	Pitch Angle rate	accelerograph				
16:NED	17:Global	18:	19:	20:	21:	22:	23:
Position and sp	Position and speed Global						

eed NED							
24:	25:	26:	27:	28:	29:	30:	31:
Integer type latitude	Precision of integer type						

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

Note: The corresponding value is 1 when position control is used only, 2 when speed control is used only, 8 when yaw Angle is used only, and 16 when Angle rate is used only. If multiple controls need to be combined, the values for separate controls are added.

inSILFloats protocol. inSILFloats are used to store actual data. Their meanings can change depending on the Settings of inSILInts. For example, the first three represent positions, but the specific coordinate system is controlled by inSILInts.

inSILFloats[0-2] =pos;

inSILFloats[3-5]=vel; // Speed | remote control pitch, roll, yaw signals |inSILFloats[3] can be used as speed

inSILFloats[6-8]=acc;

inSILFloats[9-11]=att; // Attitude control uses Euler angles and is more intuitive for the user.

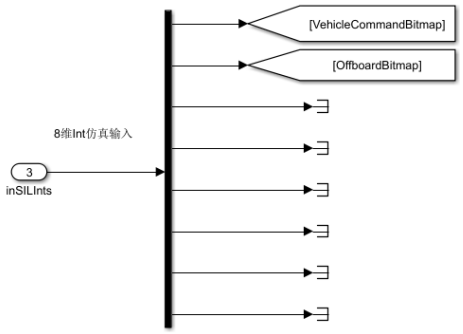
inSILFloats[12-14]=attRate;

inSILFloats[15]=thrust; // accelerograph

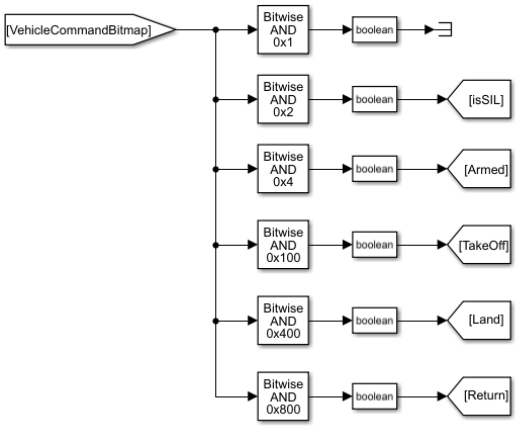
3.2.2. Rotor integrated model control interface

For the comprehensive model, its output is similar to that of the ordinary high-precision model, and the difference is mainly in the input, so the input is introduced in detail in the following.

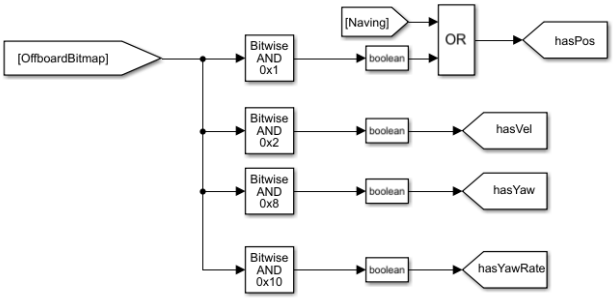
Rotorcraft protocol parsing is the parsing of network packets received by CopterSim into the instructions in Section 3.2.1. inSILInts is an 8-dimensional input that currently uses only the 0th number instruction and the 1st number Offboard mode. Subsequent 6th and 7th numbers are represented as integers of latitude and longitude in the Global coordinate system. In the figure below, the inSILInts vector is broken down into eight separate numbers.



Each bit of the instruction has a corresponding meaning, so each bit needs to be furt her parsed. In the following figure, Bitwise module is used to parse bits, and the 0th bit is not used for the time being. The first bit identifies whether it is emulation mode. Whe n the bit is 1, emulation mode is used; when the bit is 0, the hardware is in ring mode. The controller in the integrated model takes effect only when emulation mode is used. Th e second digit indicates whether it is unlocked. The other digits also include take-off, land ing, and return functions. For details, see Section 3.2.2.

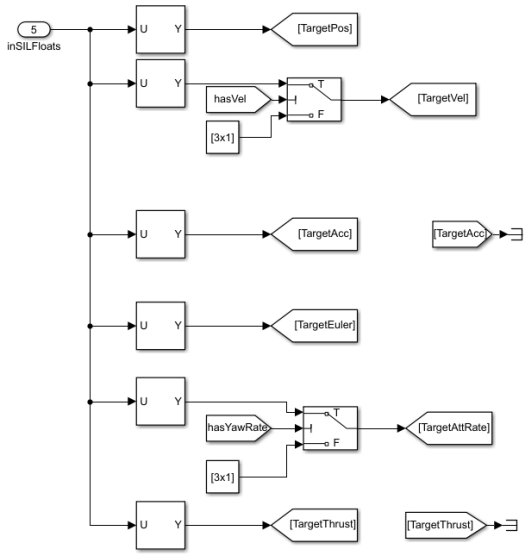


The following flags identify which offboard control information is available. The prot ocol in Section 3.2.1 supports full PX4 offboard control, following which the most urgent requirements of the current project are supported, including position, speed, yaw, yaw Ang le rate.



Further, the actual position and speed values sent by offboard are analyzed as shown in the figure below. In the figure above is to identify the corresponding value does not e

xist, while the figure below is the specific value. Currently only position, speed, yaw, yaw Angle rate are used. In the following parsing, the hasVel flag and the hasYawRate flag a re used. When the two flags are false, it indicates that there is no input of speed and ya w Angle rate, then the remote control mode will be switched. In remote control mode, 15 00 represents the desired speed or expected yaw Angle rate of 0.



3.2.3. Fixed wing integrated model control interface

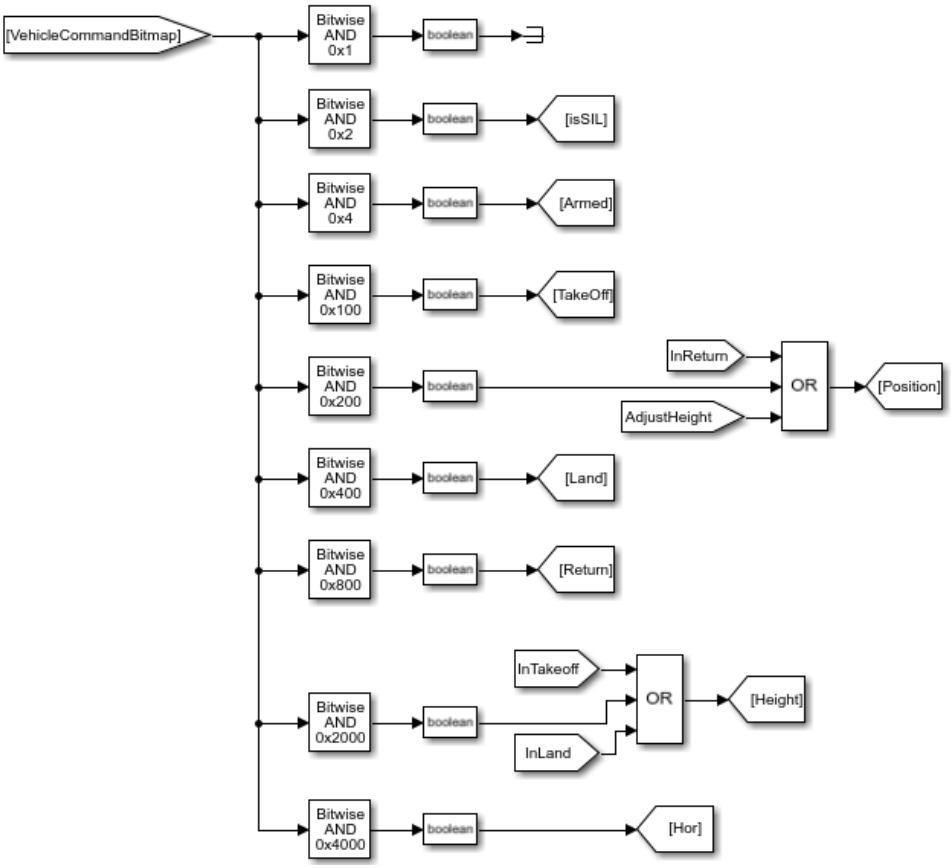
Fixed wings and rotors follow mutually compatible protocols, but the mode of fixed wings is different from that of rotors, because fixed wings cannot hover like quadrotors. T he following figure shows the protocol analysis in the fixed-wing integrated model. Posito n, Height and Hor are increased, and the behavior of take-off, return and landing is also different from that of rotorcraft.

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

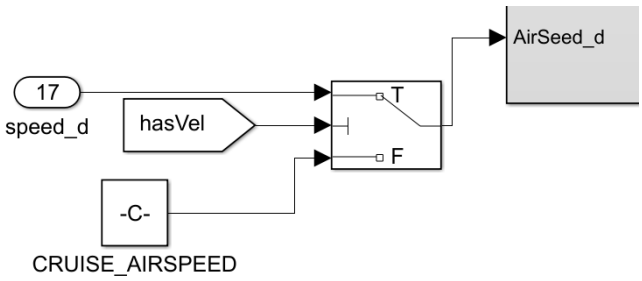
Position, while controlling horizontal position and height. The user can directly specif y the mode as position control through inSILInts, and the system can automatically trigger the position mode when returning or landing. In position mode, if the corresponding posi tion is set, fly to the corresponding position first. After reaching the specified position, if no next position is specified, it will hover automatically. In the return mode, it is essentia lly the horizontal Position to return to the starting point and supports specifying the return height, so this function can be implemented using the Position mode. When landing, the

altitude of the aircraft will first be adjusted, and the AdjustHeight in the figure is the des cription of this process. Adjusting the altitude of the aircraft is accomplished by circling, and the horizontal position will also change during circling, so the position mode is also used for control.

Height mode controls the altitude and airspeed, but does not control the horizontal po sition, that is, the aircraft will only fly forward. Height mode is used during takeoff, and will also be entered at the end of landing (when the aircraft has reached the correspondin g altitude). Hor mode is to control the horizontal position separately. This pattern is suppo rted by current models, but is generally less used.



Unlike rotorcraft, fixed wings are not free to control speed in all directions. However, you can set the rate in the horizontal direction. In the full protocol, the desired speed ha s 3 components, and in fixed wings only the first component is used as the horizontal sp eed.



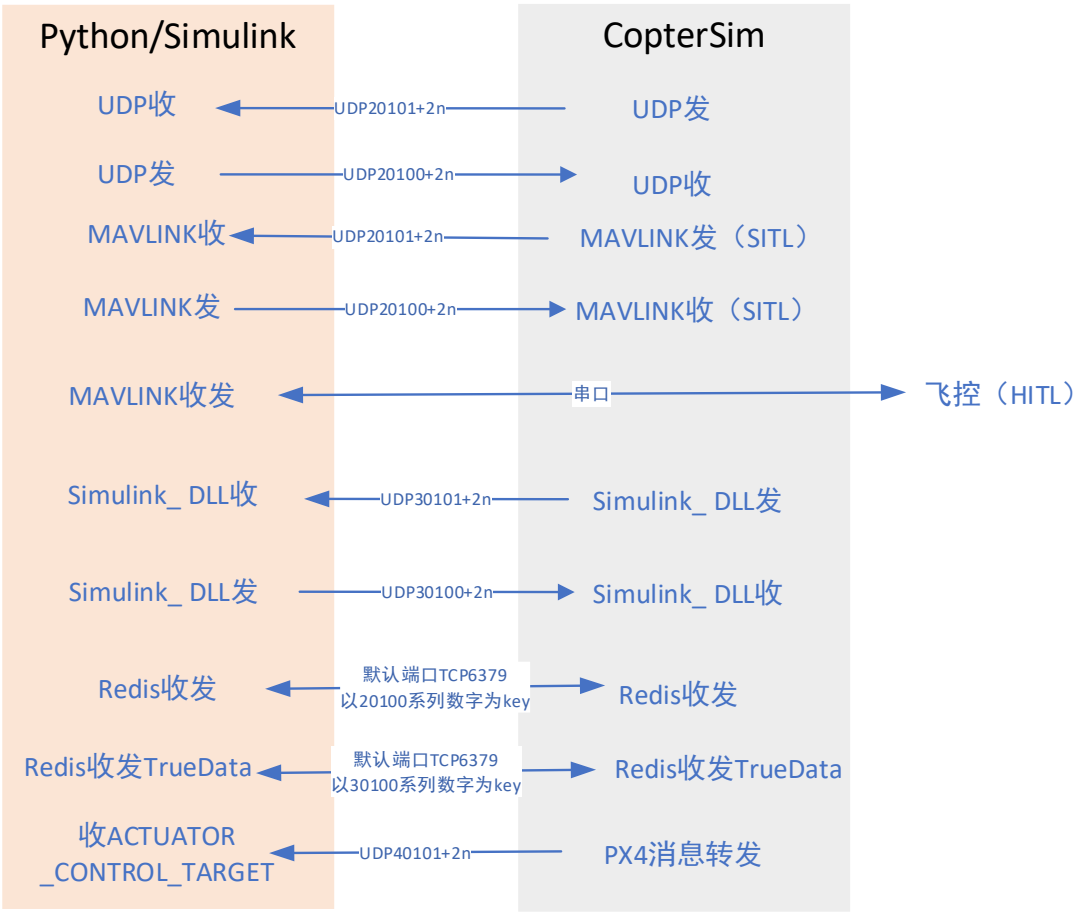
3.3. Simplified comprehensive model based on particle model（Rely on Python rotors to copy from the cluster API）

4. RflySim control protocol（Only the enterprise customized version supports Redis）

4.1. General introduction

The external control usually obtains the state of the vehicle through Python or Simulink, and sends control instructions according to the state of the vehicle. CopterSim is the communication hub for the entire platform. CopterSim supports different simulation modes, and the components of the whole simulation system are quite different under different simulation modes. In the same simulation mode, there are different communication modes, including UDP/MAVLink/Redis, etc. Different communication modes can support the number of simulations, the specific degree of state information, stability, etc.

The diagram below is an overall block diagram of CopterSim interacting with Python/Simulink. In terms of communication mode, three main modes UDP/MAVLink/Redis are supported, of which Redis is only supported in the enterprise customized version. In addition, two sub-modes, FULL/Simple, are also supported, and the size of packets in these two modes is different. RflySim supports the traditional MAVLink message format, and MAVLink messages can be forwarded over UDP or directly to the flight control through the serial port. The Simulink_DLL mode is the mode of the integrated model, in which the controller is integrated in the CopterSim. And in Redis mode，



4.1.1. Emulation modes supported by CopterSim

Simulation mode mainly refers to software in the loop, hardware in the loop, etc. Dif ferent simulation modes often involve different components. For example, software in the l oop relative to hardware in the loop does not require flight control hardware, and the inte grated model relative to software in the loop does not need to run a linux subsystem.

The full CopterSim version supports four simulation modes: HITL = 0, SITL = 1, SI TL_RFLY = 2, Simulink_DLL = 3. HITL indicates that the hardware is in the loop, and t he flight control hardware needs to be connected to run this mode. SITL indicates softwar e in the ring, and SITL_RFLY is also software in the ring. However, compared with SIT L, the port configuration is optimized and port conflicts are not easy to occur. Simulink_ DLL is dedicated to comprehensive model simulation, that is, the controller is also built i n the model, which is more suitable for large-scale cluster simulation.

4.1.2. Connection modes supported by CopterSim

The connection mode of CopterSim refers to the communication mode between Copte rSim and other components. Currently, there are 8 supported modes and 6 conventional m odes. UDP_Full = 0, UDP_Simple = 1, MAVLink_Full = 2, MAVLink_Simple = 3, MAV Link_NoSend = 4, MAVLink_NoGPS = 5, Redis_Full = 6. Redis_Simple = 7.

UDP, MAVLINK and Redis are the three basic communication modes, and they are d ivided into two sub-modes, Full and Simple, according to whether the packet is compact or complete. The result is a combination of six general patterns. For large-scale clusters, Redis_Simple is recommended.

4.2. Data Protocol

The platform defines some basic data structures related to control, which are independ ent of the specific communication mode, that is, whether using UDP, MAVLink, or redis, the data parsing follows these agreed formats.

4.2.1. outHILStateData

The outHILStateData structure is where CopterSim forwards the data estimated by PX 4, including timestamp, aircraft ID, location, speed, etc. In PX4, EKF2 converts the raw s ensor data into a state estimate of the vehicle, and CopterSim forwards the data received from PX4 to the upper layer application through the following data structure.

```
    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 positi
ve
    int32_t GpsVel[3]; //Estimated GPS velocity, NED, m/s*1e2->cm/s
    int32_t gpsHome[3]; //Home GPS position, lat&long: deg*1e7, alt: m*1e3 and up is positive
    int32_t relative_alt; //alt: m*1e3 and up is positive
    int32_t hdg; //Course angle, NED,deg*1000, 0~360
    int32_t satellites_visible; //GPS Raw data, sum of satellite
    int32_t fix_type; //GPS Raw data, Fixed type, 3 for fixed (good precision)
    int32_t 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
    }
```

4.2.2. SOut2Simulator

The definition of SOut2Simulator is exactly the same as the new MavVehileInfo struc ture added in Simulink, and can directly package the MavVehile3DInfo output data of the new simulink model. These states are real data from the model.

```
struct SOut2Simulator {
        int copterID;   //Vehicle ID
        int vehicleType;   //Vehicle type
```

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

4.2.3. inHILCMDData

inHILCMDData is the key data structure for the underlying control of the system. The designation of copterID allows the same Simulink program to control multiple machines, while the designation of mode and flag allows the user to send more upper-level instructions, such as mode switching, locking, etc. ctrls are the control quantity, usually the pulse width of the PWM wave.

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

4.2.4. outHILStateShort

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

```
}
```

4.2.5. inOffboardShortData（Minimal control protocol, supported by CopterSim in UDP/Redis Simple mode）

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

checksum-- Check bit 1234567890, a fixed value;

ctrlMode-- Select the mode. For details, see the following table;

controls[4]-- Four digit control quantity.

ctrlMode Supported control mode

Pattern label	description
0	Velocity mode in navigation coordinate system [vx,vy,vz, yaw rate]
1	Velocity mode in body coordinate system [vx,vy,vz, yaw rate]
2	Position mode in navigation coordinate system [x,y,z, yaw]
3	Position mode in body coordinate system [x,y,z, yaw]
4	Attitude throttle control command [roll, pitch, yaw (radian), throttle (0~1)], can be automatically unlocked, can automatically enter the OffBoard mode
5	Attitude throttle increment control command [roll, pitch, yaw, throttle increment]- can be automatically unlocked, can automatically enter OffBoard mode
6	Acceleration control mode [ax,ay,az,yaw]
7	Acceleration control mode [ax,ay,az,yaw rate]
8	Acceleration control mode [ax,ay,az,yaw rate]
9	Unlock the mode shown [Unlock,-,-,-]
10	Set the speed and hover radius of the fixed wing aircraft, [speed, radius, -, -]
11	Indicates Mavlink take-off command, automatic unlock, navigation coordinate system position [x, y, z, -]
12	Indicates Mavlink take-off command, automatic unlock, GPS coordinate system position [latitude, longitude, altitude, -]

13	Speed altitude heading command, automatically unlock and enter offBoard mode, GPS coordinate position [speed, altitude, heading, -]
14	Position mode in Global coordinates, GPS coordinates position [lat_int, lon _int, alt_float, yaw_float]
30	Used for VTOL mode switching, representing flight mode switching instruc tion, corresponding to mavlink command MAV_CMD_DO_VTOL_TRANSI TION, controls[0] indicates the State bit. Define see link (https://mavlink.io /en/messages/common.html#MAV_VTOL_STATE). controls[1] indicates the Immediate position (1: Force immediate front swit ch, 0: normal transition).

4.3. Communication ports

4.3.1. UDP14540 series +TCP4560 series（communication with PX4, PX4 default port when the software is s imulated in the loop）

The UDP14540 series ports are used by PX4 for software-in-the-loop simulation by d efault, with UDP ports for sending less important data and TCP ports for sending more i mportant data. See Section 3.1.1 for a more detailed description.

4.3.2. UDP16540 series (communication with PX4, RflySi m private port during ring simulation)

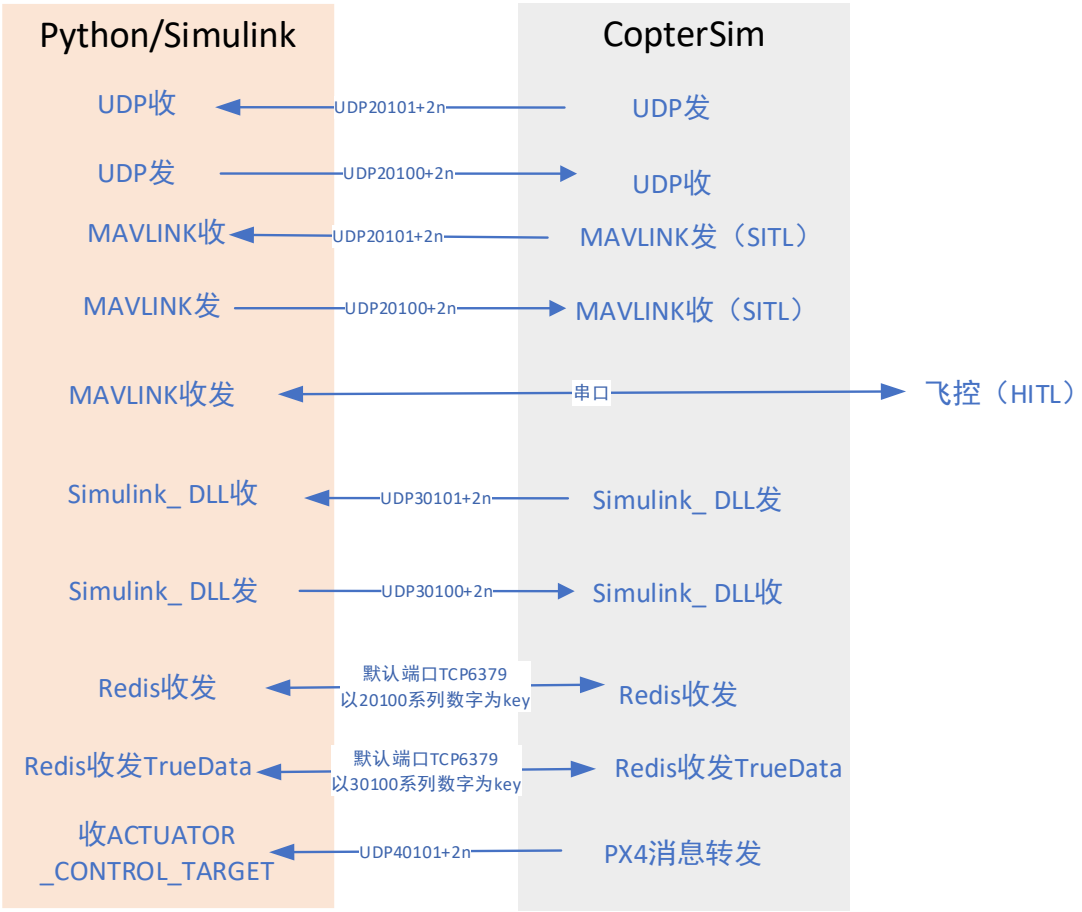
The UDP16540 series port is a custom series port of the RflySim platform, which ca n support a larger number of vehicles than the PX4 default port/See Section 3.1.1 for a more detailed description.

4.3.3. Serial port（communication with PX4, hardware-in -the-loop emulation port）

When a MAVLink message packet is transmitted over a serial port, parameters such a s the baud rate can be set. When the baud rate is small (less than 115200), the message sending frequency decreases. Some messages are not sent directly.

4.3.4. UDP20100 series (Python/Simulink to obtain status information and issue control commands)

The following figure is the port information of the Python/Simulink program to interact with CopterSim, and Redis can be supported in the enterprise customized version. The 20100 series port refers to python/Simulink sending messages to CopterSim using UDP20101+2n, where n is the ID-1 of the aircraft (assuming ID is numbered from 1), and receiving messages from CopterSim using UDP20100+2n. The messages sent by Python/Simulink to CopterSim include instructions, Offboard position, speed, acceleration control messages, etc. Python/Simulink receives messages from CopterSim mainly about the current aircraft position, speed and other information, which can be used for closed-loop control.



Python从30101+2n读数据时，实际上读的是发给UE的数据。

4.3.5. UDP30100 Series (Get True status information or issue control commands through inSIL)

The UDP30100 series port is used to obtain True status information or send control c

ommands through inSIL. True information refers to the true position and speed of the vehicle, which can be directly obtained during simulation. The inSIL sends control commands, which are currently used to control the integrated model. (Currently fault injection also uses this port, so the control synthesis model and fault injection cannot be used simultaneously).

4.3.6. UDP40100 Series (Get User-Defined messages)

The UDP40100 port is used to obtain user-defined messages. Currently, it is mainly used to obtain the ACTUATOR_CONTROL_TARGET message, which contains the pulse width of the PWM wave.

4.3.7. TCP6379（Redis port）

This is the port of Redis communication, when supporting multiple aircraft, Redis uses the Key to distinguish between different aircraft。

4.4. RflySim UDP protocol

The platform can send control commands through UDP, which is a simplification of the MAVLink protocol, and users can use the regular control interface through the protocol.

The UDP mode can be UDP_Full or UDP_Simple. The biggest difference between UDP_Full and UDP_Simple is the degree of data simplification. The packet size of UDP_Simple is smaller than that of UDP_Full. For CopterSim, use port 20100+2n to receive messages and port 20101+2n to send messages, where n=0,1,2... .

The platform can send control commands through UDP, which is a simplification of the MAVLink protocol, and users can use the regular control interface through the protocol.

The UDP mode can be UDP_Full or UDP_Simple. The biggest difference between UDP_Full and UDP_Simple is the degree of data simplification. The packet size of UDP_Simple is smaller than that of UDP_Full. For CopterSim, use port 20100+2n to receive messages and port 20101+2n to send messages, where n=0,1,2... .

4.4.1. CopterSim UDP_Simple

UDP_Simple mode is suitable for cluster development and provides the simplest implementation for basic position and speed control. In UDP_Simple mode, the received control instructions are described by inOffboardShortData. Contains checksum, ctrlMode, and four float controls[4]. ctrlMode can have multiple protocols (RflyUdpFast.cpp and Cop

terSim should be changed). ctrlMode If set to <0 (less than 0), it indicates that the command is empty and the module will not release messages. The following table lists the protocols supported by ctrlMode.

In UDP_Simple mode, once an inOffboardShortData message is received, Copter automatically sends a message to PX4 to unlock and enter Offboard mode. This simplifies the process of user control of the aircraft。

```
struct inOffboardShortData{
    int checksum; // Check bit 1234567890
    int ctrlMode; // mode selection
    float controls[4]; // Four-bit control quantity
}
```

The above control mode is used to send control commands to the PX4. Usually, when the control algorithm calculates the control command, it needs to rely on the current position and speed information of the vehicle. The UDP_Simple schema provides the data structure to retrieve this information.

As shown below, UDP_Simple mode receives simplified data from CopterSim. Specifically, the checksum needs to be 1234567890 to verify that the data is correct. gpsHome aircraft take-off point longitude and latitude data. The Euler Angle of an AngEular aircraft, pitch roll yaw, in degrees. localPos The relative coordinates of the aircraft with respect to the take-off point, in northeast earth coordinate system, in meters. localVel Speed of the aircraft, in m/s. gpsHome is an int, first obtain the degree (multiplied by 1e-7) degree (multiplied by 1e-7) meters (multiplied by 1e-3) format, and then use the module of Simulink, combined with the unified GPS origin, can obtain the offset position of the aircraft relative to the unified GPS origin, combined with localPos, The real-time position of the relative uniform origin of the aircraft can be obtained.

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

As can be seen from the above description, UDP_Simple mode only provides the simplest instructions and obtains the simplest information.

In order for users to fully support the Offboard control protocol in Section 2.2.2, the UDP_Full mode is designed. In UDP_Full mode, the Offboard control message is the complete MVLINK protocol, and UDP only forwards the message. However, in UDP_Full mode, the messages received by the user are still greatly simplified compared to MAVLINK.

4.4.2. CopterSim UDP_Full

In UDP_Full mode, the system automatically unlocks and enters the Offboard mode when receiving Offboard messages. Therefore, in UDP_Full mode, the user does not need to manually send instructions to unlock and enter the Offboard mode. The data received in UDP_Full mode is shown in 4.1.1 and is represented by outHILStateData. It includes position and speed in GPS coordinate system and position and speed in NED coordinate system. In actual use, it is not necessary to parse out all the data in outHILStateData, only the data of interest can be extracted. outHILStateData is the result of sensor data processed by PX4 EKF2.

For CopterSim to transmit UDP data, the packet needs to be further encapsulated. The following is the data structure of netDataShortShort, which supports a maximum of 112 data. It can be seen that netDataShortShort fully supports the transmission of outHILStateData data.

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

The user can also obtain real data, SOut2Simulator, which can be read in both UDP_Simple mode and UDP_Full mode, because this data uses additional ports. The corresponding incoming port of CopterSim is 30100 and the outgoing port is 30101. SOut2Simulator is real state information from the model, without added noise, which is also data that can only be obtained during the simulation.

Similarly, the data in SOut2Simulator is transferred through the netDataShort structure, which means that the total length of all the data in SOut2Simulator is 192 bytes.

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

4.4.3. CopterSim Redis_Simple/Full

4.4.4. Simulink control mode Full/Simple/UltraSimple

4.4.5. Python control mode（Full support for CopterSim mode）

5. The MAVLink protocol

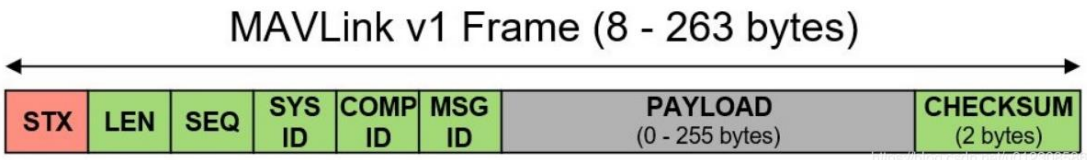
5.1. Introduction to MAVLin

MAVLink (Micro Air Vehicle Link) is a communication protocol for small unmanned vehicles, first released in 2009. The protocol is widely used in the communication betwee n Ground Control Station (GCS) and Unmanned vehicles, as well as in the internal comm unication between the onboard computer and Pixhawk. The protocol defines the rules of p arameter transmission in the form of message library. The MAVLink protocol supports a v ariety of vehicles such as unmanned fixed-wing aircraft, unmanned rotorcraft, and unmanne d vehicles. Using document official website: https://mavlink.io/en/messages/common.html.

5.1.1. Format of the MAVLink packet

MAVLink is divided into MAVLink 1 and MAVLink 2. MAVLink 2 offers more functiona lity and security than MAVLink 1, especially in applications that require more complex co mmunications and higher security. However, it is important to note that MAVLink 2 May require more computing resources and bandwidth to support variable-length packet and me ssage signing, so there may be trade-offs in resource-constrained systems.

The packet format of MAVLink 1 is as follows:



MAVLink 2 的数据包如下：

5.1.2. MAVLink data parsing

All byte streams are stored in the buffer, and byte data in the buffer is read successively. When the STX flag bit is encountered (the flag bit of MAVLink v1 is 0xFE, and the flag bit of v2 is 0xFD), a message is recognized until the end of the message. If the message verification is correct, the message is sent to the processor. Given a byte stream buffer of a certain length, the length of which is length, the onMavLinkMessage function is executed every time a mavlink packet is parsed through the following script

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

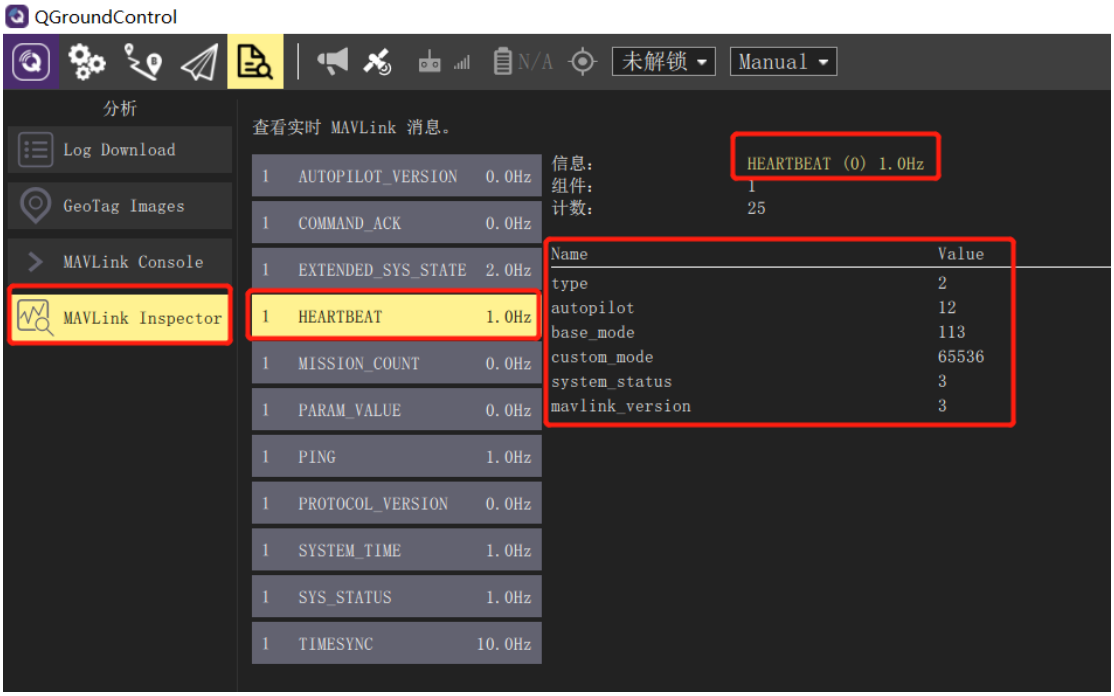
void onMavLinkMessage(mavlink_message_t message); It is a processing function after getting a MAVLink message packet, which needs to identify the purpose of the current packet (heartbeat packet, GPS position, attitude, etc.) according to the ID of the message, and extract the data of interest.

The parsing function is implemented as follows, jumping to the corresponding _decode function according to message.msgid to decode the data

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

```
            outHilData.GpsVel[0]=gp.vx;
            outHilData.GpsVel[1]=gp.vy;
            outHilData.GpsVel[2]=gp.vz;
            outHilData.hdg = gp.hdg;
            break;
    }}}
```

In QGC's MAVLink Inspector page, you can browse all MAVLink packets sent by Pixhawk to see the frequency of each packet and the specific value.



[wxme] start ->

You can modify the MavlinkConsolePage.qml file to change the page layout. Related information is processed using QGCMAVLinkMessage

->end[wxme]

5.2. Common MAVLink messages

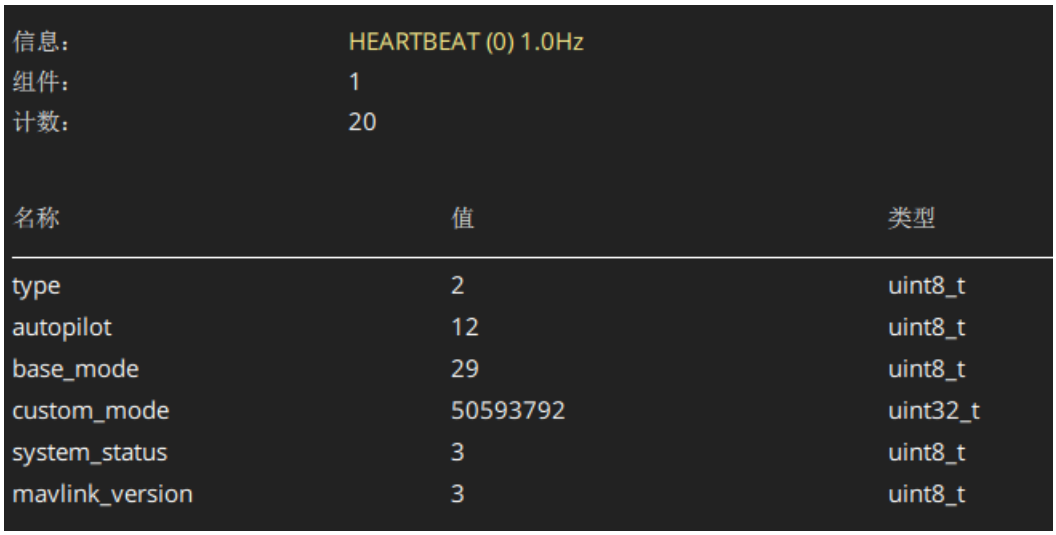
SET_ATTITUDE_TARGET, SET_POSITION_TARGET_LOCAL_NED, and SET_POSITION_TARGET_GLOBAL_INT are the three most commonly used external control messages. They are described in detail in Section 2.2.1.

5.2.1. HEARTBEAT

HEARTBEAT is a heartbeat packet. A heartbeat message indicates that the system or component is present and responding. The type and Autopilot fields (along with the messa

ge component ID) allow the receiving system to appropriately process further messages fro m this system (for example, configuring the user interface according to Autopilot). The mi cro service document at https://mavlink.io/en/services/heartbeat.html.

name	data type	description
type	uint8_t	Vehicle or component type. For flight control components, this refers to the vehicle type (quadcopter, helicopter, etc.). For other components, it refers to the type of component (e.g. camera, head, etc.). This information should be used first to identify the component type rather than the component ID.
autopilot	uint8_t	The flight control type, which for PX4 is MAV_AUTOPILOT_PX4, is a fixed value.
base_mode	uint8_t	MAVLink basic system mode bitmap, including system unlock, manual input, HIL, self-stabilization, autonomous and other modes.
custom_mode	uint32_t	Flight control defines modes, such as PX4, which defines preparation, take-off, hover, landing, etc.
system_status	uint8_t	System status, system startup, correction, failsafe, etc
mavlink_version	uint8_t	MAVLink version



HEARTBEAT messages displayed in QGC

5.2.2. ATTITUDE（Attitude - Euler Angle）

ATTITUDE is the Euler Angle of attitude. Attitude in the carrier coordinate system (r

ight-handed coordinate system, Z-axis down, Y-axis right, X-axis forward, ZYX sequence, body coordinate system).

naem	data type	units	description
time_boot_ms	uint32_t	ms	The timestamp is calculated from the boot of the system
roll	float	rad	Rolling Angle [-π, π]
pitch	float	rad	Angle of pitch [-π, π]
yaw	float	rad	Yaw Angle [-π, π]
rollspeed	float	rad/s	Roll Angle rate
pitchspeed	float	rad/s	Pitch Angle rate
yawspeed	float	rad/s	Yaw Angle rate

The ATTITUDE message displayed in QGC

5.2.3. ATTITUDE_QUATERNIO

ATTITUDE_QUATERNION is the attitude quaternion. The attitude expressed in the carrier coordinate system (right-handed coordinate system, Z axis down, X axis forward, Y axis right) is expressed in quaternion form. The order of the quaternions is w, x, y, z, and the zero rotation will be expressed as (1 0 0 0 0).

name	data type	units	description
time_boot_ms	uint32_t	ms	The timestamp is calculated from the boot of the system
q1	float		w, real part, square root of the cosine of half the Angle of rotation
q2	float		x, (x,y,z) indicates the direction of the axis of

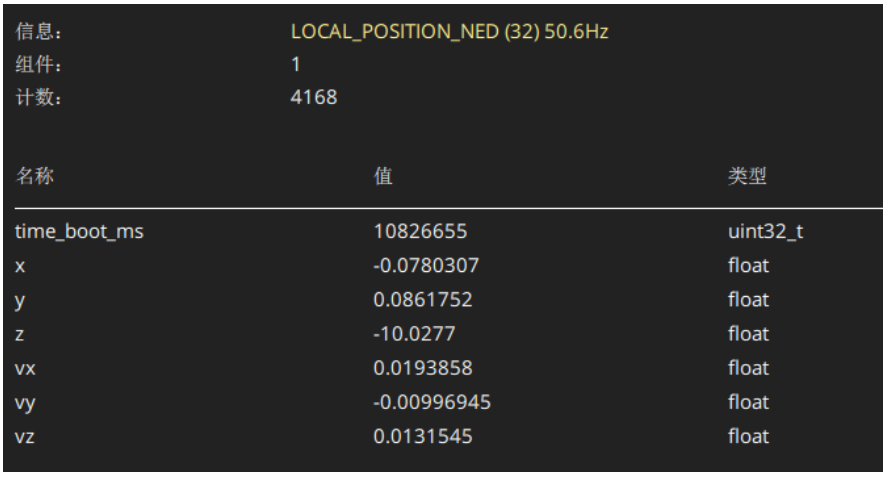
			rotation
q3	float		y
q4	float		z
rollspeed	float	rad/s	Roll Angle rate
pitchspeed	float	rad/s	Pitch Angle rate
yawspeed	float	rad/s	Yaw Angle rate

QUATERNION as shown in QGC

5.2.4. LOCAL_POSITION_NED

LOCAL_POSITION_NED indicates the position and velocity in the NED coordinate s ystem. Local locations that have been filtered (for example, incorporating computer vision and accelerometer data). The coordinate system is right-handed with the Z-axis down (aero -coordinate system, NED/north-east-down convention).

name	data type	units	description
time_boot_ms	uint32_t	ms	The timestamp is calculated from the boot of the syste m
x	float	m	x position
y	float	m	y position
z	float	m	z position
vx	float	m/s	Velocity in x direction
vy	float	m/s	Velocity in y direction
vz	float	m/s	Velocity in z direction



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

The LOCAL POSITION NED shown in QGC

5.2.5. GLOBAL_POSITION_IN

GLOBAL_POSITION_INT. Filtered global position (for example, integrated GPS and accelerometer data). The position is represented in the GPS coordinate system (right-hande d coordinate system, z-axis up). It is designed to be a scaled integer message because the resolution of floating-point numbers is not enough to meet the requirements. The velocity is still in NED coordinates。

name	data type	units	description
time_boot_ms	uint32_t	ms	The timestamp is calculated from the boot of the syste m
lat	int32_t	degE7	latitude
lon	int32_t	degE7	longitude
alt	int32_t	mm	altitude
relative_alt	int32_t	mm	Height from the ground, upward is positive
vx	int16_t	cm/s	Velocity in the x direction, positive to the north
vy	int16_t	cm/s	Velocity in the y direction, positive to the east
vz	int16_t	cm/s	Velocity in the z direction, positive down
hdg	int16_t	cdeg	Vehicle yaw Angle, ranging from 0.0 degrees to 359.99 degrees, if unknown, can be set to: UI NT16_MAX. cdeg is the magnitude. One degree is equal to 100.

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

The GLOBAL POSITION INT displayed in QGC

5.2.6. ACTUATOR_OUTPUT_STATUS

ACTUATOR_OUTPUT_STATUS. The raw value of the servo output (for example, on Pixhawk, from the MAIN and auxiliary ports). This message replaces SERVO_OUTPUT_RAW.

name	data type	units	description
time_uesc	uint64_t	us	The timestamp is calculated from the boot of the system
active	uint32_t		
actuator	float[32]		The value of the servo/motor output array. A zero value indicates an unused channel.

5.2.7. ATTITUDE_TARGE

ATTITUDE_TARGET. Reports the current desired attitude of the vehicle as specified by the autopilot. If the vehicle is controlled by sending a SET_ATTITUDE_TARGET message, this should match the command sent.

name	data type	units	description
time_boot_ms	uint32_t	ms	The timestamp is calculated from the boot of the system
type_mask	uint8_t		Desired attitude bitmap, indicating which desired attitude information should be ignored
q	float[4]		Expected quaternion (w, x, y, z order, zero rotation corresponding to 1, 0, 0, 0)
body_roll_rate	float	rad/s	Desired roll Angle rate
body_pitch_rate	float	rad/s	Desired pitch Angle rate

body_yaw_rate	float	rad/s	Expected yaw Angle rate
thrust	float		Expected throttle

QGC 中显示的 ATTITUDE_TARGET

5.2.8. POSITION_TARGET_LOCAL_NE

POSITION_TARGET_LOCAL_NED. Reports the current desired vehicle position, speed, and acceleration as specified by the autopilot. If the aircraft is controlled in this way by sending SET_POSITION_TARGET_LOCAL_NED messages, then these reports should match the commands sent. **POSITION_TARGET_LOCAL_NED also has a value when SET_POSITION_TARGET_LOCAL_NED is not set,** Because PX4's built-in Navigator can publish POSITION_TARGET_LOCAL_NED.

name	data type	units	description
time_boot_ms	uint32_t	ms	system time
coordinate_frame	uint8_t		The most commonly used coordinate system is MAV FRAME LOCAL NED = 1
type_mask	uint16_t		The bit flag indicates that control information should be ignored
x	float	m	x position in NED coordinates
y	float	m	y position in NED coordinates
z	float	m	NED coordinate height
vx	float	m/s	NED coordinate system velocity in the x direction
vy	float	m/s	NED coordinate system velocity in the y direction

vz	float	m/s	NED coordinate system velocity in the z direction
afx	float	m/s^2	NED coordinates acceleration or force in the x direction
afy	float	m/s^2	NED coordinates acceleration or force in the y direction
afz	float	m/s^2	NED coordinates acceleration or force in the z direction
yaw	float	rad	yaw angle
yaw_rate	float	rad/s	Yaw Angle rate

信息： POSITION_TARGET_LOCAL_NED (85) 50.0Hz
组件： 1
计数： 3472779

名称	值	类型
time_boot_ms	80199785	uint32_t
coordinate_frame	1	uint8_t
type_mask	0	uint16_t
x	-0.0597772	float
y	0.0779866	float
z	-10.013	float
vx	0.00549753	float
vy	-0.00251879	float
vz	-0.0141983	float
afx	-0.00935648	float
afy	-0.000793941	float
afz	-0.0157687	float
yaw	0.488225	float
yaw_rate	0	float

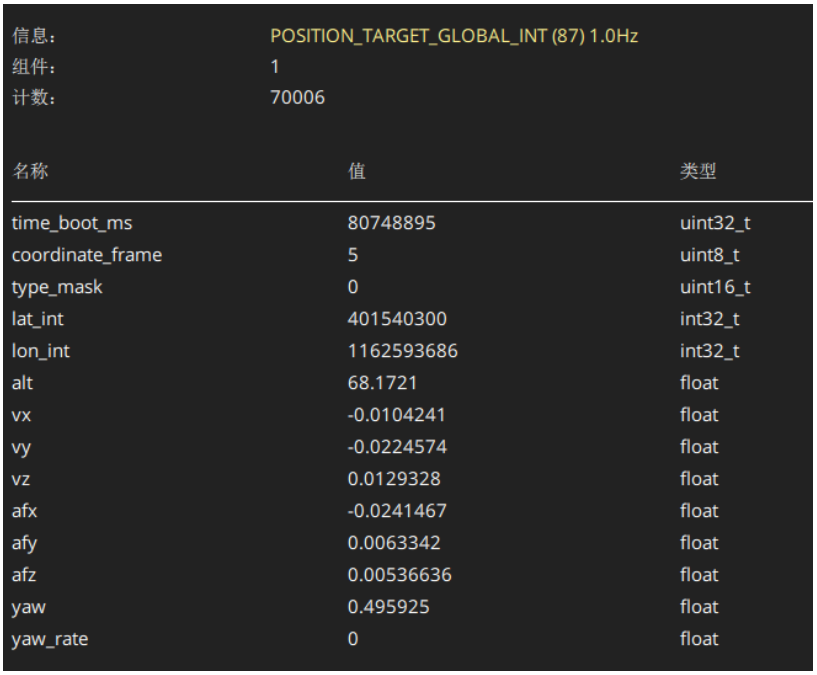
POSITION_TARGET_LOCAL_NED shown in QGC

5.2.9. POSITION_TARGET_GLOBAL_IN

POSITION_TARGET_GLOBAL_INT. Reports the current desired vehicle position, speed, and acceleration as specified by the autopilot. If the aircraft is controlled in this way by sending SET_POSITION_TARGET_GLOBAL_INT messages, then these reports should match the commands sent.

name	data type	units	description
time_boot_ms	uint32_t	ms	system time
coordinate_frame	uint8_t		The most commonly used FRAME is MAV FRAME GLOBAL INT = 5

type_mask	uint16_t		The bit flag indicates that control information should be ignored
lat_int	int32_t	degE7	WGS84 lower x position
lon_int	int32_t	degE7	WGS84 lower y position
alt	float	m	Altitude, or relative altitude, depends on the coordinate system
vx	float	m/s	NED coordinate system velocity in the x direction
vy	float	m/s	NED coordinate system velocity in the y direction
vz	float	m/s	NED coordinate system velocity in the z direction
afx	float	m/s^2	NED coordinates acceleration or force in the x direction
afy	float	m/s^2	NED coordinates acceleration or force in the y direction
afz	float	m/s^2	NED coordinates acceleration or force in the z direction
yaw	float	rad	yaw angle
yaw_rate	float	rad/s	Yaw Angle rate



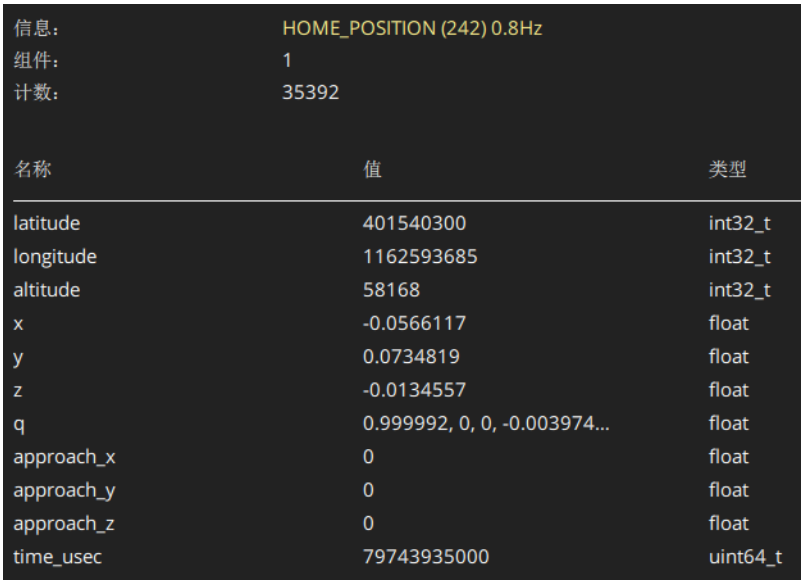
POSITION TARGET GLOBAL INT displayed in QGC

5.2.10. HOME_POSITIO

HOME_POSITION contains home location information. The home location is the defa ult location to which the system will return and land. This position must be set automatic ally by the system during takeoff or can be set explicitly using the MAV_CMD_DO_SET _HOME command. The global and local positions encode the positions in their respective coordinate systems, while the q parameter encodes the orientation of the surface. Under no rmal circumstances, it describes the heading and slope of the terrain, and the aircraft can use this information to adjust its flight en route to landing. A 3D vector on the way to l anding describes the point to which the system should fly in normal flight mode, and the n performs a landing sequence along that vector. Note: This message can be requested by sending the MAV_CMD_REQUEST_MESSAGE and setting param1=242 (or the deprecated MAV_CMD_GET_HOME_POSITION command).

name	data type	units	description
latitude	int32_t	degE7	WGS84 lower x position
longitude	int32_t	degE7	WGS84 lower y position
altitude	int32_t	mm	Altitude, up is positive
x	float	m	NED coordinate system position x
y	float	m	NED coordinate system position y
z	float	m	NED coordinate system position z
q	float[4]		The quaternion represents the take-off pos ition of the world to the surface normal and the heading change. Used to indicate the course and grade of the ground. If an accurate heading and quaternion of surfa ce slope cannot be provided, all fields sh ould be set to NaN.
approach_x	float	m	The local X position near the end of the vector. Multi-rotor vehicles should set this position according to their takeoff path. A fixed-wing vehicle that lands on grass should be set up in the same way as a multi-rotor vehicle. A fixed-wing aircraft l anding on a runway should set it in the opposite direction to takeoff, assuming th

			at takeoff starts from the threshold/touchdown area of the runway.
approach_y	float	m	ditto
approach_z	float	m	ditto
time_usec	uint64_t	us	Timestamp (UNIX epoch time or time since system boot). The receiving end can infer the format of the timestamp (since January 1, 1970 or since the system started) by examining the size of the number.



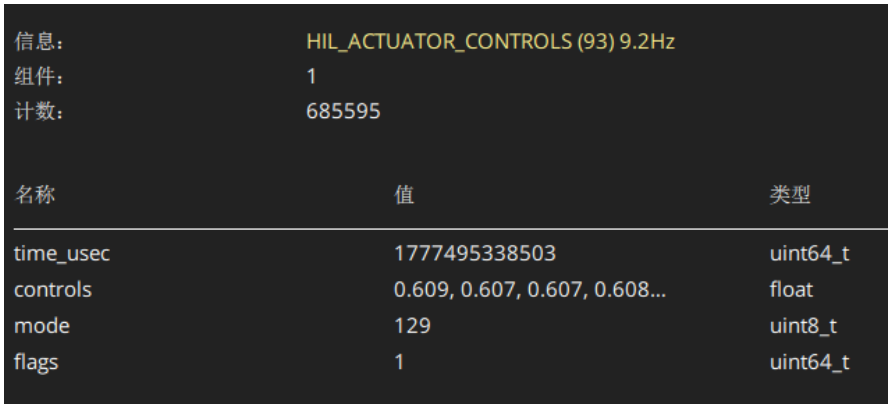
The HOME POSITION displayed in QGC

5.2.11. HIL_ACTUATOR_CONTROLS（PX4 to Sim control output）

HIL_ACTUATOR_CONTROLS. From the autopilot to the simulator. Message for Hardware in the Loop (HIL) control output (used instead of HIL_CONTROLS).

name	data type	units	description
time_uecs	uint64_t	us	Timestamp (UNIX epoch time or time since system boot). The receiving end can infer the format of the timestamp (since January 1, 1970 or since the system started) by examining the size of the number.
controls	float[16]		The control output ranges from -1 to 1. Channel allocation depends on the configuration of the emulation hardware.

mode	uint8_t		System status, including introduction status.
uint8_t	uint64_t		This is a flag bit field where a 1 indicates emulation using a lock step.



HIL ACTUATOR CONTROLS shown in QGC

5.2.12. HIL_SENSOR（Sim to PX4 sensor information）

HIL_SENSOR. IMU (Inertial Measurement Unit) readings are expressed in the International System of Units (SI units) in the NED (North-east-lower) body coordinate system. This means that measurements such as acceleration and angular velocity are measured in meters per second ² and radians per second, relative to the forward, right, and down directions of the vehicle.

name	data type	units	description
time_usec	uint64_t	us	Timestamp (UNIX epoch time or time since system boot). The receiving end can infer the format of the timestamp (since January 1, 1970 or since the system started) by examining the size of the number.
xacc	float	m/s^2	Acceleration in x direction
yacc	float	m/s^2	Acceleration in y direction
zacc	float	m/s^2	Acceleration in z direction
xgyro	float	rad/s	Angular velocity in the x direction
ygyro	float	rad/s	Angular velocity in the y direction
zgyro	float	rad/s	Angular velocity in the z direction
xmag	float	gauss	Magnetic field strength in the x direction
ymag	float	gauss	Magnetic field strength in the y direction

zmag	float	gauss	Magnetic field strength in the z direction
abs_pressure	float	hPa	absolute pressure
diff_pressure	float	hPa	Differential pressure
pressure_alt	float		pressure altitude
temperature	float		air temperature
fields_updated	uint32_t		This is a bitmap that represents the fields that have been updated since the last message. Each bit in a bitmap usually represents a field, and the corresponding bit is set to 1 when the corresponding field has been updated, or 0 otherwise. The use of this bitmap can help efficiently transmit and identify which fields have changed in communication without having to transmit all the data for the entire message.
id	uint8_t		Sensor ID (indexed from zero). When dealing with multiple sensor inputs, this ID can be used to distinguish between different sensors. This helps the system identify and distinguish data from different sources, especially in multi-sensor environments. Each sensor is usually assigned a unique ID for identification and data processing.

Note: Data passed from the simulation model to PX4 cannot be displayed in QGC.

5.2.13. HIL_GPS（Sim to PX4 GPS information）

HIL_GPS.Global location, returned by the Global Positioning System (GPS). This is not a global position estimate of the system, but rather a raw sensor value. To get a global location estimate for the system, look at the GLOBAL_POSITION_INT message. This message contains the raw location information received directly from the GPS, rather than the global location that has been processed and estimated.

name	data type	units	description
time_usec	uint64_t	us	Timestamp (UNIX epoch time or time since syst

			em boot). The receiving end can infer the format of the timestamp (since January 1, 1970 or since the system started) by examining the size of the number.
fix_type	uint8_t		0-1: no fix, 2: 2D fix, 3: 3D fix. Some applications will not use the value of this field unless it is at least 2, so always fill in the correction value correctly. This field is commonly used to indicate the positioning quality of the GPS receiver, with 0 indicating no positioning, 2 indicating 2D positioning, and 3 indicating 3D positioning. In some applications, location information is only used if the GPS has at least a two-dimensional correction, so when reporting GPS location information, ensuring that the correction field is filled in correctly is important for the accuracy of the data.
lat	int32_t	degE7	WGS84 latitude
lon	int32_t	degE7	WGS84 Longitude
alt	int32_t	mm	Altitude, up is positive.
eph	uint16_t		GPS horizontal position accuracy factor (HDOP, no unit * 100). If it is unknown, it can be set to UINT16_MAX. HDOP is a factor that represents the accuracy of GPS positioning, which is usually a unitless decimal number multiplied by 100 to obtain an integer value. A lower HDOP value indicates a higher positioning accuracy, while a higher HDOP value indicates a lower positioning accuracy. If you do not know the GPS horizontal position precision factor, you can set this field to UINT16_MAX to indicate the unknown value.

epv	uint16_t		GPS vertical position accuracy factor (VDOP, none unit * 100). If it is unknown, it can be set to UINT16_MAX. VDOP is a factor that represents the accuracy of GPS positioning, which is usually a unitless decimal number multiplied by 100 to obtain an integer value. A lower VDOP value indicates a higher vertical positioning accuracy, while a higher VDOP value indicates a lower vertical positioning accuracy. If you do not know the GPS vertical position accuracy factor, you can set this field to UINT16_MAX to indicate unknown values.
vel	uint16_t	cm/s	GPS ground speed. If it is unknown, it can be set to UINT16_MAX, which indicates an unknown value. GPS ground speed represents the speed at which the device is moving on the ground as measured by the GPS receiver. This value is usually expressed in meters per second (m/s) and indicates the speed of the device. If an exact value for the ground speed is not available, the field can be set to UINT16_MAX to represent an unknown value.
vn	int16_t	cm/s	The northbound component of GPS speed in the Earth's fixed NED coordinate system.
ve	int16_t	cm/s	The eastern component of GPS speed in the Earth's fixed NED coordinate system.
vd	int16_t	cm/s	The downward component of GPS speed in the Earth's fixed NED coordinate system.
cog	uint16_t	cdeg	Ground heading (not heading, but direction of movement), ranging from 0.0 degree

			s to 359.99 degrees. If it is unknown, it can be set to UINT16_MAX, which indicates an unknown value. Ground heading refers to the direction of the device's movement relative to the ground, rather than the device's orientation. Usually expressed in degrees, it indicates the direction of the device relative to the true north direction. If the exact value of the ground course cannot be obtained, the field can be set to UINT16_MAX to represent an unknown value.
satellites_visible	uint8_t		Number of visible satellites
id	uint8_t		GPS ID number
yaw	uint16_t	cdeg	The yaw Angle of the vehicle relative to the north of the Earth, where zero is not available and 36,000 is used to represent the true north direction. This Angle represents the orientation of the vehicle relative to the true north of the Earth, usually expressed in hundredths of a degree (centidegrees). If the exact value of the yaw Angle cannot be obtained, it can be set to zero, indicating unavailability, or set to 36,000, indicating true north. This value is used to indicate the orientation of the vehicle with respect to the Earth's North Pole.

Note: Data passed from the simulation model to PX4 cannot be displayed in QGC.

5.3. Microservice

The high-level protocol adopted by the MAVLink system is called microservices, which are used for better interoperation. For example, QGroundControl, ArduPilot, and PX4 autopilot all share a common command protocol for sending peer-to-peer messages that require confirmation.

Microservices are used to exchange many types of data, including parameters, tasks, trac

ks, images, and other files. If the data is much larger than the capacity of a single messa ge, the service will define how to split and reassemble the data, and how to ensure that a ny lost data is retransmitted. Other services provide command validation and/or error repor ting.

Most services use a client-server model, where GCS (the client) initiates the request and t he vehicle (the server) responds to the data. MAVLink defines the following microsuits: H eartbeat/Connection protocol, Task protocol, Parameter protocol, Extended parameter protoc ol, Command protocol, Manual control (joystick) protocol, Camera protocol, Camera Defini tion, Universal Protocol v2, Arm License protocol, Image Transfer Protocol, File Transfer Protocol (FTP), login target protocol, Ping Protocol, path planning protocol (trajectory inter face), battery protocol, terrain protocol, tunnel protocol, open drone ID protocol, high laten cy protocol, Component Metadata Protocol (WIP), payload protocol, Traffic Management (UTM/ADS-B), Event Interface (WIP), time synchronization protocol, etc. Specific visible athttps://mavlink.io/en/services/。

This section mainly introduces several protocols involved in the secondary developmen t process of QGC

5.3.1 Heartbeat/connection protocol

The heartbeat protocol is used to notify the presence of a system (QGC or other ope rating software) on the MAVLink network, along with its system and component ID, vehic le type, flight stack, component type, and flight mode.

The heartbeat protocol can perform the following functions: 1. Discover systems conn ected to the network and infer when they are disconnected. If a component regularly recei ves HEARTBEAT messages, it is considered to be connected to the network. If the expect ed message is not received, the component is considered disconnected. 2, appropriately pro cess additional messages from the component based on the component type and other prop erties (for example, layout of the GCS interface based on vehicle type). 3. Route message s to systems on different interfaces.

The HEARTBEAT protocol must be defined at the broadcast rate HEARTBEAT, as w ell as defining how many messages the system is "missing" before they are considered ti meout/disconnected from the network. On an RF telemetry link, for example, a component typically posts its heartbeat at a frequency of 1 Hz, and if four or five messages are not received, the other system is considered disconnected.

If the component does not detect another system, it can choose not to send or broadc ast information on the channel (other than the channel HEARTBEAT), and it will continue to send messages to the system when the heartbeat is received. Therefore, the system mu

st: (1) broadcast the heartbeat even if the remote system is not commanded; (2) Do not b roadcast heartbeats while they are in a failed state (that is, do not publish heartbeats from a separate thread that does not know the state of the rest of the component).

The specific code for connecting to QGroundControl can be found in multivehicleMan ager.cc (see Resources void MultiVehicleManager::_vehicleHeartbeatInfo)。

See the detailed description of related protocols https://mavlink.io/en/services/heartbeat.h tml

5.3.2 Task protocol

Mission sub-protocols enable GCS or developer apis to exchange mission (flight plan), geofencing, and safety point information with UAVs/components.

The protocol covers: (1) Upload, download, clear tasks, set/get the current task item number, and receive notifications when the current task item changes. (2) Message types a nd enumerations used to exchange task items. (3) Task items common to most systems (" MAVLink command "). The protocol supports re-requesting messages that have not yet arr ived, allowing tasks to be reliably transmitted over lossy links.

The agreement mainly includes three types of "missions" : flight plans, geofencing, an d assembly/security points.

Flight plan Task type Command format: 1. NAV command (MAV_CMD_NAV_*) for navigation/movement (for example, MAV_CMD_NAV_WAYPOINT, MAV_CMD_NAV_LAN D) 2. DO command (MAV_CMD_DO_*) for immediate operation. For example, change th e speed or activate the servo system (for example, MAV_CMD_DO_CHANGE_SPEED). T he CONDITION command (MAV_CMD_CONDITION_*) is used to change the execution of the task based on conditions-for example, to pause the task for a period of time before executing the next command (MAV_CMD_CONDITION_DELAY).

Geofencing task type Command format: prefix MAV_CMD_NAV_FENCE_ (for examp le, MAV_CMD_NAV_FENCE_RETURN_POINT).

Aggregation point Task type Command format: Only one aggregation point MAV_CM D: MAV_CMD_NAV_RALLY_POINT.

The task item () is transmitted/encoded in the MISSION_ITEM_INTMAV_CMD messa ge. The message includes fields that identify a specific task item (the command ID) and up to seven optional parameters specific to the command.

Can in SRC/QGroundControl MissionManager/PlanManager. See specific implementatio n in cc

See the detailed description of related protocols https://mavlink.io/en/services/command.

html

5.3.3 Parameter protocol

Parameter protocol services are used to exchange configuration Settings between MAV Link components. Each parameter is represented as a key/value pair. The key is usually a human-readable parameter name (up to 16 characters) and a value - which can be one of many types.

Key/value pairs are defined according to the following principles:

1. The readable name is small but useful (it encodes the parameter name, from which the user can infer the purpose of the parameter).

2, can "out of the box" support the implementation of the protocol of the unknown a utopilot.

3. GCS does not need to know in advance which parameters exist on the remote syst em (although in fact GCS can provide a better user experience by adding parameter meta data (e.g., maximum and minimum values, default values, etc.).

4. Adding parameters only requires changes to the system with parameters. The GCS and MAVLink communication libraries that load parameters do not require any changes.

Can in SRC/QGroundControl FactSystem/ParameterManager. See specific implementatio n in cc

See the detailed description of related protocols https://mavlink.io/en/services/parameter. html

5.3.4 Command protocol

The MAVLink command protocol ensures the transmission of MAVLink commands. T he general MAV_CMD command defines the value of a maximum of 7 parameters. These parameters and command ids are encoded as COMMAND_INT or COMMAND_LONG for sending.

The protocol provides reliable delivery by expecting a match acknowledgement (COM MAND_ACK) from a command to indicate command arrival and result. If no acknowledg ement is received, the command must be resend automatically.

COMMAND_INT Flight stack support for specific commands, used when sending co mmands that contain location or navigation information. This is because it allows specifyin g a coordinate system for position and height values that might otherwise be "unspecified. " In addition, latitude/longitude can be sent as scaled integers in the COMMAND_INT par ameters 5 and 6 with greater precision (more precise than when sent as floating-point valu es in COMMAND_LONG).

COMMAND_LONG is used to send commands that MAV_CMD sends floating-point properties in parameters 5 and 6, because if sent in COMMAND_INT, these values will be truncated to integers.

If the flight stack supports it, you can use non-positional commands in either message or commands that specify integers in parameters 5 and 6.

The COMMAND_INT flight stack can support the command COMMAND_LONG in either message or/or both, despite the loss of precision, rounding error, and/or undefined reference frame. However, they are encouraged to support only the command COMMAND_INT in the position and the command COMMAND_LONG with floating point values in parameters 5 and 6 in the position. The flight stack can use COMMAND_ACK.result as needed or refuse to send MAV_RESULT_COMMAND_LONG_ONLY commands with the "error" message type. MAV_RESULT_COMMAND_INT_ONLY Flight stacks that support only specific commands in specific message types can more generally use these result values to indicate the correct message type for a command

See the detailed description of related protocols https://mavlink.io/en/services/command.html

5.3.5 Manual control protocol (joystick)

The manual control protocol allows the system to be controlled using a "standard joystick" (or joystick-like input device that supports equivalent coaxial nominations).

This protocol implements MANUAL_CONTROL only through messages. It defines the system to be controlled by target, the motion of the four main axes (x, y, z, r) and two extension axes (s, t), and two 16-bit fields to represent the state of up to 32 buttons (buttons, buttons2). You can disable unused axes, and you must explicitly enable the extension axis enabled_extensions with bits 0 and 1 of this field.

The purpose of the protocol is relatively simple and abstract, and provides a simple way to control the main movement of the vehicle, as well as several arbitrary functions that can be triggered using buttons.

This enables the GCS software to provide simple control for multiple types of vehicles and allows new vehicle types with unusual features to operate with minimal, if any, changes to the MAVLink protocol or existing ground control station (GCS) software.

You can see the implementation in src/Joystick/ joystick.cc of QGroundControl

See the detailed description of related protocols https://mavlink.io/en/services/manual_control.html

5.3.6 Camera protocol

The camera protocol is used to configure the camera payload and request its state. It supports photo shooting, video shooting and streaming. It also includes messages for querying and configuring on-board camera storage.

Built-in camera support for MAVLink camera protocol, Workswell cameras: WIRIS Pro, WIRIS Pro SC, WIRIS Security, WIRIS Agro, GIS-320 (source); PhaseOne camera (source).

The Camera Manager provides the MAVLink Camera Protocol interface for cameras that do not directly implement MAVLink support. These usually run on companion computers: MAVLink Camera Manager (Active maintenance), Drone Code Camera Manager.

The camera assembly should follow the heartbeat/connection protocol and send a constant heartbeat stream (usually 1Hz). Each camera must use a different predefined camera component ID: MAV_COMP_ID_CAMERA to MAV_COMP_ID_CAMERA6. The first time a heartbeat is detected from a new camera, the GCS (or other receiving system) should initiate the camera recognition process.

See the detailed description of related protocols https://mavlink.io/en/services/camera.html

5.3.7 Image transfer protocol

The image Transfer protocol uses MAVLink as a communication channel to transfer any type of image (raw image, Kinect data, etc.) from one MAVLink node to another. It basically takes live camera images, splits them up into small pieces and sends them through MAVLink.

The image flow component uses two MAVLink messages: the handshake message DATA_TRANSMISSION_HANDSHAKE for starting image streams and describing images to send, and the data container message ENCAPSULATED_DATA for transporting image data.

The communication is initiated by QGroundControl and the DATA_TRANSMISSION_HANDSHAKE request starts the flow. The message specifies (1) type: enumerates any type in MAVLINK_DATA_STREAM_TYPE in mavlink.h; (2) jpg_quality: Image quality required (for lossy formats such as JPEG) (3) All other fields in the initial request must be zero.

When the target MAV receives a handshake request, it sends back a DATA_TRANSMISSION_HANDSHAKE. This behavior provides confirmation of the request and information about the image to be streamed:

type: The type of image to stream (same as the type requested)

size: Image size in bytes.

width: Image width in pixels.

height: indicates the height of the image in pixels.

packetsENCAPSULATED_DATA: indicates the number of MAVLink packets to be sent

payload: The size of each packet (usually 252 bytes)

jpg_quality: Image quality (same as required)

The image data is then broken into chunks to fit ENCAPSULATED_DATA messages and sent over MAVLink. Each packet contains a serial number and the ID of the image s tream to which it belongs.

The image stream transmitter periodically sends new images without further interactio n. Each new image comes with a new DATA_TRANSMISSION_HANDSHAKEACK packe t size that contains the updated image packets and payload fields. After this ACK packet, the new image arrives as a series of ENCAPSULATED_DATA packets.

To stop the image stream, the GSC must send a new DATA_TRANSMISSION_HAN DSHAKE request packet with all values of 0. The MAVLink node will confirm this by s ending back DATA_TRANSMISSION_HANDSHAKE that also contains a 0 value.

See the detailed description of related protocols https://mavlink.io/en/services/image_tra nsmission.html

5.3.8 File Transfer Protocol (FTP)

File Transfer Protocol (FTP) supports file transfer via MAVLink. It supports common FTP operations such as reading, truncating, writing, deleting and creating files, listing and deleting directories.

The protocol follows a client-server model, where all commands are sent by the GCS (client) and the drone (server) responds with an ACK containing the requested informatio n or a NAK containing an error. GCS sets a timeout after most commands and may rese nd the command if triggered. If a request is received with the same serial number, the dr one must resend the response.

All messages (commands, ACK, NAK) are exchanged within the FILE_TRANSFER_P ROTOCOL message. The message type definition is minimal and has fields for specifying the target network, system, and component, as well as "any" variable-length payload.

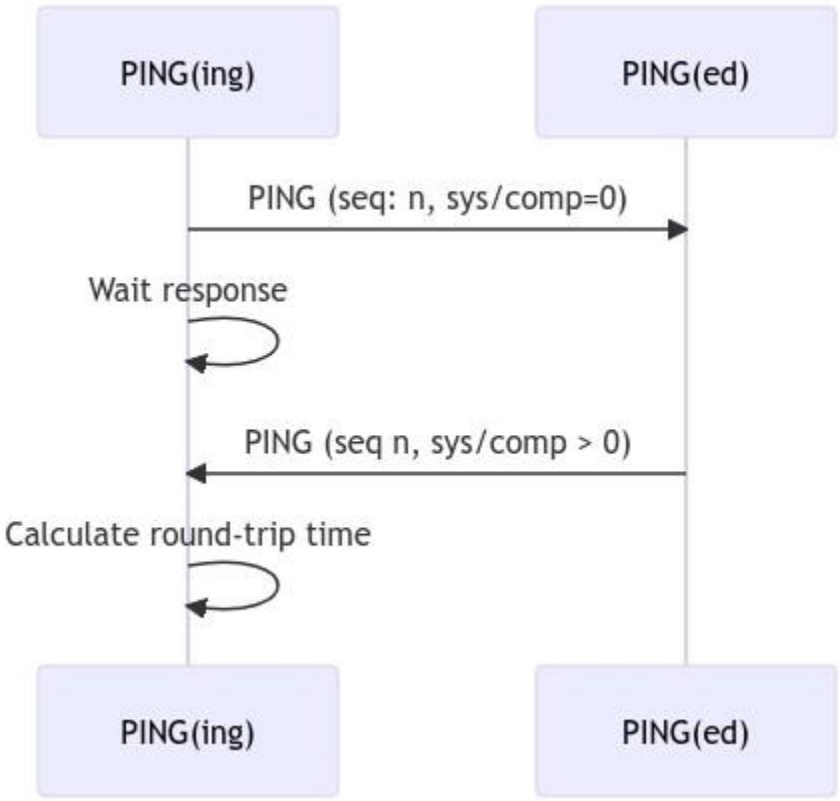
The different commands and other information needed to implement the protocol are encoded in the payload FILE_TRANSFER_PROTOCOL. This topic explains coding, packa ging formats, commands and errors, and the order in which commands are sent to implem ent core FTP functionality.

You can see the implementation in src/uas/FileManager.cc and FileManager.h of QGroundControl

See the detailed description of related protocols https://mavlink.io/en/services/ftp.html

5.3.9 PING protocol

The PING protocol enables the system to measure system latency on any connection: serial ports, radio modems, UDP, etc. The simplified timing diagram is as follows:



The PING system initially populates the PING message with the following

time_usec: indicates the current system time stamp.

seq: indicates the current PING sequence number (n, n+1,...). . This should PING iteratively for each message sent and overflow back to zero.

target_system and target_component: 0 (for the PING request).

The message header automatically contains the sender system.

The message can be received by multiple systems. All ping ed systems should respond with another PING message where:

The original timestamp and serial number PING received is sent back in the response.

target_system and the target_component is set to the ID of the ping system from the incoming ping message header.

You can see the implementation in src/uas/FileManager.cc and FileManager.h of QGroundControl

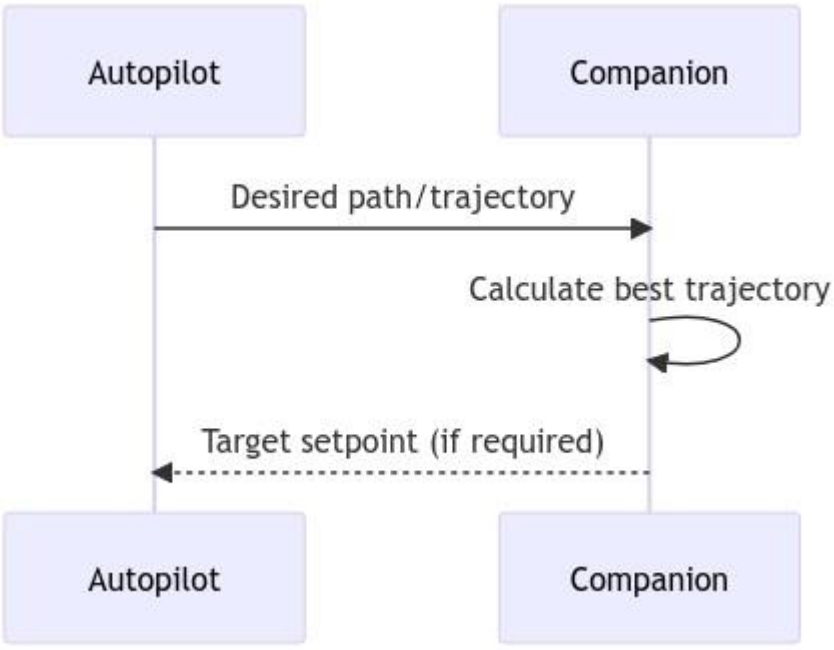
See the detailed description of related protocols https://mavlink.io/en/services/ftp.html

5.3.10 Path Planning Protocol (Trajectory Interface)

The path planning protocol (also known as the trajectory interface) is a common protocol for a system to request a dynamic path planning from another system (i.e. the autopilot requests a path from the companion computer).

This protocol is primarily suitable for situations where the path constraints to a destination are unknown or may change dynamically, but it can also be used for any other path management activity. Examples include avoiding obstacles while performing a pre-planned task, determining the path of a self-forming/repairing swarm, offloading geofencing management to a matching computer, and more.

A (autopilot) system that requires path planning sends a message containing its current location and the desired trajectory. The path planning system (companion computer) analyzes the required route and sends back a message flow with a new path setting value. The relevant process is shown in the figure.



See the detailed description of related protocols https://mavlink.io/en/services/trajectory.html

5.3.11 Battery protocol

MAVLink provides a number of messages for providing battery information: battery status information that can be periodically accessed through BATTERY_STATUS. The SMA

RT_BATTERY_INFO command enables you to obtain some battery information, such as th e device name.

Messages are sent individually for each battery in the system (messages have an insta nce ID field that identifies the corresponding battery). GCS is responsible for providing ap propriate mechanisms that allow users to evaluate the overall battery status on systems wit h multiple batteries.

A smart battery connected to a flight controller via a non-MAVLink bus is considered part of the flight controller assembly. Specifically, the battery message is sent along with the automated driving system and component ID and the MAV_TYPE vehicle type.

Smart batteries that are different components on the MAVLink network must: issue H EARTBEAT = MAV_TYPE_BATTERY - have a unique component ID HEARTBEAT.type within the MAVLink system. By default, the first two battery instances should use MAV_ COMP_ID_BATTERY and MAV_COMP_ID_BATTERY2. Subsequent instances can use any alternate/unused ID.

See the detailed description of related protocols https://mavlink.io/en/services/battery.ht ml

5.3.12 Event Interface (WIP)

An event interface is a general and flexible mechanism that allows a component to re liably notify the GCS (or any other component) of unexpected events and state changes. F or example, the interface can be used to notify readiness, calibration completion, and targe t takeoff altitude.

This interface provides common events shared by the flight stack or other components as well as implementation-specific events. MAVLink "common" events in MAVLink/libeve nts/events/common definition in json.

The interface provides the following main functions:

- Reliable delivery with retransmission
- A consistent interface for reporting system health and defense checks.
- Minimizes buffer requirements on the autopilot side.
- Minimizes binary message length
- General: not related to autopilot and GCS.
- Long-term stability and scalability
- allows parameters to be attached to events.
- Possible type: uint8, int8, uint16, int16, uint32, int32, int64, uint64, float
- enumerations and bit fields can be built upon these types
- Enable automatic processing (e.g. from a flight log containing events).

- Minimizes the amount of automatically generated code for embedded implementations.
- Average event volume <1 Hz (may change as protocol parameters are adjusted, such as retransmission timeout).
- Events can be targeted or broadcast
- Any component can send events, including cameras, companion computers, ground stations, and more.
- events have metadata, such as log levels. They can also have detailed, broader descriptions, possibly with urls.
- Supports message text and message translation.

See the detailed description of related protocols https://mavlink.io/en/services/events.html

5.3. CopterSim MAVLink_Simple

By encapsulating the details of the MAVLink protocol, MAVLink_Simple makes it easier for users to establish communication links, send and receive messages without having to delve into the underlying details of MAVLink

5.4. CopterSim MAVLink_Full

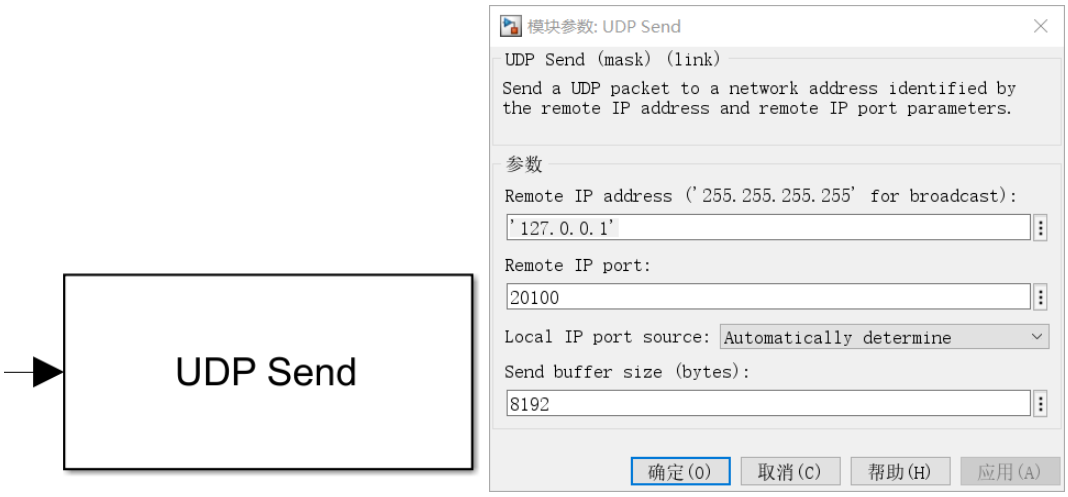
6. Control interface（original,PX4MavCtrlV4.py）

Currently, although CopterSim supports UDP_Simple control posture, the implementation of PX4MavCtrlV4.py does not.

6.1. UDP control interface of Simulink

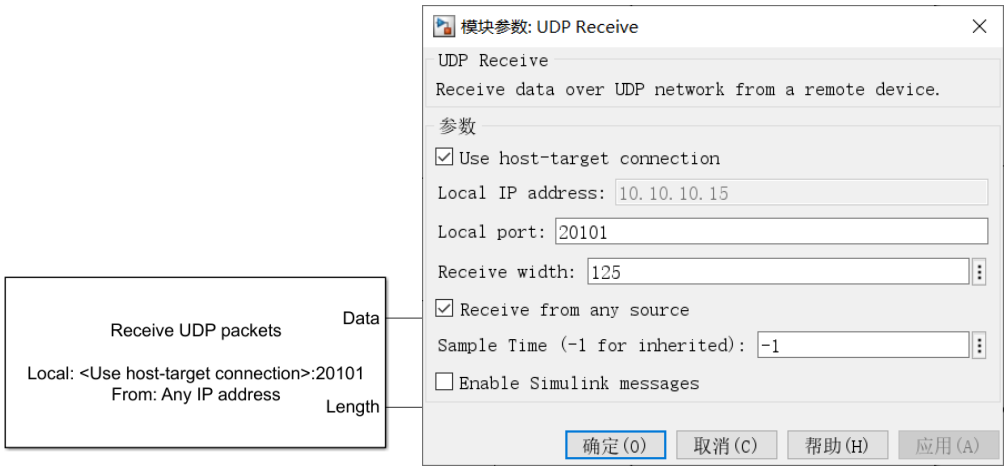
6.1.1. UDP Send (UDP Send Byte Stream Module)

UDP Send is a byte stream sending module provided by Simulink, whose input is a byte stream and output to a specific port in the network. The module has three parameters: network IP, network port, and cache size. After the original data is packaged, the user can use the UDP Send module to send the data.



6.1.2. Receice UDP (UDP Receive Byte Stream Module)

Simulink provides Receive UDP packets for receiving UDP data. UDP data sent by C opterSim can be received using this module. Receive UDP packets The local host can be selected as the host or the IP address can be specified if this parameter is not selected. Y ou also need to specify the port and the length of the received data.

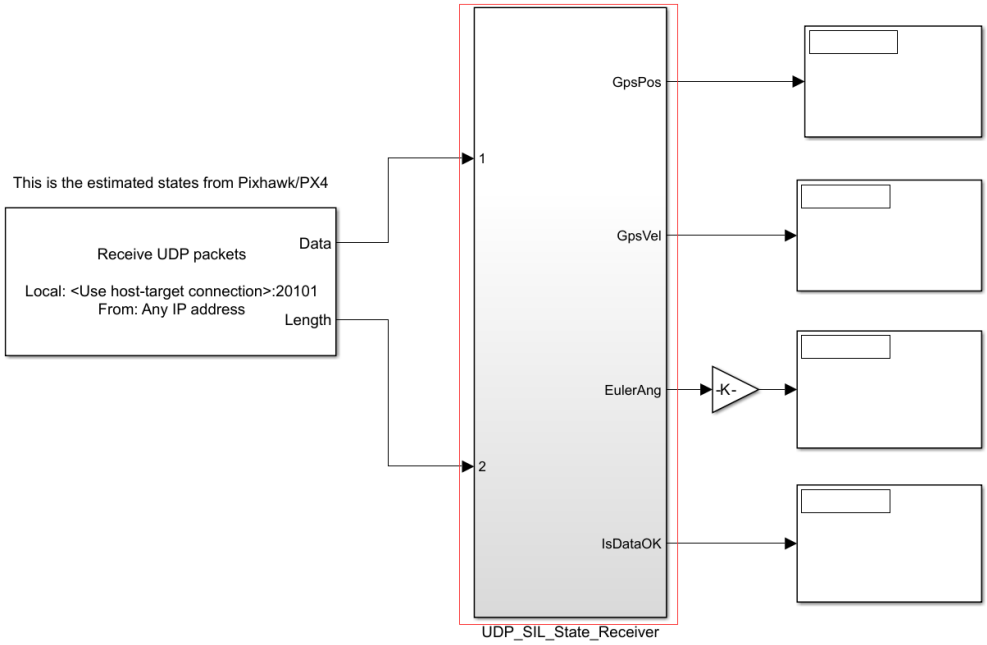


6.1.3. UDP_SIL_State_Receiver (Module for receiving sim ulation position, speed, attitude, etc.)

UDP_SIL_State_Receiver Used for interface PX4 EKF2 estimation information. By def ault, only position, speed, and attitude information are parsed in the platform, while the ac tual message data format from PX4 is outHILStateData, including relative height, NED po sition, speed, etc. Users who want to use this information need to modify the internal imp lementation of UDP_SIL_State_Receiver to parse this information.

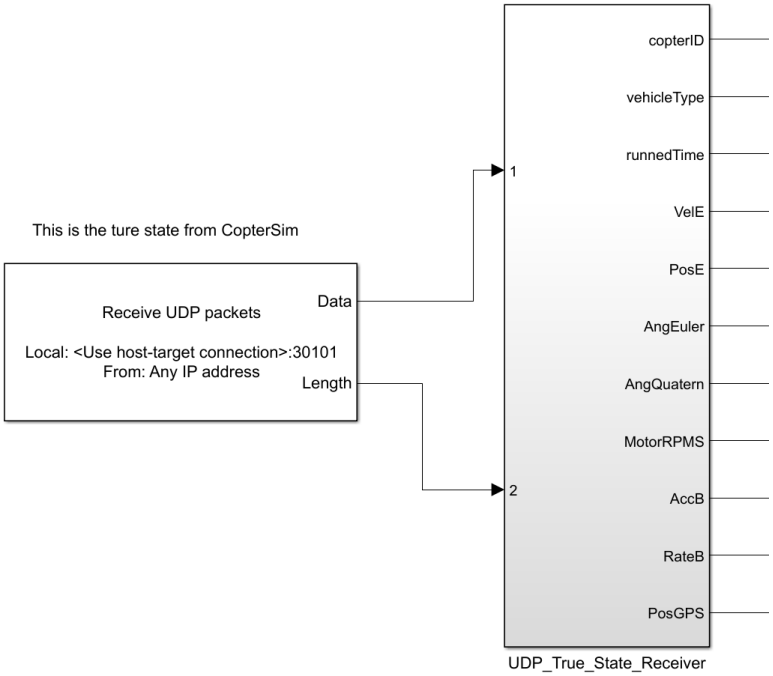
output parameter	GpsPos	GPS location, int32_t, lat&long: deg*1e7, alt: m*1e3

	GpsVel	GPS speed, int32 t, NED, m/s*1e2->cm/s
	EulerAng	Euler's Angle, float
	IsDataOK	Indicates whether the data is valid. The value bool is 1 when valid
output parameter	data	Byte stream, unit8
	len	Byte stream length, unit16



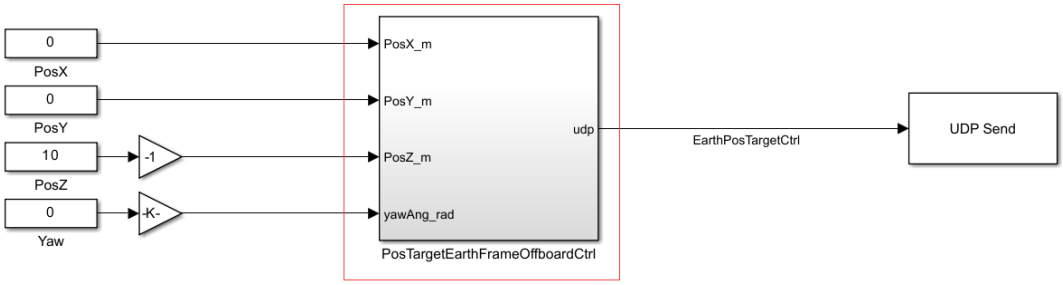
6.1.4. UDP_True_State_Receiver (Module for receiving information such as real position, speed, and attitude)

UDP_True_State_Receiver is used to parse SOut2Simulator data. The module used to get data from UDP is the same as that used to receive SIL data, but the port has been changed from 20101 to 30101. Similarly, users can modify the internal implementation of UDP_True_State_Receiver to reduce or increase the amount of specific data parsed.



6.1.5. Location control (location messages packaged into byte streams)

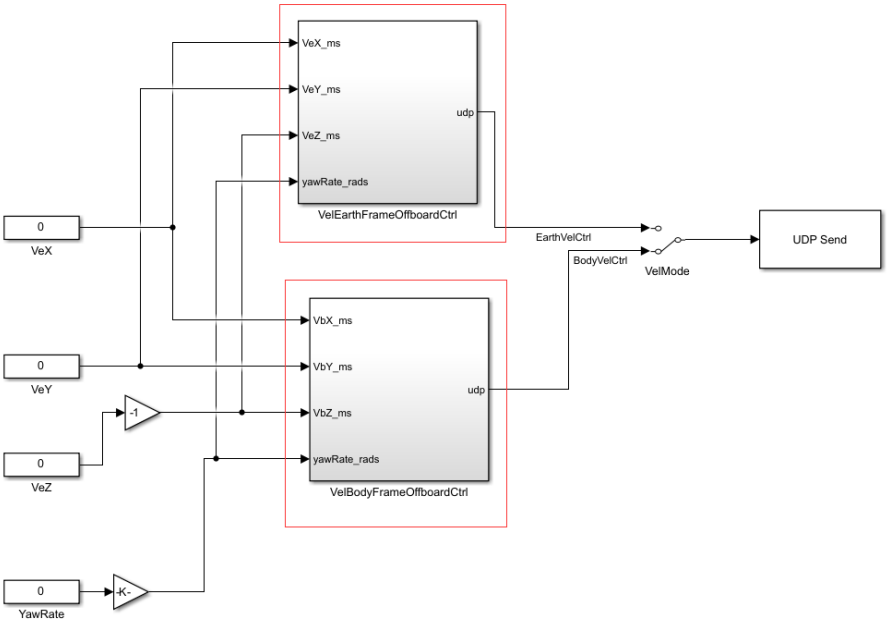
Platform provides PosTargetEarthFrameOffboardCtrl position control module is used to package information into a byte stream. The user can control the position through this mo dule. The values of PosX, PosY, PosZ, and Yaw on the left can be modified during the r unning of the program. For more complex control, you can replace PosX, PosY, PosZ, an d Yaw with sinusoidal signals or custom functions.



6.1.6. Speed control (speed messages packaged into byte streams)

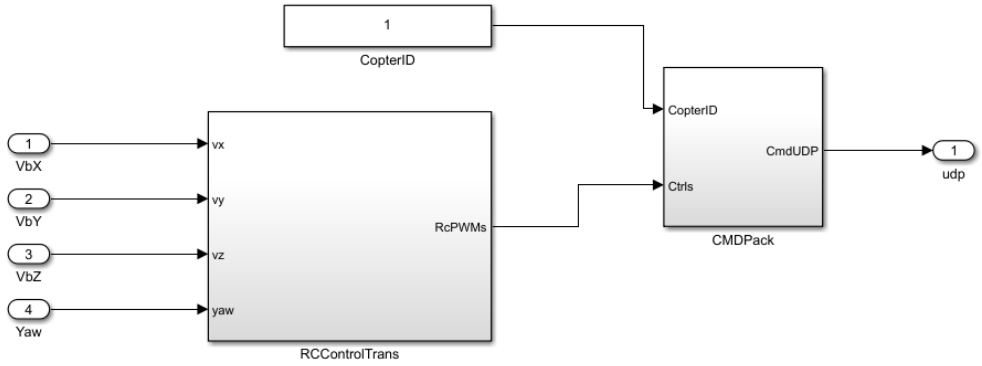
The speed control usually supports NED coordinate system and body coordinate syste m. VelEarthFrameOffboardCtrl and VelBodyFrameOffboardCtrl within the specified coordina te system is different. Simulink speed control uses the platform's simplified UDP_Simple

mode, i.e. [vx, vy,vz, yaw_rate].



6.1.7. Analog remote control PWM control

The key to analog remote control is to convert the speed signal into PWM waves. T he input to the module is still the speed, but the output is the pulse width of the PWM wave. RCOverrideMavlink module, the final encapsulated data needs to be parsed accordin g to inHILCMDData.



6.2. Python UDP control interface

PX4MavCtrler is the core implementation class for python control, and it is also the core class for users to obtain and control vehicle status through python.

6.2.1. PX4MavCtrler:__init__()（Initialization of parameters）

Function prototype: __init__(self, port=20100,ip='127.0.0.1')

Parameters: (self, port=20100,ip='127.0.0.1')

Returned value: Object of type PX4MavCtrler

PX4MavCtrler:__init__() is a type initialization function that initializes vehicle status and connection parameters. You can specify two parameters, port and ip. __init__() is called automatically when the object is initialized.

	name	description	default
output parameter	port	port	20100, number
	ip	UDP communication ip address	“127.0.0.1”,string
returned value	PX4MavCtrler	A PX4MavCtrler object	--

6.2.2. InitMavLoop()（Initialize CopterSim to listen to MAVlink）

Function prototype: InitMavLoop(self,UDPMode=2)

Parameter: InitMavLoop(self,UDPMode=2). Parameter Settings during object initialization affect the behavior of this function. UDPMode Specifies the communication mode.

Returned value: None

PX4MavCtrler: The InitMavLoop() function is used to establish the receive and send connections, and almost every routine uses this interface. This function supports the parameters UDPMode, 0-UDP_Full, 1-UDP_Simple, 2-MAVLink_Full, 3-MAVLink_Simple, and 4-MAVLink_NoSend. InitMavLoop() initializes different connection objects depending on the mode. Then, two threads are initialized to get status information and to send control instructions. The receiving thread calls getMavMsg(), which is started at InitMavLoop(). The sending thread calls OffboardSendMode(), which is only called when the thread is in Offboard mode.

6.2.3. endMavLoop()（Stop Mavlink listening）

Function prototype: endMavLoop (self)

Parameter: None

Returned value: None

PX4MavCtrler: endMavLoop() is used to stop listening for Mavlunk messages and running. It calls the stopRun() method to stop receiving messages from port 20100 or the serial port.

6.2.4. initOffboard()（Send offboard to PX4）

Function prototype: initOffboard(self)

Parameter: None

Returned value: None

PX4MavCtrler: initOffboard() is used to switch the mode of the vehicle to Offboard mode, and then the user can send Offboard position, speed, acceleration, yaw Angle, yaw Angle rate and other control information. This function not only causes the vehicle to enter Offboard mode, but also causes the vehicle to unlock. After initOffboard() is run, Offboard messages are continually sent via OffboardSendMode(). initOffboard() sets the speed, yaw Angle rate to 0 and sends.

6.2.5. initOffboard2()（Send offboard to PX4）

Function prototype: initOffboard2(self)

Parameter: None

Returned value: None

PX4MavCtrler: initOffboard2() is also used to enter the Offboard mode. Unlike initOffboard(), initOffboard2() does not reset the control information to 0. initOffboard2() is suitable for over-the-air switching to Offboard mode where control information is not expected to be set to 0.

6.2.6. InitTrueDataLoop()（Example Initialize listening for UDP True）

Function prototype: InitTrueDataLoop(self)

Parameter: None

Returned value: None

PX4MavCtrler: InitTrueDataLoop() is to initialize the UDP True data listening loop. It first binds the UDP socket to the specified port (self.port+1+10000) and then sets stopFlagTrueData to False, indicating that the listening loop is not stopped. Next, create a thread tTrue with the object function getTrueDataMsg, and start the thread tTrue. Then bind another UDP socket to another port (self.port+1+20000), and set stopFlagPX4Data to False to

not stop the listening loop. Create another thread tPX4 with the object function getPX4DataMsg and start thread tPX4.

6.2.7. EndTrueDataLoop()（Example End UDP True listening）

Function prototype: EndTrueDataLoop(self)

Parameter: None

Returned value: None

EndTrueDataLoop is the end of the True data schema. It starts by setting stopFlagTrueData to True, which stops the listening loop for True data. Then set hasTrueDataRec to False, indicating that no True data was received. Finally close the UDP socket udp_socket True. Finally, stopFlagPX4Data is set to True to stop the listening loop for PX4 data. Then wait for the tPX4 thread to finish. Finally close the UDP socket udp_socketPX4

6.2.8. Access the read status of PX4MavCtrler member variables

The PX4MavCtrler class has the following member variables

variate	annotation	variate	annotation
uavTimeStmp	Aircraft time stamp	trueTimeStmp	True timestamp
uavAngEular	Estimated Euler Angle	trueAngEular	The real Euler Angle
uavAngRate	Estimated angular velocity	trueAngRate	The true angular velocity
uavPosNED	Estimated local position (NED coordinates)	truePosNED	Real position (NED coordinate system)
uavVelNED	Estimated local speed	trueVelNED	Real speed
uavPosGPS	Estimated GPS position (NED coordinates)	uavPosGPSHome	Estimated GPS starting position (NED coordinates)
uavGlobalPos	Estimated global position (converted to	trueAngQuatern	A real quaternion

	UE4 map coordinate system)		
trueMotorRPMS	Real motor speed	trueAccB	True acceleration
truePosGPS	Real GPS location	trueSimulinkData	Real Simulink data
useCustGPSOri	Whether to use custom GPS directions	trueGpsUeCenter	Real GPS UE central location
GpsOriOffset	GPS 方向偏移量	uavThrust	Estimated thrust
pos	位置	vel	speed
acc	加速度	yaw	Yaw Angle
yawrate	偏航角速率		

6.2.9. SendVelNED()（Send maximum speed to PX4）

Function prototype: SendVelNED(self,vx=0,vy=0,vz=0,yawrate=0)

Arguments: (self,vx=0,vy=0,vz=0,yawrate=0), self indicates that the PX4MavCtrler shared object affects the behavior of this function. The default value for each parameter is 0.

Returned value: None

Sends velocity control information in NED coordinates with yaw Angle rate. The function contains four input parameters, vx=0, vy=0, vz=0, yawrate=0. The default value for each argument is 0, that is, if SendVelNED() takes no arguments, the function will send the expected speed 0 and the expected yaw Angle rate 0.

6.2.10. SendVelNEDNoYaw()（Maximum transmission speed without yaw）

Function prototype: SendVelNEDNoYaw(self,vx,vy,vz)

Arguments: (self,vx,vy,vz), self indicates that the PX4MavCtrler shared object affects the behavior of this function. Parameters have no default value, vx,vy,vz must be specified.

Returned value: None

Sends velocity control information in NED coordinate system without yaw Angle rate. This function takes three parameters, vx,vy,vz. There are no default values for these three parameters. It is worth noting that no yaw rate and a yaw rate of 0 are two different things/

6.2.11. SendVelFRD()（Maximum sending speed under FRD framework）

Function prototype: SendVelFRD(self,vx=0,vy=0,vz=0,yawrate=0).

Arguments: (self,vx=0,vy=0,vz=0,yawrate=0), self indicates that the PX4MavCtrler shared object affects the behavior of this function. The default value for each parameter is 0.

Returned value: None

This function is the expected velocity in the specified carrier coordinate system, in m/s. In the absence of any parameters it will send the expected velocity 0 and the expected yaw Angle rate 0.

6.2.12. SendVelNoYaw()（Send maximum speed under FRD frame without rolling down）

Function prototype: SendVelNoYaw(self,vx,vy,vz)

Arguments: (self,vx,vy,vz), self indicates that the PX4MavCtrler shared object affects the behavior of this function. Parameters have no default value, vx,vy,vz must be specified.

Returned value: None

Send the velocity control information in the FRD carrier coordinate system without yaw Angle rate. This function takes three parameters, vx,vy,vz. There are no default values for these three parameters. It is worth noting that no yaw rate and a yaw rate of 0 are two different things.

6.2.13. SendPosNED()（Send coordinates to PX4）

Function prototype: SendPosNED (self,x=0,y=0,z=0,yaw=0)

Parameters: x, y and z represent the position in the NED coordinate system respectively, yaw represents the yaw Angle, and the default value is 0.

Returned value: None

Send position and yaw Angle in NED coordinates.

6.2.14. SendVelYawAlt()（Send gesture to PX4）

Function prototype: SendVelYawAlt(self,vel=10,yaw=6.28,alt=-100)

Parameters: vel represents the horizontal speed, yaw represents the yaw Angle, and alt represents the altitude.

Returned value: None

Send the altitude, yaw Angle and horizontal speed in the NED coordinate system.

6.2.15. SendPosGlobal()（Send target location to PX4）

Function prototype: SendPosGlobal(self,lat=0,lon=0,alt=0,yawValue=0,yawType=0)

Parameters: lat&lon indicates latitude and longitude, unit °; alt indicates height, down ward is positive, unit m. The meaning of yawValue is determined by yawType. yawType 0, yaw is not specified. yawType 1, yaw Angle control; yawType 2, yaw Angle rate.

Returned value: None

To send the global position, the input latitude and longitude of the function are meas ured in °, and the height is measured in m. However, data is automatically converted to MAV_FRAME_GLOBAL_INT when it is sent.

6.2.16. SendPosNEDNoYaw()（Send target position with out yaw control）

Function prototype: SendPosNEDNoYaw (self,x=0,y=0,z=0)

Parameters: x, y and z represent the position in the NED coordinate system, respectiv ely.

Returned value: None

Control position in NED coordinate system without specifying yaw Angle.

6.2.17. SendPosFRD()（Send the location under FRD to PX4）

The original intention of this function is to control the position in the FRD carrier co ordinate system, but PX4 does not support this format, so the function does not work as expected.

6.2.18. SendPosFRDNoYaw()（Send position under FRD without yaw control to PX4）

The original intention of this function is to control the position in the FRD carrier co ordinate system, but PX4 does not support this format, so the function does not work as expected.

6.2.19. SendPosNEDExt()（Send target position to fixed wing）

Function prototype: SendPosNEDExt(self,x=0,y=0,z=0,mode=3,isNED=True)

Parameters: x, y, z indicate position, mode is used to control the flight mode of the fixed wing. mode 0: coasting mode. mode 1, take-off mode; mode 2, landing mode; mode 3, hover mode; mode 4, zero throttle, roll and pitch.

Returned value: None

This function is for fixed wing only and supports multiple modes.

6.2.20. sendPX4UorbRflyCtrl()（Send data to CopterSim）

The function prototype: sendPX4UorbRflyCtrl (self, data =,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 [0], modes = 1, flags = 1)

Parameter: data is a 16-dimensional control quantity, the default is all 0; modes are modes and flags are flags.

Returned value: None

This function implements sending inHILCMDData data to CopterSim.

6.2.21. SendAccPX4()（Send acceleration information to PX4）

Function prototype: SendAccPX4(self,afx=0,afy=0,afz=0,yawValue=0,yawType=0,frameType=0)

Parameters: afx, afy, afz represent acceleration, yawValue represents yaw Angle or angular velocity.

yawType 0: no yaw; yawType 1: yaw Angle control; yawType 2: yaw Angle rate control.

frameType 0: NED coordinate system; frameType 1: FRD carrier coordinate system.

Returned value: None

This function is used for acceleration control and the interface can be supported in UDP_Full and MAVLINK modes. CopterSim already supports acceleration control in UDP_Simple mode, but the current python interface of PX4MavCtrlV4.py does not support it.

6.2.22. endOffboard()（Send out offboard mode to PX4）

Function prototype: endOffboard(self)

Parameter: None

Returned value: None

Set isInOffboard to False and restore the corresponding flag for PX4 to the state it was as before entering Offboard.

6.2.23. stopRun()（Stop listening for mavlink messages）

Function prototype: stopRun(self)

Parameter: None

Returned value: None

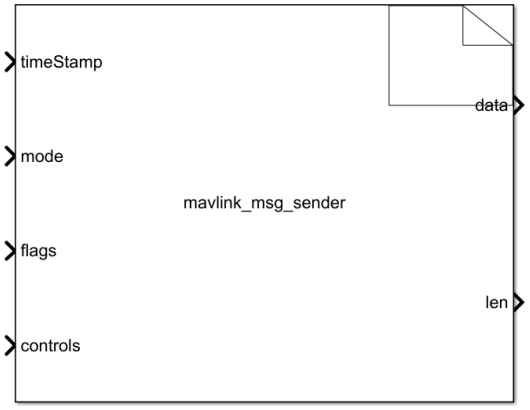
stopRun is used to stop the mavlink listening loop. It first checks if it is unlocked (isArmed), and if it is, sends an unlock command to the flight controller. Then set stopFlag to True to stop the listening loop. Then wait 0.5 seconds for the t1 thread to finish. If you are currently in Offboard mode, the endOffboard method is called to endOffboard mode. If UDPMode is greater than 1.5, the connection to flight control is closed. Otherwise, close the UDP socket if it is not a serial port connection

6.3. Simulink's MAVLink control interface (serial connection)

6.3.1. mavlink_msg_sender

The following interface is provided in Simulink for packaging data into MAVLink format. mavlink_msg_sender is an S function that provides the default interface for sending control messages. If the user is just sending regular control instructions, the data can be packaged directly using the module. If there are some special needs, you can modify the input and output data format according to the Simulink S function writing method. 【7.RflySimExtCtrl\1.BasicExps\e0_ExtAPIUsage】

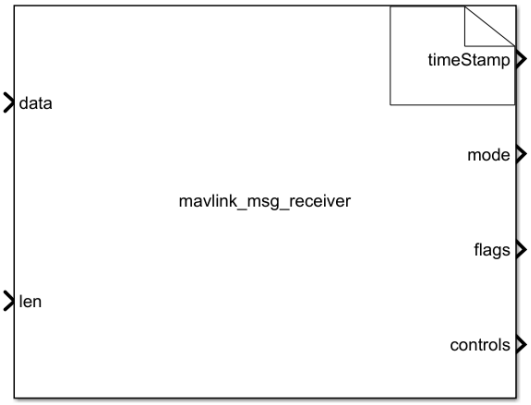
input parameter	timeStamp	Time stamp, unit32
	mode	Mode, unit8
	flags	Flag, uint32
	controls	16-bit control signal single
output parameter	data	Byte stream, unit8
	len	Byte stream length, unit16



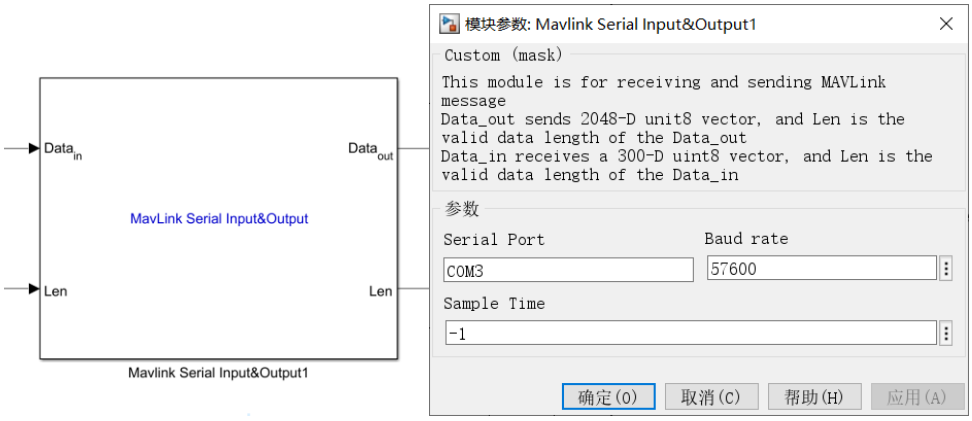
6.3.2. mavlink_msg_receiver

mavlink_msg_receiver is the reverse procedure of mavlink_msg_sender for parsing byte stream data. The data format of the mavlink_msg_receiver must strictly correspond to that of the mavlink_msg_sender. 【7.RflySimExtCtrl\1.BasicExps\e0_ExtAPIUsage】

output parameter	timeStamp	Time stamp, unit32
	mode	Mode, unit8
	flags	Flag, uint32
	controls	16-bit control signal single
input parameter	data	Byte stream, unit8
	len	Byte stream length, unit16



6.3.3. MavLink Serial Input&Output



6.4. Python's MAVLink control interface (based on pymavlink)

6.4.1. SendMavArm () (issue a disarmament order to PX 4)

Function prototype: SendMavArm(self, isArm=0)

Parameter: isArm indicates whether the drone is interpreted, 1 indicates unlocking, 2 indicates locking

Returned value: None.

SendMavArm sends a command to PX4 to unlock or lock the drone.

6.4.2. SendMavCmdLong() (Sending the command length to PX4)

Function prototype: SendMavCmdLong(self, command, param1=0, param2=0, param3=0, param4=0, param5=0, param6=0, param7=0)

Parameter: command indicates the command to be sent.

param1: Hold time (fixed wing ignore, rotor stop time)

param2: Acceptance radius (if in the sphere of this radius, the route is reached)

param3: Through radius 0 indicates that WP is passed. If the radius is greater than 0, WP is passed. Positive clockwise orbits and negative counterclockwise orbits allow trajectory control.

Param4: yaw Angle (rotor) required for yaw waypoint. Use the current system yaw headin g mode (for example, yaw to next waypoint, yaw home, etc.).

Param5: Latitude

Param6: Longitude

Param7: Height (in meters)

Returned value: None.

SendMavCmdLong is a command length message that sends Mavlink to PX4. And ac cording to the current connection mode and flight mode, select different sending methods. If it is a serial port connection or in real flight mode, the function sends the command u sing the mav.mand_long_send method of the the_connection object. For UDP connection m ode, the function generates the command using the command_long_encode method of the mav0 object.

6.4.3. sendMavOffboardCmd() (send an off-board comma nd to PX4)

Function prototype: sendMavOffboardCmd(self,type_mask,coordinate_frame, x, y, z, v x, vy, vz, afx, afy, afz, yaw, yaw_rate)

Parameters: type_mask Indicates the bitmask of the control mode. coordinate_frame re presents the type of coordinate system;

x, y, z represent the coordinates of the target location; vx, vy, vz represent the comp onents of the target velocity; afx, afy, afz represent the components of the target accelerati on; yaw represents the target course Angle; yaw_rate represents the angular rate of the tar get course;

Returned value: None.

sendMavOffboardCmd sends the Offboard command to PX4.

6.4.4. initRCSendLoop() (Initializing remote control)

Function prototype: initRCSendLoop(self, Hz=30)

Parameter: Hz Indicates the frequency of remote transmission. The default value is 30 Hz.

Returned value: None.

initRCSendLoop initializes the remote control and sends a loop

6.4.5. SendRCPwms() (Update PWM value of remote control)

Function prototype: SendRCPwms(self, Pwms)

Parameter: Pwms is a list of PWM values

Returned value: None.

SendRCPwms sends a list of the given PWM values to the remote control.

6.4.6. endRCSendLoop() (Stopping the remote transmission loop)

Function prototype: endRCSendLoop(self)

Parameter: None

Returned value: None.

endRCSendLoop Ends the remote control sending loop.

6.4.7. SendSetMode() (Send the mavlink command to switch the flight mode)

Function prototype: SendSetMode(self,mainmode,cusmode=0)

Parameter: mainmode indicates the main flight mode; cusmode Specifies the user-defined mode. This parameter is optional. The default value is 0.

Returned value: None.

SendSetMode is the MAVLink command sent to PX4 to change the flight mode.

6.4.8. SendAttPX4() (sends speed control signal to PX4)

Function prototype: SendAttPX4(self,att=[0,0,0],thrust=0.5,CtrlFlag=0,AltFlg=0)

Parameter: att indicates attitude and thrust indicates accelerator.

CtrlFlag 0, Euler Angle, unit degree;

CtrlFlag 1, Euler Angle, unit rad;

CtrlFlag 2, Euler Angle, quaternion;

CtrlFlag 3, Euler Angle, angular rate rad/s;

CtrlFlag 4, Euler Angle, angular rate °/s.

AltFlg 0, throttle values zoom to 0~1.

AltFlg>0, the throttle value is the desired height.

Returned value: None

This function is used to control the attitude of the vehicle, and the interface is only supported in MAVLink mode. CopterSim can perform hardware-in-the-loop emulation in UDP mode, in which case Python scripts can bypass CopterSim and send MAVLink messages directly to PX4. CopterSim already supports control attitude in UDP_Simple mode, but the current python interface of PX4MavCtrlV4.py does not support it.

6.4.9. enFixedWRWTO () (order the aircraft to take off from the runway)

Function prototype: enFixedWRWTO(self)

Parameter: None

Returned value: None.

enFixedWRWTO is to send the command to enable the aircraft to take off on the runway.

6.4.10. SendCruiseSpeed()（Send the command to change the cruising speed of the aircraft）

Function prototype: SendCruiseSpeed(self,Speed=0)

Parameter: Speed Indicates the cruising speed. The default is 0.

Returned value: None.

SendCruiseSpeed is the command sent to change the cruising speed of the aircraft (m/s).

6.4.11. SendCopterSpeed()（Set the maximum multi-rotor speed）

Function prototype: SendCopterSpeed(self,Speed=0)

Parameter: Speed Indicates the maximum speed. The default value is 0.

Returned value: None.

SendCopterSpeed is the command sent to set the maximum speed of a multi-rotor vehicle.

6.4.12. SendGroundSpeed()（Set the ground speed of the aircraft）

Function prototype: SendGroundSpeed(self,Speed=0)

Parameter: Speed Indicates the ground speed. The default value is 0

Returned value: None.

SendGroundSpeed is the command sent to change the ground speed of the aircraft (m/s).

6.4.13. SendCruiseRadius()（Set the cruising radius of the aircraft）

Function prototype: SendCruiseRadius(self,rad=0)

Parameter: rad indicates the cruising radius. The default is 0.

Returned value: None.

SendCruiseRadius is the command sent to change the cruising radius of the aircraft (in meters).

6.4.14. sendTakeoffMode()（Take-off order）

Function prototype: sendTakeoffMode(self,alt=0)

Parameter: alt indicates the takeoff height. The default value is 0.

Returned value: None.

sendTakeoffMode sends the command to make the plane take off.

6.4.15. sendMavTakeOff()（Order the plane to take off to the desired position）

Function takeoff: sendMavTakeOff(self,xM=0,yM=0,zM=0,YawRad=0,PitchRad=0)

Parameters: xM, yM, zM represent the local coordinates of the target location; YawRad represents the target course Angle (radian);

PitchRad represents the pitch Angle of the target (radian);

Returned value: None.

sendMavTakeOff Sends the command to make the aircraft take off to the specified local position (in meters).

6.4.16. sendMavTakeOffLocal()（Order the aircraft to fly to the desired local location）

Function prototype: sendMavTakeOffLocal(self,xM=0,yM=0,zM=0,YawRad=0,PitchRad=0,AscendRate=2)

Parameters: xM, yM, zM represent the local coordinates of the target location; YawRa

d represents the target course Angle (radian);

PitchRad represents the pitch Angle of the target (radian); AscendRate: Rate of ascent (unit: m/s)

Returned value: None.

sendMavTakeOffLocal sends the command to take off to the specified local position (in meters) with an additional parameter passed in to the sendMavTakeOff function

6.4.17. sendMavTakeOffGPS()（Order the aircraft to fly to the desired global location）

Function prototype: sendMavTakeOffGPS(self,lat,lon,alt,yawDeg=0,pitchDeg=15)

Parameters: lat, lon, alt represent the global coordinates of the target location; yawDeg represents the target course Angle (degree);

pitchDeg represents target pitch Angle (degree);

Returned value: None.

sendMavTakeOffGPS sends a command to take off an aircraft to a specified global location (in degrees)

6.4.18. sendMavLand()（Land in position）

Function prototype: sendMavLand(self,xM,yM,zM)

Parameters: xM, yM, zM indicates the local coordinates of the target location

Returned value: None.

sendMavLand sends the command to land the aircraft at the specified local location (in meters).

6.4.19. sendMavLandGPS()（Land at the designated global location）

Function prototype: sendMavLandGPS(self,lat,lon,alt)

Parameters: lat, lon, alt represent the global coordinates of the target location.

Returned value: None.

sendMavLandGPS sends the command to land the aircraft to the specified global position (in degrees).

6.4.20. sendMavSetParam()（Send a command to PX4 to change the expected parameters）

Function prototype: sendMavSetParam(self,param_id, param_value, param_type)

Parameter: param_id Specifies the ID of the parameter to be changed. param_value Specifies the parameter value to be set.

param_type Specifies the type of the parameter.

Returned value: None.

sendMavSetParam sends the command to PX4 to change the ID, size, and type of the specified parameter values.

6.4.21. SendHILCtrlMsg() (PX4 send to ComSim)

Function prototype: SendHILCtrlMsg(self,ctrls)

Parameter: ctrls is a list of control signals to be sent.

Returned value: None.

SendHILCtrlMsg sends the hil_actuator_controls command to PX4 to control the attitude and motion of the aircraft. For a serial connection or in real flight mode, the function sends the command using the mav.hil_actuator_controls_send method of the the_connection object. In UDP connection mode, the function sends the command using the hil_actuator_controls_encode method of the mav0 object.

6.4.22. SendHILCtrlMsg1()（Send debugging instruction）

Function prototype: SendHILCtrlMsg1(self)

Parameter: None

Returned value: None.

SendHILCtrlMsg1 is used to send the debug_vect command to PX4 to send debugging vector information. If it is a serial port connection or in real flight mode, the function will send the command using the mav.debug_vect_send method of the the_connection object. For UDP connection mode, the function sends the command using the debug_vect_encode method of the mav0 object.

6.5. Ros-based MAVLink control interface (based on mavros, copy the visual group)

7. rflysim Standard Library Edition control interface (new version, support Redis)

7.1. ctrl

The ctrl module is an external control module that can obtain the position, speed and attitude information of the UAV through UDP_Full, UDP_Simple, MAVLINK_Full, MAVLINK_Simple, Redis_Full, Redis_Simple and other modes. The position, speed and heading of the UAV are controlled.

7.1.1. Code structure

The ctrl module code is placed in the Python38/Lib/site-packages/rflysim/ctrl directory. It contains __init__.py, api.py, offboard.py, and principal.py. __init__py and api.py mainly provide user interfaces, which are described in detail in the next section, "User apis."

7.1.2. offboard.py

offboard.py is the core source code of external control, which supports the acquisition of UAV position, speed and attitude information through UDP_Full, UDP_Simple, MAVLINK_Full, MAVLINK_Simple, Redis_Full, Redis_Simple and other modes. The position, speed and heading of the UAV are controlled. offboard.py is aimed at advanced developer users and supports many features, but the interface is relatively complex to invoke.

offboard.py type table

type	function
SimMode	Simulation mode, including HITL, SITL, SITL_RFLY, Simulink_DLL and other modes.
CtMode	It refers to the connection mode, including UDP/MAVLINK/Redis and other major modes. According to the size of data packets, they are d

	ivided into Simple mode and Full mode. To further reduce data overhead, there are two modes, MAVLink_NoSend and MAVLink_NoGPS.
Coordinate	Currently, only three coordinate systems are supported for sending control information, LOCAL_NED, GLOBAL_INT, and NED_BODY. The corresponding encoding is referred to the MAVLINK protocol.
PX4MainMode	Main modes supported by PX4, manual, fixed height, fixed point, etc.
PX4SubMode	PX4 supports sub-modes, take-off, mission, landing, follow, etc.
RedisKey	In Redis mode, aircraft are distinguished by key values.
PX4CmdLong	PX4 Long command, containing 7 data.
FIFO	It is used to read and write MAVLINK data.
RflySimCP	Integrated model control protocol.
OffboardSimpleMode	Simplified control protocol, in which one int type data identifies the pattern, and the remaining 4 float type data are actual data. Since float and int are both 4 bytes, float data can also be replaced with int data.
OffboardPosType	Indicate what information is in the Offboard message, fully compatible with the PX4 Offboard position message, which includes position, speed, acceleration, throttle, yaw Angle, yaw Angle rate, etc.
OffboardPosSend	Used to build data packets for sending.
VehicleStatus	It is used to represent the status of the aircraft, including ID, position, speed, attitude and other information.
EarthFrame	It provides the conversion between the geographic coordinate system (lla), the Earth-centered solid system (ecef), and the site-centered coordinate system (NED, ENU).

PID	PID controller, single channel.
Offboard	The most core class, described in detail below

The Offboard class is the core class of Ctrl and even the entire RflySim. Functionally speaking, it is mainly divided into two types of functions, receiving data and sending data. When encoding, functions that receive data start with receive, while functions that send data start with send.

As shown in the following figure, are the functions associated with receiving messages. The entry function for receiving the message is receive_msg_loop(), which is a large while loop that is started by a separate thread when the Offboard class is initialized. receive_msg_loop() is used to call different receiving functions and data parsing functions depending on the connection mode. Because redis mode and UDP mode are very similar, a lot of code is similar, redis and udp are encapsulated together. In a nutshell, receive_msg_loop() will call receive_udp_redis_msg() when the mode is UDP/Redis, or receive_mav_msg() otherwise. The data will then be parsed by calling the respective update functions, and the parsed data will be placed in vehicle_status. So an advanced developer user who wants to access the drone's status information would simply read the self.vehicle_status.

receive_udp_redis_msg() also makes a further distinction between Simple and Full mode for the control instructions in UDP/Redis mode. Although there are differences between Simple and Full modes in MAVLINK, the Simple and Full modes in MAVLINK are not reflected in the control instructions, so there is no need to distinguish between Simple and Full for external control. It is worth noting that when using the MAVLINK mode, the actual communication mode is UDP if the software is in the loop. However, unlike the UDP mode, the MAVLINK mode encodes

the packets. In UDP mode, the MAVLINK encoding process takes place in CopterSim.

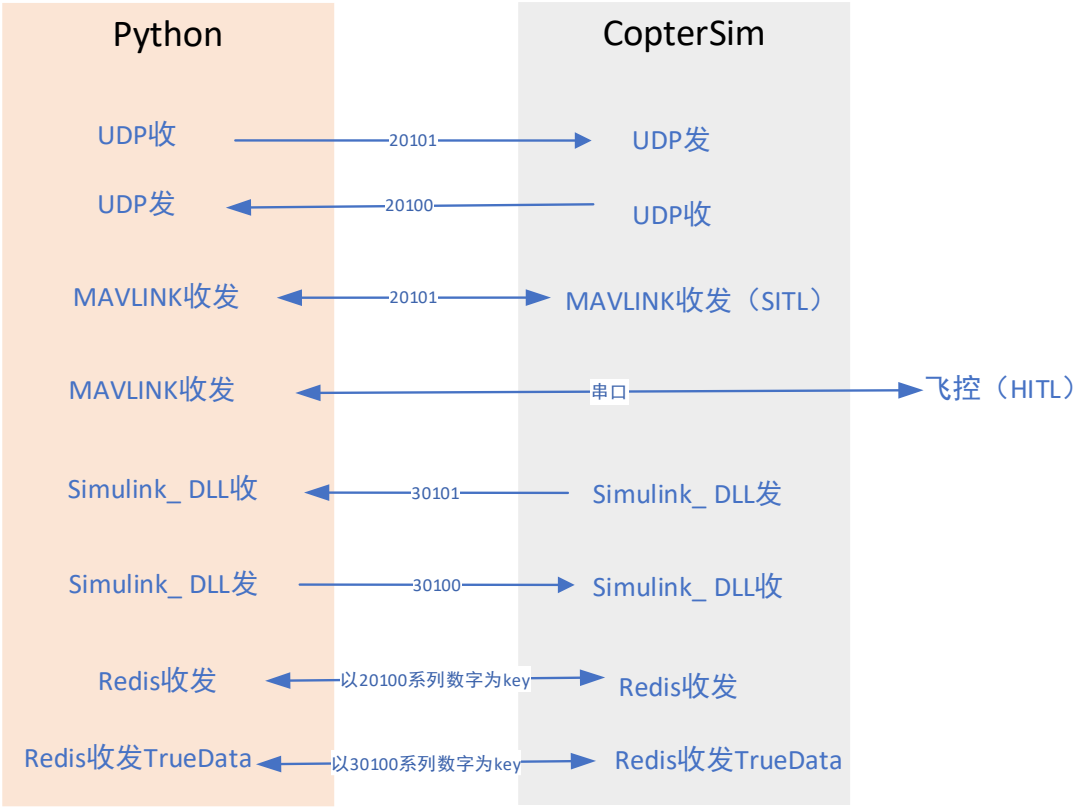
```
def receive_mav_msg(self):...

def update_vehicle_status_by_mav(self):...

def receive_full_udp_redis_msg(self):...

def receive_simple_udp_redis_msg(self):...

def receive_udp_redis_msg(self):...

def update_vehicle_status_by_udp_redis(self):...

def receive_msg_loop(self):...
```

It should be added that when Simulink_DLL mode is used for simulation, the data is sent through the 30100 series port, and the data format is different from that of UDP/Redis in other modes. Therefore, the Simulink_DLL schema is handled separately in receive_full_udp_redis_msg().

As shown in the following figure, it is the interface that sends the message. Similarly, the entry function to send the message is send_msg_loop(), which is also a large while loop that is started by a separate thread when the Offboard class is initialized. Since Simulink_DLL mode instructions are placed in inSIL, which is different from (HITL, SITL), send_msg_loop() determines the Simulink_DLL mode separately and calls send_simulink_dll(). Sending messages also distinguishes between MAVLINK/UDP_Redis by calling send_position_target_mav() and send_position_target_udp_redis() in send_msg(). In both UDP and Redis modes, CopterSim automatically unlocks and enters Offboard mode when it receives Offboard location information. With the exception of send_cmd_udp(), the other send functions support only MAVLINK mode.

```
def send_position_target_mav(self):...

def send_position_target_udp_redis(self):...

def send_cmd_udp(self):...

def send_param_mav(self, param_id, param_value, param_type):...

def send_cmd_mav(self, cmd_long):...

def send_flight_mode(self, main_mode, sub_mode=0):...

def send_msg(self):...

def send_simulink_dll(self):
    self.send_sil_int_float(self.in_sil_ints, self.in_sil_floats)

def send_msg_loop(self, time_interval_s=0.01):...

def send_redis_data(self, key, buf):...

def send_sil_int_float(self, in_sil_ints, in_sil_floats):...
```

As mentioned above, a separate thread is used to receive and send messages. That is, there are 3 threads to control an aircraft, namely the main thread, the sending thread, and the receiving thread. When there are n planes, the total number of threads is 3n. Initialized by init_connection_loop() in offboard.py, this function supports mode as a parameter, which means that the initialization of the process that sends and receives messages will vary depending on the connection mode UDP/MAVLINK/Redis(Full\Simple). For non-SIMULINK_DLL mode, the 20100 series port is used for communication. As shown in the figure below, for CopterSim, port 20100 is used to receive messages and port 20101 is used to send messages. When the number of aircraft is increased, the port value is gradually increased, such as adding a second aircraft, its port is 20102, 20103. For Simulink_DLL mode, CopterSim uses port 30100 to receive information and port 30101 to send information. Similarly, when there are multiple aircraft, the ports also increase from 30100. For Redis mode, the communication is carried out in the pub/sub mode, and there is a topic for each aircraft's sending and receiving. In the same way that UDP communication is numbered, the original port is used as the key, that is, the name of the topic, for Redis communication.



7.1.3. topics.py

topics.py encapsulates the MAVLINK message package with only the necessary pose and location information, as shown in the following table.

MAVLINK message encapsulated by pythonSDK

Message name	description
HeartBeat	Heartbeat packets, including the main mode, submode, and system status.
Attitude	Attitude, including time, roll, pitch, yaw and the corresponding angular rate.
AttitudeTarget	Desired attitude, including time, desired roll, pitch and corresponding angular rate
LocalPositionNED	Position, position and velocity in NED coordinates.
PositionTargetLocalNED	Desired position, desired position and velocity in the NED coordinate system.

HomePosition	home point location, latitude, longitude, height, etc.
GlobalPositionInt	The global position, latitude and longitude are represented as INT type, that is, the original basis is multiplied by 107 to convert to INT, which can avoid the loss of significant digits of floating point numbers.

7.2. User Api

7.2.1. ctrl

__init__.py is used to describe externally accessible classes in the ctrl module. Including offboard.py "EarthFrame" geodetic Coordinate system conversion class, "PID" controller class, "Ctrl" external control interface class, "CtMode" connection mode class, "SimMode" simulation mode class, "coordinate" coordinate class.

The Ctrl class is a wrapper around the Offboard class, providing a more user-friendly user interface. In implementation, Ctrl is a subclass of Offboard. The arguments of the Ctrl class are passed to its parent class Offboard through a super call. The following table shows how to build a Ctrl instance, using the default parameters of the interface by default. When a user sets a custom port, be careful of port conflicts.

Interface for creating an external control instance

Ctrl(port=20100, ip='127.0.0.1', redis_host='127.0.0.1', redis_port=6379, redis_pass=None, connect_to_hw=False,	The Ctrl constructor is used to create an external control instance and contains 7 parameters port - Base port, generally use the default value. If you change the port value here, on the one hand, ensure that CopterSim is changed simultaneously and on the other hand, avoid conflict with other ports. ip¬ - ip when communicating in UDP mode redis_host -- the ip address of the server side when communicating with Redis. redis_port: specifies the port number of Re

sim_mode= SimMode.SITL_RFLY)	dis. If the Redis mode is used and the default port number of Redis is changed, set this parameter. redis_pass -- password of Redis connect_to_hw - Indicates whether the hardware is connected. This is True when the hardware is in the loop and flying. sim_mode - includes HITL, SITL, SITL_RFLY, Simulink_DLL and other modes. Example: 1) Use default parameters for all parameters ctrl = rflysim.Ctrl() 2) Multi-machine integrated model simulation ctrl = rflysim.Ctrl(port=20100 + i * 2, sim_mode=SimMode.Simulink_DLL)

Initializes the data input/output interface init_loop(). This function must be called during normal use, and the specification of parameters is often required. For example, developers will choose UDP_Simple or Redis_Simple when doing cluster simulation, and MAVLINK_FULL mode when doing fault injection.

The interface initializes the sending and receiving data thread.

Procedure

init_loop(ct_mode=CtMode.MAVLink_Full, offboard=False)	Used to initialize the data transceiver interface, start the corresponding thread, call the Ctrl class control aircraft always need to call the interface. Parameters: ct_mode -- Indicates the connection mode offboard - Whether the offboard control is performed Example: ctrl.init_loop(self.ct_mode, offboard=True)

Take-off, return, and landing command interfaces that support onl

y integrated models. For the take-off, return and landing of the comprehensive model, only the following functions can be called, and the following functions all support specifying the height in the NED coordinate system.

Take-off, return, landing command interface

takeoff(height=0)	Calling this function allows the synthesis model to take off and supports specifying the height, which is the NED coordinate. The rotorcraft will hover after takeoff, and the fixed-wing aircraft will continue to fly forward at roughly the same altitude after takeoff.
return_home(height=0)	Calling this function can control the return of the integrated model, that is, the horizontal position returns to the home point, and supports setting the return height (NED). When the rotor reaches the corresponding point, it will hover, and the fixed wing will hover.
land(height=0)	Calling this function can control the integrated model landing and support setting the landing height (NED). Fixed-wing landing generally has a runway in the real scene, and the ground support force is also a complex process, at present, it can only land to the corresponding height, because the ground information is unknown, there may be a situation of drilling into the ground.

Desired location send interface. Send position in NED coordinate system and global coordinate system. When the user is programming, the pos passed in by calling send_pos_global() is floating-point, and the daemon multiplies the value by 107 and converts it to an integer and sends it to CopterSim or directly through the serial port.

Location sending interface

send_pos_ned(pos, yaw)	Send the desired position and yaw Angle in the NED coordinate system
send_pos_global(pos, yaw)	Send the position in the Global coordinate system. The interface supports floating point number Settings, but automatically converts to INT when sending

Expected speed sending interface. The expected speed and yaw Angle rate can be set in NED coordinate system and carrier coordinate system.

send_vel_ned(vel, yaw_rate)	Send the desired velocity and yaw Angle rate in NED coordinates
send_vel_body(vel,yaw_rate)	Expected velocity and yaw Angle rate in the sending carrier coordinate system

Attitude sending interface. The interface currently only supports fixed-wing integrated models.

send_att(att, thrust)	Send Euler Angle and throttle

Get status information on the drone. The interface to obtain UAV status information starts with get, mainly to obtain information such

as position and attitude. After initializing the messaging interface, the following status is updated. It is worth noting that when communicating in Simple mode, the position in the Global coordinate system is not available.

Get drone status information interface

get_pos_ned()	Gets the position in the NED coordinate system
get_pos_global()	Gets the position in the Global coordinate system
get_home_pos()	Get the home point location, Global coordinates
get_vel_ned()	Get the NED coordinate velocity
get_euler()	Obtain the Euler Angle
get_vehicle_id()	Get vehicle ID
get_time_s()	Get simulation time or flight control run time

The SimMode class. Users can directly access members of the SimMode class to control the emulation mode. The specific simulation mode is shown in the following table. Users need to set the correct simulation mode during simulation.

Interface of the SimMode class

HITL	hardware-in-loop
SITL	Software in loop
SITL_RFLY	The software in the ring is optimized for port Settings and supports the operation of more than 100 aircraft
Simulink_DLL	Comprehensive model simulation Example: rflysim.Ctrl(port=20100 + i * 2, sim_mode=

	SimMode.Simulink_DLL)

The CtMode class. Users can access members of the CtMode class directly to configure the mode of the connection. The supported modes include UDP_Full, UDP_Simple, MAVLink_Full, MAVLink_Simple, Redis_Full, and Redis_Simple. MAVLink_NoSend and MAVLink_NoGPS are not fully supported.

If the user performs a stand-alone experiment and needs to obtain low-level data, the MAVLINK_Full mode is recommended. For cluster experiments, you can select UDP_Simple/Redis_Simple. Redis supports only the enterprise customized version. Redis is more stable, reliable, and efficient than UDP. Users who are familiar with the basic routines of the platform can generally choose UDP_Full. Example: The following self.ct_mode can be any mode in a table. self.ctrl.init_loop(CtMode.UDP_Full, offboard=True)

CtMode interface

UDP_Full	UDP mode, complete control instructions
UDP_Simple	UDP mode, simplified control instructions
MAVLink_Full	MAVLink mode: complete MAVlink protocol
MAVLink_Simple	The MAVLink mode simplifies the MAVlink protocol
MAVLink_NoSend	CopterSim does not forward all MAVLINK messages
MAVLink_NoGPS	CopterSim does not forward GPS messages
Redis_Full	Redis mode: sends and receives data through Redis, which has the same data format as UDP_Full
Redis_Simple	Redis mode: sends and receives data through Redis, and has the same data format as UDP_Simple

Coordinate class. Supports local NED, Global position, velocity and acceleration in carrier coordinate system, and the specific protocol is consistent with MAVLINK.

Coordinate class interface

LOCAL_NED	Coordinate class interface NED coordinate system
GLOBAL_INT	Longitude and latitude multiplied by 10^7 is converted to INT transmission
NED_BODY	Position in NED coordinates, velocity and acceleration in BODY coordinates

The EarthFrame class. EarthFrame provides the conversion between the geographic coordinate system, the geostationary system, and the stationocentric coordinate system. The geographical coordinate system (Latitude, Longitude, Altitude, lla), also known as the global coordinate system: latitude and longitude, latitude and longitude unit is °, height unit is m. Earth-centered, Earth-fixed (ecef) : The origin (0,0,0) is the center of the Earth's mass, the Z-axis points parallel to the Earth's axis to the North Pole, the X-axis points to the intersection of the prime meridian and the equator, and the Y-axis is perpendicular to the xOz plane (i.e. the intersection of the equator at 90 degrees east longitude) to form a right-handed coordinate system. Station centered coordinate system: the coordinate system with the station as the origin. There are generally two types: East, North, Up, ENU and North, East, Down, NED. PX4 uses the NED coordinate system, and the starting point of the UAV is generally the station, that is, the origin. EarthFrame can realize the conversion between the above three coordinate systems. When converting with the station center coordinate system, it is necessary to know the global position of the station center, that is, the latitude and longitude height.

EarthFrame interface

lla2ecef(pos_global)	The latitude is converted to x, y, and z with the Earth's center of mass as the origin
enu2ecef(pos_global, pos_global_home)	The northeast sky coordinates are converted to x, y, and z with the Earth's center

	of mass as the origin
ecef2enu(pos_global, pos_global_home)	x, y, and z with the Earth's center of mass as the origin are converted to the northeast day
lla2enu(pos_global, pos_global_home)	Latitude and longitude are converted to northeast sky coordinates
lla2ned(pos_global, pos_global_home)	Latitude and longitude are converted to northeastward coordinates
ecef2lla(pos_global)	x, y, convert to longitude and latitude with the Earth's center of mass as the origin
enu2lla(pos_enu, pos_global_home)	The northeast day is convert to longitude and latitude
ned2lla(pos_ned, pos_global_home)	Northeastward convert to longitude and latitude

PID class. The PID class provides a simple controller that can be used to calculate the desired speed from the desired position when Offboard control is done.

PID(p=0.0, i=0.0, d=0.0)	PID constructor, which can specify PID parameters. By default, all parameters are 0
pid(err)	The input parameter is the status error and the output PID result
reset()	Replacement integral

OffboardSimpleMode class. This class is unique to the RflySim platform, and the entire control command takes only 20 bytes of space. This mode is only applicable to advanced developer users, and users who do upper-level control do not need to pay attention to this class.

OffboardSimpleMode class interface

Vel_Yaw_Rate_NED	Velocity mode in navigation coordinate system [vx,vy,vz, yaw_rate]
Vel_Yaw_Rate_BODY	Velocity mode in body coordinate system [vx,vy,vz, yaw_rate]
Pos_Yaw_NED	Position mode in navigation coordinate system [x,y,z,

	yaw]
Pos_Yaw_BODY	Position mode in body coordinate system [x,y,z, yaw]
Att_Throttle	Attitude throttle control command [roll, pitch, yaw (radian), throttle (0~1)]- can be automatically unlocked, can automatically enter OffBoard mode
Att_Throttle_Add	Attitude throttle increment control command [roll, pitch, yaw, throttle increment]- can be automatically unlocked, can automatically enter OffBoard mode
Acc	Acceleration control mode [ax,ay,az,--]
Acc_Yaw	Acceleration control mode [ax,ay,az,yaw]
Acc_Yaw_Rate	Acceleration control mode [ax,ay,az,yaw_rate]
Arm	Unlock the mode shown [Unlock,-,-,-]
Speed_Radius_FW	Set the speed and hover radius of the fixed wing aircraft, [speed, radius, -, -]
Takeoff_NED	Indicates Mavlink take-off command, automatic unlock, navigation coordinate system position [x, y, z, -]
Takeoff_Global	Indicates Mavlink take-off command, automatic unlock, GPS coordinate system position [latitude, longitude, altitude, -]
Speed_Height_Dir_Global	Speed altitude heading command, automatically unlock and enter offBoard mode, GPS coordinate position [speed, altitude, heading, -]
Pos_Yaw_Global_Int	Position mode in Global coordinates, GPS coordinates position [lat_int, lon_int, alt_float, yaw_float]
VTOL_Switch	It is used to switch the VTOL mode

The RflySimCP class. This class is a control protocol for the comprehensive model and is only suitable for advanced developer users. This class is a code representation of the protocol in Section 3.2.2. Not all of the flags in the following table are currently used, and some of the flags are reserved for later more complex functions.

RflySimCP class interface

ILen	Length of inSILInts
ICmd	Corresponding to inSILInts[0], indicates the subscript corresponding to the instruction
IOffboard	Corresponding to inSILInts[1], indicates that Offboard mode corresponds to subscript
ILat	Corresponding to inSILInts[6], latitude is stored when global coordinates are supported
ILon	Corresponding to inSILInts[7], longitude is stored when global coordinates are supported
CmdEn	inSILInts[0] Indicates that the 0 bit is enabled, indicat

	ing that the command is reserved
CmdSIL	inSILInts[0] Indicates that the first bit is enabled, indicating that the simulation mode is entered
CmdArmed	inSILInts[0] The second bit is enabled, indicating that the device is unlocked
CmdTakeoff	inSILInts[0] The eighth bit is enabled, indicating take-off
CmdPosition	inSILInts[0] Enable the ninth bit, which indicates a fixed point
CmdLand	inSILInts[0] The 10th enabled position indicates landing
CmdReturn	inSILInts[0] Enables the 11th position, indicating return
CmdOffboard_Pos	inSILInts[0] The 16th bit is enabled, indicating that the Offboard position is reserved
CmdOffboard_Att	inSILInts[0] Enables the 17th position, which indicates that the Offboard posture is reserved
CmdBase	CmdEn+CmdSIL: This header is required when sending commands
HasPos	inSILInts[1] The 0th bit is enabled, indicating a position
HasVel	inSILInts[1] The first bit is enabled, indicating speed
HasAcc	inSILInts[1] The second position enables, representing acceleration
HasYaw	inSILInts[1] The third bit is enabled, indicating a yaw Angle
HasYawRate	inSILInts[1] The fourth bit is enabled, indicating a yaw Angle rate
HasAtt	inSILInts[1] The eighth bit is enabled, indicating posture
HasRollRate	inSILInts[1] The 9th bit is enabled, indicating that there is a roll Angle rate
HasPitchRate	inSILInts[1] The 10th enable indicates that the pitch Angle rate is present
HasThrust	inSILInts[1] The 11th enabled position indicates a throttle
NED	inSILInts[1] The 16th bit is enabled, which means reserved in the NED coordinate system
Global	inSILInts[1] The 17th bit enables, which means reserved in the Global coordinate system
FLen	inSILFloats length
FPos	Position start subscript
FVel	Velocity start subscript

FAcc	Acceleration starting coordinates
FAtt	Attitude starting coordinate
FAttRate	Angular rate starting coordinates
FThrust	Throttle coordinate

7.2.2. test

The test folder is used to store use cases. These use cases are used both to babysit the functionality of the pythonSDK and to teach users how to use the platform api for secondary development.

7.3. test_ctrl.py

This file mainly tests the interface of external control. The key logic and usage of each test case are described below.

TestTakeoff class. This class is the simplest routine to use rflysim pythonSDK to send the position in the NED coordinate system for takeoff. The constructor of this class supports the configuration of connection modes, including UDP/MAVlink/Redis (Full+Simple) for a total of 6 modes. The TestTakeoff constructor defines a Ctrl() instance self.ctrl. The run() function initializes the data sending and receiving thread with a call to self.ctrl.init_loop(). self.ctrl.send_pos_ned() sends the desired position for takeoff. Wait for takeoff altitude, program exit.

TestTakeoffVel class. Speed control allows the drone to take off and reach the specified altitude. In order to more accurately reach the specified position through speed control, a PID controller is used in the use case. The input of the PID controller is the altitude error, and the output is the desired speed in the altitude direction. In this use case, the desired speed is generated by the loop function and sent by self.ctrl.send_vel_ned().

The TestTracking class. This class uses position control to implement "figure 8" tracking, and the routine shows how to implement more

complex trajectory control with self.ctrl.send_pos_ned(). Its core function is tracking_8_shape(), which represents the position of the drone on the "figure 8" by Angle, equivalent to control in the polar coordinate system. When the drone reaches a position, the corresponding polar Angle increases by an interval. However, the control interface only supports Cartesian coordinates, so the polar coordinates are converted to Cartesian coordinates and sent to the aircraft.

The TestGetPos class. Gets the current position and home point in different coordinate systems. This type also supports six modes, but in simple mode, there is no global location. Therefore, the global location in simple mode is 0.

The TestPosFrame class. Compare position control in NED coordinate system and Global coordinate system. In large-scale simulation, it is often necessary to set the target point using the Global location. TestPosFrame is a routine that combines NED and Global coordinates. The run() function first takes off in the NED coordinate system and waits to reach the specified altitude. After reaching the specified height, horizontal position x+10, height +5. The EarthFrame instance converts a position in the NED coordinate system to a Global position. Then, send_pos_global() is called to send it to the plane. Finally, the position in NED coordinate system is used to determine whether the specified position is reached. If it can be reached, it not only proves that send_pos_global() is successfully sent, but also proves that the conversion of EarthFrame is correct.

The TestVelFrame class. Compare the difference between the speed instruction in NED coordinate system and the speed instruction in body coordinate system. This class can be viewed as an extension of TestTakeoffVel. The key modification is that the program selects send_vel_ned or send_vel_body depending on the coordinate system specified. W

hen no yaw motion is performed, carrier coordinates and NED coordinate system coincide, so it is necessary to do a yaw motion to see the different phenomena caused by sending the same data in two coordinate systems. Specifically, when a speed is specified in the x direction of NED, it will travel north, and when x direction is specified in the body coordinate system, the speed will advance beyond the nose direction.

TestSynModel class. This class is a test case for a system of integrated models that supports specifying the number of aircraft and modes. Each aircraft is controlled using a separate thread, and each thread needs to pass in the aircraft instance and the initial position of the aircraft. The initial position of the incoming aircraft and the initial position of the CopterSim need to be consistent to avoid horizontal movement during takeoff. The initial position is calculated through init_poses() in the routine. loop() is the action performed by each aircraft in order to cover as many interfaces of the integrated model as possible.

TestFWSynModel class. This is the class that tests the fixed wing integrated model, which supports the specified number of aircraft, mode, and whether attitude control is enabled. When att=True, the test case will use attitude mode to control take-off, level flight, turn, hover, etc. Otherwise, the takeoff(), send_pos_ned(), return_home(), land() and other takeoff() functions will be used to control the trajectory.

8. QGC secondary development

This paper mainly introduces the secondary development of QGroundContorl based on Windows environment, using Qt Creator 5.15.2 + Visual Studio 2019 tools, and realizing some customization functions. Including QGC secondary development environment preparation and construction, module introduction, new function case introduction.

8.1. Development environment preparation and setup

Before fetching QGC source code, you need to configure GitHub to ensure that local code can be pulled from GitHub.

Reference QGC website HTTPS: / / github.com/mavlink/qgroundcontrol fetching code, including the module, through gitcode grab is completed. You need to refer to the.gitmodules file to confirm whether the submodule is successfully captured, for example, check whether the qgroundcontrol directory exists./ src/GPS/Drivers/src source code. If it does not exist, grab the relevant source code separately and put it in the corresponding directory, and use the same method for other subdirectories. If you do not grab the relevant subdirectory source code, the compilation will fail. In addition, please note that the submodules corresponding to the Master branch and the Tag branch are not shared. If different branch codes are used, the corresponding submodules need to be captured.

Use instructions can not capture the source Code (GitHub configuration failure), you can use the Code->Local->Download ZIP download package, refer to the.gitmodules file to confirm whether the submodule capture successfully. It is recommended to grab the latest Tag source code, which is more stable than the Master version. After the source code and submodules are downloaded, download the IDE tool.

Reference QGC website HTTPS: / / github.com/mavlink/qgroundcontrol, install Visual Studio 2019 and Qt 5.15.2 respectively, pay attention to the need to install Visual Studio 2019 first, then install Qt. When installing Qt 5.15.2, compile tool option MVSC_2019 64-bit, after the compile tool is installed, and ensure that it can be used normally. Open the Qt Creator program and load the project by going to Qt Creator-> File -> Open File or Project -> select QgroundControl.pro in the qgroundcontrol directory. If the compilation fails, check that the Qt build kit is Qt5.15.2+MVSC_2019 64-bit, and if more than one Qt version is configured in the environment variable, put the Qt5.15.2 configuration before the other versions.

8.2. Module introduction

QGC is designed to provide a single code base that can run across multiple operating system platforms and multiple device sizes and styles.

The QGC user interface is implemented using Qt QML. QML provides hardware acceleration, which is a key feature for low-power devices such as tablets or phones. QML also provides features that make it easier to create a single user interface that can be adapted to different screen sizes and resolutions.

QGC UI is aiming for more of a tablet + touch style UI than a desktop mouse-based UI. This makes a single UI easier to create, as tablet-style UIs also tend to work fine on a desktop/laptop.

This chapter mainly explains how QGC works.

8.2.1 Communication process

The communication between QGC and devices is mainly carried out through MavLink protocols. For details about MavLink, see Section 5 of this chapter. QGC mainly processes MavLink messages through LinkManager, MAVLinkProtocol, and MissionManager related classes.

LinkManager always opens the UDP port to wait for heartbeat packets sent by the device, and creates a new SerialLink when it detects that the device (Pixhawk, SiK Radio, PX4 Flow) has established a UDP connection with the computer. The incoming bytes from Link are then sent to the handlers of the MAVLinkProtocol class, which convert the bytes into MAVLink messages. After the undo analysis, if the message is HEARTBEAT, the MultiVehicleManager class will be notified and create a new vehicle object based on the information in the HEARTBEAT message. The vehicle object instantiates the plug-in that matches the vehicle. The ParameterLoader PARAM_REQUEST_LIST associated with the vehicle object sends a message to the connected device to load the parameters using the parameter protocol. Once the parameters are loaded, the MissionManager associated with the vehicle object requests the task item from the connected device using the task protocol, and the VehicleComponents will display their UI in the Settings view after the parameters are loaded

8.2.2 Plug-in architecture

This is despite the fact that the MAVLink specification defines a standard communication protocol for communicating with vehicles. Many aspects of the specification need to be interpreted by firmware developers. Therefore, in many cases, communication with a vehicle running one firmware is slightly different from communication with a vehicle running a different firmware in order to accomplish the same task. In addition, each firmware can implement a different subset of the MAVLink command set.

Another major issue is that the MAVLink specification does not cover vehicle configurations or common parameter sets. As a result, all of the code associated with vehicle Settings is ultimately firmware-specific. In addition, any code that must refer to specific parameters is also firmware specific.

Given all these differences between firmware implementations, it can be tricky to crea

te a single ground station application that can support every ground station application wit hout degenerating the code base into a lot of if/then/else statements based on the firmware used by the vehicle.

QGC uses a plug-in architecture to isolate firmware-specific code from code common to all firmware-allowing for further customization beyond what standard QGC can provide. At present, two classes, FirmwarePlugin and FirmwarePlugin, are mainly used to manage the expanded plug-in functions.

The FirmwarePlugin class is commonly used to create standard interfaces for parts of the Mavlink that are not standardized

The AutoPilotPlugin class is used to extend the user interface that provides vehicle S ettings

QGCCorePlugin is used to expose vehicle-independent functionality of the QGC applic ation itself through a standard interface. Custom builds then use it to adapt the QGC feat ure set to meet their needs.

8.2.3 Important class introduction

The LinkManager, LinkInterface class is a pipeline that handles specific types of com munication with devices. Such as serial port or UDP over WiFi. The base class for all lin ks is LinkInterface. Each link runs on its own thread and sends bytes to MAVLinkProtoco l. The LinkManager object keeps track of all open links in the system. LinkManager also manages automatic connections through serial and UDP links.

The MAVLinkProtocol class takes the incoming bytes from the link and converts the m into a MAVLink message. The MAVLink HEARTBEAT message is routed to the Multi VehicleManager. All MAVLink messages are routed to the vehicle associated with that lin k.

The MultiVehicleManager class creates only one object within the system. When it re ceives HEARTBEAT on a link it hasn't seen before, it creates a Vehicle object. The Multi VehicleManager also keeps track of all vehicles in the system and handles the switch fro m one active vehicle to another and correctly handles the vehicles that are removed.

The Vehicle class is the main interface that implements communication with the physi cal device. Note: there is also a UAS object associated with each Vehicle, which is a dep recated class and is slowly being phased out, with all functionality transferred to the Vehi cle class. No new code should be added here.

The FirmwarePlugin class is the base class for firmware plug-ins. Firmware plug-ins contain firmware specific code, so vehicle objects are clean in terms of supporting a singl e standard interface to the UI.

FirmwarePluginManager is a factory class that creates FirmwarePlugin instances based on the vehicle's MAV_AUTOPILOT/MAV_TYPE combination.

8.2.4 Main components of the user interface

The main pattern of UI design in QGC is UI pages written in QML, which often co mmunicate with custom "controllers" written in C++. This follows some modified variation of the MVC design pattern.

QML code is bound to system-related information through several mechanisms: (1) cu stom controllers, (2) global objects that use QGroundControl to provide access to things li ke moving vehicles, and (3) FactSystem to provide access to parameters and, in some cas es, custom facts. Note: Due to the complexity of QML used in QGC and its reliance on communication with C++ objects to drive the ui, it is not possible to edit QML using Q ML Designer provided by Qt.

QGC does not have different coded UIs for different screen sizes and/or form factors. In general, it uses the QML layout feature to rearrange a set of QML UI code to fit dif ferent form factors. In some cases, it will provide less detail on a small screen size to m ake it fit. But this is a simple pattern of visibility. The FactSystem system is used to ma nage all the individual data within the system. The data model is then connected to the c ontrol. The QGC UI is developed from a basic set of reusable controls and UI elements. This way, any new functionality added to the reusable control is now available throughout the UI. These reusable controls also connect to FactSystem Facts, which then automaticall y provide the appropriate UI.

QGC has a standard set of fonts and color palettes that are used by referring to the following two modules.

import QGroundControl.Palette 1.0

The import QGroundControl. ScreenTools 1.0

QGC software themes are designed in two styles: light and dark. Light palettes are s uitable for outdoor use and dark palettes are suitable for indoor use. In general, you shoul d not specify a color directly for the UI, but should always use one of the colors in the palette. If you do not follow this rule, the user interface you create will not be able to c hange from light/dark styles.

The QGCMapPalette is used to draw colors on the map. Because of the different ma p styles, especially satellite maps and street maps, you need to use different colors to dra w them clearly. Satellite maps need lighter colors to be seen, while street maps need dark er colors to be seen. The QGCMapPalette project provides a set of colors for this purpose

as well as the ability to switch between light and dark colors on the map.

The ScreenTools item provides a value that you can use to specify the font size. It a lso provides information about screen size and whether QGC is running on mobile device s.

The following controls are QGC variants of the standard Qt QML controls. They pro vide the same functionality as the corresponding Qt controls, except that they are drawn u sing the QGC palette.

- QGCButton
- QGCCheckBox
- QGCColoredImage
- QGCComboBox
- QGCFlickable
- QGCLabel
- QGCMovableItem
- QGCRadioButton
- QGCSlider
- QGCTextField

The following defined controls are unique to QGC and are used to create standard U I elements.

- DropButton - The RoundButton that pops up in the Options panel when clicked. An example is the Sync button in a floor plan.
- ExclusiveGroupItem - Used as the base item for custom controls that support the QML ExclusiveGroup concept.
- QGCView - Basic controls for all top-level views in your system. Provides supp ort for FactPanels and displays QGCViewDialogs and QGCViewMessages.
- QGCViewDialog - The dialog box that pops up from the right side of QGCView. You can specify the accept/reject button for the dialog box and the dialog conte nt. An example use is when you click on a parameter, it pops up the Value Edi tor dialog box.
- QGCViewMessage - A simplified version of QGCViewDialog that allows you to specify buttons and simple text messages.
- QGCViewPanel - The main view content in QGCView.
- RoundButton - A round button control that uses an image as its internal content.
- SetupPage - All the basic controls for setting up the vehicle components page. P rovides title, description, and component page content areas

8.2.5 Factual system

The Fact System provides a set of features that standardize and simplify the creation of the QGC user interface. It is mainly implemented by the following classes:

The Fact class implements a single value within the system.

The FactMetaData class implements each fact to be associated. It provides detailed in formation about facts to drive automatic user interface generation and verification

A Fact control is a QML user interface control that connects to a fact and provides FactMetaData users with a control to modify/display values associated with the fact.

The FactGroup class implements a set of facts. It is used to organize facts and mana ge user-defined facts.

You can add user-defined facts by overriding the factGroups function in the custom fi rmware plug-in class. FirmwarePlugin These functions return the name of the fact group map, which identifies the added fact group. You can add a custom fact group, FactGroup, by extending the class. FactGroup can define FactMetaDatas by providing a json file with the necessary information, using the appropriate constructor.

adjustMetaData can also change the metadata FirmwarePlugin for existing facts throug h the rewriting class.

You can use or factGroups to access vehicle-related facts (including facts that belong to the group of facts returned by the vehicle firmware plugin feature)getFact(" factName") getFact("factGroupName.factName"), for more information, See the comments in FirmwareP lugin.h.

8.2.6 Primary view

AppSettings.qml draws the Appliction Settings page, and each button loads a separate QML page

The Vehicle Setup page is drawn in SetupViey.qml, fixed button/page set: summary, fi rmware, and the rest of the buttons/pages are from the AutoPilotPlugin VehicleComponent list.

The main visual UI in PlanView.qml is the FlightMap control QML communicates wi th the MissionController (C++), which provides the task item data and methods for the vi ew. Commands for dynamically editing a specific task item from the json metadata hierarc hy. This hierarchy is called the task command tree. This way, you only need to create jso n metadata when you add new commands.

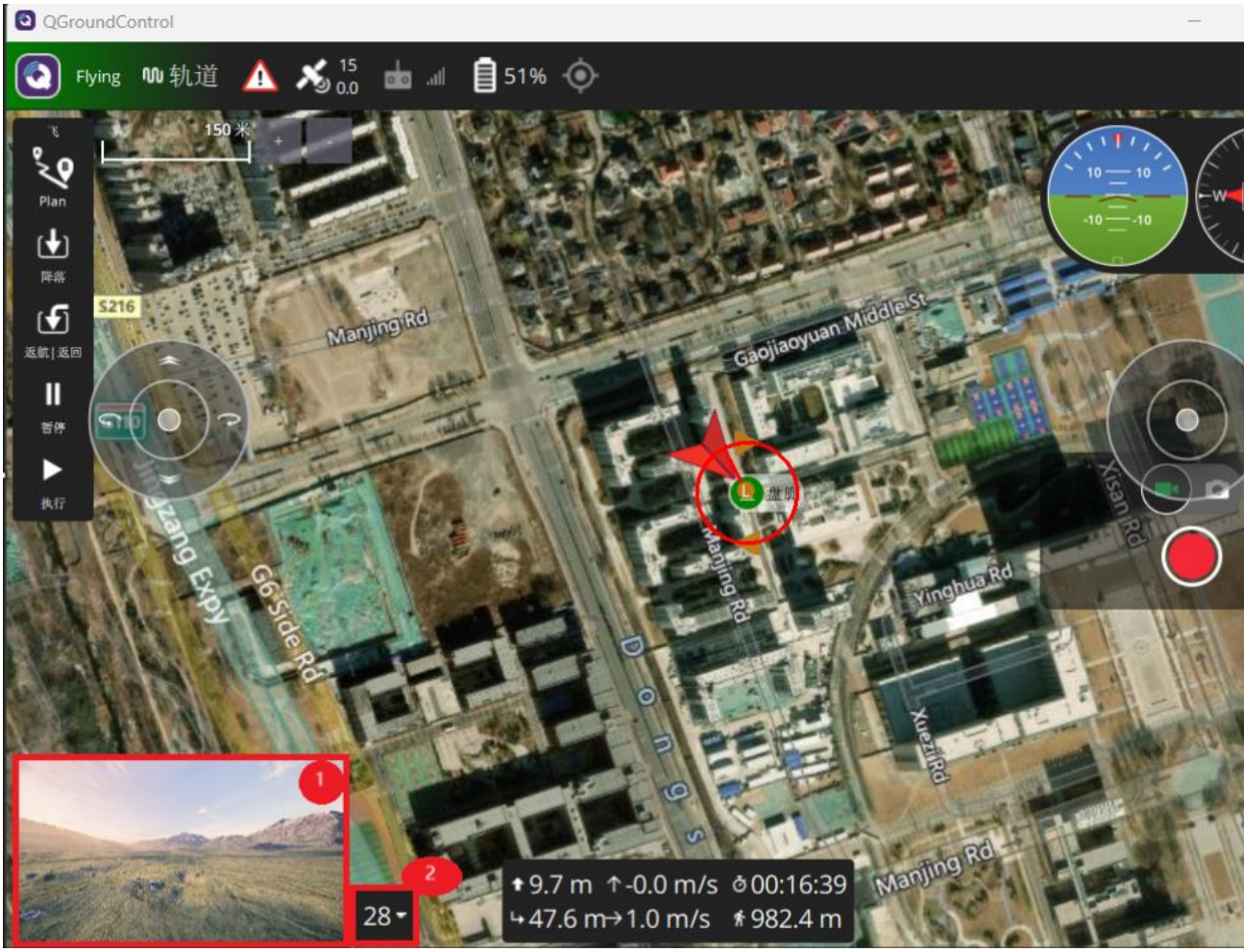
In FlightDisplayView.qml, the QML code communicates with the (C++) MissionContro

ller for task display, the instrument widget communicates with the moving vehicle object, and the two main internal views are: FlightDisplayViewMap, FlightDisplayViewVideo

8.3. New feature case

Based on the RflySim platform, add the ability to read images from shared memory for display in QGC, as well as create and set up RflySim3D cameras.

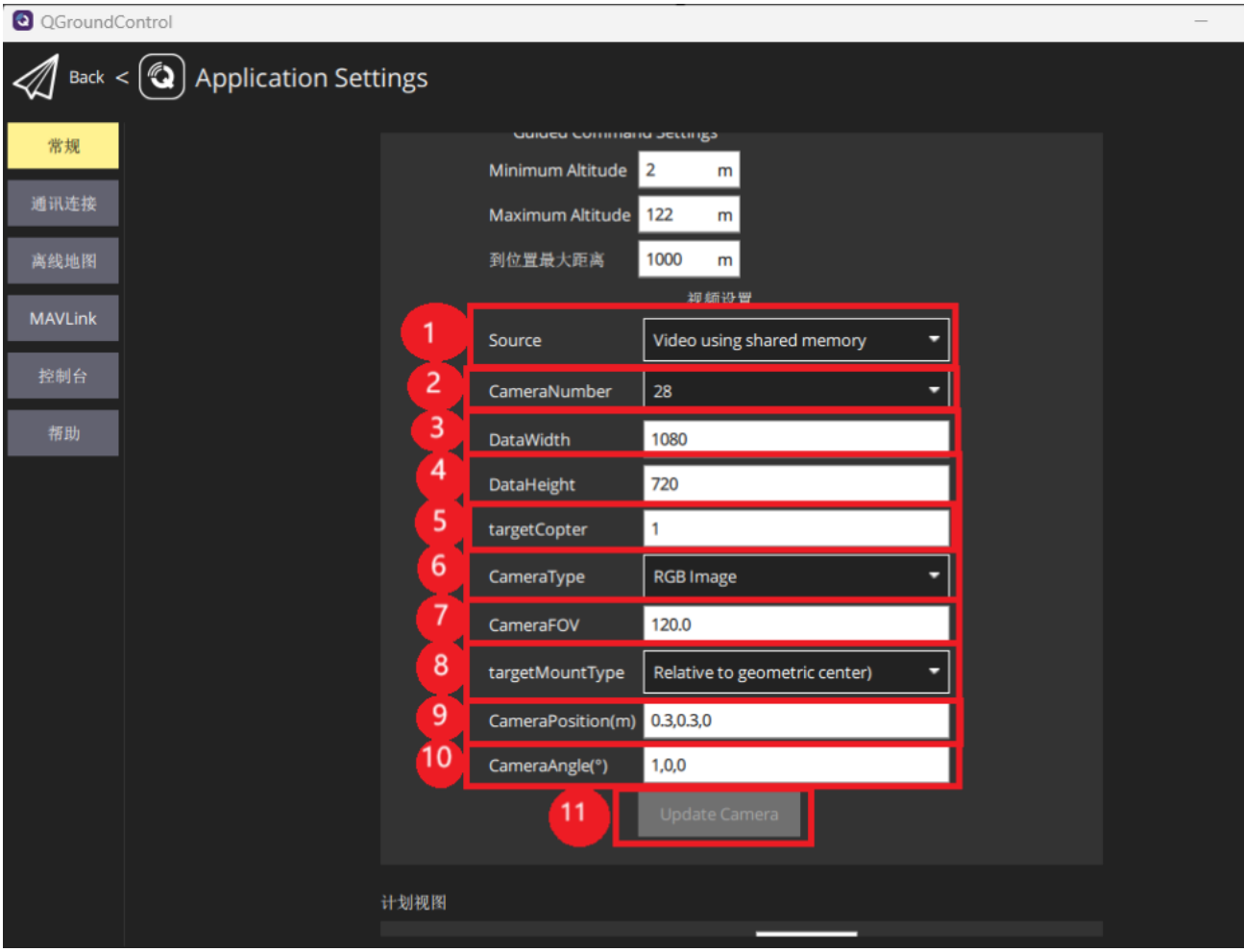
Added UI interface after reading image display function from shared memory:



① Shared memory image display window, you can scale and switch the window like QGC display camera data. Image shows the UI changes visible FlyViewVideo. Add the QML FlightDisplayViewSharedMemory. QML files, by adding relevant QML file Object Name, Ensure that the specified QML object is available in QGCCorePlugin.cc, and then draw the shared memory data onto the QML.

RflySim3D camera ID switch, select the corresponding camera ID through the drop-down box, the loaded camera configuration will be different. The FactComboBox component added in FlyView.qml can be changed by changing the cameraID. Based on the fact system of QGC, add the cameraID function to create the object and bind the _cameraIDChanged parameter function in the VideoManager class. The change of UI value is sent to Rfl

ySim3D through UDP to realize the switch of camera ID.



①Switch the video display source. Select the shared memory data for the video displ ay from the drop-down list box. The data obtained from the shared memory will be displ ayed on the video display page. For details about how to add an interface, see videoSourc e modifications in GeneralSettings.qml.

②RflySim3D Camera ID Switch, select the corresponding camera ID from the drop-dow n box. The component with id cameraNumberLabel is added to GeneralSettings.qml. Based on QGC's de facto system, the cameraID function is added to create an object and _cam eraIDChanged in the VideoManager class is bound. The change of UI value is sent to Rfl ySim3D through UDP to realize the switch of camera ID.

③、④RflySim3D camera pixel switch, modify the relevant value, you can set to get the image of the large camera ID switch change visible GeneralSettings.qml add the id of pixel* component, based on QGC fact system, add dataWidth function to create an object, Bind the _dataWidthChanged function in the VideoManager class, and send the change of UI value to RflySim3D through UDP to realize the switch of camera parameters.

⑤The plane ID number attached to the camera.

⑥Camera image type Settings, the image can be set to RGB image, depth image, gray image and other formats.

⑦ Camera Angle of view setting

⑧ Set the camera binding type

⑨ Camera binding position Settings

⑩ Camera mounting Angle setting

⑪Updates camera parameter Settings Add the UI related to the above Settings in the GeneralSettings.qml file, create Fact objects in VideoSettings.cc based on QGC's fact system, Then bind slot functions in VideoManager, C++ related classes, and values that respond to UI modifications in QML, and save the modified values in the VisionSensorReq structure provided. Finally by clicking on the Update Camera button call VideoManager: : clickCreatBtn slot () function, will give RlySim3D VisionSensorReq structure through UDP broadcast program, realize the Update of the Camera