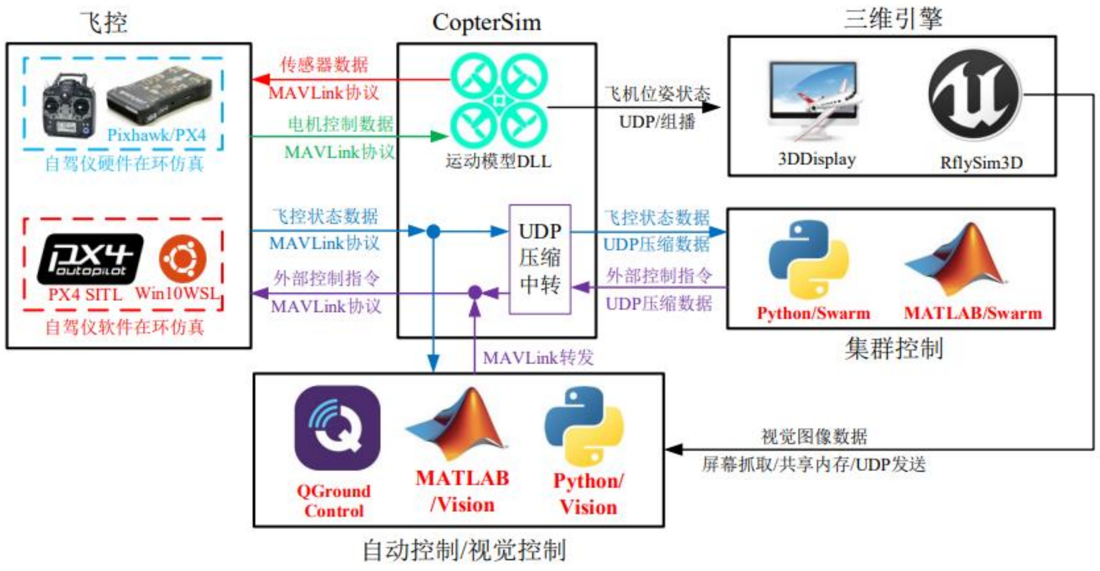
API Outline

1. Introduction and use of RflySim platform clus ter architecture

1.1 Introduction to cluster communication architecture



1. The RflySim communication architecture is shown in the figure above. The cluster control module mainly uses CopterSim for compression and transfer mechanism.
2. The amount of data used in MAVLink communication for real machines is too large and is not suitable for direct transmission in the communication network in large-scale clusters. Therefore, MAVLink data needs to be compressed and transferred.
3. Python or Simulink cluster controller receives flight control status data and sends Offboard instructions (top-level controls such as speed, position, acceleration, etc.) to control the aircraft to complete the specified flight mission.
4. Multiple CopterSim+ flight control combinations will simultaneously send data to the 3D engine, cluster control, and visual control modules in the LAN to achieve cluster communication simulation.

1.2 Cluster control communication optimization

As the number of aircraft increases, the network communication load becomes larger and larger. In order to achieve a larger number of UAV cluster simulations under limited bandwidth, communication needs to be optimized.

There are currently two main data protocols on the platform: 1) MAVLink data (contains a large amount of irrelevant data, and the data packets are very large, not suitable for clusters) and UDP compression structure (contains only required information, and the data packets are small, suitable for clusters).

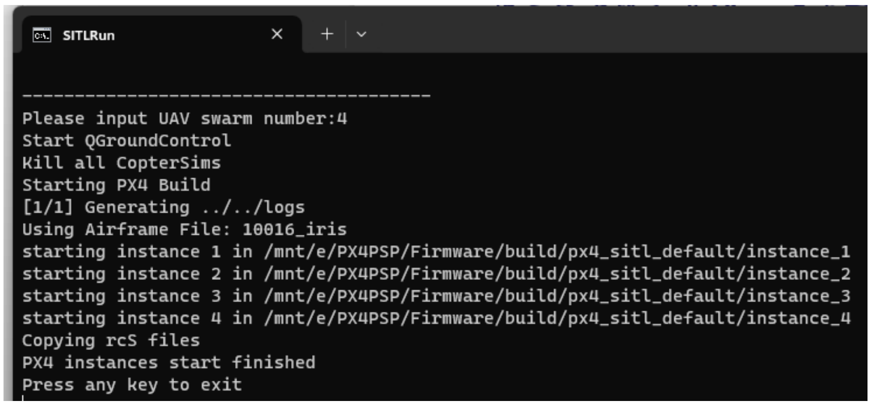
There are currently two ways to optimize communication on the platform: 1) Full mode, which contains as much data as possible and sends it as frequently as possible to ensure integrity (suitable for aircraft ≤ 8); Simple mode, which only contains necessary data and reduces the sending frequency to ensure Real-time and smooth communication under large-scale clusters (suitable for the number of aircraft > 4).

The combination of the two has four modes: UDP_Full, UDP_Simple, MAVLink_Full, and MAVLink_Simple; in addition, MAVLink communication is optimized for multi-computer vision hardware-in-the-loop simulation, and a MAVLink_NoSend protocol is also added.

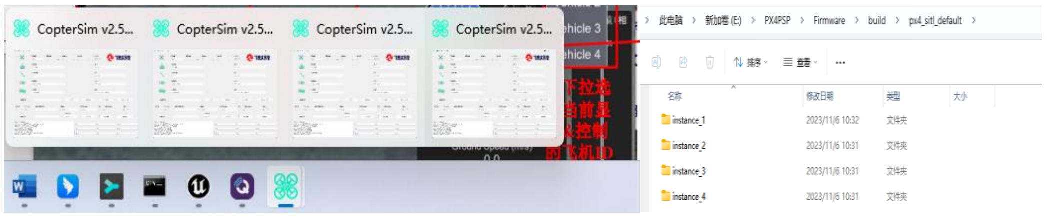
1.3 Cluster software-in-the-loop simulation

Double-click to run "RflySimAPIs\SITLRun.bat" (or the shortcut with the same name in the Rflytools folder), enter "4" in the pop-up command prompt, and click the Enter key to quickly start the cluster simulation system of four aircraft.

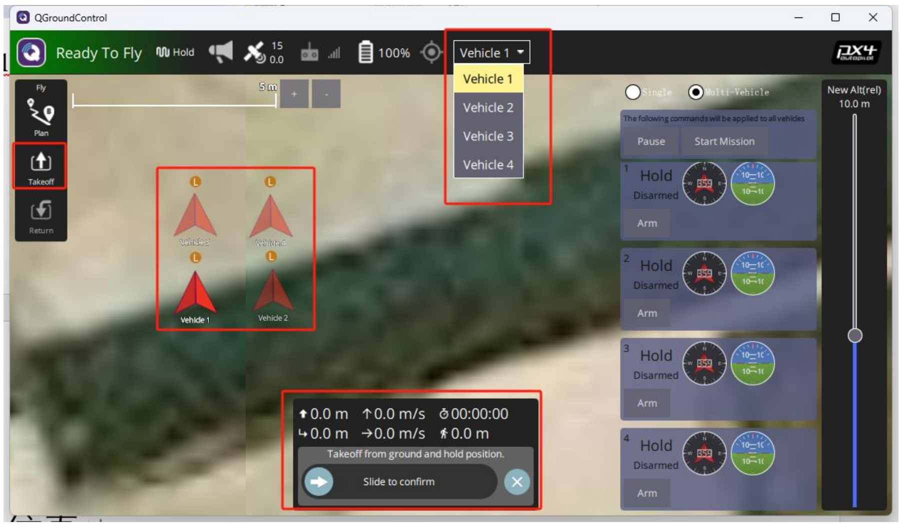
The automatically opened software includes 4 CopterSim (one for each aircraft), 1 RflySim3D (will display all aircraft at the same time), 1 QGC ground station (display and control all aircraft at the same time) and 1 command prompt window (calling Win10WSL The compiler opens four PX4 SITL controllers, and the logs and parameters of each controller can be accessed PX4PSP\Firmware uild\px4_sitl_default\instance_*)



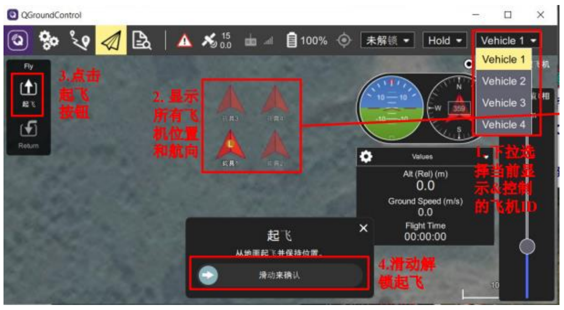
```
SITLRun

---------------------------------------
Please input UAV swarm number:4
Start QGroundControl
Kill all CopterSims
Starting PX4 Build
[1/1] Generating ../../logs
Using Airframe File: 10016_iris
starting instance 1 in /mnt/e/PX4PSP/Firmware/build/px4_sitl_default/instance_1
starting instance 2 in /mnt/e/PX4PSP/Firmware/build/px4_sitl_default/instance_2
starting instance 3 in /mnt/e/PX4PSP/Firmware/build/px4_sitl_default/instance_3
starting instance 4 in /mnt/e/PX4PSP/Firmware/build/px4_sitl_default/instance_4
Copying rcS files
PX4 instances start finished
Press any key to exit
```


CopterSim v2.5...
CopterSim v2.5...
CopterSim v2.5...
CopterSim v2.5...
此电脑 › 新加卷 (E:) › PX4PSP › Firmware › build › px4_sitl_default
名称
修改日期
类型
大小
instance_1 2023/11/6 10:32 文件夹
instance_2 2023/11/6 10:31 文件夹
instance_3 2023/11/6 10:31 文件夹
instance_4 2023/11/6 10:31 文件夹

As shown in the figure below, different aircraft can be switched in QGroundControl. Switch each aircraft (Vehicle 1~4) in sequence, click the "Take Off" button in turn, and then click "Slide to Confirm" to control 4 aircraft to take off in sequence.

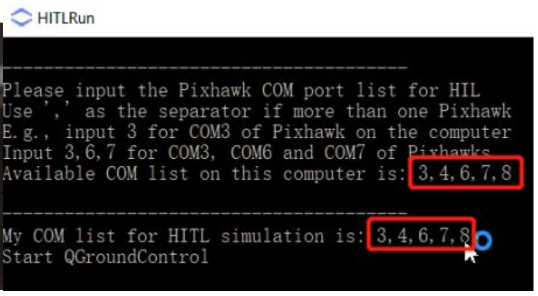


After taking off, press the "S" key in RflySim3D to display each aircraft ID, press the "T" key to display the trajectory, press the "D" key to display the aircraft information, press the "B" key to switch aircraft, and press the "V" key to switch the perspective , the effect is as follows.



1.4 Cluster hardware-in-the-loop simulation

As shown in the picture below, plug all Pixhawk flight controllers into the computer (please configure and restore the Pixhawk firmware in advance, and set it to enter HITL mode)
Double-click to run "RflySimAPIs\HITLRun.bat" (or the desktop shortcut with the same name), enter the flight control serial number (separated by commas) as prompted in the pop-up window shown on the right, and click the Enter key to quickly open multiple Cluster simulation system for aircraft.
The following two figures correspond to the situation of connecting 5 flight controllers for hardware-in-the-loop simulation. The running results are the same as SITL.



```
----------------------------------------
Please input the Pixhawk COM port list for HIL
Use ',' as the separator if more than one Pixhawk
E.g., input 3 for COM3 of Pixhawk on the computer
Input 3,6,7 for COM3, COM6 and COM7 of Pixhawks
Available COM list on this computer is: 3,4,6,7,8

----------------------------------------
My COM list for HITL simulation is: 3,4,6,7,8
Start QGroundControl
```

1.5 Cluster bat one-click script modification

The cluster's bat script is stored in the directory: PX4PSP\RflySimAPIs
Parameter description in Bat file:
START_INDEX: The starting aircraft serial number. If it is set to 1, and SITL enters the number of aircraft 4 (or HITL enters 4 serial numbers), then the created aircraft serial number is 1 to 4; if START_INDEX is set to 5, then it is 5 to 8. airplane.
TOTOAL_COPTER: Non-essential parameter, needed for multi-computer distributed network simulation, describing the number of all aircraft in the LAN (advanced full version only); used to automatically arrange the aircraft in a rectangular manner based on the given initial position and interval.
UE4_MAP: RflySim3D map name, please go to the directory PX4PSP \CopterSim xternal \map to select the map name.
ORIGIN_***: The initial position and yaw of the aircraft on the map
VEHICLE_INTERVAL: multi-machine arrangement interval (meters)
IS_BROADCAST: Whether to enable multi-computer online simulation, or specify a list of online computer IP addresses (advanced full version only)

UDPSIMMODE: Communication data sending and receiving mode

Modified example:
Requirement 1: The SITLRun script automatically sets the number of aircraft (for example, 4) without manual input. Just delete the code from ":Top" to ":StartSim" and add the line "SET VehicleNum=4". Please refer to RflySimAPIs\SimulinkSwarmAPI\RflyUdpFullFour.bat
Requirement 2: Two computers simulate 20 aircraft (full version only). Two bat scripts are required, IS_BROADCAST is set to 1, and the online function is enabled; the first bat script START_INDEX=1 and TOTOAL_COPTER=20, enter 10 when running the SITL script, and the second bat script START_INDEX=11 and TOTOAL_COPTER=20, enter 10 . Then select the appropriate map, starting location and aircraft interval.
Requirement 3: Change the map name. Simply modify UE4_MAP to require a map.
Requirement 4: Manually specify each aircraft position. Please refer to the HITLRunPos.bat and SITLRunPos.bat shortcuts in the RflySimAPIs directory. It supports manually specifying the x-coordinate sequence, y-coordinate sequence and yaw angle sequence of each aircraft. The number of aircraft is automatically determined based on the input sequence.

1.6 RflySim3D cluster shortcut usage

F1: Pop up the help menu prompt;
ESC: Clear all aircraft
S: Show/hide aircraft ID;
B: Switch the focus between different aircraft;
B+Number*: Switch to aircraft No.*
T: Turn on or off the aircraft track recording function
T+Number*: Turn on/change the track thickness to * number
P: Turn on the physical collision engine (different aircraft will collide and crash, only in the advanced full version)
CTRL+mouse wheel: zoom in and out of all aircraft sizes (for easy observation when there are multiple aircraft);
CTRL + C: Switch all aircraft 3D styles

2. Cluster control model

2.1 Software/hardware-in-the-loop simulation model composed of high-precision model + PX4 controller

2.1.1 PX4MavCtrler: __init__()

Parameters: self, ID=1, ip='127.0.0.1',Com='udp',port=0
ID: Simulation ID
IP: IP address where data is sent out
Com: connection mode with Pixhawk
Port: port number
The function is used to initialize the aircraft and its communication mode.

2.1.2 InitMavLoop()

Parameters: UDPMode=2
UDPMode: 0 and 1 correspond to UDP_Full and UDP_Simple Mode, 2 and 3 correspond to MAVLink_Full and MAVLink_Simple mode, 4 corresponds to MAVLink_NoSend
isCom: the symbol of serial communication
isRealFly: Flag of network direct connection mode
isRedis: Flag of Redis mode
Functions are used to establish communication. The default mode is MAVLink_Full

2.1.3 initOffboard()

This function requires no input parameters to start.
The function sends an Offboard command to make the aircraft enter Offboard mode and start sending Offboard information at a frequency of 30Hz.
The function sends information by calling the sendMavOffboardAPI() function.

2.1.4 sendMavOffboardAPI()

Parameters: type_mask=0, coordinate_frame=0, pos=[0,0,0], vel=[0,0,0], acc=[0,0,0],yaw=0,yawrate=0
type_mask: control mode, such as local position control, global position control, etc.
coordinate_frame: coordinate system type
pos: location information
vel: speed information
acc: acceleration information
yaw: yaw angle information
yawrate: Yaw rate information
The function determines the offboard mode based on the offMode variable and calls the corresponding function to send information.

2.1.5 SendPosNED()

Parameters: x=0,y=0,z=0,yaw=0
x: target location information
y: Target location information
z: target location information
yaw: yaw angle
The function will assign the offMode variable to 0, which means entering the local northeast coordinate system position control.

2.1.6 endOffboard()

The call does not require parameters and is called directly.
The function will send the command for PX4 to exit offboard mode and stop the information sending cycle.

2.1.7 stopRun()

The call does not require parameters, just call it directly.
The function will stop mavlink's information listening loop.

2.1.8 initPointMassModel()

Parameters: intAlt=0, intState=[0,0,0]
intAlt: aircraft initial altitude
intState: aircraft initial position, initial yaw angle
Function used to start particle model simulation.

2.1.9 PointMassModelLoop()

The function is the update loop of the particle model. Will be called by the initPointMassModel() function. No need for user to start it.

2.1.10 sendUE4PosNew()

Parameters:
copterID=1,vehicleType=3,PosE=[0,0,0],AngEuler=[0,0,0],VelE=[0,0,0],PWMs=[0]*8,runnedTime= -1,windowID=-1
copterID: simulated aircraft ID
vehicleType: aircraft style
PosE: North East coordinate system position
AngEuler: aircraft attitude angle
VelE: aircraft speed
PWMs: rotational speed
runnedTime: current time
windowID: window number
The function is to package the incoming data and send it to RflySim3D to create a new 3D model or update the status information of the model.

2.1.11 InitTrueDataLoop()

After the function is started, the UDP real data listening loop will be initialized. Listen for data from CopterSim via the 30100 series port.

2.1.12 EndTrueDataLoop()

The function will close the UDP real data listening loop.

2.1.13 endMavLoop()

function has the same functionality as the stopRun() function. Data monitoring of 20100 series ports will be turned off.

2.1.14 SendMavCmdLong()

Parameters: command, param1=0, param2=0, param3=0, param4=0, param5=0, param6=0, param7=0
For the definition of command and param1~7, please refer to:
https://mavlink.io/en/messages/common.html#COMMAND_LONG
https://mavlink.io/en/messages/common.html#MAV_CMD
The function is to send messages to PX4

2.1.15 sendMavOffboardCmd()

Parameters: type_mask, coordinate_frame, x, y, z, vx, vy, vz, afx, afy, afz, yaw, yaw_rate
type_mask: control mode
coordinate_frame: coordinate system information
x: location information
y: location information
z: location information
vx: speed information
vy: speed information
vz: speed information
afx: acceleration information
afy: acceleration information
afz: acceleration information
yaw: yaw angle
yaw_rate: Yaw angle rate
Function used to send offboard instructions to PX4

2.1.16 sendUDPSimpData()

Parameters: ctrlMode,ctrls
ctrlMode: airplane mode
ctrls: control parameters. For example: if the flight mode is takeoff, the control parameters should be [x, y, z, yaw]
Function sends a simple UDP message to the communication port.

2.1.17 SendVelNED()

Parameters: vx=0,vy=0,vz=0,yawrate=0
vx: speed control command
vy: speed control command
vz: speed control command
yawrate: yaw rate command
The function is used to switch the flight mode to the earth speed control mode, switch the offboard mode, send the judgment bit of the message, and set the coordinate system.

2.1.18 SendVelNEDNoYaw()

Parameters: vx,vy,vz
vx: speed control command
vy: speed control command
vz: speed control command
The function has similar functions to SendVelNED(). The difference is that the function does not set the yaw angle rate command, but only sends the speed control command.

2.1.19 SendVelFRD()

Parameters: vx=0,vy=0,vz=0,yawrate=0
vx: speed control command
vy: speed control command
vz: speed control command
yawrate: yaw rate command
The function will switch the flight mode to the body speed control mode, and the offboard mode to the speed control mode. Switch the working coordinate system to the collective coordinate system. The positive direction of the coordinate system is front, right, and down.

2.1.20 SendAttPX4()

Parameters: att=[0,0,0,0],thrust=0.5,CtrlFlag=0,AltFlg=0
att: determined based on CtrlFlag
CtrlFlag 0: att is a three-dimensional vector, including roll angle, pitch angle, and yaw angle. The unit is angle.
CtrlFlag 1: att is a three-dimensional vector, including roll angle, pitch angle, and yaw angle, the unit is radians
CtrlFlag 2: att is a four-dimensional vector containing quaternions
CtrlFlag 3: att is a three-dimensional vector, including roll angular rate, pitch angular rate, and yaw angular rate, the unit is rad/s
CtrlFlag 4: att is a three-dimensional vector, including roll angular rate, pitch angular rate, and yaw angular rate, the unit is degree/s
thrust: determined according to AltFlg
AltFlg 0: Total thrust, in the range of 0-1, (-1~1 is suitable for aircraft with reverse thrust)
AltFlg>0: desired height
CtrlFlag: Flag used to determine the input att definition
AltFlg: Flag used to determine the input thrust definition
The function will switch the aircraft's flight mode to attitude control mode and switch offboard mode. Send the aircraft target attitude information in the FRD (front, right, down) coordinate system to PX4

2.1.21 SendAccPX4()

Parameters: afx=0,afy=0,afz=0,yawValue=0,yawType=0,frameType=0
afx: acceleration information
afy: acceleration information

afz: acceleration information
yawValue: yaw information
yawType: 1, yaw angle control. 2. Yaw angle rate control
frameType: 0, north-east coordinate system. 1. Body FRD coordinate system
The function will switch the aircraft's flight mode to acceleration control mode and switch offboard mode. Switch coordinate systems based on frametype. Send acceleration control information to PX4

2.1.22 SendVelNoYaw()

Parameters: vx,vy,vz
vx: speed control command
vy: speed control command
vz: speed control command
The function has the same function as SendVelNEDNoYaw(), except that the function sets the current coordinate system to the body's FRD coordinate system.

2.1.23 SendVelYawAlt()

Parameters: vel=10,yaw=6.28,alt=-100
vel: speed command
yaw: yaw angle command
alt: height command
The function will set the flight mode to speed altitude yaw mode. Switch offboard mode. Switch the coordinate system to the northeast coordinate system. Send the aircraft's yaw information, speed information and altitude information to PX4

2.1.24 SendPosGlobal()

Parameters: lat=0,lon=0,alt=0,yawValue=0,yawType=0
lat: latitude information
lon: longitude information
alt: height
yawvalue: yaw
yawType: 0, no yaw. 1. yawvalue is the yaw angle. 2. yawvalue is the yaw angle rate
The function will switch the flight mode to the earth position control mode, switch the offboard mode, and the coordinate system will be the northeast earth coordinate system. Send the aircraft's position control information and yaw information to PX4

2.1.25 SendPosNEDNoYaw()

Parameters: x=0,y=0,z=0,
x: target location information
y: Target location information
z: target location information
The function is similar to 2.1.5 SendPosNED(). Fly using position control. The difference is that the function does not send the aircraft's yaw information.

2.1.26 SendPosFRD()

Parameters: x=0,y=0,z=0,yaw=0
x: target location information
y: Target location information
z: target location information
yaw: yaw information
The function sets the flight mode to the body position control mode. Set the offboard mode to the local northeast coordinate system position control mode. Set the working coordinate system to the body FRD coordinate system, and the function sends the position control command and yaw angle control command to PX4.

2.1.27 SendPosFRDNoYaw()

Parameters: x=0,y=0,z=0
x: target location information
y: Target location information
z: target location information
The function of the function is similar to the 2.1.26 SendPosFRD() function, except that the function does not send the aircraft yaw angle control command.

2.1.28 SendPosNEDExt()

Parameters: x=0,y=0,z=0,mode=3,isNED=True
x: target location information
y: Target location information
z: target location information
mode: 0, gliding mode. 1. Take-off mode. 2. Landing mode. 3. Rotorcraft hovering mode and fixed-wing hovering mode. 4. Fixed-wing aircraft, no throttle, no roll/pitch
isNED: True, local northeast coordinate system. False, North East of the body coordinate system
Function used to issue aircraft position control. Issue position control instructions after resetting the aircraft's working mode based on the input.

2.1.29 enFixedWRWTO()

The function is used to send a takeoff permission command to the aircraft on the runway. The function is called without parameters.

2.1.30 SendCruiseSpeed()

Parameters: Speed=0
Speed: The cruising speed of the aircraft.
Function is used to change the cruise speed of the aircraft.

2.1.31 SendCruiseRadius()

Parameter: rad=0
rad: The cruising radius of the aircraft.
Function is used to modify the cruise radius of the aircraft.

2.1.32 sendMavTakeOffLocal()

Parameters: xM=0,yM=0,zM=0,YawRad=0,PitchRad=0,AscendRate=2
xM: location information
yM: location information
zM: location information
YawRad: Yaw angle rate
PitchRad: Pitch rate
AscendRate: Ascend rate
The function is used to send the desired local position command to the aircraft to make the aircraft take off to the desired position.

2.2 High-precision comprehensive model composed of high-precision model + Simulink controller

2.2.1 outHILStateData

uint32_t time_boot_ms; message timestamp
uint32_t copterID; aircraft ID
int32_t GpsPos[3]; simulated GPS position, latitude and longitude *1e7, altitude *1e3, the positive direction of altitude is up
int32_t GpsVel[3]; simulated GPS speed, northeast coordinate system
int32_t gpsHome[3]; GPS initial position, the unit is the same as GpsPos
int32_t relative_alt; height, positive direction is up
int32_t hdg; heading angle, northeast coordinate system.
int32_t satellites_visible; GPS raw data, satellite sum
int32_t fix_type; GPS raw data, fixed wing type
int32_t resrveInit; reserved for future use
float AngEular[3]; simulated attitude angle, unit: radians
float localPos[3]; Simulation position, northeast coordinate system. Unit: m
float localVel[3]; Simulation velocity, northeast coordinate system, unit: meters/second
float pos_horiz_accuracy; GPS horizontal accuracy, unit: meters
float pos_vert_accuracy; GPS vertical accuracy, unit: meters
float resrveFloat; Reserved for future use

2.2.2 outHILStateShort

int checksum; Message check bit
int32_t gpsHome[3]; GPS initial position, latitude and longitude *1e7, altitude *1e3, the positive direction of altitude is up

float AngEular[3]; simulated attitude angle, unit: radians
float localPos[3]; Simulation position, northeast coordinate system. Unit: m
float localVel[3]; Simulation velocity, northeast coordinate system, unit: meters/second

2.2.3 inHILCMDData

uint32_t time_boot_ms; Message timestamp
uint32_t copterID; Aircraft ID
uint32_t modes; Flight mode
uint32_t flags; Flight phase flags
float ctrls[16]; Control command information

2.2.4 inOffboardShortData

int checksum; Message check bit
int ctrlMode; Control mode flag bit
float controls[4]; Control command information

2.2.5 CreateStructure function

Parameters: SimStruct *S, int indx
The function is to store a dynamically allocated memory pointer of type SInfo in the input structure. This pointer can be accessed or modified via index indx.

2.2.6 mdlStart function

Parameters: SimStruct *S
The main purpose of the function is to create a UDP server for each aircraft and start it. The function will call the StartUDPServer() function to create and start the UDP server. Create a new structure for each aircraft through the CreateStructure() function. For non-Linux systems, this code activates the Winsock DLL. Winsock DLL is a library that provides network communication functions on Windows systems.

2.2.7 mdlOutputs function

Parameters: SimStruct *S, int_T tid
The function will obtain the UDP mode and update the aircraft information according to different modes.

2.2.8 mdlUpdate function

Parameters: SimStruct *S, int_T tid
Function used to update aircraft status information.

2.2.9 mdlTerminate()

Function used to terminate the simulation.

2.2.10 StartUDPServer()

Parameters: SimStruct *S, SInfo *info, int index
Function used to initialize and start a UPD server.

2.2.11 mdlInitializeSizes()

The function is used to initialize the Simulink custom module of MATLAB. Its main function is to set the model parameters, including the number of input and output ports, data type and width, etc.

2.2.12 netServerInit()

The main function of the function is to create a UDP server on the specified port and return the socket.

2.2.13 mdlCheckParameters()

The function is used to check whether the incoming parameters meet the requirements. Parameter values, which are port number, number of vehicles and UDP mode, are converted into integer format.

2.3 based on mass point model fixed wing model

An example has been added at the end of this section to more clearly illustrate how to use the interface of the fixed wing mass model to implement swarm control.

2.3.1 Vehicle: __init__()

Function prototype: __init__(self,CopterID = 1,Vehicletype = 3,mapName='Grasslands', updatefreq = 100, isBroadCast=False)
Parameters: (self,CopterID = 1,Vehicletype = 3,mapName='Grasslands', updatefreq = 100, isBroadCast=False)
CopterID: Aircraft ID
Vehicletype: Aircraft style
mapName: map name
updatefreq: update frequency
isBroadCast: Whether to broadcast, that is, whether aircraft movement data is sent to other computers
RecIPPort: Mavlink receiving interface 20100 series, 20100+(CopterID-1)*2
SendIpPort: Mavlink sending interface 20101 series, 20100+(CopterID-1)*2+1
isInPointMode: flag to enable the particle model
CurFlag: Current control mode. 0: Offboard control mode. 2: Location model. 10: Rolling phase of takeoff mode. 11: Climb phase of takeoff mode. 12: Level flight mode. 121: Hovering mode. 13: Speed-height yaw model.
CircleDir: The circling direction when the aircraft is circling. 1: Circle clockwise. 2: Circle counterclockwise.
EnList: Flag bit of control mode. EnList[0] position control. EnList[1] speed control. EnList[2] acceleration control. EnList[3] force control. EnList[4] Yaw angle control. EnList[5] Yaw rate control.

2.3.2 initSimpleModel()

This function is used to establish receiving connections and sending connections, and initialize objects.
intState: intState[0] Initial position of the aircraft intState[1] Initial position of the aircraft intState[2] Initial yaw angle Indicates the offset of the initial position of the aircraft relative to the origin of the scene
targetIP: data sending target IP address
GPSOrigin: map origin location

This function is the starting point for starting the particle model simulation and will be used to create a new particle model object. This is the default parameter when the function is called. You can modify the corresponding parameters according to your own needs.
def initSimpleModel(self,intState=[0,0,0],targetIP = '127.0.0.1', GPSOrigin = [40.1540302,116.2593683,50]):
This function will call the getTerrainAltData interface in the UE service to obtain the altitude of the aircraft's initial position. This is the line of code that is called:
self.intAlt=self.map.getTerrainAltData(self.intStateX,self.intStateY)
Determine self.updatefreq (data update frequency) and decide whether to call SimpleModelLoop() to enable the aircraft's own data update.
Call this RecMavLoop() function to listen to data from the bound 20100 series interface.

2.3.3 SimpleModelLoop()

This function will be called by the initSimpleModel() function.
This function is the root function of the particle model. The main function is to continuously update the model information by calling the ModelStep() function in a loop when self.isInPointMode is true, and transmit the model information to the external interface.
The frequency of data updates is controlled by using the self.updatefreq parameter to control the frequency of calling the ModelStep() function. When self.isInPointMode is true, it indicates that the current model is a particle model.

2.3.4 ModelStep()

Function to start updating model data. Including opening the loop function that sends data to the external interface. This function has been called in the SimpleModelLoop() function.
This function first ensures that the data is sent on the correct timeline by judging the time difference between the current time and the last time it was called. First call the ProssInput() function to process the instructions sent by the user and update the status information of the aircraft.
In the Step() function, the data obtained by the ProssInput() function is further processed according to the different control models, and the attitude information of the aircraft is updated, etc.
Call the SendUavState() function to send the aircraft's status information data to the 20101 series port to simulate the internal state of the PX4.
Call the SendOutput() function to send the data to RflySim3D to simulate the three-dimensional true value data of the aircraft.

2.3.5 ProssInput()

This function will be called by the ModelStep() function. Used to process user instructions and generate desired control information for the aircraft.
Issue control instructions based on the mode. If it is Offboard control mode, it is also necessary to distinguish between speed control, position control and other methods. Differentiate control modes according to self.CurFlag. This parameter definition has been explained in 2.3.1 above.
self.velOff: Desired speed control.
self.yawRateOff: Desired yaw rate.
The function of the fixedPitchFromVel() function called in this function is to generate the pitch angle of the aircraft based on the speed command. The unit of pitch angle is radians.

2.3.6 Step()

This function is called from the ModelStep() function.
In this function, the Runge-Kutta method is used to push forward one step further. For different modes, the expected state information of the aircraft is calculated.
uavVelNED UAV speed in the north-east coordinate system
uavPosNED the offset position of the drone relative to the scene origin in the north-east coordinate system.
This function will determine the current model based on the self.CurFlag parameter. The definition of this parameter has been explained in 2.3.1 above.
This function will call the getTerrainAltData interface again to determine the map altitude of the aircraft's true position.

2.3.7 SendUavState()

This function will be called by the ModelStep() function.
Send the aircraft status information updated by the Step() function to the bound 20100 series interface.
The specific operation of the function is to set the check digit of the sent data. Pack the check digit, the original position of the scene with high latitude and longitude, the attitude angle information of the aircraft, the offset position of the aircraft and the speed of the aircraft. Send data to the 20100 series interface.
The interface setting method is explained in 2.3.1.

2.3.8 SendOutput()

This function will be called by the ModelStep() function.
This function will call the sendUE4PosNew() function to send the updated data to RflySim3D.
The data sent in a package includes: the ID of the aircraft, the style of the aircraft, the true position of the aircraft in the North East coordinate system, the attitude angle information of the aircraft, the speed information of the drone in the North East coordinate system, and the speed of the aircraft motor.

2.3.9 fixedPitchFromVel()

This function will be called by the ProssInput() function.

The input to the function is the flight speed of the aircraft. The calculation returns the desired pitch angle of the aircraft.

2.3.10 RecMavLoop()

This function will be called by the initSimpleModel() function.
The functions that the function will call are SendPosNED() function, SendMavArm() function, SendCruiseSpeed() function, SendCruiseRadius() function, sendMavTakeOff() function, sendMavTakeOffGPS() function, and SendVelYawAlt() function.
The main functions of the called functions are to send position control instructions, send aircraft unlocking instructions, send aircraft cruise speed instructions, send aircraft circling radius instructions, send takeoff instructions, send takeoff instructions in the longitude and latitude coordinate system, and send aircraft flight control instructions.
The main function of the function is to start the data listening loop. According to the control mode, different data sending functions are called to send data. Different calling functions will be selected according to ctrlMode, where ctrlMode is determined by ListenDate[1].
ListenDate is the received information. ListenDate[0] Data check digit. ListenDate[1] control mode flag. ListenDate[2:6] control information bits.
Control mode bits:
0: Empty command
2: Position control, the SendPosNED() function will be called.
9: Unlock, the SendMavArm() function will be called.
10: When circling, the SendCruiseSpeed() function and SendCruiseRadius() function will be called.
11: Take off and call the sendMavTakeOff() function.
12: The global coordinate system takes off and calls the sendMavTakeOffGPS() function.
13: Speed, altitude and yaw control, call the SendVelYawAlt() function.

2.3.11 SendPosNED()

This function is called by the RecMavLoop() function.
Send position control instructions. Modify EnList to adjust the control mode and send desired position information and desired yaw.
When ctrlMode == 2, ListenDate[2:6] represents x, y, z, yaw control information respectively.

2.3.12 SendMavArm()

This function will be called in the RecMavLoop() function.
Send aircraft unlock command.
self.isArmed The aircraft unlock flag.

2.3.13 SendCruiseSpeed()

This function will be called in the RecMavLoop() function.
When ctrlMode == 10, ListenDate[2:4] represents the cruising speed and hovering radius of the drone.
Set the cruise speed of the fixed-wing drone.

2.3.14 SendCruiseRadius()

This function will be called in the RecMavLoop() function.
Sets the hover radius of the fixed wing.

2.3.15 sendMavTakeOff()

This function will be called in the RecMavLoop() function.
Enter the takeoff mode and set the parameters required for subsequent takeoff.

2.3.16 sendMavTakeOffGPS()

This function will be called in the RecMavLoop() function.
It has the same function as 2.3.15, the difference is that this function uses latitude and longitude coordinate information as input.

2.3.17 SendVelYawAlt()

This function will be called in the RecMavLoop() function.
When ctrlMode == 13, ListenDate[2:5] represents the speed, altitude, and yaw information of the drone.
Interface for sending speed, altitude and yaw control.

2.3.18 sendUE4PosNew()

This function will be called by the SendOutput() function.
This function will call the sendBuf() function.
The main function of this function is to send the received data to RflySim3D by calling the sendBuf() function.
This function will package the data that needs to be sent. The packaged data includes: data check digit, simulated aircraft ID, aircraft style, aircraft motor speed, aircraft speed, aircraft attitude information, and aircraft offset in the scene. Location, program running time. The description order is the same as the packaging order in the program.

2.3.19 sendBuf()

This function will be called by sendUE4PosNew() function and sendUE4Cmd() function.
The function first determines whether the WindowID is confirmed. If the WindowID is confirmed, data is sent to the specified port. If not determined, it is determined by the program loop.
The function of the function is to determine whether it is in broadcast mode based on the self.isBroadCast parameter. If it is not in broadcast mode, it will send data to the local machine. If it is in broadcast mode, it will send data to all computers in the network.
The sending port is a 20010 series port.

2.3.20 EndSimpleModel()

This is the function that ends the particle model simulation. The function sets the two parameters self.isInPointMode and self.t4Flag to False, thereby ending the update cycle of the model's own data and

the cycle of monitoring data.
When calling this function, no parameters are required.

2.3.21 Model usage examples

mav = VehicleApi.Vehicle(1,100,'OldFactory')
First use the Vehicle class of VehicleApi to generate objects. Here is the aircraft ID, data update frequency, and map name.
__init__(self,CopterID = 1,Vehicletype = 3,mapName='Grasslands', updatefreq = 100, isBroadCast=False) This is a list of parameters that the function can set, followed by aircraft ID, aircraft style, map name, and data update frequency. Whether it is broadcast mode. As you can see, the default aircraft ID is 1, the aircraft style is 3, the map is grass, and the update frequency is 100, which is not broadcast mode.
mav.initSimpleModel([-250,-119,0])
Call the initSimpleModel() function to initialize the simulation object. Here, the offset position of the aircraft relative to the scene origin is set to [-250,-119,0].
initSimpleModel(self,intState=[0,0,0],targetIP = '127.0.0.1',GPSOrigin=[40.1540302,116.2593683,50])
The parameters that can be set include the offset position of the aircraft relative to the origin of the scene, the data sent IP address, scene origin coordinates. By default, the aircraft has no offset relative to the scene origin. The target IP address for data transmission is '127.0.0.1', and the scene origin is set to [40.1540302,116.2593683,50], which is the latitude and longitude coordinate information.
mav.sendMavTakeOff(500, 0, -100, 0)
The aircraft takeoff function is called here. Here, the coordinate center of the aircraft's hovering is set to (500,0,-100). The unit is m, and the coordinate system of the coordinate center is the northeast coordinate system.
sendMavTakeOff(self,xM=0,yM=0,zM=0,YawRad=0,PitchRad=20/180.0*math.pi) The parameters that can be set when calling include the aircraft's circling center coordinates and the yaw direction during taxiing , the slope during the climb phase.
mav.SendMavArm(True)
The unlocking function of the aircraft is called here. There is only one parameter here.
mav.EndSimpleModel()
This is the function that ends the simulation.

2.3.22 Model illustration

2.4 based on mass point model rotor Model

2.4.1 PX4MavCtrler:__init__()

Function prototype: def __init__(self, port=20100, ip='127.0.0.1'):
Parameters: (self, port=20100, ip='127.0.0.1')
This function will create a data output interface and a data listening interface. And initialize drone data.
Port: port number
IP: correspondence address

isInPointMode: flag to enable the particle model
isCom: hardware-in-the-loop simulation flag bit
type_mask: control mode flag bit
coordinate_frame: coordinate system selection
EnList: Flag bit of control mode. EnList[0] position control. EnList[1] speed control. EnList[2] acceleration control. EnList[3] force control. EnList[4] Yaw angle control. EnList[5] Yaw rate control.
uavAngEular: PX4 attitude angle
trueAngEular: True simulated attitude angle of the CopterSim DLL model
uavAngRate: PX4 attitude angular rate
trueAngRate: True simulated angular rate for the CopterSim DLL model
uavPosNED: The current position of PX4 relative to the take-off position (North East coordinate system)
truePosNED: The true simulated position relative to the UE4 map origin CopterSim DLL model
uavVelNED: velocity in PX4 north east coordinates
trueVelNED: True simulation speed for CopterSim DLL models
isVehicleCrash: Flag of collision
isVehicleCrashID: ID of the aircraft that collided with this aircraft
uavPosGPS: The GPS position of PX4 is in the northeast coordinate system, longitude, latitude, altitude, time, relative altitude, speed, and heading.
truePosGPS: Real simulated GPS position for CopterSim DLL model
uavPosGPSHome: PX4 take-off position in the northeast coordinate system (GPS origin)
uavGlobalPos: The position information obtained by PX4 is converted to the position in UE4
trueAngQuatern: True simulation quaternion for CopterSim DLL models
trueMotorRPMS: Real simulated motor speeds for CopterSim DLL models
trueAccB: True simulated acceleration for CopterSim DLL models

2.4.2 initPointMassModel()

This function is used to enable the particle model
Parameters: (self,intAlt=0,intState=[0,0,0])
intAlt: CopterSim's initial height relative to the ground height of the current map
intState: [0]: Initial position [1]: Initial position [2]: Yaw angle

2.4.3 PointMassModelLoop ()

This function will be called by the initPointMassModel() function.
This function will initialize the aircraft state. And start the model cycle. The EnList flag bit determines the control mode. Update aircraft status information, including position, speed, angular velocity, etc.
This function will call sendUE4PosNew() to send the update data to UE4.

2.4.4 sendUE4PosNew()

This function is called by PointMassModelLoop().
This function is used to package the aircraft information and send it to the specified address and port. Use 20010 series ports for data sending. The packaged data includes: data check digit, simulated aircraft ID, aircraft style, aircraft motor speed, aircraft speed, aircraft attitude information, aircraft offset position

in the scene, and program running time. The description order is the same as the packaging order in the program.

2.4.5 SendPosNED()

Send position control instructions. Modify EnList to adjust the control mode and send desired position information and desired yaw.
When ctrlMode == 2, ListenDate[2:6] represents x, y, z, yaw control information respectively.

2.4.6 EndPointMassModel()

This is the function that ends the particle model simulation. The function sets the self.isInPointMode parameter to False, thereby ending the update cycle of the model's own data and the cycle of monitoring data. The PointMassModelLoop() function will be based on
isInPointMode determines when the program is running. Therefore, after setting isInPointMode to False, it will automatically stop.
When calling this function, no parameters are required.

2.5 Fixed wing guidance model

2.5.1 Vehicle: __init__()

Parameters: self,CopterID = 1,Vehicletype = 3,mapName='Grasslands', updatefreq = 100, isBroadCast=False
CopterID: simulation ID in CopterSim
Vehicletype: The display style of the aircraft in RflySim3D
mapName: Scene in RflySim3D
updatefreq: The frequency of sending messages to RflySim3D, which is also the default update frequency of the model.
isBroadCast: Indicates whether the aircraft's 3D data is broadcast to other computers on the LAN
The function of this function is to initialize the aircraft and scene. At the same time, determine the information interaction method for model operation.

2.5.2 initSimpleModel()

Parameters: self, intState=[0,0,0], targetIP = '127.0.0.1', GPSOrigin=[40.1540302,116.2593683,50]
intState: initial position of the aircraft, initial yaw angle
targetIP: the sending IP address of aircraft data
GPSOrigin: GPS location of the aircraft’s initial position
The function of this function is the initial aircraft model. Determine the initialization status of the aircraft and how it interacts with information.

2.5.3 SendMavArm()

Parameters: isArm
isArm: unlock flag

2.5.4 sendMavTakeOff()

Parameters: xM=0,yM=0,zM=0,YawRad=0,PitchRad=20/180.0*math.pi
xM: The position of the next waypoint
yM: The position of the next waypoint
zM: The position of the next waypoint
YawRad: Yaw direction of taxiing
PitchRad: Slope of Climb

2.5.5 SendPosNED()

Parameters: x=0,y=0,z=0,yaw=0
x: target position
y: target position
z: target position
yaw: yaw angle

2.5.6 SendCruiseRadius()

Parameter: rad=20
rad: hovering radius of fixed wing

2.5.7 SendCruiseSpeed()

Parameters: Speed=10
speed: cruise speed of fixed wing

2.5.8 sendMavTakeOffGPS()

Parameters: lat,lon,alt,yawDeg=0,pitchDeg=15
lat: dimension information
lon: longitude information
alt: height information
yawDeg: Yaw direction setting, the unit here is °
pitchDeg: climb angle setting, the unit here is °

2.5.9 SendVelYawAlt()

Parameters: vel=10, alt=-100, yaw=6.28
vel: speed control
alt: height control
yaw: yaw angle

2.5.10 EndSimpleModel()

Parameters: No parameters required
Call the function to close model simulation.

2.5.11 UEMapServe: LoadPngData()

Parameters: name
name: the file name to be read

2.5.12 getTerrainAltData()

Parameters: xin, yin
xin: x coordinate of the location where height information is to be obtained
yin: y coordinate of the location where height information is to be obtained

2.5.13 EarthModel: lla2ecef()

Parameters: lat, lon, h
lat: latitude
lon: longitude
h: height
Convert from latitude and longitude coordinate system to geocentric coordinate system

2.5.14 ecef2enu()

Parameters: x, y, z, lat0, lon0, h0
x: location information
y: location information
z: location information
lat0: latitude information
lon0: longitude information
h0: height information
Convert from geocentric coordinate system to northeast celestial coordinate system

3. Cluster communication interfaces and protocols

3.1 CopterSim communication mode

3.1.1 UDP_Simple model

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 checksum, coordinate mode selection ctrlMode and 4 float control quantities controls[4]. ctrlMode can have multiple protocols (RflyUdpFast.cpp and CopterSim need to be changed). If ctrlMode is set to <0 (less than 0), it means that it is an empty command and the module will not publish messages to the outside. The specific protocols supported by ctrlMode are shown in the following table.
In UDP_Simple mode, once the inOffboardShortData message is received, Copter will automatically send a message to PX4 to unlock and enter Offboard mode. This simplifies the process for users to control the aircraft.
struct inOffboardShortData {

```
int checksum; // Check digit 1234567890
int ctrlMode; //Mode selection
float controls[4]; //Four-digit control amount
}
```

Control modes supported by UDP_Simple

pattern label	describe
0	Speed mode in navigation coordinate system [vx, vy, vz, yaw_rate]
1	Speed mode in the body coordinate system [vx, vy, vz, yaw_rate]
2	Position mode in navigation coordinate system [x, y, z, yaw]
3	Position mode in the body coordinate system [x, y, z, yaw]
4	Attitude throttle control commands [roll, pitch, yaw (radian), throttle (0~1)], can be automatically unlocked and can automatically enter OffBoard mode
5	Attitude throttle increment control command [roll, pitch, yaw, throttle increment] - can be automatically unlocked and 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	Indicates setting the speed and circling radius of the fixed-wing aircraft, [speed, radius, -, -]
11	Indicates Mavlink takeoff command, automatic unlocking, navigation coordinate system position [x, y, z, -]
12	Indicates Mavlink takeoff command, automatic unlocking, GPS coordinate system position [latitude, longitude, altitude, -]
13	Speed altitude heading command, automatically unlock and enter offBoard mode, GPS coordinate system position [speed, altitude, heading, -]
14	Position mode in Global coordinate system, GPS coordinate system position [lat_int, lon_int, alt_float, yaw_float]
30	Used for VTOL mode switching, indicating the flight mode switching command, corresponding to the mavlink command of MAV_CMD_DO_VTOL_TRANSITION, controls[0] represents the State bit. See the link for definition (https://mavlink.io/en/messages/common.html#MAV_VTOL_STATE). controls[1] represents the Immediate bit (1: Force immediate pre-switching, 0: normal transition normal switching.).

The above control mode is used to send control instructions to PX4. Usually, when the control algorithm calculates the control command, it needs to be based on the current position and speed information of the vehicle. UDP_Simple mode provides data structures to obtain this information.
As shown below is the simplified data received by CopterSim in UDP_Simple mode. Specifically, the checksum needs to be 1234567890 to verify the correctness of the data. gpsHome Longitude and latitude height data of aircraft take-off point. AngEular Euler angle, pitch, roll and yaw of the aircraft, in degrees. localPos is the relative coordinates of the aircraft relative to the take-off point, in the northeast coordinate system, in meters. localVel The speed of the aircraft, north east, unit m/s. gpsHome is of int type. First get the format of degrees (multiplied by 1e-7) degrees (multiplied by 1e-7) meters (multiplied by 1e-3). Then use the Simulink module, combined with the unified GPS origin, to get the relatively unified GPS of this aircraft. The offset position of the origin, combined with localPos, can be used to obtain the real-time position of the aircraft relative to the unified origin.

```
struct outHILStateShort{
int checksum; //Check digit 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.

3.1.2 UDP_Full model

In UDP_Full mode, receiving an Offboard message will also automatically unlock and enter Offboard mode. Therefore, in UDP_Full mode, users do not need to manually send instructions to unlock and enter Offboard mode. The data received in UDP_Full mode is shown in 4.1.1 and is represented by outHILStateData. Including position and speed in the GPS coordinate system, position and speed in the NED coordinate system, etc. In actual use, there is no need 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 data packets need 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 ;
```

Users can also get real data SOut2Simulator, which can be read in UDP_Simple mode or UDP_Full mode, because this type of data uses additional ports. The corresponding receiving port of CopterSim is 30100, and the sending port is 30101 port. SOut2Simulator is the real state information from the model without adding noise. This is also data that can only be obtained during the simulation process.
Similarly, SOut2Simulator data is transmitted through the netDataShort structure, which means that the total length of all data in SOut2Simulator is 192 bytes.

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

3.1.3 UDP_Ultra Simple model

3.2 Cluster communication interface

3.2.1 20100 series ports

For non-Simulink_DLL modes, use 20100 series ports for communication. For CopterSim, port 20100+2n is used to receive messages and port 20101+2n is used to send messages, where n=0,1,2…. When the number of aircraft is increased, the port value gradually increases. For example, if a second aircraft is added, its ports are 20102 and 20103.

3.2.2 30100 series port

For Simulink_DLL mode, CopterSim uses port 30100 to receive information and port 30101 to send information. Similarly, when there are multiple aircraft, the port is also increased starting from 30100.

3.2.3 20010 series port

Used for UE communication.
SocketReceiverMap: Receive one of the 20010~20029 ports of the local machine (this is because multiple RflySim3D on one computer will compete for the port, so these 20 ports will be allocated in sequence according to the serial number)

3.3 Cluster communication protocol

3.3.1 basic data structure

Each data structure contains a reset() function to reset the data.

3.3.1.1 outHILStateData

uint32_t time_boot_ms; message timestamp
uint32_t copterID; aircraft ID
int32_t GpsPos[3]; simulated GPS position, latitude and longitude *1e7, altitude *1e3, the positive direction of altitude is up
int32_t GpsVel[3]; simulated GPS speed, northeast coordinate system
int32_t gpsHome[3]; GPS initial position, the unit is the same as GpsPos
int32_t relative_alt; height, positive direction is up
int32_t hdg; heading angle, northeast coordinate system.
int32_t satellites_visible; GPS raw data, satellite sum

int32_t fix_type; GPS raw data, fixed wing type
int32_t resrveInit; reserved for future use
float AngEular[3]; simulated attitude angle, unit: radians
float localPos[3]; Simulation position, northeast coordinate system. Unit: m
float localVel[3]; Simulation velocity, northeast coordinate system, unit: meters/second
float pos_horiz_accuracy; GPS horizontal accuracy, unit: meters
float pos_vert_accuracy; GPS vertical accuracy, unit: meters
float resrveFloat; Reserved for future use

3.3.1.2 outHILStateShort

int checksum; Message check bit
int32_t gpsHome[3]; GPS initial position, latitude and longitude *1e7, altitude *1e3, the positive direction of altitude is up
float AngEular[3]; simulated attitude angle, unit: radians
float localPos[3]; Simulation position, northeast coordinate system. Unit: m
float localVel[3]; Simulation velocity, northeast coordinate system, unit: meters/second

3.3.1.3 inHILCMDData

uint32_t time_boot_ms; Message timestamp
uint32_t copterID; Aircraft ID
uint32_t modes; Flight mode
uint32_t flags; Flight phase flags
float ctrls[16]; Control command information

3.3.1.4 inOffboardShortData

int checksum; Message check bit
int ctrlMode; Control mode flag bit
float controls[4]; Control command information

3.3.2 Platform private UDP protocol

There are three modes, including FullData mode, SimpleData mode, and UltraSimple mode.
In Fulldata mode, the module input is a 15-dimensional double vector, and the module output is a 28-dimensional double vector.
In SimpleData mode, the input is a 5-dimensional double vector, and the output is a 12-dimensional double vector.
In UltraSimple mode, the input is a 5-dimensional double vector and the output is a 12-dimensional double vector.

3.3.3 Mavlink protocol

The MAVLink protocol stipulates the vehicle type, flight mode, vehicle status, system components, message frame coordinate format, ground station command format, micro-vehicle data flow, etc. Below

we will introduce some key contents of the agreement.
(1) Vehicle types: including universal, fixed-wing, quad-rotor, coaxial propeller structure, ordinary tail-rotor helicopters, free-flying balloons, rockets, ground vehicles, submarines, etc.
(2) Vehicle status: This protocol requires that the vehicle status can be transmitted back to the ground station in real time, including: uninitialized/unknown status, starting up, calibrating, on standby, driving, system abnormal but navigable, completely abnormal, executing shutdown instruction.
(3) System components: The agreement requires at least support for the following components: GPS, mission manager, route manager, map, camera/video camera, 3 attitude sensors, network and data transmission relay, system controller, 14 rudders machine. GPS and maps play an important role in the MAVLink protocol. GPS can provide important longitude and latitude information to the flight controller and assist in providing altitude information. The micro-vehicle can move according to the planned trajectory completely relying on the positioning and navigation function of GPS. Therefore, the ground station needs to display the number of GPS satellites, positioning accuracy, vehicle location and other information in real time. MAVLink fully provides these capabilities.
(4) Route planning instructions: For ordinary drones, being able to accurately fly along the route is the core task of the aircraft. MAVLink can be divided into two types of instructions: immediate execution and task script. There are 7 instruction parameters, each representing different data types. These regulations refer to the ARINC424 navigation database standard for civil aviation aircraft, including instructions such as waypoint, continuous circling, circling at the waypoint for N circles, circling at the waypoint for N seconds, return to the take-off point, and landing at the set point. These rich instructions ensure that the aircraft can fly according to the route. During aerial survey, the aircraft is required to fly strictly according to the route to cover all areas to be surveyed and mapped, so that the captured pictures can be spliced in the later stage. Immediate execution of instructions allows ground operators to flexibly operate drones for aerial photography operations. Micro-drone aerial photography is not only low-priced, but also has the characteristics of flying lower than manned aircraft and flying faster than handheld devices, and can complete different tasks. Videos shot from a different perspective have received widespread attention. However, MAVLink cannot realize video transmission due to data throughput rate limitations. Integrating data transmission and image transmission has become the direction to perfect MAVLink.

4. RflyUdpFast interface

4.1 Simulink components

4.1.1 UDP IP Address

Broadcast mode

Use broadcast communication method to simulate LAN networking.
The advantage is that the configuration is simple, and there is no need to check the addresses of the two computers in the LAN. Copying it to other computers can run it directly, but the communication efficiency is low, unstable, and the delay is large.

Use IP mode

Use the method of specifying the IP addresses of two computers to create a network simulation.
The configuration is more troublesome (you need to specify the IPs of two computers), but the performance is high, the communication speed is fast, and the delay is small. This mode is recommended for simulations of more aircraft.

4.1.2 UDP Port

UDP Port is the initial port number of the first aircraft, and the default starting port is 20100. Each CopterSim needs to occupy a port to send and receive messages.

4.1.3 Vehicle Number

Vehicle numberThe number of aircraft indicates the number of CopterSims that need to be connected. The number of input and output ports of the module is controlled by this option. If 10 is entered, the module will automatically generate 10 pairs of input and output interfaces.

4.1.4 UDP Mode

FullData mode

The module input is a 15-dimensional double vector. The specific definition (implementing MAVLink's Offboard message) is as follows
Dimension 1: time_boot_ms; % current timestamp (just fill in 0, not currently used)
Dimension 2: copterID; % aircraft ID (just fill in 1, not currently used)
The third dimension: type_mask; % input control mode (same as Offboard definition)
4th dimension: coordinate_frame; % coordinate system mode (same as Offboard definition)
The 5th to 15th dimensions: ctrls[11]; % respectively correspond to the 3-dimensional expected position pos, the 3-dimensional expected velocity vel, the 3-dimensional expected acceleration acc, the 1-dimensional expected yaw angle yaw, and the 1-dimensional expected yaw. Yaw rate yawRate. (Same as Offboard definition)

The module output is a 28-dimensional double vector (all forwarded from Pixhawk internal filter values), which is specifically defined as follows
Dimensions 1~3: gpsHome[3]; %The longitude and latitude high coordinates of the Home point (which will not change after power on). The longitude and latitude need to be divided by 1e7 to get the longitude and latitude in degrees, and the height needs to be divided by 1e3 to get the unit of m. The height of (upward is positive)
Dimensions 4~6: AngEular[3]; %Euler angle of attitude estimated by Pixhawk, unit radians
Dimensions 7 to 9: localPos[3]; %The relative north-east position vector estimated by Pixhawk with gpsHome as the origin, unit m, and the z-axis is positive downward.
Dimensions 10~12: localVel[3]; % Northeast movement velocity vector, unit m/s
Dimensions 13~15: GpsPos[3]; %Real-time GPS position, the unit is the same as gpsHome, but it will change in real time

Dimensions 16~18: GpsVel[3]; %GPS speed, need to be divided by 100 to get the speed in m/s
Dimension 19: time_boot_ms; % power-on time
Dimension 20: copterID; % aircraft ID
Dimension 21: relative_alt; % GPS relative altitude, need to be divided by 1000 to get the altitude in m, upward is positive
22nd dimension: hdg; % GPS heading angle, need to be divided by 1000 to get the angle in the range of 0~360 degrees
Dimension 23: satellites_visible; % number of visible satellites
Dimension 24: fix_type; % positioning accuracy
Dimension 25: resrveInit; % reserved bits of type int
Dimension 26: pos_horiz_accuracy; % horizontal positioning accuracy, unit m
Dimension 27: pos_vert_accuracy; % vertical positioning accuracy, unit m
Dimension 28: resrveFloat; % float type reserved bit, not enabled

SimpleData mode

The input is a 5-dimensional double vector. The specific definition (implementing MAVLink's Offboard message) is as follows
1st dimension: ctrlMode; The first bit of % is the flag bit, 0: indicates the earth speed control mode Earth Vel; 1: the body speed control mode Body Vel; 2: the earth position control mode Earth Pos; 3: the body position control mode Body Pos
Dimensions 2~5: If ctrlMode =0, then these four dimensions correspond to the vx, vy, vz speed + yawRate yaw rate signal in the earth coordinate system (at the position when unlocked); if ctrlMode =1, then these four dimensions correspond to The vx, vy, vz speed + yawRate yaw rate signal in the body coordinate system (based on the body attitude and position when unlocked) is obtained; if ctrlMode =2, then these four dimensions correspond to the earth coordinate system (taking the position when unlocked as the origin) The x, y, z position + yaw signal under
Generally, ctrlMode=0 is more common, and the global speed is directly given for trajectory control. If you want to switch between position and speed, you need to modify the value of ctrlMode in real time and adjust the definition of the 2nd to 5th dimension signals.
The output is a 12-dimensional double vector, and the order is defined as follows
Dimensions 1~3: gpsHome[3]; % Longitude and latitude high coordinates of the Home point (which will not change after power on). The longitude and latitude need to be divided by 1e7 to get the longitude and latitude in degrees, and the height needs to be divided by 1e3 to get the unit of m. The height of (upward is positive)
Dimensions 4~6: AngEular[3]; % Attitude Euler angle estimated by Pixhawk, unit radians
7th to 9th dimensions: localPos[3]; % The relative north-east position vector with gpsHome as the origin, estimated by Pixhawk, in m, and the z-axis is positive downward.
Dimensions 10~12: localVel[3]; % Movement velocity vector of the northeastern land, unit m/s

UltraSimple mode

The input is a 5-dimensional double vector. The specific definition is exactly the same as the SimpleData mode.
The output is a 12-dimensional double vector, and the order is defined as follows

Dimensions 1~3: GlobalPos [3]; % CopterSim global position, unit m
Dimensions 4~6: localPos[3]; % Estimated by Pixhawk, the relative north-east position vector with gpsHome as the origin, unit m, the z-axis is positive downward
Dimensions 7~9: localVel[3]; % The movement velocity vector of the northeastern land, unit m/s
Dimensions 10~12: AngEular[3]; % Attitude Euler angle estimated by Pixhawk, unit radians

4.1.5 Sample Time

Sample Time sampling time, this time should correspond to Simulink simulation time.

4.1.6 RflySwarmAPI module advantage

High efficiency: directly implemented in C/C++ language, more efficient than M language
Small operation: Simulink's built-in UDP module input and output are high-dimensional (several hundred dimensions) unit8 vectors. When the number of aircraft increases, the entire project dimension expands sharply; while the S function method directly outputs double-type position, speed and other data, the dimension Low (several dimensions), small amount of calculation
Low latency: In order to prevent data loss, the UDP module that comes with Simulink will save all received data in the cache and call it in sequence. In this way, when the external program sending frequency is greater than the running frequency of Simulink, a large delay will occur; while writing The S function method can avoid this problem
More reliable: The UDP module that comes with Simulink reads one data per simulation step. If the external program sending frequency is less than the Simulink running frequency, Simulink does not read the data and will output 0, resulting in an operation error; while writing the S function yourself will This problem can be avoided.
Strong scalability: S function can be easily expanded to other communication protocols such as serial port, Tcp, shared memory, MAVLink, etc.

4.2 RflyUdpFast.cpp interface

4.2.1 CreateStructure function

Parameters: SimStruct *S, int indx
The function is to store a dynamically allocated memory pointer of type SInfo in the input structure. This pointer can be accessed or modified via index indx.

4.2.2 mdlStart function

Parameters: SimStruct *S
The main purpose of the function is to create a UDP server for each aircraft and start it. The function will call the StartUDPServer() function to create and start the UDP server. Create a new structure for each aircraft through the CreateStructure() function. For non-Linux systems, this code activates the Winsock DLL. Winsock DLL is a library that provides network communication functions on Windows systems.

4.2.3 mdlOutputs function

Parameters: SimStruct *S, int_T tid
The function will obtain the UDP mode and update the aircraft information according to different modes.

4.2.4 mdlUpdate function

Parameters: SimStruct *S, int_T tid
Function used to update aircraft status information.

4.2.5 mdlTerminate()

Function used to terminate the simulation.

4.2.6 StartUDPServer()

Parameters: SimStruct *S, SInfo *info, int index
Function used to initialize and start a UPD server.

4.2.7 mdlInitializeSizes()

The function is used to initialize the Simulink custom module of MATLAB. Its main function is to set the model parameters, including the number of input and output ports, data type and width, etc.

4.2.8 netServerInit()

The main function of the function is to create a UDP server on the specified port and return the socket.

4.2.9 mdlCheckParameters()

The function is used to check whether the incoming parameters meet the requirements. Parameter values, which are port number, number of vehicles and UDP mode, are converted into integer format.

4.3 Interface usage example

4.3.1 Single machine control example

A. UDP FullData mode

B. UDP SimpleData mode

C. UDP UltraSimpleData mode

4.3.2 Multi-machine control example

A. UDP FullData mode

B. UDP SimpleData mode

C. UDP UltraSimpleData mode

4.3.3 External machine control example

A. Notes on bat script writing

B. Broadcast mode

C. Use IP mode

4.3.4 Formation-free cluster experiment with automatic collision avoidance

4.3.5 Data recording and analysis

5. PX4MavCtrlV4 interface

5.1 PX4MavCtrler

5.1.1 UAV parameters illustrate

5.1.2 Simplify model function illustrate

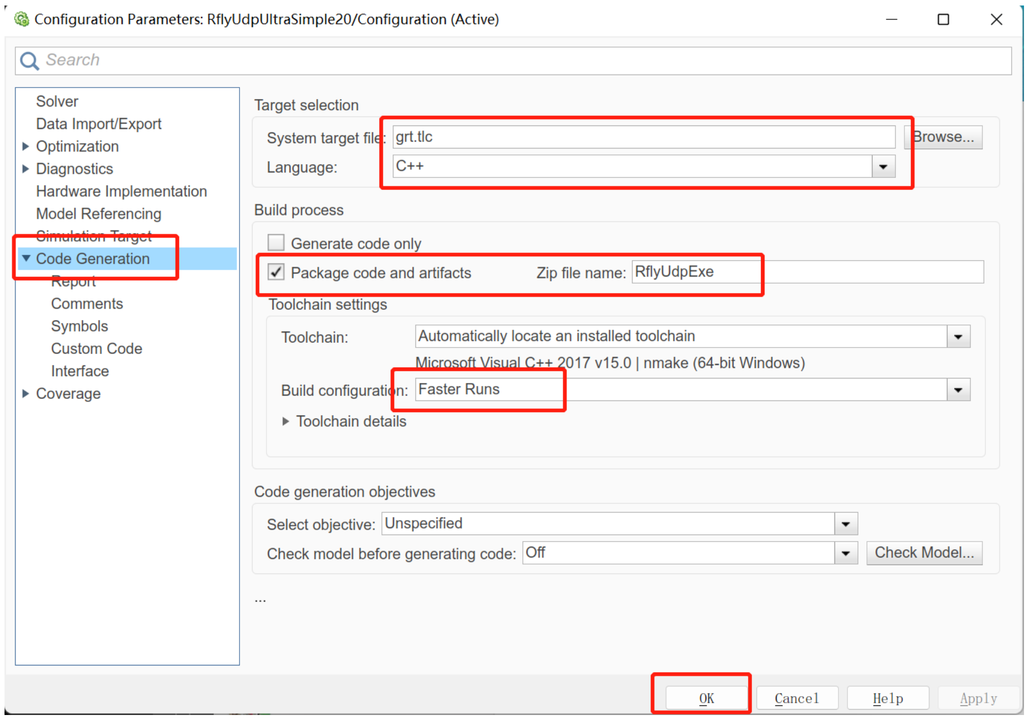
6. Simulink cluster algorithm compilation EXE

6.1 General introduction

M A T L A B itself will occupy a large amount of CPU and memory resources. When running complex Simulink control programs, on the one hand, the amount of calculation is too large, causing the algorithm to run slowly and unable to meet real-time requirements (running in Simulink for 1 second is greater than the real clock 1 second), so it cannot Control a swarm of aircraft in a simulated system (or real system) in real time. Secondly, if Simulink occupies a large amount of computing resources during simulation, it will lead to less allocation of computing resources for RflySim3D and CopterSim, resulting in poor aircraft simulation, violent aircraft shaking and even crashing. After the Simulink controller is compiled into exe, the algorithm can be run without MATLAB, and it is a binary executable file itself, which has very high running efficiency and can ensure real-time control even for large-scale control algorithms.

6.2 Simulink Configuration method

Click the Simulink settings button and set as shown below. For versions after 2019b, you need to enter the APP interface and select Simulink Coder in CODE GENERATION to pop up the code generation toolbar.


Configuration Parameters: RflyUdpUltraSimple20/Configuration (Active)

Target selection
System target file: grt.tlc
Language: C++

Build process
Generate code only: ☐
Package code and artifacts: ✓ Zip file name: RflyUdpExe

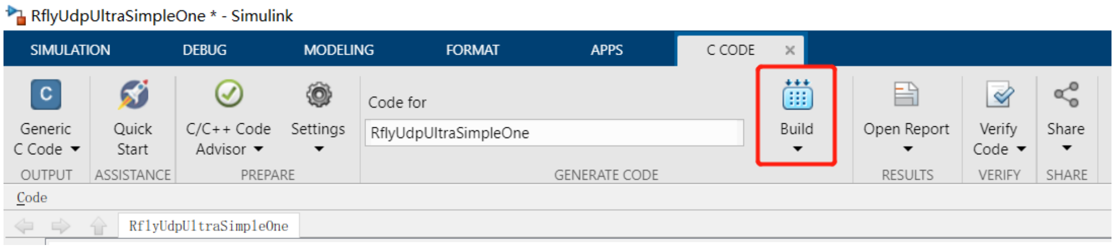
Toolchain settings
Toolchain: Automatically locate an installed toolchain
Microsoft Visual C++ 2017 v15.0 | nmake (64-bit Windows)
Build configuration: Faster Runs

Code generation objectives
Select objective: Unspecified
Check model before generating code: Off

6.3 Steps

Select the slx file to generate exe and ensure that GenerateSwarmExe.p, RflyUdpFast.cpp and RflyUdpFast. mexw64 are copied to the same directory.

Open the slx file, generate the toolbar and click the Build button as shown below to generate the code.



After the compilation is completed, we will get a successful compilation prompt in the "Diagnostic Viewer" (please ignore the yellow warning). We can get a "RflyUdpExe.zip" compressed package file, which contains all the generated code and will be used for us. The exe file is subsequently generated.

Note: An exe file was generated when compiling the command in the previous step. Double-click it to run it, but it does not meet the requirements of real-time control (always runs at the fastest speed and is not synchronized with the real clock), so we need to run our script to generate real-time control. exe file.

Right-click the GenerateSwarmExe.p file and click "Run." Or enter "GenerateSwarmExe" directly on the MATLAB command line to get the .exe file corresponding to the Simulink file name as shown below.

6.4 Experimental example

For experimental cases, see RflySimSwarm