Contents

1、 RflySim3D Architecture and basic functions of

The position of RflySim3D in the RflySim simulation platform is shown in the following figure:
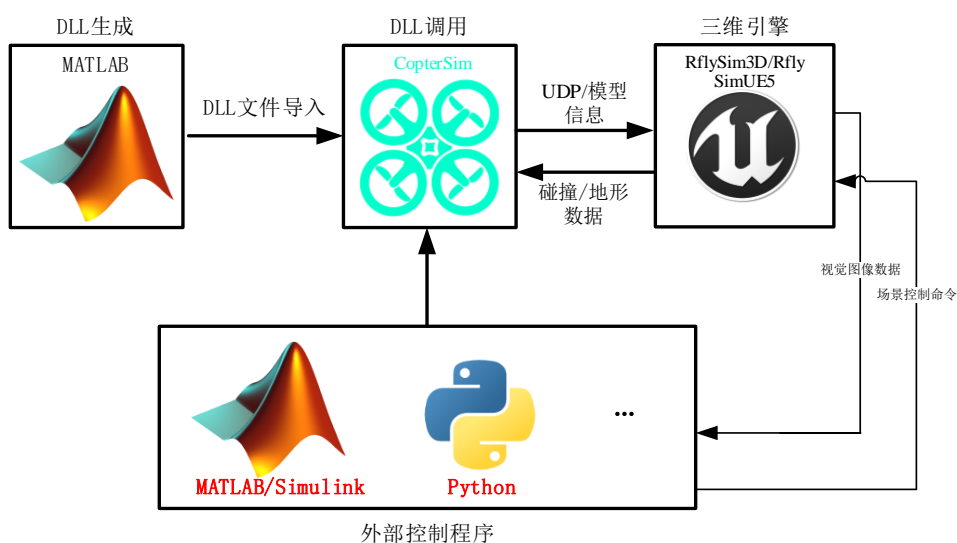
CopterSim will calculate the current state of UAV (mainly position and attitude data) based on the motor control data from Pixhawk (or PX4 SITL), and then send these data to RflySim3D, which will apply these data to the corresponding UAV in the scene. So that we can see the state of the UAV more intuitively.

RflySim3D uses UDP communication and can accept some external commands, such as switching scenes, creating UAVs, and opening the built-in physical collision of UE. The details of the commands will be described in the introduction to the interface and usage of RflySim3D. In summary, RflySim3D can accept UDP commands from CopterSim, Python, Simulink, and return collision/ terrain data as well as visual image data.

RflySim3D also supports some configuration through XML files, mainly using XML to configure the configuration of UAV (four-rotor, six-rotor, fixed-wing, etc.), the priority of the model in the list, the name of the aircraft, the initial position and attitude of the aircraft, the initial position and attitude of each actuator (usually rotor). You can also define the position of the camera, as well as some obstacle components (such as pillars, rings), and so on.

2. UE4/UE5 Development Environment

Corresponding platform routine: UE Installation and Blueprint Programming Introductory Lab

U E4

Visual Studio Configuration

Unreal Engine version	Visual Studio version
4.25 or later	VS 2019 (Default)
4.22 or later	VS 2017 / VS 2019
4.15 or later	VS 2017
4.10 - 4.14	VS 2015

Unreal Engine version	Visual Studio version
4.2 - 4.9	VS 2013

See for details Set up the Unreal Engine Visual Studio

U E4 Installation and Editor interface

The installation steps are detailed in Install the Unreal Engine

The editor interface is shown in Unreal editor interface

U E5

Visual Studio Configuration

Unreal Engine version	VS 2019 version	VS 2022 version
5.3	16.11.5 or later	17.4 or later, 17.6 recommended (default)
5.2	16.11.5 or later	17.4 or later (Default)
5.1	16.11.5 or later (Default)	17.4 or later

See for details For the Unreal Engine C++ Project settings Visual Studio Development Environment

UE 5 Installation and Editor interface

The installation steps are detailed in Install the Unreal Engine

The editor interface is shown in Unreal editor interface

3. Scenario development depends on software

Demonstration effect of corresponding platform routine 3.1: 0. ApiExps/e0_DevToolsUsage/5.SketchUpUsage/Readme.pdf

Demonstration effect of corresponding platform routine 3.2: 0. ApiExps/e0_DevToolsUsage/2.3dsMaxUsage/Readme.pdf

Demonstration effect of corresponding platform routine 3.3: 0. ApiExps/e0_DevToo

3.1 SketchUp

SketchUp is a popular 3D modeling software developed by Trimble Navigation. It is known f or its easy-to-use interface and powerful modeling tools. SketchUp is suitable for a variety of appli cations, including architectural design, interior design, landscape design and more. It allows users t o quickly create and edit 3D models, and provides a rich library of plug-ins to extend functionality.
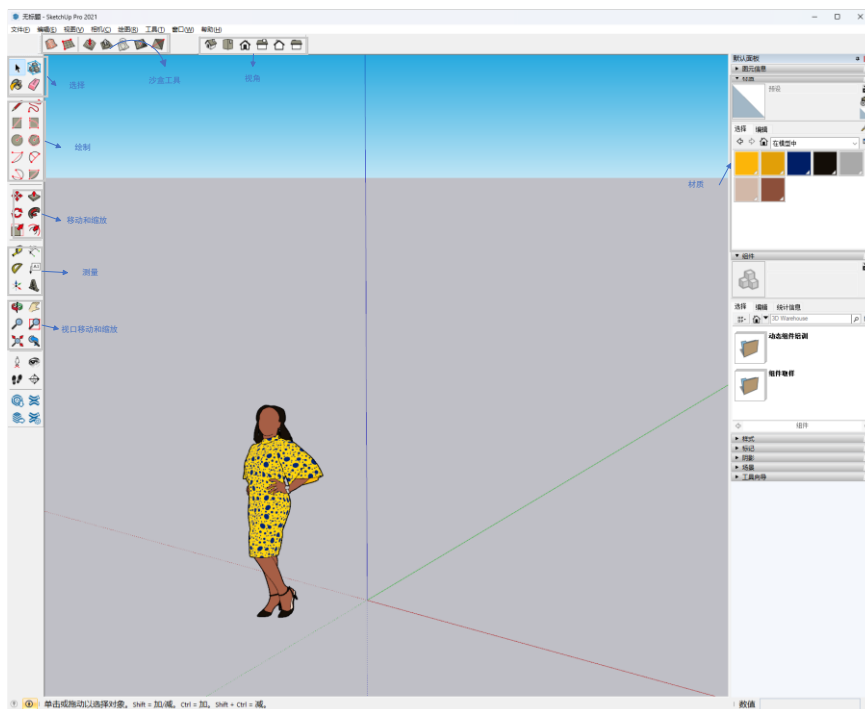
The detailed installation and use tutorial is as follows:

 Download SketchUp | 3D Modeling Software Free Trial : https://www.sketchup.com/zh-CN /try-sketchup#for-personal

 SketchUp Campus ：https://learn.sketchup.com/collections

 [Collection] Very detailed novice tutorial! SketchUp Complete set of excellent introductory co urses for beginners _ Bilibili _bilibili ：https://www.bilibili.com/video/BV1Tb411E7Co/?spm_id _from=333.337.search-card.all.click&vd_source=49928ee3b2e585b60a2ebc fbd79ca829

3.2 3DsMax

3ds Max is a powerful 3D modeling and animation software developed by Autodesk. It provid es sophisticated modeling tools that allow users to create anything from simple cubes to complex c haracters and scenes.
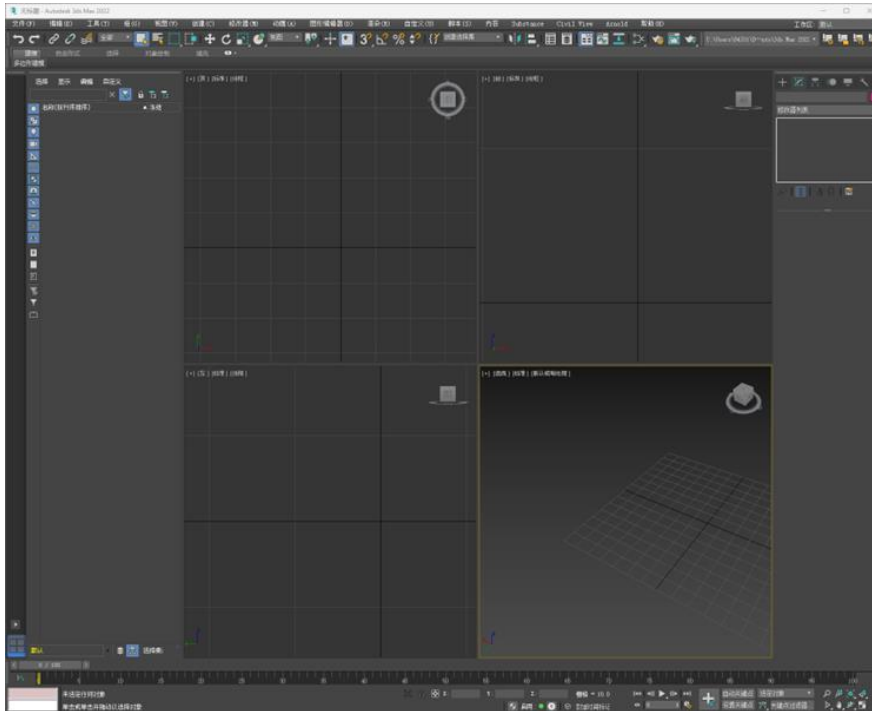
Detailed installation and use methods are as follows:

 3dsMax Official website https://www.autodesk.com.cn/products/3ds-max/overview?term=1-YEAR&tab=subscription

 3Ds Max Tutorial: Full Beginner Crash Course (New for 2022) | RedefineFX \u2014 YouTube： https://www.youtube.com/watch?v=P-lWkbDXxVU

3.3   T winmotion

Twinmotion is a real-time visualization software based on Unreal Engine, which provides fast rendering and high-quality rendering. It has a large number of built-in architectural scene materials and materials, enabling users to quickly create realistic scenes. Twinmotion has an intuitive and easy-to-use user interface that can be used to enhance scene details, change materials, and add vegetation. In addition, Twinmotion supports integration with other software and engines, such as Revit, SketchUp, and Unreal Engine. Although Twinmotion provides some basic 3D elements (such as furniture, plants, people, weather, light, etc.) And allows users to use these elements for scene assembly and decoration, it is not a dedicated 3D modeling tool in nature. When using custom scenes in the RflySim platform, you can make full use of the built-in materials and objects provided by Twinmotion to make the existing scenes richer and more realistic.

Interface of 2022 version:

2023 version of the interface:

Detailed installation and use methods are as follows:
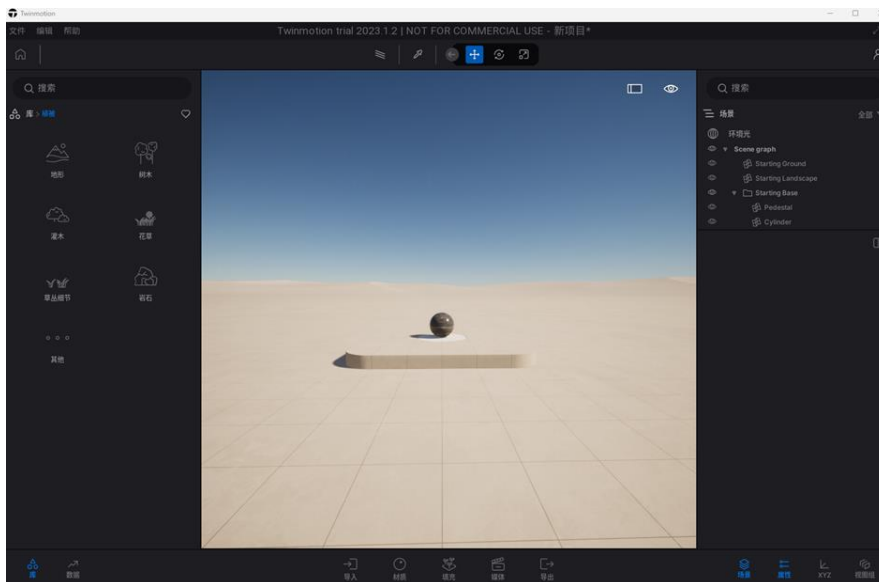
https://www.twinmotion.com/learning-resources

【 Twinmotion 】than UE5 Simple, 10 Minutes to create a beach with a brand new rendering software _ Bilibili _ bilibili ：https://www.bilibili.com/video/BV1NY411w7MD/?spm_id_from=333.337.search-card.all.click&vd_source=49928ee3b2e585b60a2ebc fbd79ca829

3.4 C esium

Cesium is an open source platform for creating and presenting 3D geographic information. It is based on WebGL technology and can realize high-performance 3D earth rendering and geographic data visualization in modern web browsers. Cesium has powerful geospatial analysis and visualization functions, which can be used to realize virtual earth, satellite image browsing, geographic information system and other applications, providing users with interactive and scalable geographic i

nformation display solutions.

The detailed usage method is as follows:

 Cesium for Unreal – Cesium ： https://cesium.com/learn/unreal/

 Cesium for Unreal Quickstart – Cesium ： https://cesium.com/learn/unreal/unreal-quickstart/

https://www.bilibili.com/video/BV16p4y1t7Mc

3.5 Python

Python was designed to emphasize code readability and to allow programmers to express con cepts in a small amount of code. It provides a rich standard library and a wide variety of framewor ks, libraries, and tools; supports object-oriented programming, but also allows procedural and func tional programming, and supports a variety of operating systems. It is widely used in the field of m achine learning and artificial intelligence, such as TensorFlow, PyTorch, and can be used to write v arious automation scripts. The python 3.8.1 environment is automatically installed during the Rfly Sim platform installation.

The detailed usage method is as follows:

 Python Basic course https://www.runoob.com/python/python-tutorial.html

3.6   S imulink

Simulink is a graphics-based simulation and model design environment developed by MathW orks, which is often used with MATLAB.

The detailed usage method is as follows:

 Simulink official tutorial https://ww2.mathworks.cn/products/simulink/getting-started.html

4. Shortcut key control interface

Demonstration effect 4 of the corresponding platform routine: Demonstration of us ing shortcut key interface

In order to improve user interaction and operation convenience, RflySim3D/RflySimUE5 also has a series of built-in shortcut keys on the basis of built-in global commands, some of which will interact with CopterSim.

These shortcuts can be used to manage the simulation environment and the aircraft, such as p opping up the help menu, showing or hiding information, switching between the map and the aircr aft, activating the collision engine, showing or hiding the minimap, etc. (F1, ESC, S, H, D, M, B, C, P, L). And you can switch to the specified map, focus on the specified aircraft, modify the mode l 3D style (M + number *, B + number *, C + number *).

F1 (Help):

Help menu prompt pops up;

ESC (Clear Copter):

Clear all aircraft (close all CopterSim first)

S (explicit/implicit CopterID):

Show/hide aircraft ID;

H (Hide/Show all on-screen text):

Hide/show all screen text;

D (explicit/implicit Copter data):

Display/hide current aircraft data;

M (switch map):

Switch the map (close all CopterSim first);

M + number * (switch to map No. *):

Switch to Map No. *;

B (Toggle focus Copter):

Witching visual angle focus among different aircrafts;

B + Number * (Focus on Copter No. *):

Switch to aircraft number *

C (Toggle current Copter style):

Toggle the current aircraft (newly created) 3D style;

C + Number * (switch to 3D style no. *):

Switch to 3D style No. *;

CTRL + C （Toggle all Copter Style）:

Toggle All Aircraft 3D Style

P (active Collision engine） Limited to personal premium version or above：

Turn on the physical collision engine (it will collide with scene objects and the ground. This function only supports the full version)

L (explicit/implicit minimap):

Show/hide the minimap (appears in the lower right corner by default, double click on the minimap to show more modifiable content)

After starting CopterSim, the shortcut keys are often used for visual angle control, aircraft track recording, etc. (V, N, mouse operation, T), as well as switching to the specified visual angle, specifying the thickness of the running track, creating an obstacle with a specified number, and switching to the specified communication mode (V + number *, N + number *, T + P + number *).

V (switch following view angle):

Angle of view switch on the plane, 0: follow angle of view, 1: front view camera, 2: right view camera, etc ⋯ ;

V + number * （Switch to No. Follow Perspective）：

Switch to the * perspective

N （Switch God Perspective）：

Switch to the perspective of the plane God.

N + number * （Switch to No. God Perspective）：

Switch to the * God perspective.
0: Follow the aircraft view (do not change the view angle with the aircraft attitude) 1: Fix the ground view and always look to the current aircraft, 2: Fix the ground view to the north, 3: Fixed the ground to the south, etc. ;

Left mouse button press drag (adjust Angle of view）：

And switch that angle of the visual angle;

Press and drag the right mouse button (adjust the vertical position of the viewing angle):

Switch the vertical YZ position of the viewing angle

Mouse wheel (to adjust the landscape position of the viewing angle):

Lateral X position at which that view angle is switch

CTRL + mouse wheel (adjust all Copter sizes):

Zoom all aircraft dimensions (easy to observe in case of multiple aircraft);

ALT + mouse wheel (adjust the current view angle Copter size):

Zoom current view aircraft dimension

T (on/off Copter trace):

Turn on or off the aircraft track recording function

T + number * (change track thickness to *):

Open Change track thickness to * Number

Double-click the mouse（Display hit Point Information）:

Display the position, size, object and other information of the hit point 。 Note: Press the N key immediately after double-clicking to quickly switch the perspective to the double-clicking position for object creation

O + number * (object generated C lass ID as "*"):

An object (obstacle) with a style ID of "*" is generated at the mouse double click.

P + number （Switch communication mode）Limited to personal premium version or above :

After the P mode is enabled, RflvSim3D will transmit the obstacle information back to each CopterSim at a high speed 30100 Series port . 0, 1 and 2 respectively correspond to the communication modes of local sending, LAN sending and LAN minimalist sending (sending only in case of collision).

- In P0 mode (press the P + 0 key, which is triggered by pressing the P key by default), RfvSim3D will transmit the ambient distance data of each aircraft to all CopterSims on this computer (not on the LAN) at high frequency.
- In P1 mode, RflvSim3D will transmit the distance data around each aircraft to each CopterSim in the LAN at high frequency (by specifying IP and port to improv

e efficiency).

● In P2 mode, RfvSim3D will only send the obstacle data to CopterSim in the LAN (by specifying IP and port) during the aircraft collision (and within 1 second), thus optimizing the communication from the data frequency and the target IP number.

I + Number（Switch LAN shield status ？This feature is only available in the full version. Above）：

● In the I-1 mode (press the I + -1 key, and press the I key by default to trigger this mode), the current LAN shielding state will be switched. These two commands cause the LAN to become masked if it is currently unmasked, and vice versa.

● In I0 mode, LAN masking is enabled. When this command is executed, all instances of RflySim3D on the LAN will not receive or send data.

● In I1 mode, the LAN is unmasked, and after executing this command, the RflySim3D instance on the LAN will resume normal communication.

5. Command line control interface

Corresponding platform routine demonstration effect 5: Command line interface use effect demonstration

RflySim3D has some built-in global commands, which can complete most of the functions related to RflySim3D. Press the tilde (~) of the keyboard in the program window of RFlySim3D to open the console terminal, where some built-in commands of the UE engine itself can be triggered. You can also trigger commands built into the RflySim3D platform. These commands are equivalent to global functions, and the specific commands are as follows:

5.1 RflySim Command interface

RflyShowTextTime（Display text)

Function interpretation

Make the UE display txt for time seconds. The method of use is as follows

```
RflyShowTextTime(String txt, float time);
```

RflyLoad3DFile (execute TXT script)

Function interpretation

Load and execute the TXT script file under the path. The method of use is as follows

```
RflyLoad3DFile(FString FileName);
```

RflySim3D supports txt files as scripts, and the commands described here can be written as txt scripts.

An example of operation

For example, let's create a "Test. Txt" "file on Drive D, enter the console command on line, then open RflySim3D and enter the console command:

```
RflyLoad3DFile "D:/Test.txt"
```

RflySetIDLabel (display at setting CopterID label) This function is only supported above the personal premium version.

Function interpretation

Set the display content of the top ID position of a Copter (it will replace the default Copter ID displayed by pressing the S key). The use method is as follows:

```
RflySetIDLabel(int CopterID,FString Text, FString colorStr, float size);
```

Parameter interpretation

- CopterID (int): The ID of the Copter whose ID location content is to be set.
- Text (FString): The text content to be displayed in the ID position.
- ColorStr (FString): The color of the text. You can specify this with a string that represents a color value, such as "FF0000" for red and "00FF00 " for green.
- Size (float): The size or scale of the text.

An example of operation

When you use this function, you need to pass the appropriate values to the parameters. Examples of operation commands are as follows:

```
RflySetIDLabel 1000 "Hi, here's the test string" FFFF00 20
```

Modify the content, color and font size of the label on the Copter with ID 1000. This color is a hexadecimal RGB code. FFFF00 means that the value of red and green is 255, and the value of blue is 0. When they are mixed, they become yellow.

RflySetMs gLabel (displayed below the set CopterID label) This function is only supported above the personal premium version.

Function interpretation

Set the content displayed by the Message on the top of a Copter. The usage method is as follows:

```
RflySetMsgLabel(int CopterID, FString Text, FString colorStr, float size, float time, int
flag);
```

Parameter interpretation

- CopterID (int): The ID of the Copter to set the Message display content.
- Text (FString): Text content to be displayed in Message.
- ColorStr (FString): The color of the text. You can specify this with a string that represents a color value, such as "FF0000" for red and "00FF00" for green.
- Size (float): The size of the text.
- Time (float): The display time of the text, in seconds. At the end of the specified time, the message disappears.
- Flag (int): The number of message lines for the operation. There are 5 message labels in total. When flag < 0, it means a new message is added.

An example of operation

```
RflySetMsgLabel 1000 "Hi, here's the test string2" FF00FF 20 5 -1
```

A purple text string appears below the ID label. In this way, using the RflySetMsgLabel function, you can set the Message display content of the Copter. You can specify different parameters to create multiple lines of messages, and each line of messages can be displayed and disappear within a specified time. By setting different flag values, you can manipulate different numbers of lines to update or add new message content.

RflyChangeMapbyID (According to I D Toggle Map)

Function interpretation

Switch RflySim3D scene map according to the ID of the map. The use method is as follows:

```
RflyChangeMapbyID(int id)
```

An example of operation

For example, in a console terminal

```
RflyChangeMapbyID 5;
```

Quickly switch to map 5.

RflyChangeMapbyName （Switch map by name）

Function interpretation

Switch the RflySim3D scene map according to the map name. The use method is as follows:

```
RflyChangeMapbyName(String txt)
```

An example of operation

For example, in a console terminal

```
RflyChangeMapbyName 3Ddisplay
```

Switch to the map 3D Display.

RflyCesiumOriPos （Modify the origin of the map）

Function interpretation

Modify the origin position of the Cesium map (Cesium global scene is required), allowing the latitude and longitude coordinates of the origin of the map to be set. The use method is as follows:

```
RflyCesiumOriPos(double lat, double lon, double Alt)
```

Parameter interpretation

- Lat (double): The latitude value of the map origin.
- Lon (double): Longitude value of the map origin.
- Alt (double): The elevation (height) value of the map origin.

An example of operation

```
RflyCesiumOriPos 39.9042 116.4074 50
```

This will set the location of the map origin to the latitude and longitude coordinates of Beijing.

Notice

- Before using this function, you need to go into a map that uses the Cesium plug-in, and preferably be in a networked state to load the online map image.
- The latitude and longitude ranges are [-180,180] and [-90,90]. Make sure that the latitude and longitude values provided are within the valid range.

RflyCameraPosAng A dd （Offset Camera）

Function interpretation

Add an offset value to the position and angle of the camera (adjust based on the current

position and angle of the camera) as follows:

```
RflyCameraPosAngAdd(float x, float y, float z,float roll,float pitch,float yaw)
```

Parameter interpretation

- X (float): The offset value of the camera in the x-axis direction.

- Y (float): The offset value of the camera in the y-axis direction.

- Z (float): The offset value of the camera in the z-axis direction.

- Roll (float): The offset angle (roll) of the camera rotation around the x-axis.

- Pitch (float): The offset angle (pitch) of the camera rotation around the y-axis.

- Yaw (float): The offset angle of the camera rotation around the z-axis (yaw).

Operation example

```
RflyCameraPosAngAdd 0 0 -10 0 -30 0
```

Move the camera up 10 meters and look down 30 degrees

RflyCameraPosAng (reset camera)

Function interpretation

Directly set the position and angle of the camera as follows:

```
RflyCameraPosAng(float x, float y, float z, float roll, float pitch, float yaw);
```

Parameter interpretation

- X (float): X-axis position of the camera in the world coordinate system.

- Y (float): Y-axis position of the camera in the world coordinate system.

- Z (float): The Z axis position of the camera in the world coordinate system.

- Roll (float): The angle of rotation (roll) of the camera around the X axis.

- Pitch (float): The angle of rotation (pitch) of the camera around the Y axis.

- Yaw (float): The angle of rotation of the camera around the Z axis (yaw).

Operation example

```
RflyCameraPosAng 0 0 1000 0 0 0
```

Set the camera at a height of 1000 units above the origin of the world coordinate system at the default straight ahead angle.

RflyCameraFovDegrees (Set View)

Function interpretation

Set the FOV angle of the camera (Field of View) as follows:

```
RflyCameraFovDegrees(float degrees)
```

Perspective projection (P)
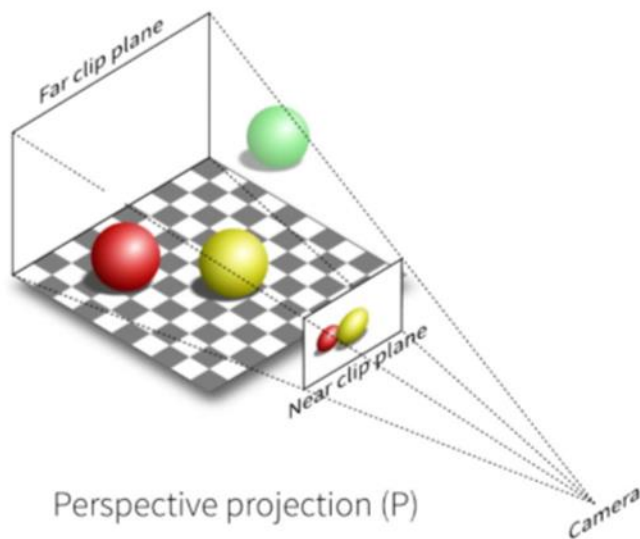
This function modifies the horizontal field of view of the camera, which is the angle between the camera and the leftmost and rightmost midpoints on the top image plane. The default angle is typically 90 °.

An example of operation

Change the angle to 120 °. The operation command is as follows:

```
"RflyCameraFovDegrees 120
```

RflyChange3Dmodel （Modify Copter Style）

Function interpretation

To modify the model style of a drone, use the following

```
RflyChange3DModel(int CopterID, int veTypes=0)
```

This command modifies the 3D model style of the specified UAV

Parameter interpretation

- **CopterID: The ID of the drone, used to specify the drone to be modified.**
- **VeTypes: The number of the 3D model style. This parameter is optional and defaults to 0. You can specify different numbers to correspond to different model styles.**

An example of operation

For example, the ID of the drone here is 1000, so if you want to change its style to No.1, you can use the command

```
RflyChange3DModel 1000 1
```

RflyChangeVehicleSize（Resize Copter）

Function interpretation

Modify the zoom size of a UAV as follows:

```
RflyChangeVehicleSize(int CopterID, float size=0)
```

Parameter interpretation

- CopterID: The ID of the drone, used to specify the drone to be modified.

- Size: The multiple of the zoom. This parameter is optional and defaults to 0, indicating no scaling. Make the drone bigger by specifying a positive multiple and smaller by specifying a multiple less than 1.

An example of operation

Suppose you want to increase the size of a drone with a drone ID of 1000 by a factor of 10. You can use the following commands:

```
RflyChangeVehicleSize(1000, 10);
```

RflyMoveVehiclePosAng（Offset C opter）

Function interpretation

Set an offset value for the position and angle of the UAV, and isFitGround sets whether the UAV adapts to the ground. The usage is as follows:

```
RflyMoveVehiclePosAng(int CopterID, int isFitGround, float x, float y, float z, float roll, float pitch, float yaw)
```

Parameter interpretation

- CopterID: The ID of the drone that specifies the drone for which you want to set the offset.

- IsFitGround: whether to adapt to the ground height. If set to 1, the Z coordinate of the drone is adjusted according to the height of the terrain; if set to 0, the Z coordinate of the drone remains unchanged.

- X, y, Z: three axis components of position offset. By setting these parameters, you can move the drone a specified distance in three-dimensional space along the X, y, and Z axes.

- Roll, pitch, yaw: Euler angle components of the rotation angle. By setting these parameters, you can rotate the drone through a specified angle about its own X, y, and Z axes.

An example of operation

```
RflyMoveVehiclePosAng 1000 1 -10 -10 -10 0 -20 0
```

The aircraft can be moved (-10, -10, -10) meters and the pitch angle is -20 °. If isFitGround is set to 1, the Z coordinate of the drone is determined based on the height of the terrain.

RflySetVehiclePosAng （Reset Copter）

Function interpretation

Set the position and angle of the UAV as follows:

```
RflySetVehiclePosAng(int CopterID, int isFitGround, float x, float y, float z, float roll,
float pitch, float yaw)
```

This command is similar to the RflyMoveVehiclePosAng function, but instead of giving an increment, it sets the position of the drone directly.

Parameter interpretation

- CopterID: The ID of the drone, which is used to specify the drone whose position and angle you want to set.
- IsFitGround: whether to adapt to the ground height. If set to 1, the Z coordinate of the drone is adjusted according to the height of the terrain; if set to 0, the Z coordinate of the drone remains unchanged.
- X, y, Z: three components of position coordinates. By setting these parameters, the UAV can be moved directly to the specified location.
- Roll, pitch, yaw: Euler angle components of the rotation angle. By setting these parameters, you can directly set the UAV to the specified angle.

An example of operation

Let's say you want to directly set the drone with drone ID 1000 to position (-10, -10, -10) meters and set it to a pitch angle of -20 °. You can use the following commands:

```
RflySetVehiclePosAng(1000, 1, -10, -10, -10, 0, -20, 0);
```

RflySetActuator PWMs (Trigger Blueprint Interface, this function is limited to Personal Advanced Edition and above)

Function interpretation

Pass in 8 values and trigger the interface function of the target UAV's blueprint

```
RflySetActuatorPWMs(int CopterID, float pwm1, float pwm2, float pwm3, float pwm4, float p
wm5, float pwm6, float pwm7, float pwm8);
```

In fact, it is to directly set the 8-bit motor data of the UAV. Generally, the data is sent by UDP, but it can be set directly with this command.

Parameter interpretation

- CopterID: The ID of the UAV, which is used to specify the UAV to set the motor data.
- PWM1, PWM2, PWM3, PWM4, PWM5, PWM6, PWM7, PWM8: 8 motor data. This data controls the rotational speed or other behavior of the UAV's individual rotors or other actuators. Refer to the settings of the relevant blueprint model for the specific meaning.

An example of operation

Suppose you have created a drone with ID 1000 through RflySim3D and want to start turning the propeller of the drone, where the first four motor data controls the rotation speed of 10 units per second.

```
RflySetActuatorPWMs(1000, 10, 10, 10, 10, 0, 0, 0, 0);
```

RflySetActuator PWMsExt (Trigger extension blueprint interface, this function is limited to personal advanced version and above)

Function interpretation

It is used to pass in 16 values and trigger the blueprint interface function of the target UAV. The usage is as follows

```
RflySetActuatorPWMsExt(int CopterID, float pwm9, float pwm10, float pwm11, float pwm12, float pwm13, float pwm14, float pwm15, float pwm16, float pwm17, float pwm18, float pwm19, float pwm20, float pwm21, float pwm22, float pwm23, float pwm24)
```

Explanation of parameters:

- CopterID: The ID of the UAV, which is used to specify the target UAV to call the blueprint interface function.
- Pwm9-pwm 24:16 parameters, as the blueprint interface function for data transfer to the target UAV. The specific meaning and role depend on the blueprint design of the target UAV.

Notice

This function requires a full version of RflySim to use, and the target drone must be an object of a custom Blueprint class. This 16-dimensional data is passed directly to the "ActuatorInputsExt" function in the target UAV's blueprint class, and RflySim3D does not process it by default.

By calling the "RflySetActuatorPWMsExt" function, you can pass various control data to the blueprint of the target UAV to achieve various control effects. This function provides more parameters for flexible control and interaction.

RflyDelVehicles (Clear Copter)

Function interpretation

Delete the specified Copter, using the following

```
RflyDelVehicles(FString CopterIDList);
```

This function can delete the UAV currently present in the scene based on the ID. RflySim3D does not actively delete UAVs in the scene, even if they have not received their data. (This is why the drones did not disappear from the scene when CopterSim was turned off.) RflySim3D must be explicitly asked to remove these drones.

Parameter interpretation

- CopterIDList: A parameter of type FString that specifies the list of IDs of the drones to be removed. Multiple IDs are separated by commas.

An example of operation

Suppose there are two UAVs with IDs 1000 and 1001 in the scenario. If you want to delete these two drones, you can use the following command:

```
RflyDelVehicles("1000,1001");
```

Notice

Before deleting a drone, you should ensure that you stop sending data from these drones to RflySim3D to avoid situations that could cause the drone to be recreated.

RflyScanT errainH

Function interpretation

To obtain the terrain data, use the following

```
RflyScanTerrainH(float xLeftBottom(m), float yLeftBottom(m), float xRightTop(m), float yRightTop(m), float scanHeight(m), float scanInterval(m))
```

This function scans the 3D terrain and generates a PNG height map and txt, which the CopterSim program will need to know what terrains the UE has and their elevations. But this function needs to know the size and height of the terrain and the accuracy of the scan.

Parameter interpretation

- XLeftBottom, yLeftBottom: coordinates of the lower left corner of the terrain, in meters. This specifies the starting location for the terrain scan.
- XRightTop, yRightTop: coordinates of the upper right corner of the terrain, in meters. This specifies where the terrain sweep ends.
- ScanHeight: The maximum height limit of the terrain, in meters. This parameter is used to specify the height range of the terrain for generating height maps and t

ext files.

- ScanInterval: The scan sampling interval, in meters. This parameter determines the accuracy with which the terrain is sampled during the scan, that is, the distance between the sampling points.

An example of operation

For example, you can use the following command to scan the terrain:

```
RflyScanTerrainH(-1000, -1000, 1000, 1000, 200, 1);
```

This command specifies the coordinates of the lower left corner of the terrain as (-1000, -1000) meters and the upper right corner as (1000, 1000) meters. Terrain height is limited to a maximum of 200 meters. The scanning interval is sampling every 1 meter.

After executing this command, RflySim3D will scan the terrain within the specified range and generate a PNG image file of the height map and a text file (TXT format). You can find 2 files of the same name for this map at "\PX4PSP\RflySim3D".

RflyReqVehicleData (activation data return)

Function interpretation

Activates the data postback mode, in which once RflySim3D receives an update from Copter, it sends the relevant data out again. The usage is as follows

```
RflyReqVehicleData(FString isEnable);
```

Parameter interpretation

If the parameter isEnable passed in is not 0, RflySim3D starts sending the Copter's data.

By default, RflySim3D only receives information from an external Copter, and the structure of the received UDP data is similar to the following structure:

```
struct SOut2SimulatorSimple {
int checkSum;
int copterID;
int vehicleType;
float MotorRPMSMean;
float PosE[3];
float AngEuler[3];
}
```

Member variables and types of the struct:

- Check Sum: Integer variable, representing the check code of the data packet.
- CopterID: An integer variable representing the ID number of the aircraft.
- VehicleType: Integer variable, indicating the type of aircraft.
- MotorRPMSMean: a floating point variable representing the average value of t

he motor speed.

- PosE: An array of three floating-point variables representing the position of the aircraft. The three elements are X, Y, and Z coordinates.
- AngEuler: An array of three floating-point variables representing the Euler angles of the aircraft. The three elements are Roll, Pitch, and Yaw.

By default, RflySim3D will not send the position and attitude information of the UAV in the 3D scene, but if necessary, the command "RflyReqVehicleData 1" can be called. When RflySim3D receives this command, it goes into "data postback mode" and starts sending the requested Copter's data using UDP. In this mode, once RflySim3D receives the updated data from Copter, it will send the relevant data out again.

The specific data structure to be sent is as follows reqVeCrashData , including the following fields:

- Check sum: packet checksum (fixed value 1234567897)
- CopterID: ID number of the current aircraft
- VehicleType: The style of the current aircraft
- CrashType: Collision object type, -2 for ground, -1 for scene static object, 0 for no collision, 1 and above for ID number of the collided aircraft
- RunnedTime: Time stamp of the current aircraft
- VelE: Current aircraft speed
- PosE: Current position of the aircraft
- CrashPos: Coordinates of the collision point
- TargetPos: Center coordinates of the object touched
- AngEuler: Euler angle of the current aircraft
- MotorRPMS: Current aircraft motor speed
- Ray: front, rear, left, right, up and down scan lines of the aircraft
- CrashedName: The name of the touched object

An example of operation

```
RflyReqVehicleData 1
```

RflySetPosScale（Global Zoom）

Function interpretation

Scaling of the global position is used as follows

```
RflySetPosScale(float scale);
```

Parameter interpretation

This function affects the position information of the Copter passed to RflySim3D via UDP. The default is scale == 1.

RflySim3D The received structure is as follows:

```
struct SOut2SimulatorSimple {
  int checkSum;
  int copterID;
  int vehicleType;
  float MotorRPMSMean;
  float PosE[3]; //An array of three floating-point variables representing the position of the aircraft. The three elements are X, Y, and Z coordinates.
  float AngEuler[3];
}
```

After receiving the position information, RflySim3D will also perform a process: "PosE [I] = PosE [I] * scale" to perform a global scaling transformation.

An example of operation

This function is mainly used for unit unification. The space unit in RflySim3D is cm. If the unit of the external incoming data is m, you can use the command

```
RflySetPosScale 100
```

This eliminates the need for additional unit conversion logic.

RflyReqObjData (specify return data)

Function interpretation

Request to obtain the data of the object in the 3D scene. The usage is as follows

```
RflyReqObjData(int opFlag, FString objName, FString colorStr);
```

This function is similar to " RflyReqVehicleData Function ", which also returns some data of the objects in the 3D scene. But it only returns data for one specified target at a time.

Parameter interpretation

- OpFlag: An action flag that specifies the type of data returned. Values have the following meanings:

    0: Camera data of the specified ID is returned.

    1: Return the aircraft (Copter) data of the specified ID.

    2: Returns the Object data with the specified name.

    20: Clear all cameras

    21: Clear all aircraft

    22: Clear all objects

    23: Remove all aircraft and objects

    24: Clear all cameras, objects, and aircraft

10: Delete a camera

11: Delete an aircraft

12: Delete an object

- ObjName: Object name or camera ID, used to specify the target object or camera to acquire data. For camera data, you can specify the ID of the camera. For the object data, the name of the object may be specified.

- ColorStr: a color string, an optional parameter that specifies the color of the data.

When opFlag is 0, the function returns the camera data used to capture the image with the specified ID (objName = seqID). The camera data structure is as follows:

```
struct CameraData {
int checksum = 0; //checksum, constant 1234567891
int SeqID; //camera serial number
int TypeID; //camera type
int DataHeight; //Image pixel height
int DataWidth; //Image pixel width
float CameraFOV; //camera field angle
float PosUE[3]; //Center position of the camera
float angEuler[3]; //Euler angle of the camera
double timestmp; //timestamp
};
```

When op Flag is 1, the function returns data for the aircraft（ objName = Copter ID ）with the specified ID. The aircraft data structure is as follows:

```
struct CoptReqData {
int checksum = 0; //checksum, constant 1234567891
int CopterID; //Aircraft ID
float PosUE[3]; //Central position of the aircraft (specified during artificial 3D modeling, the attitude coordinate axis is not necessarily at the geometric center)
float angEuler[3]; //Euler angle of the aircraft
float boxOrigin[3]; //object geometric center coordinate
float BoxExtent[3]; //Half of the length, width and height of the object's outer frame
double timestmp; //timestamp
};
```

When opFlag is 2, the function returns data for the object with the specified ID (objName = seqID/ObjName [32]). The object data structure is as follows:

```
struct ObjReqData {
int checksum = 0; //checksum, constant 1234567891
int  seqID = 0;
float PosUE[3]; //Center position of the object (specified during artificial 3D modeling, the attitude coordinate axis is not necessarily at the geometric center)
float angEuler[3]; //Euler angles of an object
float boxOrigin[3]; //object geometric center coordinate
float BoxExtent[3]; //Half of the length, width and height of the object's outer frame
double timestmp; //timestamp
char  ObjName[32] = { 0 }; //Name of the object touched
};
```

An example of operation

When op Flag = = 0, it can create a camera to capture the image according to the request, and the data of the camera can be obtained through this function:

```
RflyReqObjData 0 0 FFFFFF
```

When op Flag = = 1, open RflySim3D, double-click the ground with the mouse + press the letter O + number 3, create an ID aircraft (ID is 1000 by default), and enter the command

```
RflyReqObjData 1 1000 FFFFFF
```

When opFlag = = 2, not all objects in RflySim3D are Copter objects (such as terrain, obstacles, buildings, etc.), and this interface can be used to obtain information about these non-Copter objects.

Open RflySim3D and double-click the ground. We can see that after the mouse hits the ground, some information is output on the screen. The first word indicates the name of the hit object, that is to say, "The name of the terrain is Landscape＿1".
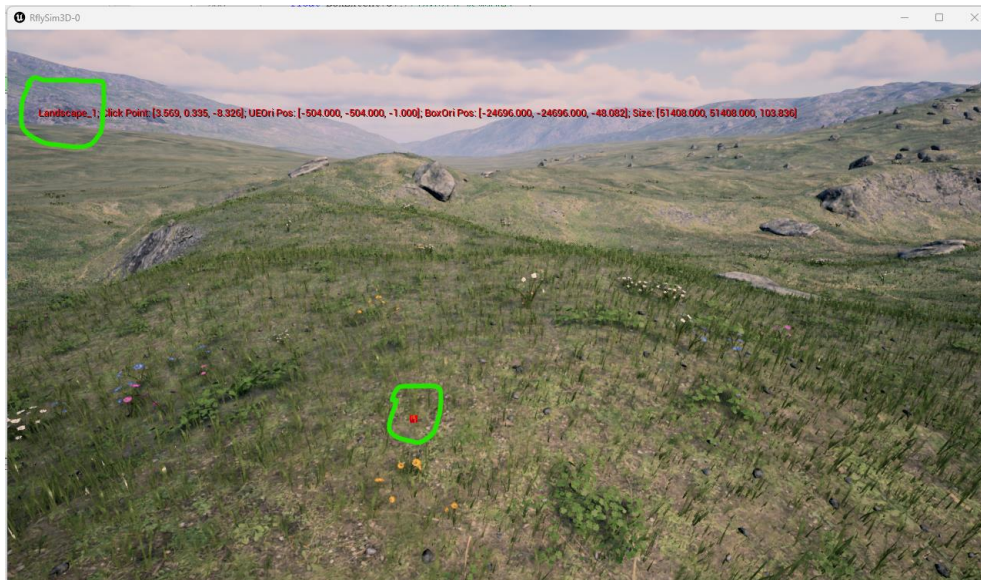
Then we use the command

```
RflyReqObjData 2 Landscape_1 FFFFFF
```

Indicates that we want to know about the object named "Landscape＿1".

RflyClearCapture（Clear Image Cache）

Function interpretation

Clear Captured Image

```
RflyClearCapture(int seqID)
```

This function clears memory for images captured by the image sensor with the specified seqI

D. If seqID < 0, memory data is cleared for images from all cameras.

Parameter interpretation

None

An example of operation

```
RflyClearCapture
```

RflyDisableVeMove（Reject the specified Copter data）

Function interpretation

Refuse to receive the message of the Copter with the specified ID

```
RflyDisableVeMove(FString CopterIDList, int disable);
```

Using this function, RflySim3D can refuse to receive some UAV data according to the ID, which is equivalent to prohibiting the creation, movement and attitude change of Copters with these IDs.

Parameter interpretation

- CopterIDList: a comma-separated list of aircraft IDs. Used to specify the aircraft ID to be disabled from receiving messages.
- Disable: disable flag, which controls whether to disable the information of the aircraft with the specified ID. When the value is 1, it is disabled; when the value is 0, it is undisabled.

An example of operation

Open SITLRun in ″\Desktop\RflyTools″, use QGC to make the UAV in mobile state, and then use the command:

```
RflyDisableVeMove 1 1
```

It will be found that the UAV in QGC is still moving, the simulation coordinates in CopterSim are still changing, but the UAV in RflySim3D is stopped.

RflyChangeViewKeyCmd（Analog shortcut key）

Function interpretation

Simulate the effect of pressing a shortcut key + entering a number. The usage method is as follows:

```
RflyChangeViewKeyCmd(String key, int num)
```

Parameter interpretation

An example of operation

For example, if you press the shortcut key M + number to switch to the map with the specified ID, you can enter the command to switch to the map No.5.

```
RflyChangeViewKeyCmd M 5
```

RflyEnImgSync (Switch the image transmission mode)

Function interpretation

Enable synchronous or asynchronous graph mode

```
RflyEnImgSync(int  isEnable )
```

Parameter interpretation

- IsEnable: a value of 0 enables asynchronous mode, and a value of 1 enables synchronous mode

An example of operation

5.2 UE4/UE5 Commonly used Built-in commands

5.2.1 Routine use

t .Maxfps (Limit Frame Rate)

Function interpretation:

You can limit the performance of the emulator by setting a maximum frame rate.

Explanation of parameters:

No parameters. This command is only used to set the maximum frame rate and requires no additional parameters to specify.

Example of operation:

```
t.MaxFPS 60
```

In the above example, the maximum frame rate is set to 60 frames.

```
t.MaxFPS 30
```

In the above example, the maximum frame rate is set to 30 frames.

slomo （Modify the run speed）

Function interpretation:

Speed up or slow down an application by adjusting its speed.

Explanation of parameters:

- Run Speed Value: Lets you specify the speed of the run. The default value is 1,

which indicates normal speed. Smaller values slow things down, larger values speed things up.

Example of operation:

```
slomo 0.5
```

In the above example, the operating speed is set to half of the normal speed.

```
slomo 2
```

In the above example, the operating speed is set to twice the normal speed.

HighResShot （Custom Size Screenshot）

Function interpretation:

You can get a higher resolution screenshot by specifying the size of the screenshot.

Explanation of parameters:

- Screenshot Size Values: Lets you specify the width and height of the screenshot. It is usually specified using a numeric value in pixels.

Example of operation:

```
HighResShot 1920x1080
```

In the example above, take a screenshot with a width of 1920 pixels and a height of 1080 pixels. The screenshot is saved at the specified size.

stat fps (Display update rate)

Function interpretation:

By using this command, you can monitor the frame rate performance of the emulator in real time.

Explanation of parameters:

No parameters. This command is only used to display frame rate information and does not require additional parameters to specify.

Example of operation:

```
stat fps
```

In the above example, frame rate display is enabled. The current frame rate information will be displayed in RflySim3D, and entering the command again will hide the frame rate display

stat unit (Show various types of consumption)

Function interpretation:

Display performance statistics in RflySim3D. By using this command, you can monitor the performance of each system in real time, such as CPU, GPU, and rendering time.

Explanation of parameters:

No parameters. This command is used only to display performance statistics and requires no additional parameters to specify.

Example of operation:

```
stat unit
```

In the example above, performance statistics display is enabled. Performance data such as CPU, GPU, and render times are displayed in Rfly Sim3D.

stat rhi （Displays the consumption details on the GPU）

Function interpretation:

Display the performance statistics of the rendering interface (Render Hardware Interface) in RflySim3D. By using this command, you can monitor the performance of the rendering interface used by RflySim3D in real time.

Explanation of parameters:

No parameters. This command is used only to display the performance statistics of the rendering interface and requires no additional parameters to specify.

Example of operation:

```
stat rhi
```

In the example above, render interface performance statistics display is enabled. Rfly Sim3D displays performance data for the rendering interface, such as the number of frames rendered, GPU utilization, and GPU memory usage.

stat game (Display Tick feedback time of each process)

Function interpretation:

Show statistics of program logic and performance in RflySim3D. By using this command, the logical operation and performance of RflySim3D can be monitored in real time.

Explanation of parameters:

No parameters. This command is only used to display RflySim3D logic and performance statistics and requires no additional parameters to specify.

Example of operation:

```
stat game
```

In the above example, RflySim3D logic and performance statistics display are enabled.

Statistics such as RflySim3D logical frame count, RflySim3D update time, and render time are displayed in RflySim3d.

stat gpu （Display Frame GPU Statistics）

Function interpretation:

Display the performance statistics of the GPU (graphics card) in RflySim3D. By using this command, the performance of the GPU in RflySim3D can be monitored in real time.

Explanation of parameters:

No parameters. This command is only used to display the performance statistics of the GPU and does not require additional parameters to be specified.

Example of operation:

```
stat gpu
```

In the example above, the display of performance statistics for the GPU is enabled. Statistics such as GPU usage, rendering time, and GPU memory usage are displayed in RflySim3D.

stat Engine （Displays the number of frames， Time, number of triangles, etc.）

Function interpretation:

Display engine-related statistics in RflySim3D. By using this command, you can monitor the performance of the engine and other related information in real time.

Explanation of parameters:

No parameters. This command is used only to display engine statistics and requires no additional parameters to specify.

Example of operation:

```
stat Engine
```

In the example above, the display of engine-related statistics is enabled. Statistics including frame rate, time, number of triangles, map memory usage, and physics simulation are displayed in RflySim3D.

stat scenerendering （Drawcall is displayed）

Function interpretation:

Show Draw Calls in RflySim3D. By using this command, you can monitor the performance of scene rendering in RflySim3D and other related information in real time.

Explanation of parameters:

No parameters. This command is only used to display the statistics of the scene rendering and does not require additional parameters to be specified.

Example of operation:

```
stat scenerendering
```

In the example above, the statistics display for the scene rendering is enabled. RflySim3D displays statistics including Draw Calls, render frames, number of triangles, render time, number of objects, lighting calculations, and shadow calculations.

r.setRes （Set the display resolution）

Function interpretation:

Sets the display resolution for the game.

```
r.setRes [Width]x[Height] [Fullscreen/Windowed]
```

Explanation of parameters:

- Resolution parameter: [Width] X [Height], used to specify the display resolution of the game. For example, 1280x720 is 1280 pixels wide and 720 pixels high.
- Display mode parameter: Full screen mode or Windowed mode, used to specify the display mode of the game.

Example of operation:

Set resolution to 1920x1080, full screen mode:

```
r.setRes 1920x1080 Fullscreen
```

Set the resolution to 1280x720, window mode:

```
r.setRes 1280x720 Windowed
```

Notice

The actual supported resolutions and display modes depend on the graphics card, monitor, and RflySim3D settings.

r.Streaming.PoolSize （Modify the texture streaming pool sizes）

Function interpretation:

Sets the size of the texture streaming pool. The texture streaming pool is a memory pool used to dynamically load and unload texture resources to optimize performance and resource management.

Explanation of parameters:

- Pool Size: Specifies the size of the texture streaming Pool. The size is expressed in MB.

Example of operation:

Set the streaming pool size to 4096 MB:

```
r.Streaming.PoolSize 4096MB
```

Notice

The default allocation of 1G video memory, do not know how to set, give zero value to R. Streaming. PoolSize 0, when the upper left corner of the screen appears "TEXTURE STREAMING POOL OVER＊＊＊MiB BUDGET", you need to modify this, when the display prompts." Will cause part of the map load LOD changes, the screen will be blurred.

5.2.2 LowGPU Scenario Usage

r.forcelod （The number of LOD points and faces）

Function interpretation:

Change the Level of Detail (LOD) setting for the scene to reduce the number of points and faces. LOD is a technique for managing the display detail of an object, using different levels of model and texture detail at different distances and camera angles to balance performance and visual impact.

Explanation of parameters:

- LOD Level: The LOD level to enforce. It is usually expressed as a numeric value from 0 (the highest level) to the maximum LOD level.

Example of operation:

Set all LOD enforcement to 0. This sets the LOD enforcement for all objects to the highest level, that is, LOD is disabled. This means that the object will always be rendered at the highest level of detail, regardless of the distance or camera angle used.

```
r.ForceLOD 0
```

Similarly, set LOD to -1, which turns off LOD for all objects, rendering the objects with the lowest level of detail.

```
r.ForceLOD -1
```

Notice

In complex scenes, modifying the LOD settings may cause changes in the display of the terrain or other objects.

r.ScreenPercentage （Render resolution percentage）

Function interpretation:

Controls the percentage of resolution used when rendering.

Explanation of parameters:

- Resolution Percentage: Specifies the Resolution Percentage. Integer values are often used, such as 100 for the original resolution, 50 for a reduction to half the original resolution, and 200 for an increase to twice the original resolution.

Example of operation:

Set the rendering resolution to 75% of the original resolution:

```
r.ScreenPercentage 75
```

r.ShadowQuality （Shadow quality）

Function interpretation:

Controls the quality level of the shadow.

Explanation of parameters:

- Parameter value and corresponding shadow quality:

    0: Disable shadow

    1: Low quality shadow

    2: Medium quality shadow

    3: High quality shadow

Example of operation:

Disable Shadows: Completely disable shadows in the game to improve performance.
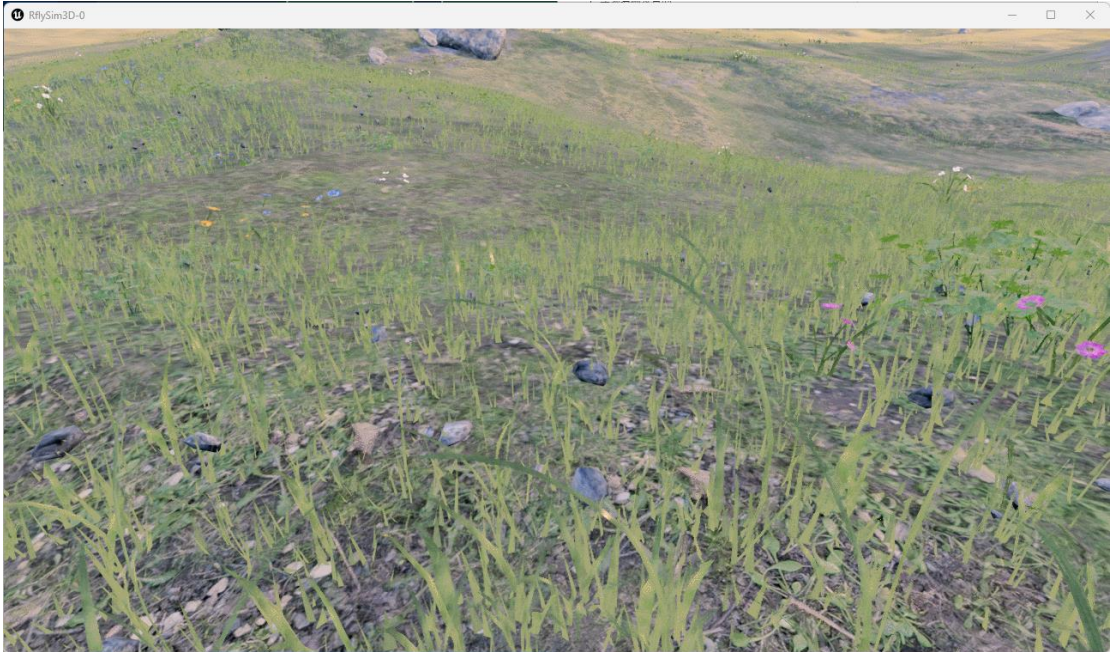
r.PostProcessAAQuality （Anti-aliasing quality）

Function interpretation:

Controls the quality level of the post-processing anti-aliasing effect.

Explanation of parameters:

- Parameter value and corresponding anti-aliasing quality:

    0: Disable post-processing anti-aliasing

    1: Low-quality post-processing anti-aliasing

    2: Medium quality post-processing anti-aliasing

    3: High-quality post-processing anti-aliasing

Example of operation:

Disable post-processing anti-aliasing: This completely disables the post-processing anti-aliasing effect to improve performance. However, the image may appear jagged or not have smooth edges.

```
r.PostProcessAAQuality 0
```

r.SetNearClipPlance （Viewport Near Clip Face）

Function interpretation:

Sets the Near Clip Plane for the camera.

Explanation of parameters:

- Near Crop Plane Distance: Expressed in units of camera space, usually centim

eters.

Example of operation:

The following is an example of a near clipping plane set to 10 units away:

```
r.SetNearClipPlane 10
```

r.MipMapLoDBias （Texture Level deviation）

Function interpretation:

Adjust the texture's MipMap level bias (LOD Bias). MipMap is a technique for pre-rendering textures that provide different levels of detail.

Explanation of parameters:

- LOD Bias value: Indicates the MipMap level deviation of the texture. A higher LOD Bias value means using a lower resolution MipMap level, reducing the level of detail displayed by the texture.

Example of operation:

The following is an example of setting the MipMap level bias of a texture to 2:

```
r.MipMapLODBias 2
```

sg.TextureQuality （Texture quality）

Function interpretation:

The texture quality level determines the resolution and detail of the texture, which has an impact on the visual effects and performance of the game.

Explanation of parameters:

- Lets you set the texture quality level.

Example of operation:

The following is an example of setting the texture quality level to the lowest (level 0):

```
sg.TextureQuality 0
```

sg.PostProcessQuality (Post-processing quality)

Function interpretation:

Postprocessing is the process of processing the image after the scene is rendered. It can adjust the color correction, blur, lighting effect and so on to enhance the visual effect of the scene.

Explanation of parameters:

- Quality level: determines the detail and computational complexity of the post-processing effect.

Example of operation:

The following is an example of setting the post-processing quality level to the lowest (level 0):

```
sg.PostProcessQuality 0
```

foliage.MaxTrianglesToRender （Number of triangular faces of vegetation model rendered）

Function interpretation:

The number of model triangular faces rendered (Foliage Triangles to Render) for a vegetation type refers to the setting that limits the number of triangular faces when the vegetation type (foliage) model is drawn in the simulation scene.

Explanation of parameters:

- Used to set the number of triangular faces of the vegetation model that are allowed to be drawn.

Example of operation:

The following is an example of setting the number of triangular faces of the vegetation model to the lowest (0):

```
foliage.MaxTrianglesToRender 0
```

6. The program starts Scripting interface

Demonstration effect of corresponding platform routines 6.1: 0. ApiExps/e3_T_X TA llCrtlScript/Readme.pdf

Demonstration effects of corresponding platform routines 6.2 ~ 6.4: 0. ApiExps\e7 UEMapCtrl\8.TXTMapCrtlScript\Readme.pdf

The FFile Helper class of the UE engine can read and write txt text files. After determining that the txt file exists in the path, you can call the FFileHelper: LoadFileToString function to read the contents of the file into an FString object, and then process or display the contents; To read each line of a text file, you can use the FFile Helper: LoadFileToStringArray function. To write to a text f

ile, you can use the FFileHelper: SaveStringToFile function; if you want to write an array of string s to a text file and wrap, you can use the FFile Helper: SaveStringArrayToFile function. A simple u se case can be found at the following link: 【 UE4 C++ 】read and write Text Paper FFileHelper

RflySim3D uses this function to develop four txt script interfaces as shown in the following fi gure

6.1 The program is started （ RflySim3D.txt ）

RflySim3D can automatically identify the txt script On the platform PX4PSP\RflySim3D Cr eate one in the directory Named " RflySim3D.txt Script in the specified directory. When RflySi m3D is opened, it will automatically execute some console commands in the script. Each comman d needs to be exclusive.

6.2 Switch the map to start （ LowGPU.txt ）

Create a script named "map name.txt" in the PX4PSP \ RflySim3D directory of the platfor m. You can also pre-store some console commands. When you switch to the map in RflySim3D, th e corresponding commands will be executed. The commands pre-stored in the LowGPU. Txt here are mainly used to improve the rendering efficiency to meet the simulation needs of low-performa nce computers, as shown in 5.2.2 LowGPU Scenario usage

6.3 Hit object log （ ClickLog.txt ）

The ClickLog. Txt records the value of the user's double mouse click starting (Restart Rfly S im 3D The previous hit point information will disappear) from the opening of RflySIm3D. Ta ke the log file generated after hitting a point on the ground and a quadrotor in 3D Display in RflyS im3D as an example:

The first line records the, Unit , coordinate system and map name information of the Current t ime last hit object.

```
2023.09.18-16.58.33, Unit:m, Frame: NED, MapName:3DDisplay
```

- Current time (accurate to seconds): 2023.09.18-16.58.33

- Unit: length m, radians rad

- Local horizontal coordinate system (Frame): N ED (North East)
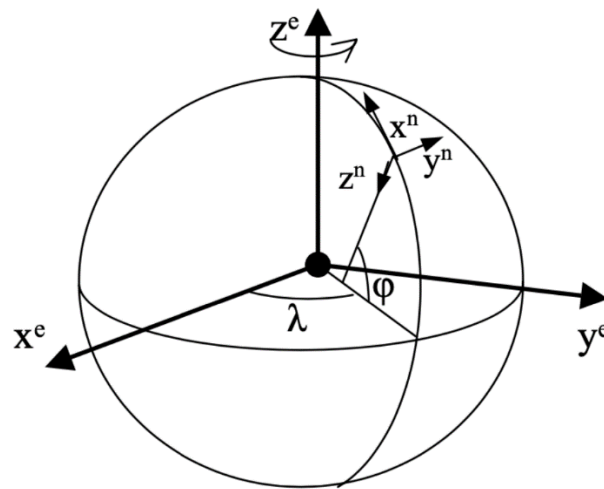
The x-axis points north, the y-axis points east, and the z-axis points to the ground

Rotation about the x-axis, called the roll angle

Rotation about the y-axis, called pitch angle

Rotate around the z-axis to become the yaw angle

The corresponding IMU carrier coordinate system is front-right-down.

Euler angle rotation order: z-y-x

- Map Name: 3D Display

Followed by the information of the midpoint hit from the front to the back

```
    Landscape_1; Click Point: [1.289, 0.486, -8.103]; UEOri Pos: [-504.000, -504.000, -1.000];
BoxOri Pos: [-24696.000, -24696.000, -48.082]; Size: [51408.000, 51408.000, 103.836]
```

- Hit object number: The object here is the ground Landscape _ 1

- Click Point: The position of this point in the world scene [1.289, 0.486, -8.103]

- Real coordinates of the hit object in the UE system (UEOri Pos): Here are the coordinates of the ground object Landscape _ 1 in the UE system [-504.000, -504.000, -1.000]

- Center Point of Bounding Box of Hit Object (BoxOri Pos): Here are the coordinates of the center point of the bounding box of the ground object Landscape _ 1 in the UE system [-24696.000,-24696.000, -48.082]

- Hit a vertex on the object bounding box (Size): Here are the coordinates of a vertex of the bounding box of the ground object Landscape _ 1 in the UE system [51408.000, 51408.000, 103.836]

```
    Copter_1000 ; ID: 1000; PosE: [-5.746, 0.601, -7.771]; AngEuler: [0.000, 0.000, 0.330], C
enterHeight: 16.00
    Copter_1000; Click Point: [-5.915, 0.524, -7.972]; UEOri Pos: [-5.746, 0.601, -7.931]; BoxOri Pos: [0.002, 0.006, 0.01
```

- Hit object number: The object here is created by itself, so it is the naming rule of Copter _ * * *, which is Copter _ 1000 here.

- Hit Object ID: 1000

- Position Sent (PosE): The position information of the hit object for transmission

- Sent attitude angle (AngEuler): The hit object is used to transmit the attitude angle information.

- Center Height of the object sent to the ground (Center Height): The height information of the hit object for transmission.

- Click Point: The position of this point in the world scene [-5.915, 0.524, -7.972]

- Real coordinates of the hit object in the UE system (UEOri Pos): Here are the coordinates of the quadrotor object Copter _ 1000 in the UE system [-5.746, 0.601, -7.931]

- Center Point of Bounding Box of Hit Object (BoxOri Pos): Here are the coordinates of the center point of the bounding box of Copter _ 1000 in the UE system [0.002, 0.006, 0.016]

- Hit a vertex on the object bounding box (Size): here is the coordinates of a vertex of the bounding box of Copter _ 1000 in the UE system [0.549, 0.535, 0.294]

6.4 Create an object log （ CreateLog.txt ）

The CreateLog. Txt records the information of the object created by pressing the O key (after restarting RflySim3D, the previous information of the created object will disappear). The format is the same as the txt loaded by using RflyLoad3DFile the command. Therefore, the CreateLog. Txt is renamed as * * *.txt. Open RflySim3D again and switch to the * * * map. You can see that the scene created with the O key and the Rfly * * command will be loaded automatically. Take the log file generated after creating a quadrotor in RflySim3D as an example:

The first line records the current time, unit, coordinate system and map name information of the latest created or moved object

```
2023.09.19-10.09.25, Unit:m or rad, Frame: NED, MapName:3Ddisplay
```

- Current time (accurate to seconds): 2023.09.19-10.09.25

- Unit: length m, radians rad

- Local horizontal coordinate system (Frame): NED

- Map Name: 3D Display

The format and information for creating an object are as follows

```
    CMD:CreateVehicle|Rfly***, copterID,  vehicleType , PosE[0], PosE[1], PosE[2], AngEuler
[0], AngEuler[1], AngEuler[2]
    CreateVehicle, 1000, 3, -10.45, -1.77, -6.97, 0.00, 0.00, -0.83
```

- Create object ID (copterID): 1000

- Create vehicleType: 3 (corresponding to quadrotor)

- Create object position coordinates (PosE [0], PosE [1], PosE [2]): position of this quadrotor in the world scene -10.45, -1.77, -6.97

- Create object attitude angles (AngEuler [0], AngEuler [1], AngEuler [2]): attitude angles 0.00, 0.00, -0.83 for this quadrotor

Notice

The vehicleType is a variable that controls the 3D object (the specific style is determined by the model category class_ID and style DisplayOrder in the XML).

- VehicleType = 3 corresponds to quadrotor. RflySim3D has multiple quadrotor styles to choose from. Enter vehicleType = 1003, 2003, 3003 and so on to select the specific style.

- VehicleType = 5 or 6 corresponds to a six-rotor

- VehicleType = 30 corresponds to a person, where 3030 represents a person of style 3

- VehicleType = 100 corresponds to fixed wing.

- VehicleType = 150 corresponds to targets such as rings and boxes

- The vehicleType = 60 corresponds to the illuminated, etc., which is used for light show display

6.5 Program startup parameter configuration command

BAT script configuration

```
    REM UE4Path
    cd %PSP PATH% RflySimUE5tasklist|find /i "RflySim3D.exe" start %PSPPATH%\RflySimUE5\RflyS
im3D.exe-key=S,N1
    choice /t 5 /d y /n >nul
```

Configure shortcuts directly

7. Scene import interface

Demonstration effect of corresponding platform routines 7.1: UE4 Default scene import

7.1 Normal import (RflySim imported by UE)

There are three kinds of scenarios that have been processed in the UE project: the UE default scenario, the scenario purchased from the Unreal Mall, and the scenario migrated from other UE projects. These scenes can be imported into RflySim3D after being baked directly in the UE, and the complete simulation scene in RflySim3D requires the following three parts of information.

7.1.1 Scene file ("****.umap")

It is required to Bake copy the scene files under the directory of " [UE project main folder] \ Saved \ Cooked \ WindowsNoEditor \ [project name] \ Content" to C: \ PX4PSP \ RflySim3D \ RflySim3D\Content Under the directory .

Notice

After baking and copying, the names of files and folders cannot be modified, and new folders cannot be created in the \ RflySim3D \ Content directory. Renaming can only be done in the UE editor.

Each "****.umap"(different before and after baking) map file corresponds to an independent 3D scene. RflySim3D/RflySimUE5 will automatically scan all the.umaps in the PX4PSP \ RflySim3D \ RFlySim3D \ Content directory.

7.1.2 Terrain Elevation information ("****.png")

Before importing the scene, observe and record the approximate range of the scene in the UE editor (in cm); After baking the scene and importing it into RflySim3D, use RflyScanTerrainH (Scan terrain) the command in RflySim3D with a slightly larger range (in meters) than the "****.umap" scene terrain. This will find the corresponding "**.png" and "**.txt" files in the PX4PSP \ RflySim3D directory, which store the elevation information and terrain calibration and range of the scene respectively. Copy these two files to PX4PSP \ CopterSim \ external \ map to complete the import.

Notice

The PNG terrain file is actually a two-dimensional matrix stored in the form of a picture, which contains the elevation map of the scene. Storing the matrix in PNG format can well realize the compression of the elevation matrix, which is convenient for saving space.

The elevation file of PNG does not contain the coordinate origin, zoom scale, scene range and other information, so a correction file is required. RflySim platform uses txt format to input 9-dimensional array or 12-dimensional array (more latitude, longitude and altitude geographic datum information) to input correction information.

7.1.3 Terrain calibration data ("****.txt")

The txt correction file in the platform stores the three-dimensional coordinate points in the upper right corner (XY is all positive, and Z upward is positive), the three-dimensional coordinate points in the lower left corner (XY is all negative, and Z upward is positive), and the third three-dimensional coordinate point, all in centimeters. The purpose of the first two points is to confirm the range and central coordinates of the terrain. The coordinates of the third point can be selected by oneself. In theory, it is necessary to have a drop in height from the first two points as far as possible to

correct the height scale.

In addition, when using cesium global large scene (only personal advanced version or above) in RflySim3D, you can also input three-dimensional GPS terrain data after the above-mentioned 9-dimensional data in txt, add them in the order of latitude, longitude and altitude, and configure the corresponding display reference coordinates in QGroundControl. See the corresponding routine 2. AdvExps\e2_CesiumScene\1.ObliModelMap\Readme.pdf
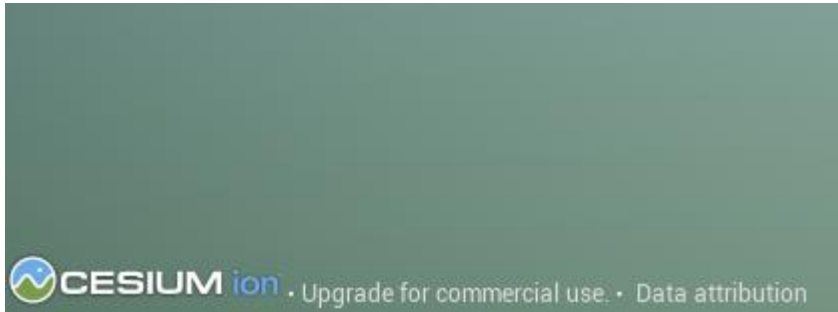
7.2 Datasmith Import

Datasmith is a collection of plug-ins developed by Epic Games for Unreal Engine, which is used to import various design data (such as CAD, 3D modeling software, etc.) into Unreal Engine to create interactive three-dimensional scenes with its real-time rendering and visualization capabilities. For a detailed tutorial, see: Datasmith Overview | Unreal Engine documentation (unrealengine.com)

Compared with ordinary scene import, Datasmith plug-in can support many different file formats, including but not limited to FBX, DWG, 3DS, SketchUp, Rhino, CATIA, SolidWorks, etc; Datasmith preserves the accuracy of the imported data, including geometry, materials, lighting, camera, animation, etc. Datasmith preserves the hierarchy and organization of the original design data to make it easier to manage and edit the imported scene; Datasmith provides automation tools that can be used to automate the processing of imported data, such as setting materials automatically, handling LOD (level of detail), creating collision bodies, and so on. In summary, using the Datasmith plug-in allows you to import more complex scenes and models, and makes the import more accurate.

7.2.1 Datasmith For Unreal (Import UE）

You can export.udata Smith files by installing the corresponding.udata Smith file exporter plug-in for various 3D processing software. For all.udata Smith file exporters, see Datasmith Exporter plug-in - Unreal Engine
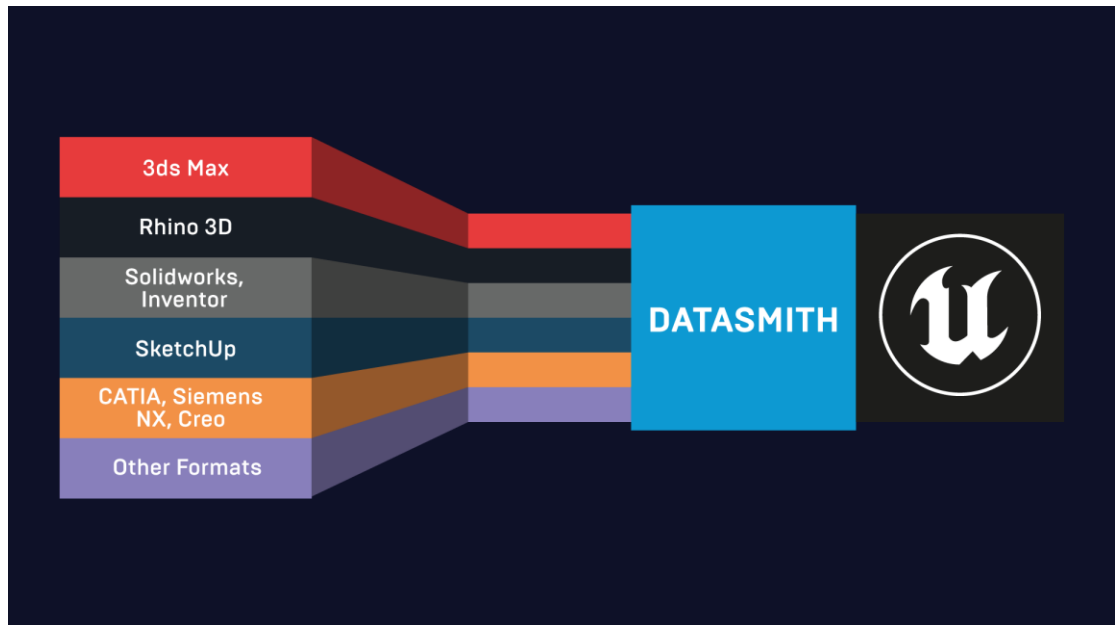
The main workflow is as follows:

UE4
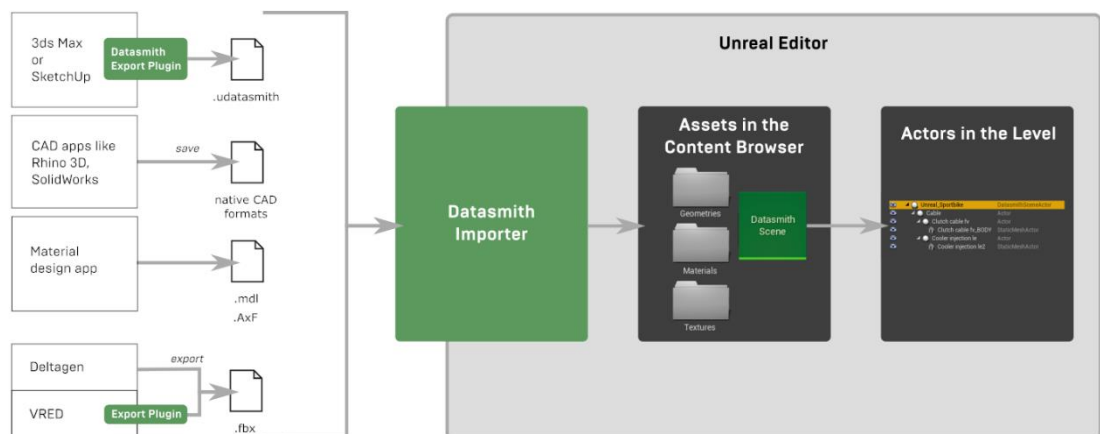
The.udatasmith format file can be imported into UE4 by enabling the following plug-ins in UE4.
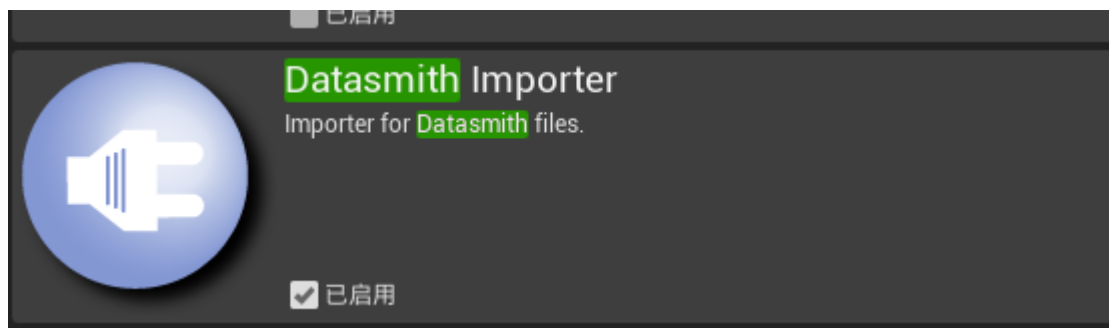
UE5
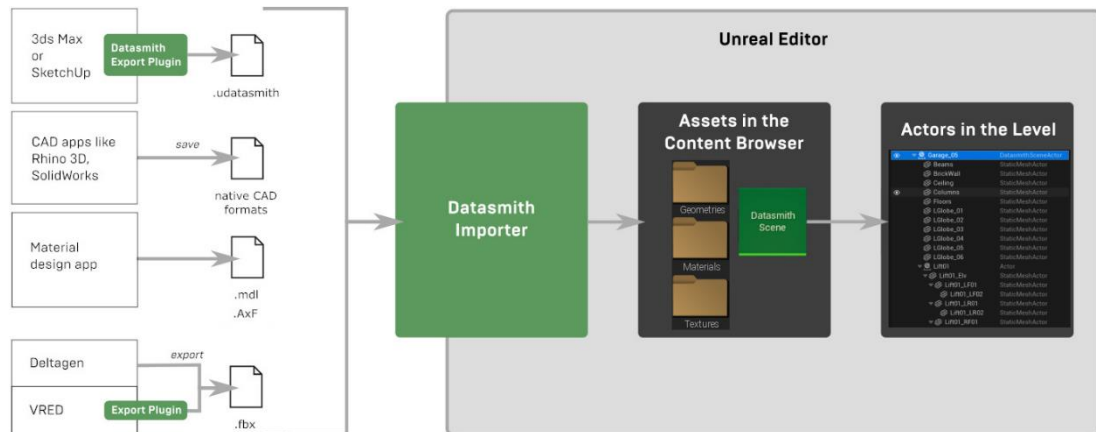
The.udatasmith format file can be imported into UE5 by enabling the following plug-ins in U
E5.

7.2.2 Datasmith For T winmotion (Import Twinmotion
）

As long as the exporter plug-in of the corresponding version is downloaded and installed for t
he 3D processing software, when the software is used to process the model, it will automatically su
pport the real-time preview of the model scene that has not been edited in Twinmotion, that is, whi
le completing the preliminary rough mold design, it will preview the effect after the later material
processing. Take SketchUp as an example, download and install the corresponding version of the e
xporter plug-in: Face_SketchUp Pro_Yes_Twinmotion Datasmith_Exporter plug-in_ - Twinmotion

Open the corresponding scene in SketchUp, then select Linear Import in Twinmotion and refr
esh the link.

You can see the scene in the twinmotion viewport

7.2.3 SKETCHUP PRO Exporter

Download and install the corresponding version of the exporter plug-in: Face SketchUp Pro Yes Twinmotion Datasmith Exporter plug-in - Twinmotion . SketchUp cannot be enabled during installation.

Export the corresponding model to.udata Smith format in SketchUp
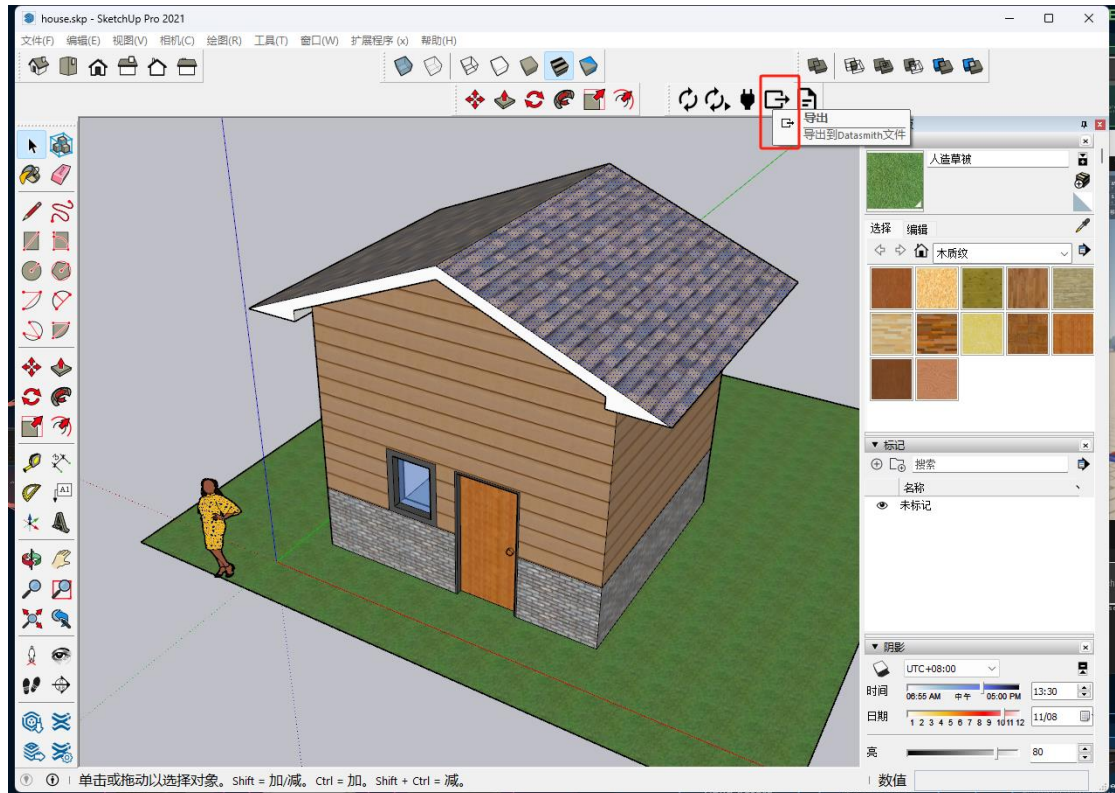
7.2.4 AUTODESK 3DS MAX Exporter

Download and install the corresponding version of the exporter plug-in: Face_3ds Max_Yes Twinmotion Datasmith_Exporter plug-in_- Twinmotion . 3DsMax cannot be enabled during installation.

Export the corresponding model to.udata Smith format in 3DsMax: File-> Export-> File Type The udata Smith file type will appear. Select Export.
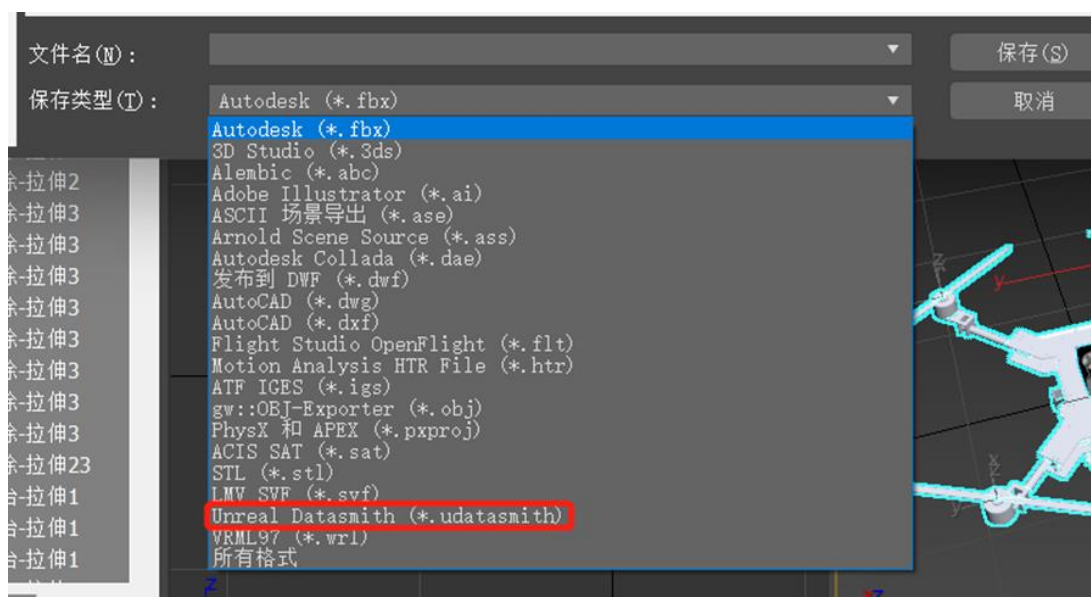
7.3 T winmotion Import

Twinmotion supports all major CAD, BIM, and modeling solution file formats (Twinmotion Plug-in - Twinmotion) and synchronizes with most of them with one click. In addition, the Twinmotion project can also be further developed in the Unreal Engine.
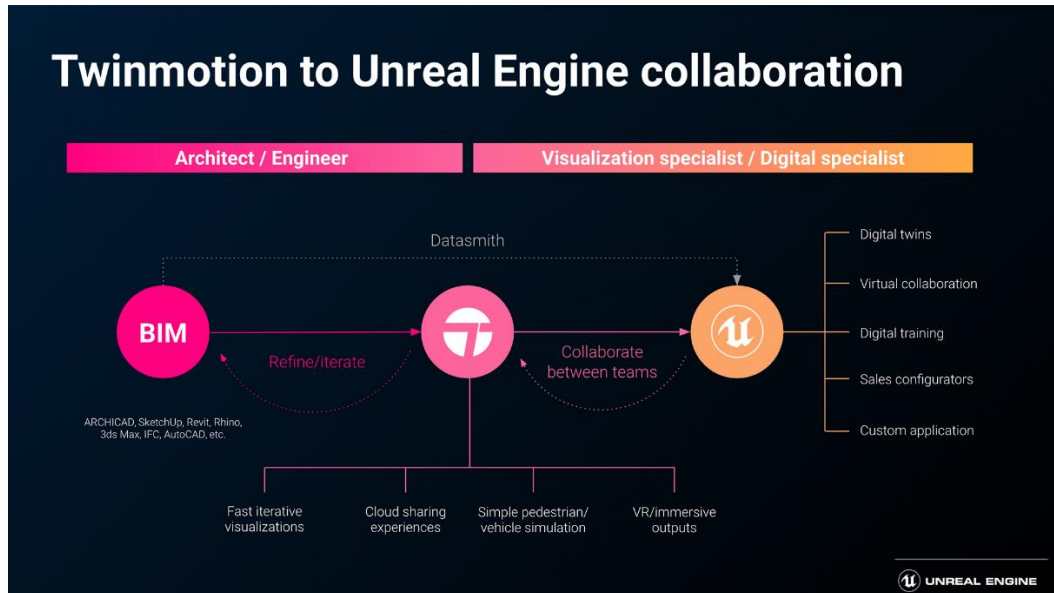
7.3.1 Twinmotion Import U E4

Twinmotion version

UE4 installs Twinmotion versions up to and including 2022.2.3.

Plug-in

UE4 installs the Datasmith Twinmotion Importer and Twinmotion Content for Unreal Engine plugins

Import method

UE4 only supports.tm files exported before Twinmotion 2022.2.3

If UE4 is used, the following two plug-ins need to be enabled

The Data Smith Twinmotion Inporter "parses the.tm file; the Twinmotion Content" contains Twinmotion's various material libraries, as well as the model library, which maps the models in the scene according to the material path.

Import the RflySim3D method

After baking, the baking file needs to be copied to the three files under PX4PSP \ RflySim3D

[Project Name]-> Saved-> Cooked-> WindowsNoEditor-> [Project Name]-> Content Copy to PX4PSP-> RflySim3D-> RflySim3D-> Content.

[Project Name]-> Saved-> Cooked-> WindowsNoEditor-> Engine-> Plugins-> Marketplace-> TMtoUnrealContent Copy to PX4PSP-> RflySim3D-> Engine-> Plugins-> Marketplace-> TMtoUnrealContent.

[Project name]-> Saved-> Cooked-> WindowsNoEditor-> Engine-> Plugins-> Enterprise-> Data smithContent Copy to PX4PSP-> RflySim3D-> Engine-> Plugins-> Enterprise-> Data smithContent.

7.3.2 Twinmotion Import U E5

Twinmotion version

UE5 installs versions after 2023.3.1.

Plug-in

UE5 installs the "Data Smith Twinmotion Content for Unreal Engine" plugin

Import method

UE5 requires a version after 2023.1 Preview 1, which Twinmotion supports Datasmith file export.

If you are using UE5, you can import the Twinmotion file into UE5 using the Datasmith import method. (The following plug-in "Data Smith Inporter" and "Data Smith Twin motion Content" shall be enabled)
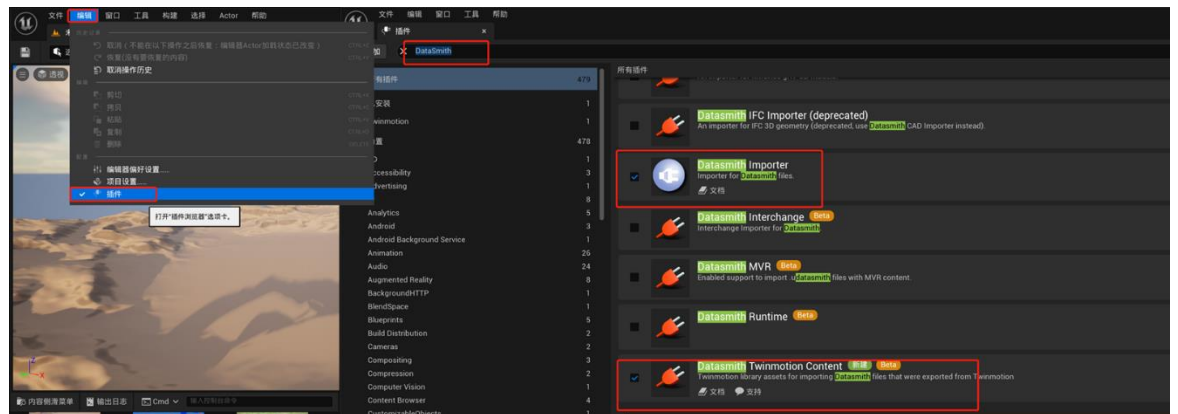
The Data Smith Inporter "will parse the.udata Smith file; the Datasmith Twinmotion C ontent" contains various material functions of Twinmotion, which will calculate the ma terial properties of various object surfaces in the scene and assign different material insta nces.

Import the RflySimUE5 method

[Project name] -> Saved -> Cooked -> WindowsNoEditor -> [Project name]-> Cont ent Copy to PX4PSP -> RflySim UE5 -> RflySim UE5 -> Content Down .

7.4 Cesium Import

7.4 .1 C esium F or U nreal

Enable the following plug-ins
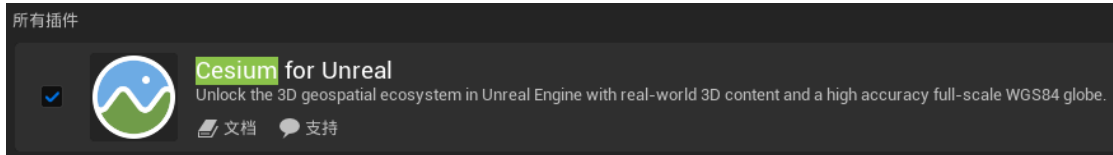
Connect to Cesium ion and log in to your Epic account
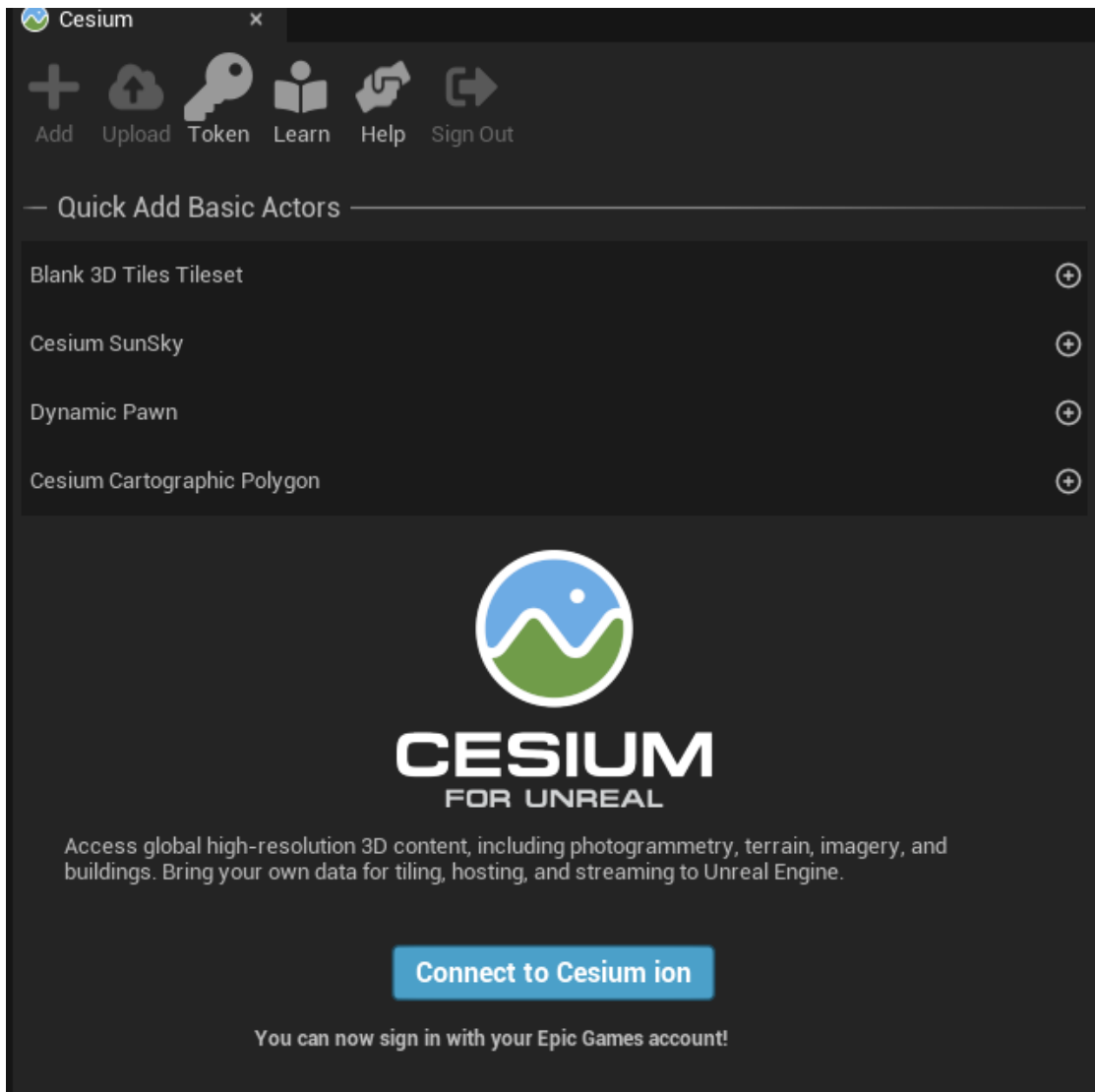
The Cesium for unreal plug-in was successfully enabled
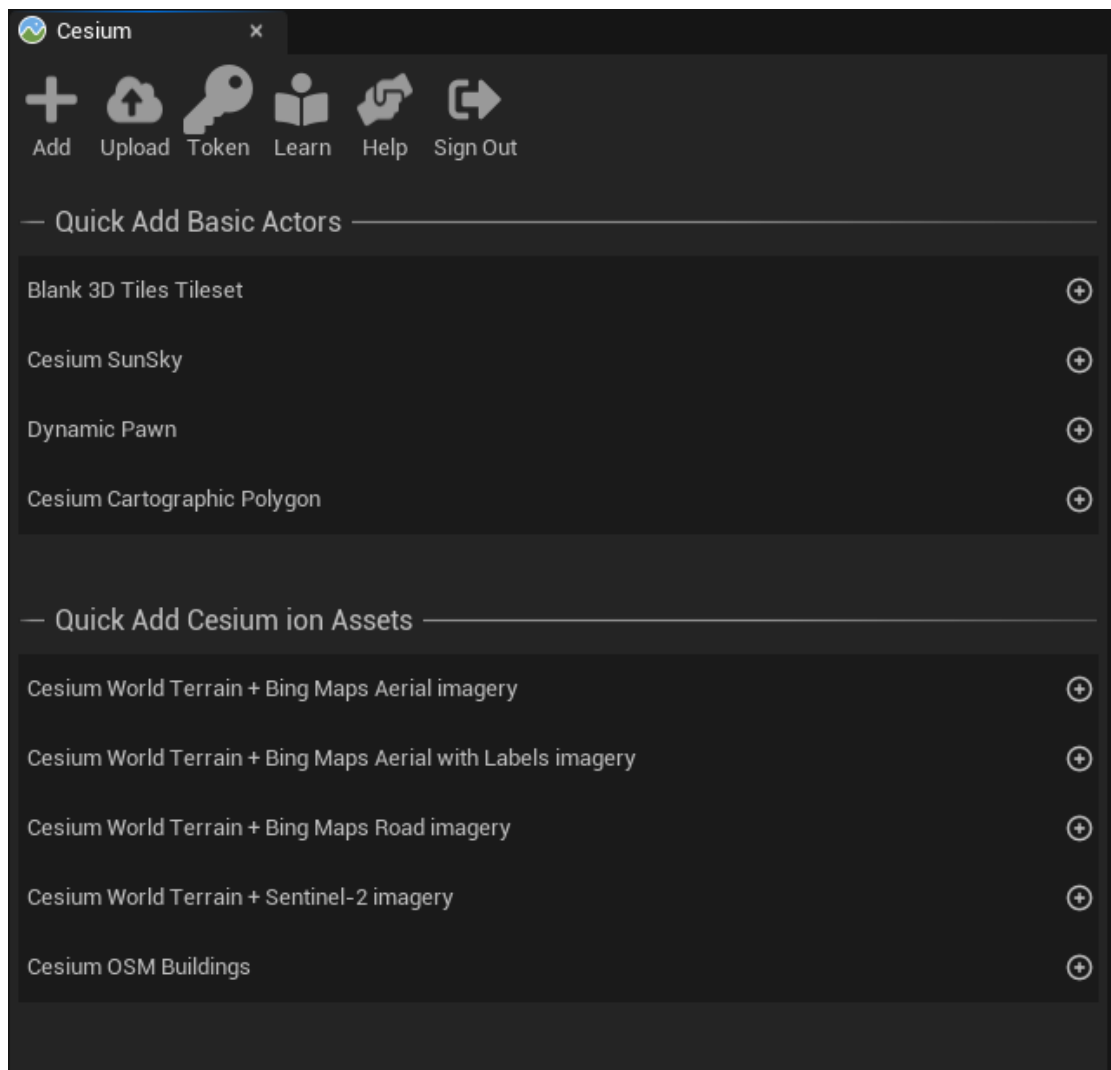
8. Model import interface

8.1 Demonstration of corresponding routine effect: [0. ApiExps/e8_UAVCtrl/1.Actua torBinding/Readme.pdf](#)

8.1 Demonstration of corresponding routine effect: [0. ApiExps\e8_UAVCtrl\2.Actua torCtrl\Readme.pdf](#)

8.1   XML Rule

XML (eXtensible Markup Language) is a markup language used to describe data. It is similar to HTML, but the tags in it can be defined by themselves. Because of this flexibility and extensibility, it is often used as the configuration file of the program. The format of XML file is similar to HTML, which is also composed of various tags. Each tag is enclosed by two pairs of <> symbols <* * *> </* * *>, and strings, numbers, or other tags are stored inside the tag. RflySim3D also uses it to configure the attributes of the UAV. In RflySim3D, each Copter 3D model has a configuration f

ile in XML format that provides information about it.

F450_Default.xml

```xml
<?xml version="1.0"?>
<vehicle>
    <ClassID>3</ClassID>
    <DisplayOrder>1000</DisplayOrder>
    <Name>F450_Default</Name>
    <Scale>
        <x>1</x>
        <y>1</y>
        <z>1</z>
    </Scale>
    <AngEulerDeg>
        <roll>0</roll>
        <pitch>0</pitch>
        <yaw>0</yaw>
    </AngEulerDeg>
    <body>
        < ModelType>0</ModelType>
        <MeshPath>/Rfly3DSimPlugin/copter/F450Body</MeshPath>
        <MaterialPath></MaterialPath>
        <AnimationPath></AnimationPath>
        <CenterHeightAboveGroundCm>16.3</CenterHeightAboveGroundCm>
        <NumberHeigthAboveCenterCm>20</NumberHeigthAboveCenterCm>
    </body>
    <ActuatorList>
        <Actuator>
            <MeshPath>/Rfly3DSimPlugin/copter/PropellersCCW</MeshPath>
            <MaterialPath></MaterialPath>
            <RelativePosToBodyCm>
                <x>15.798</x>
                <y>16.167</y>
                <z>4.5</z>
            </RelativePosToBodyCm>
            <RelativeAngEulerToBodyDeg>
                <roll>0</roll>
                <pitch>0</pitch>
                <yaw>0</yaw>
            </RelativeAngEulerToBodyDeg>
            <RotationAxisVectorToBody>
                <x>0</x>
                <y>0</y>
                <z>1</z>
            </RotationAxisVectorToBody>
            <RotationModeSpinOrDefect>0</RotationModeSpinOrDefect>
        </Actuator>
        <Actuator>
            <MeshPath>/Rfly3DSimPlugin/copter/PropellersCCW</MeshPath>
            <MaterialPath></MaterialPath>
            <RelativePosToBodyCm>
                <x>-15.634</x>
                <y>-16.299</y>
                <z>4.5</z>
            </RelativePosToBodyCm>
```

```xml
                    <RelativeAngEulerToBodyDeg>
                        <roll>0</roll>
                        <pitch>0</pitch>
                        <yaw>0</yaw>
                    </RelativeAngEulerToBodyDeg>
                    <RotationAxisVectorToBody>
                        <x>0</x>
                        <y>0</y>
                        <z>1</z>
                    </RotationAxisVectorToBody>
                    <RotationModeSpinOrDefect>0</RotationModeSpinOrDefect>
                </Actuator>
                <Actuator>
                    <MeshPath>/Rfly3DSimPlugin/copter/PropellersCW</MeshPath>
                    <MaterialPath></MaterialPath>
                    <RelativePosToBodyCm>
                        <x>16.299</x>
                        <y>−15.634</y>
                        <z>4.5</z>
                    </RelativePosToBodyCm>
                    <RelativeAngEulerToBodyDeg>
                        <roll>0</roll>
                        <pitch>0</pitch>
                        <yaw>0</yaw>
                    </RelativeAngEulerToBodyDeg>
                    <RotationAxisVectorToBody>
                        <x>0</x>
                        <y>0</y>
                        <z>1</z>
                    </RotationAxisVectorToBody>
                    <RotationModeSpinOrDefect>0</RotationModeSpinOrDefect>
                </Actuator>
                <Actuator>
                    <MeshPath>/Rfly3DSimPlugin/copter/PropellersCW</MeshPath>
                    <MaterialPath></MaterialPath>
                    <RelativePosToBodyCm>
                        <x>−16.299</x>
                        <y>15.634</y>
                        <z>4.5</z>
                    </RelativePosToBodyCm>
                    <RelativeAngEulerToBodyDeg>
                        <roll>0</roll>
                        <pitch>0</pitch>
                        <yaw>0</yaw>
                    </RelativeAngEulerToBodyDeg>
                    <RotationAxisVectorToBody>
                        <x>0</x>
                        <y>0</y>
                        <z>1</z>
                    </RotationAxisVectorToBody>
                    <RotationModeSpinOrDefect>0</RotationModeSpinOrDefect>
                </Actuator>
            </ActuatorList>
            <OnboardCameras>
                <camera>
```

```xml
                    <name>Chase_Camera</name>
                    <RelativePosToBodyCm>
                        <x>-70</x>
                        <y>0</y>
                        <z>5</z>
                    </RelativePosToBodyCm>
                    <RelativeAngEulerToBodyDeg>
                        <roll>0</roll>
                        <pitch>0</pitch>
                        <yaw>0</yaw>
                    </RelativeAngEulerToBodyDeg>
                </camera>
                <camera>
                    <name>Front_Camera</name>
                    <RelativePosToBodyCm>
                        <x>10</x>
                        <y>0</y>
                        <z>0</z>
                    </RelativePosToBodyCm>
                    <RelativeAngEulerToBodyDeg>
                        <roll>0</roll>
                        <pitch>0</pitch>
                        <yaw>0</yaw>
                    </RelativeAngEulerToBodyDeg>
                </camera>
                <camera>
                    <name>Back_Camera</name>
                    <RelativePosToBodyCm>
                        <x>-10</x>
                        <y>0</y>
                        <z>0</z>
                    </RelativePosToBodyCm>
                    <RelativeAngEulerToBodyDeg>
                        <roll>0</roll>
                        <pitch>0</pitch>
                        <yaw>180</yaw>
                    </RelativeAngEulerToBodyDeg>
                </camera>
                <camera>
                    <name>Right_Camera</name>
                    <RelativePosToBodyCm>
                        <x>0</x>
                        <y>10</y>
                        <z>0</z>
                    </RelativePosToBodyCm>
                    <RelativeAngEulerToBodyDeg>
                        <roll>0</roll>
                        <pitch>0</pitch>
                        <yaw>90</yaw>
                    </RelativeAngEulerToBodyDeg>
                </camera>
                <camera>
                    <name>Left_Camera</name>
                    <RelativePosToBodyCm>
                        <x>0</x>
```

```
                <y>-10</y>
                <z>0</z>
            </RelativePosToBodyCm>
            <RelativeAngEulerToBodyDeg>
                <roll>0</roll>
                <pitch>0</pitch>
                <yaw>-90</yaw>
            </RelativeAngEulerToBodyDeg>
        </camera>
        <camera>
            <name>Down_Camera</name>
            <RelativePosToBodyCm>
                <x>0</x>
                <y>0</y>
                <z>-10</z>
            </RelativePosToBodyCm>
            <RelativeAngEulerToBodyDeg>
                <roll>0</roll>
                <pitch>-90</pitch>
                <yaw>0</yaw>
            </RelativeAngEulerToBodyDeg>
        </camera>
        <camera>
            <name>Up_Camera</name>
            <RelativePosToBodyCm>
                <x>0</x>
                <y>0</y>
                <z>10</z>
            </RelativePosToBodyCm>
            <RelativeAngEulerToBodyDeg>
                <roll>0</roll>
                <pitch>90</pitch>
                <yaw>0</yaw>
            </RelativeAngEulerToBodyDeg>
        </camera>
    </OnboardCameras>
</vehicle>
```

ClassID (Model Class ID)

The label indicates the aircraft configuration (large style). Currently, ClassID has the following options: 3 (quadrotor), 30 (figure), 40 (calibration plate), 100 (fixed-wing aircraft), 5 and 6 (hexaptor), 60 (illuminant), 151 (square ring), 150 (circular ring), 152 (spherical) and 50 (small car).

DisplayOrder (Display order of homogeneous models)

Indicates the ranking priority in the list of quadrotor optional models (the smaller the rank, the higher the rank, and the ranking is small). The priority of built-in models starts from 1000. If it is less than 1000, it will replace the built-in aircraft, and the latest imported models will be displayed first. For example, DroneyeeX680.xml 1015 in the template corresponds to the 4th ranked aircraft;

The display style of the aircraft is determined by the large style and the small style. Send the vehicleType directly in the command of mav.sendUE4Pos to reach the required style. The format is: large style + small style * 1000.

Name (Model Display Name)

The tag represents the aircraft name displayed in the UE;

Scale (Model displays 3D dimensions)

The label indicates the size of the three-dimensional zooming of the whole machine;

AngEulerDeg (Initial display posture of the model)

Indicates that the aircraft displays a deflection angle (in degrees). For example, if you enter yaw = 180, the initial attitude of the aircraft will turn around and become backward.

B ody (Model body composition)

Is the main label of the airframe

M odel T Ype (activate vehicle blueprint)

When <ModelType>is 3, the vehicle blueprint (the vehicle class WheeledVehiclePawn of the UE) can be activated.

ModelType (Grid Body Type)

Indicates whether it is a mesh that drives the drawing. It is generally necessary to select 0 here, except for some 3D models with their own animation (0: StaticMesh 1: Animation 2: Blue prints). (PS: The tag of this XML used to be isAnimationMesh, which may be displayed if there is an old version of the XML file.)

MeshPath (Model Three dimensions File Path)

Represents the directory where the 3D file of the airframe is located, where/Game/represents the RflySim3D content directory, and DroneyeeX680/DroneyeeX680Body represents the 3D model file of the airframe just copied.

MaterialPath (Material Path)

It indicates the path of the material. If the material has been set in the UE, it can not be filled here. If it is filled, the material will be superimposed on the aircraft.

A nimation P Ath (animation path)

CenterHeightAboveGroundCm (Height of the center of mass of the model from the ground)

Indicates the height of the center of mass from the ground

isMoveBodyCenterAxisCm (Model pivot position)

Indicates to adjust the 3D pivot vectors of the model. Enter 3D vectors separated by commas.

NumberHeigthAboveCenterCm (The model shows the height of the label above the ground)

Indicates the altitude of a numerical label display on an aircraft.

ActuatorList (Actuator display parameter list)

Is a list of externally active (definable) components such as actuators (propellers, steering gears, tires, hose rigid bodies, turrets)

Actuator (An actuator)

Specific parameters of a certain actuator are in the label

MeshPath (Mesh Path)

3D model path, as defined by the body tag

MaterialPath （Material Path）

The material path is defined with the body label

RelativePosToBodyCm （Relative to the center of the body）

Indicates the position of the actuator, such as a propeller, relative to the center of the airf
rame

RelativeAngEulerToBodyDeg （Installation angle）

Indicates the installation angle of the actuator (unit degree), which is usually 0, indicatin
g that the attitude is the same as that when the UE is imported, and there is no need to ro
tate any more.

RotationModeSpinOrDefect （Sport mode）

Represents the actuator motion pattern, 0 represents rotation (input is rotational speed in
revolutions per minute), 1 represents deflection (input is rotational angle in degrees), 2 r
epresents a stationary object, and 3 represents translation in centimeters.

RotationAxisVectorToBody （Rotation axis）

Rotation axis, where the multi-rotor propeller rotates about the z-axis, the y-axis in the c
ase of a vehicle, and the x-axis in the case of a fixed-wing propeller.

OnboardCameras （On-board camera）

Some airborne cameras for observation are defined in the tab. Starting from the second c
amera, they are all fixed on the airframe, so it can be defined by filling in the installation
angle and position of the camera. The first camera is a perspective for observation, whic
h will not follow the pitch and roll of the aircraft, but will follow the yaw of the aircraft.

AttatchToOtherActuator （Additional function of actuat
or）

The label defines the ability of an actuator to attach to another actuator, and when "atta

ched", the actuator moves and rotates with the parent, but it can also rotate independently. <AttatchToOtherActuator>2</AttatchToOtherActuator> means to attach the current actuator to the second actuator in the ActuatorList.

Find XML references for other drones at "\ PX4PSP \ RflySim3D \ RflySim3D \ Plugins \ Rfly3DSimPlugin \ Content \ XML".

8.2　Normal import (Static Mesh/Skeletal Mesh)

8.2.1　A static mesh volume splice model（ModelType The label is taken 0）

The actuator needing to be controlled and the body not needing to be controlled are introduced into the UE together for assembly and material replacement, and after the baking is finished, Copy the content under [Project Name]-> Saved-> Cooked-> WindowsNoEditor-> [Project Name]-> Content and the corresponding XML file to PX4PSP-> RflySim3D-> RflySim3D-> Under Content

Notice

The model imported into the UE shall be in FBX, OBJ and other general 3D formats. Before importing, the actuators to be controlled shall be segmented in the 3D processing software. For example, if a fixed-wing aircraft usually needs to control several control surfaces, the rest parts shall be assigned to the fuselage.

When importing the model into the UE, check the "Combine Mesh" option so that all components are imported as a whole, otherwise the body will be divided into N parts.
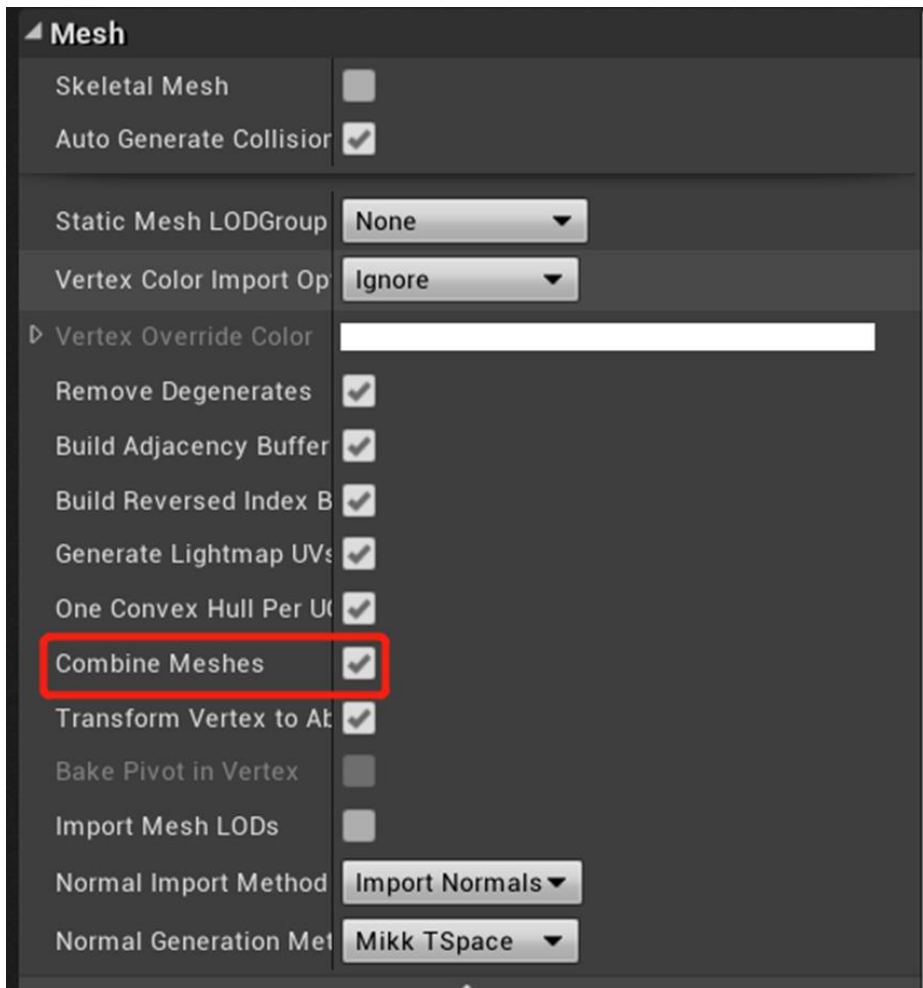
The ModelType tag in the XML file should be 0, corresponding to the static grid body

8.2.2 Built-in animation model （ModelType The label is taken 1）

The skeletal mesh body of the model and the animation sequence are imported into the UE together, and after the baking is finished, Copy the content under [Project Name]-> Saved-> Cooked -> WindowsNoEditor-> [Project Name]-> Content and the corresponding XML file to PX4PSP-> RflySim3D-> RflySim3D-> Under Content.

Notice

The models here are mainly used as some moving obstacles (such as character models). You can directly search for the required models in the Unreal Mall, so that you do not need to carry out skin binding and other operations in the 3D processing software.

The ModelType tag in the XML file should be 1, corresponding to the skeleton mesh with its own animation effect.

8.3   Blueprint import (Normal Blueprint/Vehicle Blueprint) This function is limited to personal advanced version or above.

8.3.1 Use a blank template (take 2 for the ModelType tag)

The skeletal grid body of the model and the animation sequence are imported into the UE together, and a replacement material。 Create for the model animation blueprint is added to define the animation playing effect of each actuator; a blueprint class is created for the model, and the trigger condition and the transmission data interface of the corresponding actuator are controlled in an event chart. Aft baking is complete, Copy the content under [Project Name]-> Saved-> Cooked-> WindowsNoEditor-> [Project Name]-> Content and the corresponding XML file to PX4PSP-> RflySim3D-> RflySim3D-> Under Content.

Notice

The model here is obtained directly from the Unreal Mall, with the already configured animation blueprint and collision model, just modify the blueprint event in the blueprint class.

There are two interfaces for the blueprint logic to access the platform, "Actuator Inputs" "and" Actuator InputsExt " ". These two custom events are created in the event diagram of the blueprint class. The "input" terminal on the right side of the "Actuator Inputs" creates 8-dimensional (PWM 1-pwm8) input signals as the basic 8-dimensional actuator input signals; "ActuatorInputsExt" adds 16-dimensional input signals from pwm9 to pwm24 as the receiver of 9-24-dimensional extended actuator input signals.
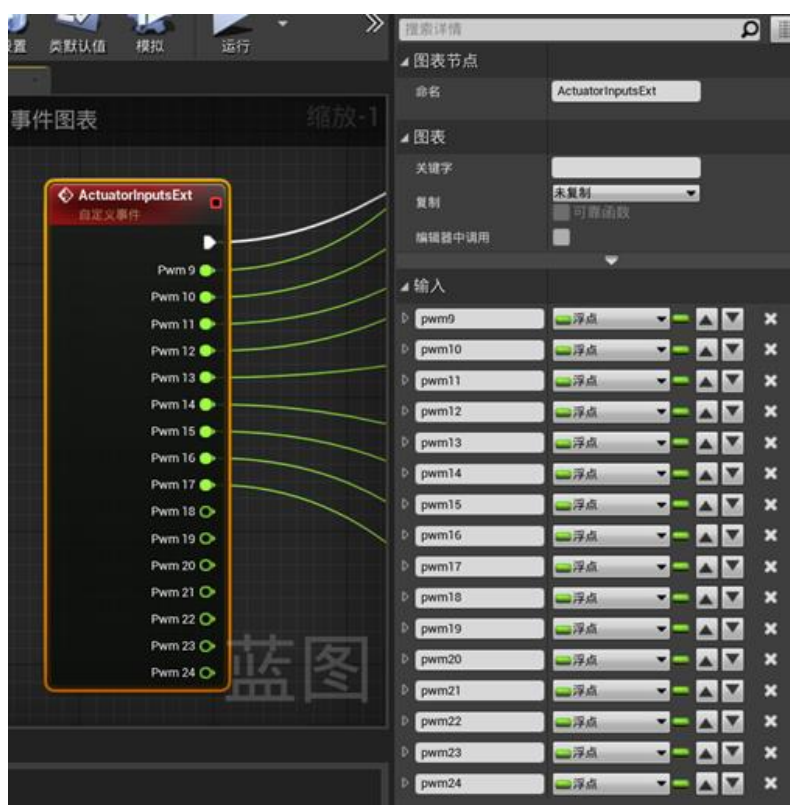
The ModelType tag in the XML file should be taken as 2, corresponding to the blueprint class with its own program logic.

8.3.2 Use the carrier template （M odel T ype The label is taken 3）

Use the carrier template when creating a UE project, and the rest of the steps are the same as using Empty White template the blueprint

Notice

If you use the built-in model of the vehicle template, you need to enable the automatic control AI in the vehicle blueprint class, as shown in the following figure
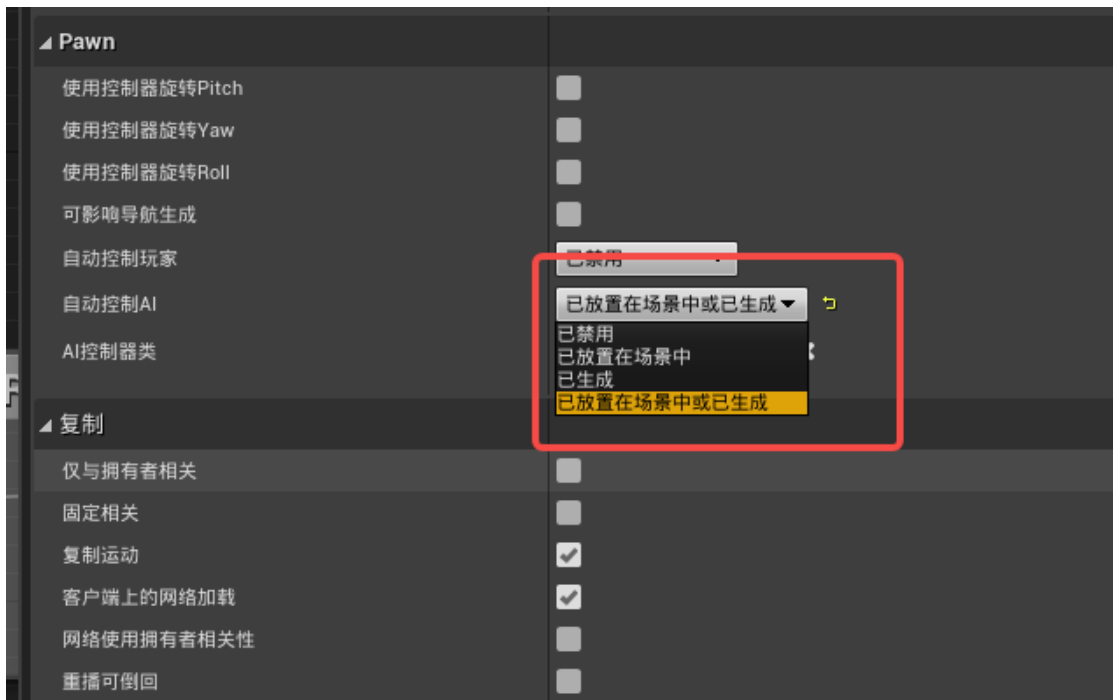
In the XML file, the ModelType tag is used instead of the ModelType tag, and the correspond ing value is 3, which corresponds to the carrier blueprint class

8.4   Blueprint control rules (Personal Premium and above only)

Each object displayed in the 3D scene is an object, and they also have member variables and member functions. The UAVs we control are the same. These objects are defined in UnrealEngine. If they are objects inherited from the UE blueprint class, the platform provides an interface to call the blueprint function. The interfaces provided by RflySim3D for blueprints mainly include "Act uator Inputs" "and" Actuator InputsExt "". These two interfaces can transmit a set of data to t he designated UAV in the scene, and can be used to display some user-defined effects (the effects n eed to be written by the blueprint system itself). For example, open the door, explode, display text, switch the material of the aircraft, rotate the wings and so on, anything that UE blueprint system c an do. It is a higher-level interface, so it also requires a basic grasp of the blueprint usage of the U E.

A blueprint interface is a function that calls the blueprint from the outside, thereby triggering the various functions written in the blueprint. This can be done with the two RflySim3D console c ommands described earlier: "RflySetActuatorPWMs" and "RflySetActuatorPWMsExt". You can also use other programs (such as Python, Simulink) to send the specified structure over UDP.

8.4.1 Actuator Inputs Interface

sendUE4Pos Series of python commands

For usage, see sendUE4Pos (Create / Update Model)

RflySetActuatorPWMs Console commands

For usage, see RflySetActuatorPWMs (Trigger Blueprint Interface)

UE4DataEncoder (Simulink submodule)

For usage, see UE4DataEncoder

8.4.2 ActuatorInputsExt Interface

sendUE4ExtAct (Python command)

For usage, see sendUE4ExtAct (Trigger Extension Blueprint Interface)

RflySetActuatorPWMsExt Console commands

For usage, see RflySetActuatorPWMsExt (Trigger Extension Blueprint Interface)

Ue4ExtMsgEncoder (Simulink submodule)

For usage, see Ue4ExtMsgEncoder

9. External interaction interface

9.1 Communication port

9.1.1 Network communication

RflySim 3D Receive

Multicast IP Address and Port 224.0.0.11： 20008

The data source is restricted to the local machine (127.0.0.1). At present, CopterSim will send data to RflySIm3D through this interface by default when the online option is not checked.

SocketReceiver1 Class

Receive 224.0.0.11: 20008 UDP packets (although it is multicast, it is set to only accept packets of the local loopback address), and also receive packets of the 20009 port of the local machine (multicast is also acceptable)

Multicast IP Address and Port 224.0.0.10： 20009

In the source code JoinedToGroup, the data source is set to be arbitrary, so it can receive messages from the LAN and perform distributed online simulation. At present, when the online option is checked, CopterSim will send data to RflySIm3D through this interface by default.

SocketReceiver Class

Receives 224.0.0.10: UDP packets from the 20009, as well as packets from the local 20009 port (not multicast is acceptable)

Native 20010 + + 1 series port (20010 to 20029)

On a computer, the first RflySim3D running will listen directly to this interface (20010, the window name is RflySim3D-0). When the second window rebinds the 20010, it will

find that it is occupied, so it will subscribe to the next port (20011, the window name is RflySim3D-1). And so on, up to 20 windows, that is, up to the 20029, RflySim3D-19. Python's UE4 series function interface, through which the function of specifying the RflySim3D window is implemented. If you want to send to RflySim3D-i, use the port 20010 + I.

SocketReceiverMap Class

Receive one of the local 20010 ~ 20029 ports (this is because multiple RflySim3D on a computer will compete for the port, so the 20 ports will be allocated in turn according to the serial number)

RflySim3D Send

S endersocket Class

Multicast ＩP Address and Port 224.0.0.10：20002

Ue4Req Structure

See for details Ue4Req (Receive / Respond to inquiries)

UTF8 A string of characters

Multicast ＩP Address and Port 224.0.0.10： 20006 （Python、Simulink)

Used for RflySim3D to send data to Python and Simulink to receive. Comprises the following structures:

reqVeCrashData (Crash data Structure ）

See for details reqVeCrashData (Crash data)

CopterSimCrash Structure

See for details [CopterSimCrash](#)

Copt R eq D ata

See for details [CoptReqData（Response received）ReqObjData ）](#)

ObjReqData（Object information data）Structure ）

See for details [ObjReqData（Response received）ReqObjData ）](#)

CameraData（Camera Data Structure)

See for details [CameraData（Response received）ReqObjData ）](#)

9999 + + 1 Series Port（Return image)

The series of ports that return images are defined in the JSON of the image sensor, usually in the form of 9999 + I, which can be modified by JSON. If an image chooses the form of network transmission, it will send the compressed image to this port.

30100 + + 2 Series Port (CopterSim)

1) CopterSim and RflySim3D have a mechanism to respond to requests. CopterSim sends one Ue4Req Structure to RflySim3D, which will set the reqIndex to 1 and return the interface body. Form a handshake mechanism. Usually, the first CopterSim window will send this message through the 20008 to confirm whether RflySim3D has received 3D data. If there is no feedback after the timeout, CopterSim will switch to the 20010 port to transmit data to RflySim3D.

2) The collision data (Ue4RayTraceDrone structure) of the returned CopterID aircraft is sent to the designated CopterSim through the 30100 + 2 * CopterID-2 port to receive the collision data.

3) Return crash data

Ue4Req Structure

See for details

Ue4RayTraceDrone Structure

See for details

9.1.2 Shared memory

UE4CommMemData （32 Vision sensors)

See for details

RflySim3DImg_i (No ⅰ Vision sensors)

9 .1.3 R edis Communication

9.2 Data Protocol (UDP interface)

9.2.1 Send

Image

ImageHeaderNew (image data structure)

```
struct imageHeaderNew {
 int checksum = 0;//1234567890;
 int PackLen = 0;
 int PackSeq = 0;
 int PackNum = 0;
 double timeStmp = 0;
};
```

Explanation of action:

The imageHeaderNew structure is used to represent the header information of the image, and it can be used to store the identification information related to the image. This struc ture can be part of the data, and when it is sent, it is followed by up to 60000 bytes of im

age data (so each packet is 60000 + 4 × 4 + 8 × 1), and there may be multiple image p ackets in 60000 bytes, each preceded by this structure.

Explanation of parameters:

- Check sum: This parameter is used to store an integer value and is typically us ed for data integrity checks. A checksum is a value generated by performing arithm etic operations on data to detect errors or corruption during transmission or storage.

- PackLen: This parameter indicates the length of the packet and is usually an in teger value. It can be used to indicate the size or length information of the data cont ained in the packet.

- PackSeq (packet sequence number): This parameter is used to store the sequen ce number of the packet, usually an integer value. Sequence numbers can help resto re the correct order of packets at the receiving end, especially when packets may be out of order as they travel through the network.

- PackNum: Similar to PackSeq, this parameter is an integer that may be used to identify the number of a packet or other similar information.

- TimeS TMP: This parameter is a double-precision floating-point number used to store timestamp information. The time stamp typically indicates the time of som e event or data, possibly recording the time of generation of the image header infor mation or other time-related information.

UE4CommMemData (Image Verification Data)

```
struct UE4CommMemData{
Int checksum;//Check digit, set to 1234567890.
Int totalNum;//maximum number of sensors
Int Width Heigh [64];//resolution wide-high sequence containing up to 32 sensor
}
```

Function explanation

Similar to imageHeaderNew, the platform now also stores the image's verification data in a shared memory named UE4CommMemData.

The UE4CommMemData structure is used to represent memory information related to U E4 (Unreal Engine 4) communication and sensor data management. This structure can b e used to store configuration information related to communication and sensor data proc essing.

Explanation of parameters:

- Check sum: This parameter is used to store the check bit, which is usually used

for data integrity checking. The default value of the check digit is set to the 12345 67890, which is a preset check value. Check bits are used to verify that data has not been corrupted or tampered with during transmission or storage.

- TotalNum (maximum number of sensors): This parameter is an integer representing the maximum number of sensors available in the system. It may be used to configure or document the number of sensors supported by the system.

- Width Height (Resolution Width Height Sequence): This parameter is an array of integers up to 64 elements. It is used to store the resolution width and height sequence of the sensor. This array can contain resolution information for up to 32 sensors.

Ask + Heartbeat

Ue4Req (receiving/responding to queries)

```
struct Ue4Req {
 int checksum;
 int reqIndex ;
}
```

Function explanation

The Ue4Req struct is used to initiate requests and receive responses to establish a handshake mechanism for communication between CopterSim and RflySim3D. It allows CopterSim to send a request to RflySim3D and confirm whether RflySim3D has successfully received the request. When the structure of the CopterSim is received, an identical structure is returned to the CopterSim

Parameter interpretation

- Check sum: This parameter is an integer used to store the check digit. Check bits are often used for data integrity checks to ensure that the received data has not been tampered with. Here, it may be used to verify the integrity of the Ue4Req struct.

- ReqIndex: This parameter is also an integer that represents the identity of the request.

  0: No aircraft data

  1: Existing aircraft data

ReqVeCrashData (crash data)

```
struct reqVeCrashData {
int checksum; //Packet check code 1234567897.
int copterID; //ID number of the current aircraft
int vehicleType; //The style of the current aircraft
Int CrashType;//Collision object type, -2 indicates ground, -1 indicates scene static obj
ect, 0 indicates no collision, and above 1 indicates ID number of the collided aircraft
double runnedTime; //Timestamp of the current aircraft
float VelE[3]; //Speed of the current aircraft
float PosE[3]; //Current aircraft position
Float CrashPos [3];//Coordinates of the collision point
Float targetPos [3];//the center coordinate of the touched object
float AngEuler[3]; //Euler angle of the current aircraft
float MotorRPMS[8]; //Current aircraft motor speed
float ray[6]; //front, back, left, right, up and down scan lines of the aircraft
Char CrashedName [20] = { 0 };//Name of the touched object
}
```

Explanation of action:

The purpose of the reqVeCrashData structure is to store data related to aircraft collisions. The fields of this structure include various details of the collision, such as collision type, timestamp, position coordinates, speed, attitude, etc.

Explanation of parameters:

- The check sum is used to verify the integrity of the data to ensure that the data has not been tampered with during transmission. Its value is the 1234567897.

- CopterID (the ID number of the aircraft): used to uniquely identify different aircraft.

- VehicleType: Indicates the style or type of the current aircraft.

- CrashType: indicates the type of colliding object, -2 indicates the ground, -1 indicates a scene static object, 0 indicates no collision, and a value above 1 indicates the ID number of the collided aircraft.

- RunnedTime (the timestamp of the current aircraft): Record the time at which the collision occurred.

- VelE (Velocity of the current aircraft): An array of 3 floating-point numbers representing the velocity components of the current aircraft, typically including eastbound, northbound, and ascending velocities.

- PosE (Position of the current aircraft): An array of 3 floating point numbers representing the position coordinates of the current aircraft, usually including east, north, and ascent positions.

- CrashPos (the coordinates of the collision point): An array of 3 floating-point numbers representing the coordinates of where the collision occurred.

- TargetPos: An array of 3 floating-point numbers representing the center coordinates of the object being bumped.

- AngEuler (Euler angles of the current aircraft): An array of 3 floating-point numbers describing the Euler angles of the current aircraft, typically including pitch, yaw, and roll angles.

- MotorRPMS (Motor RPM of the current aircraft): An array of 8 floating point numbers representing the RPM of each motor of the current aircraft.

- Ray (front, back, left, right, up and down scan lines of the aircraft): This is an array containing 6 floating point numbers, which is used to describe the data of the aircraft scanning in the front, back, left, right, up and down directions.

- CrashedName: This is a 20-character array that stores the name of the object being bumped.

Ue4RayTraceDrone structure (sent to CopterSim to which each Copter belongs)

```
struct Ue4RayTraceDrone {
 int checksum;
 int CopterID;
 float size;
 float velE[3];
Float ray [6];//front, back, left, right, up and down
 float posE[3];
}
```

Explanation of action:

It is used to store the key data of the ray-tracing UAV. Ray tracing UAVs are usually used for environment awareness, obstacle avoidance, collision detection and other tasks. These tasks require the acquisition of UAV status information and ray tracing data.

Explanation of parameters:

- Check sum: This integer field is used to store the check sum of the packet to verify the integrity of the data. It helps ensure that the received data has not been corrupted or tampered with in transit.

- CopterID: This integer field represents the unique identification number of the current ray-tracing UAV. Each UAV should have a unique ID number to identify and track different UAVs.

- Size: This floating-point field represents the size or size of the drone. The size information can be used for obstacle avoidance and collision detection to ensure that the UAV can safely pass through narrow passages or avoid collision with objects.
- VelE (velocity): This is an array of 3 floating-point numbers representing the velocity component of the drone. Typically, this includes eastbound velocity, northbound velocity, and ascent velocity.
- Ray: This is an array of 6 floating-point numbers representing the ray-traced data of the drone in the forward, backward, left, right, up, and down directions. Ray tracing data is used to detect objects in the surrounding environment to help UAV avoid obstacles and detect collisions.
- PosE (position): This is an array of 3 floating point numbers representing the current position coordinates of the drone. Typically, this includes an east position, a north position, and a rise position.

Camera

Camera Data (in response to ReqObjData received)

```
struct CameraData { //56
 int checksum = 0;//1234567891
int SeqID; //camera serial number
Int TypeID;//camera type
Int Data Height;//pixel height
Int Data Width;//pixel width
Float Camera FOV;//camera field angle
float PosUE[3]; //Camera center position
Float angEuler [3];//camera Euler angle
Double times TMP;//timestamp
}
```

Explanation of action:

The main function of the CameraData structure is to store the data related to the camera for subsequent operations such as analysis and processing.

Explanation of parameters:

- Check sum: This integer field is used to store the check sum of the packet to verify the integrity of the data.
- SeqID (camera serial number): This integer field represents the serial number or identification of the camera, which is used to distinguish different camera devices. If there are multiple cameras, this field can help identify which camera is generating the data.

- TypeID (camera type): This integer field represents the type or kind of camera and can be used to identify different kinds of camera devices.
- DataHeight: This integer field represents the height of the image data, that is, the number of pixel rows of the image. This parameter defines the resolution of the image.
- DataWidth: This integer field represents the width of the image data, that is, the number of columns of pixels in the image. This parameter also defines the resolution of the image.
- CameraFOV: This floating-point field represents the field of view of the camera, usually in degrees. It describes the horizontal range that the camera can cover and is important for the camera's field of view setting.
- PosUE (Camera Center Position): This is an array of 3 floating point numbers representing the center position coordinates of the camera. These coordinates represent the position of the camera in the UE coordinate system
- AngEuler (camera Euler angles): This is an array of 3 floating-point numbers representing the Euler angles of the camera, typically including pitch, yaw, and roll angles. These angles describe the orientation of the camera.
- Times TMP: This double-precision floating-point number field represents the time stamp of the data and is used to record the time of data collection. Timestamps are commonly used for data synchronization, time-related analysis, and event time recording.

Model

CoptReqData (in response to ReqObjData received)

```
struct CoptReqData { //64
int checksum = 0; //123456 78 91 as check
Int CopterID;//Aircraft ID
float PosUE[3]; //Object center position (specified during artificial 3D modeling, the attitude coordinate axis is not necessarily at the geometric center)
Float angEuler [3];//Euler angle of object
Float boxOrigin [3];//object geometric center coordinate
Float Box Extent [3];//half of the length, width and height of the object frame
Double times TMP;//timestamp
};
```

Parameter interpretation

- Check sum: This integer field is used to store the check sum of the packet. The

default value is 0, but the note mentions that the example check sum value is 1234 5678 91. Check codes are often used to verify the integrity of data to ensure that it has not been tampered with during transmission.

- CopterID: This integer field represents the unique identification number of the aircraft and is used to distinguish between different aircraft.

- PosUE (object center position): This is an array of 3 floating-point numbers representing the coordinates of the object's center position.

- AngEuler (Euler angles of an object): This is an array of 3 floating-point numbers representing the Euler angles of an object, typically including pitch, yaw, and roll angles. These angles describe the orientation of the object.

- BoxOrigin (object geometric center coordinates): This is an array of 3 floating-point numbers representing the geometric center coordinates surrounding the box.

- Box Extent: This is an array of 3 floating-point numbers that represent half the length, width, and height of the object's bounding box. This information is useful for describing the bounding box of an object.

- Times TMP: This double-precision floating-point number field represents the time stamp of the data and is used to record the time of data collection. Timestamps are commonly used for data synchronization, time-related analysis, and event time recording.

CopterSimCrash

```
struct CopterSimCrash {
int checksum = 0; //123456 78 91 as check
Int CopterID;//Aircraft ID
int TargetID; //Target Aircraft ID
};
```

A static object

ObjReqData (in response to ReqObjData received)

```
struct ObjReqData { //96
int checksum = 0; //123456 78 91 as check
 int seqID = 0;
float PosUE[3]; //Object center position (specified during artificial 3D modeling, the at
titude coordinate axis is not necessarily at the geometric center)
Float angEuler [3];//Euler angle of object
Float boxOrigin [3];//object geometric center coordinate
```

```
Float Box Extent [3];//half of the length, width and height of the object frame
Double times TMP;//timestamp
Char ObjName [32] = { 0 };//Name of object touched
};
```

9.2.2 Receive

Individual aircraft data

SOut2Simulator (stand-alone data 1)

```
struct SOut2Simulator {
 int copterID;
 int vehicleType;
 double runnedTime;
 float VelE[3];
 float PosE[3];
 float AngEuler[3];
 float AngQuatern[4];
 float MotorRPMS[8];
 float AccB[3];
 float RateB[3];
 double PosGPS[3];
}
```

Function explanation

This data can be used to update the state of individual aircraft, calculate new attitudes and positions, and render them in the next frame.

Parameter interpretation

- CopterID: This integer field is used to identify the unique identifier of the aircraft in order to distinguish one aircraft from another.

- VehicleType: This integer field represents the configuration category of the aircraft to distinguish different types of aircraft, such as helicopters, fixed-wing aircraft, etc.

- RunnedTime (Timestamp): This double-precision floating-point number field represents the time, in seconds, that the aircraft has been running.

- VelE (Velocity Vector): This is an array of 3 floating point numbers representing the velocity vector of the aircraft. The velocity component in the NED coordinate system that describes the current velocity of the aircraft.

- PosE (Position Vector): This is an array of 3 floating point numbers representing the position vector of the aircraft. It usually includes the position coordinates of north, east and ground, which are used to describe the position of aircraft in the curr

ent frame.

● AngEuler: This is an array of 3 floating point numbers representing the Euler angles of the aircraft. Pitch, yaw, and roll angles are typically included to describe the attitude of the aircraft.

● AngQuatern (Quaternion Angular Representation): This is an array of 4 floating point numbers that accurately represents the attitude angle of the aircraft.

● MotorRPMS (Motor RPM): This is an array of 8 floating point numbers representing the RPM of the individual motors of the aircraft.

● AccB (Acceleration in Body Coordinate System): This is an array of 3 floating point numbers representing the acceleration of the aircraft in the body coordinate system. Include fore-and-aft, left-and-right, and up-and-down acceleration components to describe the acceleration state of the aircraft.

● RateB (angular velocity in body coordinates): This is an array of 3 floating-point numbers representing the angular velocity of the aircraft in body coordinates. Angular velocity is used to describe the rotational speed of the aircraft about the various axes.

● PosGPS (GPS position): This is an array of 3 double-precision floating-point numbers representing the aircraft's Global Positioning System (GPS) position information. Includes latitude, longitude, and altitude, and is used to determine the geographic location of the aircraft.

SOut2SimulatorS imple (stand-alone data 2)

```
struct SOut2SimulatorSimple {
 int checkSum;
 int copterID;
 int vehicleType;
float  MotorRPMSMean ;
 float PosE[3];
 float AngEuler[3];
}
```

Each parameter has the same meaning as in " SOut2Simulator ".

● The checksum is the data check bit, the 1234567890 indicates normal data, and the 1234567891 indicates that the aircraft is always close to the ground

● MotorRPMSMean means to input 1-D motor data, which will be stored in 8-D motor simultaneously

SOut2SimulatorSimpleTimeF （Stand-alone data 3）

```
struct SOut2SimulatorSimpleTimeF {
 int checkSum;
 int copterID;  //Vehicle ID
 int vehicleType;  //Vehicle type
 float PWMs[8];
 float PosE[3];   //NED vehicle position in earth frame (m)
 float VelE[3];
 float AngEuler[3];  //Vehicle Euler angle roll pitch yaw (rad) in x y z
 double runnedTime;
}
```

Each parameter has the same meaning as in ” SOut2Simulator ”.

- The checksum is the data check bit, the 1234567890 indicates normal data, and the 1234567891 indicates that the aircraft is always close to the ground

SOut2SimulatorS impleTime (stand-alone data 4)

```
struct SOut2SimulatorSimpleTime {
 int checkSum; //1234567890
 int copterID;
 int vehicleType;
 float PWMs[8];
 float VelE[3];
 float AngEuler[3];
 double PosE[3];
 double runnedTime;
}
```

Each parameter has the same meaning as in ” SOut2Simulator ”.

- The checksum is the data check bit, the 1234567890 indicates normal data, and the 1234567891 indicates that the aircraft is always close to the ground

SOut2SimulatorS imple1 (stand-alone data 5)

```
struct SOut2SimulatorSimple1 {
 int checkSum;
 int copterID;
 int vehicleType;
 float MotorRPMSMean;
 float PosE[3];
 float AngEuler[3];
 float Scale[3];
}
```

Each parameter has the same meaning as in ” SOut2Simulator ”.

- The checksum is the data check bit, the 1234567890 indicates normal data, and

the 1234567891 indicates that the aircraft is always close to the ground

- MotorRPMSMean means to input 1-D motor data, which will be stored in 8-D motor simultaneously

- Scale [3] denotes the scaling scale in the XYZ direction.

Short packet

_netDataShort

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

Parameter interpretation

- The payload is interpreted as data of the ” SOut2Simulator ” structure

Multi-machine data

Multi3DData (20 machine data 1)

```
struct Multi3DData {
 uint16 IDs[20];
 uint16 VehileTypes[20];
 float PosE[60];
 float AngEuler[60];
}
```

Function explanation

It will go through a for loop, put it into the array in turn, and update the 20 aircraft data in the next frame.

Parameter interpretation

- IDs (Aircraft IDs Array): This is an array of 20 unsigned integers (uint16) used to store unique identification numbers for multiple aircraft. Each element corresponds to an aircraft ID, which can be used to distinguish different aircraft.

- VehileTypes (Aircraft Types Array): This is an array of 20 unsigned integers (uint16) used to store the types or categories of multiple aircraft. The element corresponding to each aircraft ID represents the type of the corresponding aircraft.

- PosE (Position Vector Array): This is an array of 60 single-precision floating point numbers (floats) used to store position information for multiple aircraft. Typica

lly, each set of three consecutive elements represents the position vector of an aircraft, using NED coordinates. Therefore, this array can store the position information of 20 aircraft.

- AngEuler (Euler angle array): This is an array of 60 single-precision floating point numbers (floats) used to store Euler angle information for multiple aircraft. Similar to the position vector, each set of three consecutive elements represents the Euler angles of an aircraft, which typically include pitch, yaw, and roll angles. Therefore, this array can store Euler angle information for 20 aircraft.

Multi3DDataNew （20 Machine data 2）

```
struct Multi3DDataNew {
 int checksum;
 uint16 IDs[20];
 uint16 VehileTypes[20];
 float PosE[60];
 float AngEuler[60];
 double runnedTime[20];
}
```

Similar to Multi3DData （20 Machine data 1）

- The checksum is the data check bit, the 1234567890 indicates normal data, and the 1234567891 indicates that the aircraft is always close to the ground

Multi3DData1 （20 Machine data 3）

```
struct Multi3DData1 {
 uint16 IDs[20];
 uint16 VehileTypes[20];
 float PosE[60];
 float AngEuler[60];
 uint16 ScaleXYZ[60];
}
```

Similar to Multi3DData （20 Machine data 1）

- ScaleXYZ [60] represents the scaling scale of 20 aircraft in the XYZ direction.

Multi3DData1New （20 Machine data 4）

```
struct Multi3DData1New {
 int checksum;
 uint16 IDs[20];
 uint16 VehileTypes[20];
 float PosE[60];
 float AngEuler[60];
```

```
uint16 ScaleXYZ[60];
}
```

Similar to Multi3DData （20 Machine data 1）

- The checksum is the data check bit, the 1234567890 indicates normal data, and the 1234567891 indicates that the aircraft is always close to the ground
- ScaleXYZ [60] represents the scaling scale of 20 aircraft in the XYZ direction.

Multi3DData2 （20 Machine data 5）

```
struct Multi3DData2 {
 uint16 IDs[20];
 uint16 VehileTypes[20];
 float PosE[60];
 float AngEuler[60];
 uint16 ScaleXYZ[60];
 float PWMs[160];
}
```

Similar to Multi3DData （20 Machine data 1）

- ScaleXYZ [60] represents the scaling scale of 20 aircraft in the XYZ direction.
- PWMs [160] represent 8-dimensional motor data for 20 aircraft

Multi3DData2New (20 machine data 6)

```
struct Multi3DData2New {
 int checksum;
 uint16 IDs[20];
 uint16 VehileTypes[20];
 float PosE[60];
 float AngEuler[60];
 uint16 ScaleXYZ[60];
 float PWMs[160];
}
```

Similar to Multi3DData （20 Machine data 1）

- The checksum is the data check bit, the 1234567890 indicates normal data, and the 1234567891 indicates that the aircraft is always close to the ground
- ScaleXYZ [60] represents the scaling scale of 20 aircraft in the XYZ direction.
- PWMs [160] represent 8-dimensional motor data for 20 aircraft

Multi3DData20Time （20 Machine data 7）

```
struct Multi3DData20Time {
 int checksum;
 uint16 IDs[20];
```

```
    uint16 VehileTypes[20];
    float PosE[60];
    float VelE[60];
    float AngEuler[60];
    float PWMs[160];
    double runnedTime[20];
}
```

Similar to Multi3DData （20 Machine data 1）

- The checksum is the data check bit, the 1234567890 indicates normal data, and the 1234567891 indicates that the aircraft is always close to the ground
- PWMs [160] represent 8-dimensional motor data for 20 aircraft

Multi3DData10Time （10 Machine data)

```
struct Multi3DData10Time {
 int checksum;
 uint16 IDs[10];
 uint16 VehileTypes[10];
 float PosE[30];
 float VelE[30];
 float AngEuler[30];
 float PWMs[80];
 double runnedTime[10];
}
```

Function explanation

Store data for each frame of 10 aircraft

Parameter interpretation

- Check sum: This integer field is used to store the checksum of the data to ensure its integrity and correctness. The 1234567890 indicates normal data, and the 1234567891 indicates that the aircraft is always close to the ground.
- IDs (Aircraft IDs Array): This is an array of 10 unsigned integers (uint16) used to store unique identification numbers for multiple aircraft. Each element corresponds to an aircraft ID, which can be used to distinguish different aircraft.
- VehileTypes (Aircraft Types Array): This is an array of 10 unsigned integers (uint16) that stores the categories of 10 aircraft. The element corresponding to each aircraft ID represents the type of the corresponding aircraft.
- PosE (Position Vector Array): This is an array of 30 single-precision floating point numbers (floats) used to store position information for multiple aircraft. The position information for each point in time includes 10 aircraft, and typically each set of three consecutive elements represents the position vector, NE, of an aircraft. Therefore, this array can store the position information of 10 aircraft in the same frame.

- VelE (Velocity Vector Array): This is an array of 30 single-precision floating point numbers (floats) used to store velocity information for multiple aircraft. The velocity information for each frame includes 10 aircraft, and each set of three consecutive elements represents the velocity vector of one aircraft, including the NE velocity component. Therefore, this array can store the speed information of 10 aircraft in the same frame.

- AngEuler (Euler angle array): This is an array of 30 single-precision floating point numbers (floats) used to store Euler angle information for multiple aircraft. The Euler angle information for each frame includes 10 aircraft, and each set of three consecutive elements represents the Euler angles of an aircraft, typically including pitch, yaw, and roll angles. Therefore, this array can store the Euler angle information of 10 aircraft in the same frame.

- PWMs (PWM signal array): This is an array containing 80 single-precision floating point numbers (float), which is used to store PWM (pulse width modulation) signal information of multiple aircraft. The PWM signal information of each frame includes 10 aircraft, and each group of 8 consecutive elements represents the PWM signal of one aircraft. PWM signals are used to control components such as motors and control surfaces of the aircraft.

- RunnedTime (elapsed time array): This is an array of 10 double-precision floating point numbers (doubles) that stores the timestamp for each frame. Each element represents the elapsed time, typically in seconds, at the corresponding point in time.

Multi3DData100 (100 machine data 1)

```
struct Multi3DData100 {
 uint16 IDs[100];
 uint16 VehileTypes[100];
 float PosE[300];
 float AngEuler[300];
 uint16 ScaleXYZ[300];
 float MotorRPMSMean[100];
}
```

Function explanation

Store every frame of data for 100 aircraft

Parameter interpretation

- IDs (Aircraft IDs Array): This is an array of 100 unsigned integers (uint16) used to store unique identification numbers for multiple aircraft. Each element corresp

onds to an aircraft ID, which can be used to distinguish different aircraft.

● VehileTypes (Aircraft Types Array): This is an array of 100 unsigned integers (uint16) used to store the types or classes of multiple aircraft. The element corresponding to each aircraft ID represents the type of the corresponding aircraft.

● PosE (Position Vector Array): This is an array of 300 single-precision floating point numbers (floats) used to store position information for multiple aircraft. In general, each set of three consecutive elements represents the position vector of an aircraft, including the coordinates in the north, east, and ground directions. Therefore, this array can store the position information of 100 aircraft.

● AngEuler (Euler angle array): This is an array of 300 single-precision floating point numbers (floats) used to store Euler angle information for multiple aircraft. Similar to the position vector, each set of three consecutive elements represents the Euler angles of an aircraft, which typically include pitch, yaw, and roll angles. Therefore, this array can store Euler angle information for 100 aircraft.

● ScaleXYZ: This is an array of 300 unsigned integers (uint16) that stores scale information for each aircraft. The scale bar is used to determine the size of the model in the simulation. Each set of three consecutive elements represents the scale information of an aircraft, including the scales of the X, Y, and Z axes.

● MotorRPMSMean: This is an array of 100 single-precision floating point numbers (floats) that stores the average motor speed for each aircraft. Each element represents the average motor speed of the corresponding aircraft in revolutions per minute (RPM).

Multi 3DData 100New (100 machine data 2)

```
struct Multi3DData100New {
 int checksum;
 uint16 IDs[100];
 uint16 VehileTypes[100];
 float PosE[300];
 float AngEuler[300];
 uint16 ScaleXYZ[300];
 float MotorRPMSMean[100];
}
```

Similar to Multi3DData100 （100 Machine data 1）

● The checksum is the data check bit, the 1234567890 indicates normal data, and the 1234567891 indicates that the aircraft is always close to the ground

Multi3DData100Time （100 Machine data 3）

```
struct Multi3DData100Time {
 int checksum;
 uint16 IDs[100];
 uint16 VehileTypes[100];
 float PosE[300];
 float VelE[300];
 float AngEuler[300];
 float PWMs[800];
 double runnedTime[100];
}
```

Similar to Multi3DData100 （100 Machine data 1）, but with an extra timestamp

- The checksum is the data check bit, the 1234567890 indicates normal data, and the 1234567891 indicates that the aircraft is always close to the ground

Command line for RflySim3D Data

Ue4CMD

```
struct Ue4CMD {
 int checksum;
 char data[252];
}
```

Function explanation

Deposit 1 RflySim_3D_Console commands

Parameter interpretation

- Checksum: This is an integer field that may be used to store a checksum of the console command data to ensure data integrity and correctness.
- Data: This is a character array of length 252 used to store console command data.

```
struct Ue4CMD {
 int checksum;
 char data[52];
}
```

Ue4CMD N et

```
Struct Ue 4 CM DNet {//Total length is 96
int checksum; //check digit, here should be the 1234567897.
              char data[92];
          } i92s
```

Function explanation

Deposit 1 RflySim 3D Console commands

Parameter interpretation

- Checksum: This is a 4-byte integer that stores the checksum of the console command data to ensure data integrity and correctness.

- Data: a 92-byte character array used to store command data. This is where the actual commands that will be sent to RflySim3D are stored.

Image data

VisionSensorReq

```
struct VisionSensorReq {
uint16 checksum; //Data check bit, 12345.
uint16 SeqID; //Memory serial number ID
uint16 TypeID; //Sensor type ID
uint16 TargetCopter; //Bound target aircraft//Changeable
uint16 TargetMountType; //The type of binding//can be changed
uint16 DataWidth; //data or image width
uint16 DataHeight; //Data or image height
uint16 DataCheckFreq; //Check the data update frequency
uint16 SendProtocol[8]; //Transport type (shared memory, UDP transport without compressio
n, UDP video streaming), IP address, port number..
float CameraFOV; //Camera field of view (vision sensors only)//changeable
float SensorPosXYZ[3]; //Sensor mounting position//changeable
float SensorAngEular[3]; //sensor mounting angle//changeable
float otherParams[8]; //reserved eight data bit
}
```

Function explanation

It is used to represent the request information of a vision sensor, including various parameters and attributes of the sensor. After receiving this information, send the image data packet: imageHeaderNew

Parameter interpretation

- Check sum: This is an unsigned integer (uint16) that stores the checksum of the data to ensure its integrity and correctness.

- SeqID (Memory Sequence ID): This is an unsigned integer (uint16) that identifies the sequence number or ID of the request, up to 32.

- TypeID (Sensor Type ID): This is an unsigned integer (uint16) that identifies the type or class of the sensor. TypeID belongs to [1, 6], 456 is LIDAR

> 1: RGB map (free version only supports RGB map),
>
> 2: depth map,
>
> 3: grayscale map,

4: that coordinate system of the point cloud relative to the body,

5: Point cloud relative to geodetic coordinate system;

6: Petal scanning point cloud relative to the body coordinate system

● TargetCopter: This is an unsigned integer (uint16) that specifies the target aircraft to which the sensor is bound.

● Target MountType: This is an unsigned integer (uint16) that specifies the binding type of the sensor.

● DataWidth (data or image width): This is an unsigned integer (uint16) that represents the width of the data or image generated by the sensor.

● DataHeight (data or image height): This is an unsigned integer (uint16) that represents the height of the data or image generated by the sensor.

● DataCheckFreq: This is an unsigned integer (uint16) that indicates how often to check for data updates.

● SendProtocol (transport type): This is an array of 8 unsigned integers (uint16) that specify the transport type of the data and the associated parameters. Including information such as transmission mode (shared memory, UDP transmission, etc.), IP address, port number, etc.

Bit 1 indicates the transmit mode

0: Shared memory (free version only supports shared memory),

1: UDP direct PNG compression,

2: The UDP direct image is not compressed.

3: UDP direct JPG compression

Bits 2-5 represent the IP addr

Bit 6 indicates the port number

● CameraFOV (Camera FOV): This is a single-precision floating point number (float), valid only when the sensor type is a vision-type sensor, that represents the angle of the camera's field of view.

● SensorPosXYZ: This is an array of 3 single-precision floating-point numbers (floats) representing where the sensor is mounted, including X, Y, and Z coordinates relative to the aircraft.

● SensorAngEular: This is an array of 3 single-precision floating-point numbers (floats) representing the sensor's mounting angle, including pitch, yaw, and roll angles.

● OtherParams (reserved eight data bits): This is an array of eight single-precision floating point numbers (floats) used to store data bits reserved for other related pa

rameters.

Ue4EKFInit

```
struct Ue4EKFInit {
 int checksum;
 int CopterID;
 int initCode;
}
```

Function explanation

Used to represent initialization information of the extended Kalman filter (EKF)

Parameter interpretation

- Check sum (data check bit): This is an integer field that stores the checksum of the data to ensure its integrity and correctness.
- CopterID: This is an integer field that identifies the aircraft or UAV for which the Extended Kalman Filter was initialized.
- InitCode: This is an integer field that represents the initialization state of the extended Kalman filter.

Blueprint data

Ue4 ExtMsgFloat (Extension Blueprint)

```
struct Ue4ExtMsgFloat {
 int checksum;
 int CopterID;
 double runnedTime;
 float pwm[16];
}
```

Function explanation

Used to pass status information related to the Copter, including a timestamp and PWM value. The blueprint for the specified Copter is called. ActuatorInputsExt Function

Parameter interpretation

- Check sum (data check bit): This is an integer field that stores the checksum of the data to ensure its integrity and correctness.
- CopterID: This is an integer field that identifies which aircraft the message is intended for.
- RunnedTime: This is a double-precision floating point number (double) representing the timestamp of the message.

- PWM (Pulse Width Modulation): This is an array of 16 double-precision floating point numbers (doubles) used to store values other than 8-bit PWM.

Ue4 ExtMsg (Extension Blueprint)

```
struct Ue4ExtMsg {
 int checksum;
 int CopterID;
 double runnedTime; //Current  stamp (s)
 float pwm[16];
}
```

Similar to Ue4ExtMsgFloat (Extended Blueprint)

Copters are patterns that are attached to other copters

VehicleAttatch25

```
struct VehicleAttatch25 {
 int checksum;//1234567892
 int CopterIDs[25];
 int AttatchIDs[25];
 int AttatchTypes[25];
//0: normal mode, 1: relative position not relative attitude, 2: relative position + yaw,
3: relative position + attitude
}
```

Function explanation

Describe the affiliation between multiple aircraft, with up to 25 aircraft attached to up to 25 other aircraft

Parameter interpretation

- Check sum: This is an integer field that stores the checksum of the data.
- CopterIDs: This is an array of 25 integer elements. Each element corresponds to a copter as an attachment, storing its identification number (ID).
- AttatchIDs: This is an array of 25 integer elements. Each element corresponds to an attached copter and stores its identification number (ID).
- AttatchTypes: This is an array of 25 integer elements. Each element corresponds to a copter that stores the dependency type.

  0: Normal mode (no relative relationship)

  1: Relative position and non-relative attitude

  2: Relative position + yaw

3: Relative position + attitude

GPS coordinates for Cesium map

Ue4CMDGPS

```
struct Ue4CMDGPS {
 int checksum;
 int CopterID;
 double GPS[3];
}
```

Function explanation

Modify the GPS coordinates of the target aircraft in the cesium map

Parameter interpretation

- Check sum (data check bit): This is an integer field that stores the checksum of the data to ensure its integrity and correctness.
- CopterID: This is an integer field that identifies the aircraft to which the command is sent.
- GPS (GPS Coordinate Array): This is an array of 3 double-precision floating-point numbers (doubles) that stores GPS coordinate information. GPS coordinates include latitude, longitude, and altitude.

Font displayed by Copter

Ue4CopterMsg

```
struct Ue4CopterMsg {
 int checksum; //1234567899
 int CopterID;
 int dispFlag;
 int RGB[3];
 float dispTime;
 float fontSize;
 char data[120];
};
```

Function explanation

Used to set the label content displayed at the top of each copter

Parameter interpretation

- Check sum (data check bit): This is an integer field that stores the checksum of the data to ensure its integrity and correctness.

- CopterID: This is an integer field that identifies which aircraft or drone the message is intended for. If the value of CopterID is less than or equal to 0, it means that the message is applicable to all aircraft, that is, all aircraft will process the message.
- DispFlag: This is an integer field that controls how the message is displayed.
- RGB (Color Information): This is an array of 3 integer elements representing the color of the message. The three elements of the RGB array represent the red, green, and blue color components that define the text color of the message.
- Disp Time: This is a floating-point number field that represents the display time of the message, usually in seconds.
- Font Size: This is a floating field that represents the font size of the message text.
- Data (message content): This is a character array of up to 120 characters that stores the text content of the message. If the message content (data) is not specified, the aircraft ID should be displayed by default.

10. External interface file

10.1 Simulink Interface function

10.1.1 Instructions for use

Just make sure that the function called in the Simulink module is in the same directory as the module, and make sure that it is called correctly. Run the module and open RflySim3D at the same time. RflySim3D receives the UDP packet and automatically executes the corresponding instructions.
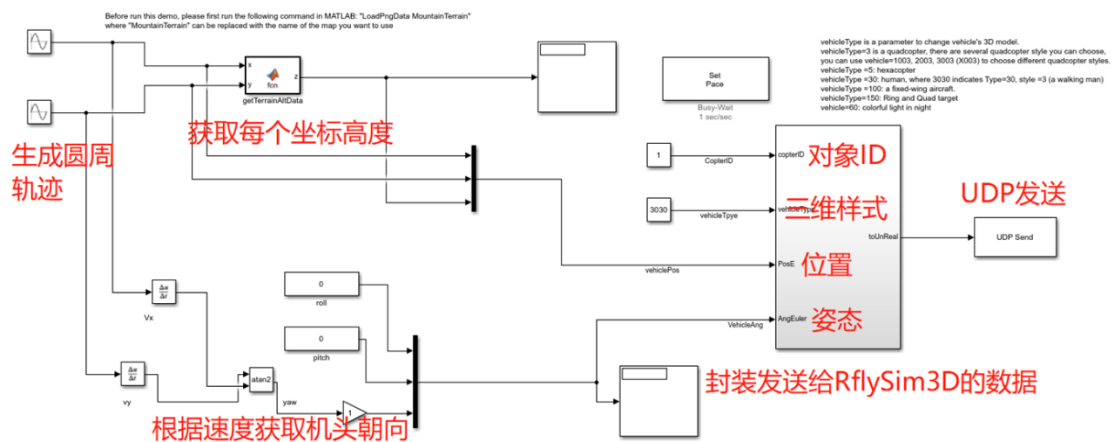
10.1.2 Introduction to the module

Data transmission interface

RflyCameraPosAng.m

Enter the path containing the function in MATLAB

Open RflySim3D and call the function in the Matlab command line.

```
RflyCameraPosAng(0,0,-10,0,-30,0)
```

Same as RflySim3D console command RflyCameraPosAng （Reset Camera）

RflySendUE4CMD.m

Enter the path containing the function in MATLAB

Open RflySim3D and call the function in the Matlab command line.

```
RflySendUE4CMD(uint8('RflyChangeMapbyName Grasslands'))
```

Functions the same as the " sendUE4Cmd " function in the python interface

UE4DataEncoder

Used to transfer 1-8 bit motor data of the specified copter to the blueprint interface 8.4.1 ActuatorInputs Interface , with the same parameters as RflySetActuatorPWMs （Trigger Blueprint Interface）
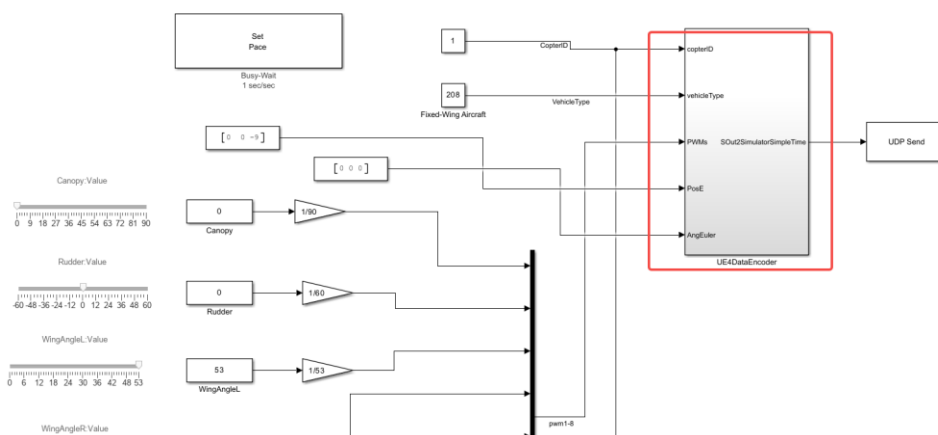
The data is stored in SOut2SimulatorSimpleTime （Stand-alone data 4） the structure
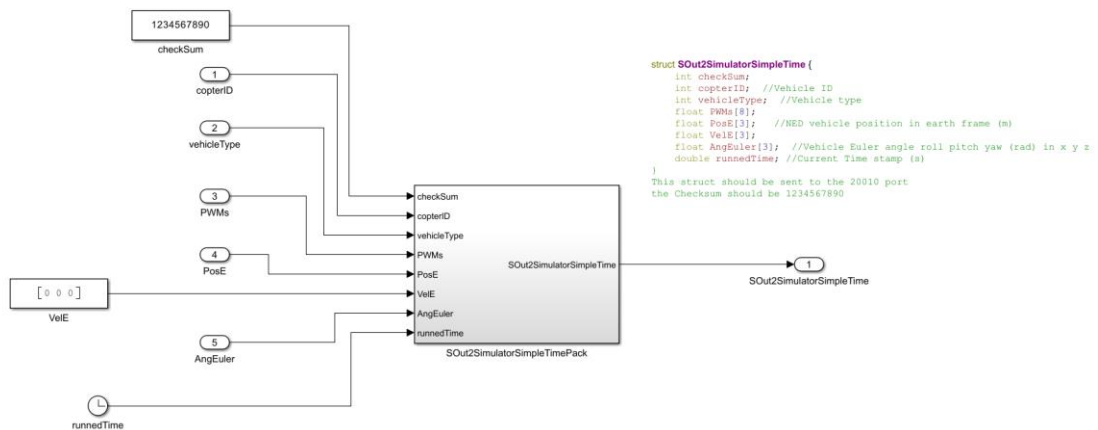
Send to This machine 20010++1 Series port (20010~20029)

Ue4ExtMsgEncoder

Used to transfer the 9-24 bit motor data of the specified copter to the blueprint extension inter face 8.4.2 ActuatorInputsExt_Interface . The parameters are the same as RflySetActuatorPWMsExt_(Trigger Extension Blueprint Interface)
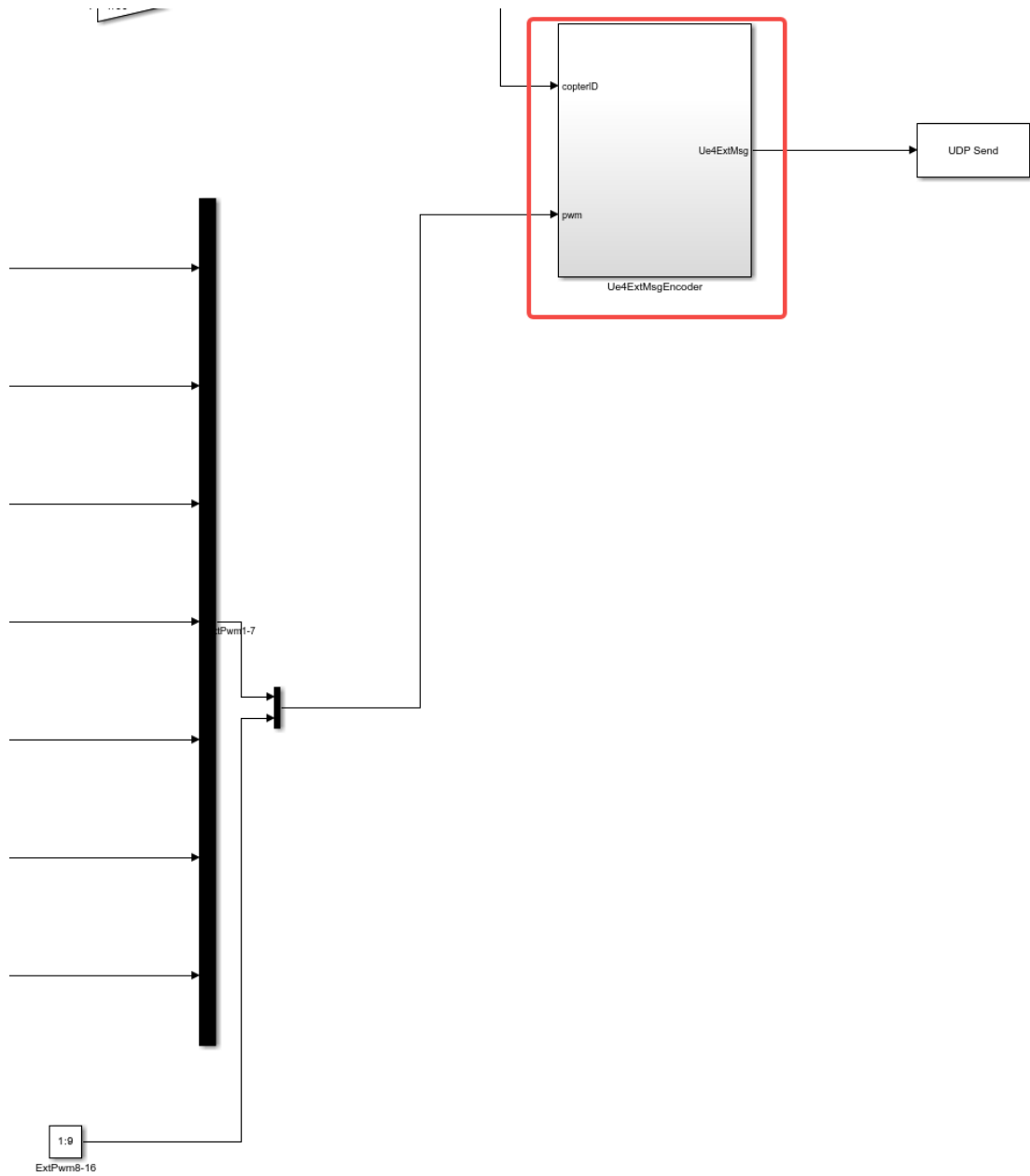
The data is stored in Ue4ExtMsg (Extended Blueprint) the structure
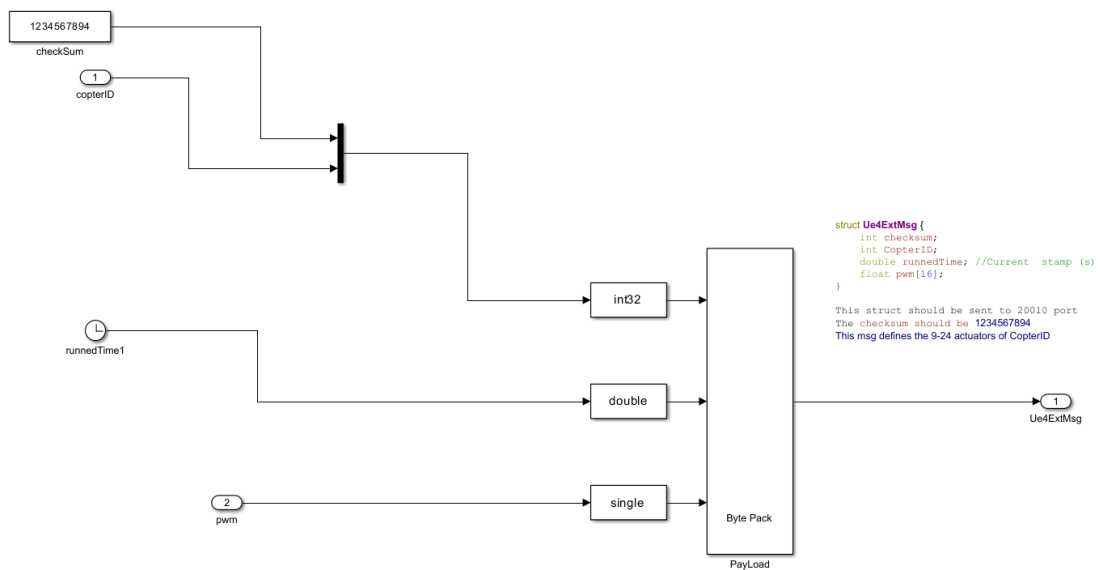
Send to This machine 20010++1 Series port (20010~20029)
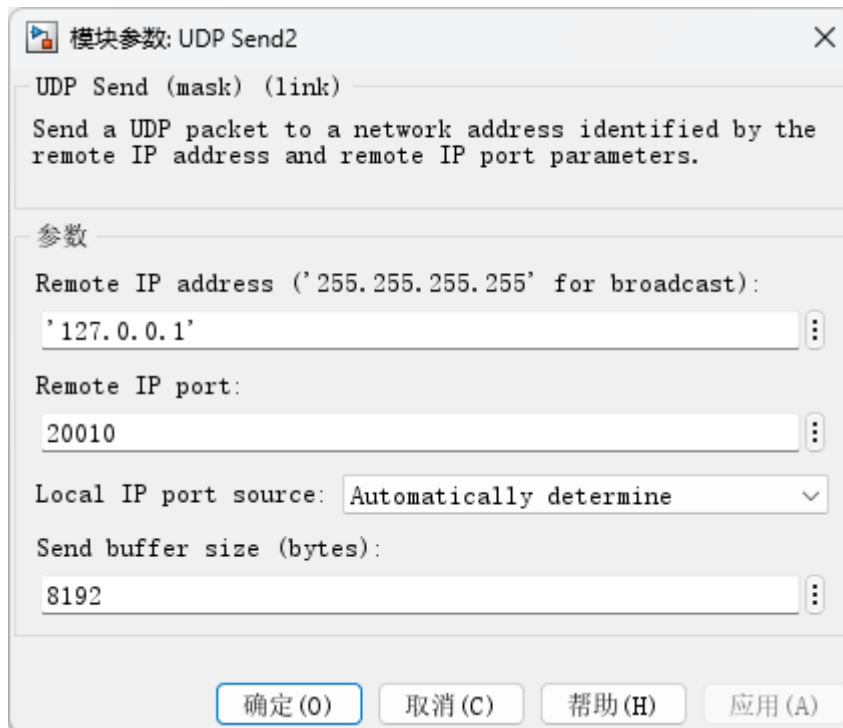
Get the terrain interface

LoadPngData.m

Enter the path containing the function in MATLAB

Open RflySim3D and call the function in the Matlab command line.

```
LoadPngData map name
```

This function will search: the "map" folder under this path and the " [installation path] \ C opterSim \ external \ map" folder in turn ([installation path] is C: \ PX4PSP by default, depending on the installation configuration of the platform), and ensure that the terrain file is in one of the above folders. After running the above script, a " Map name .mat "File." is generated to store the height map matrix data.

You can view the following algorithm by yourself, which is mainly to import PNG as a matrix file, add calibration data, and then save it as a height map matrix

```
function  LoadPngData(pngMapName)

if ~exist('pngMapName','var')
    pngMapName='MountainTerrain';
end

filelocPath='map';
fileLocPng = [filelocPath,'\',pngMapName,'.png'];
if ~exist(fileLocPng,'file')
    filelocPath='..\..\..\CopterSim\external\map';
    fileLocPng = [filelocPath,'\',pngMapName,'.png'];
    if ~exist(fileLocPng,'file')
        matPath=[userpath,'\Add-Ons\Toolboxes\PX4PSP\code\+codertarget\+pixhawk\+CMAKE_Ut
ils\FirmwareVersion.mat'];
        if exist(matPath,'file')
            AA=load(matPath);
            filelocPath=[AA.RflyPSP_Px4_Base_Dir,'\CopterSim\external\map'];
            fileLocPng = [filelocPath,'\',pngMapName,'.png'];
            if ~exist(fileLocPng,'file')
                error(['Cannot find file: ',pngMapName,'.png file']);
            end
        else
            error(['Cannot find file: ',pngMapName,'.png file']);
        end

    end

end

fileLocTxt = [filelocPath,'\',pngMapName,'.txt'];
if ~exist(fileLocTxt,'file')
    error(['Cannot find file: ',pngMapName,'.txt']);
end

fileID = fopen(fileLocTxt);
m_readData_cell = textscan(fileID,'%f',9,'Delimiter',',');
m_readData = m_readData_cell{1};
[m,n]=size(m_readData);
if m~=9 || n~=1
    error(['Cannot parse data in ',fileLocTxt]);
end

rowmap  = imread(fileLocPng);
rowmap = double(rowmap)-32768;
```

```
    [rows, columns] = size(rowmap);

    PosScaleX = (m_readData(1)-m_readData(4))/(columns-1);
    PosScaleY = (m_readData(2)-m_readData(5))/(rows-1);

    PosOffsetX = m_readData(4);
    PosOffsetY = m_readData(5);

    intCol = int32((m_readData(7)-PosOffsetX)/PosScaleX + 1);
    intRow = int32((m_readData(8)-PosOffsetY)/PosScaleY + 1);
    % constainXY(intRow,intCol);

    heightInit = double(rowmap(1,1));
    heightFirst = double(rowmap(rows,columns));
    heightThird = double(rowmap(intRow,intCol));


    if abs(heightThird-heightFirst)<=abs(heightThird-heightInit)
        if abs((heightInit-heightThird))>10
            PosScaleZ = (m_readData(6)-m_readData(9))/((heightInit-heightThird));
        else
            PosScaleZ = 1;
        end
    else
        if abs(heightThird-heightFirst)>10
            PosScaleZ = (m_readData(3)-m_readData(9))/((heightFirst-heightThird));
        else
            PosScaleZ = 1;
        end
    end

    intPosInitZ = heightInit;
    PosOffsetZ = m_readData(6);

    xMax=abs(m_readData(1)/100);
    yMax=abs(m_readData(2)/100);

    binmap= -(PosOffsetZ + ((rowmap)-intPosInitZ)*PosScaleZ)/100.0;

    save('MapHeightData','binmap','PosOffsetX','PosScaleX','PosOffsetY','PosScaleY');
```

getTerrainAltData.m

Enter the path of the existing function and "map name.mat" file in MATLAB

Open RflySim3D and call the function in the Matlab command line.

```
getTerrainAltData(x,y)
```

The getTerrainAltData function inputs the X, y coordinates of the map and outputs the current

terrain height Z. Through this function, the height information of any position in the terrain can be

obtained, so that the trajectory close to the surface can be created.

You can see the algorithm for yourself as follows:

```matlab
function zz = getTerrainAltData(xin,yin)

%The map for simulation
mapname='MapHeightData';

persistent binmap;
persistent PosOffsetX;
persistent PosScaleX;
persistent PosOffsetY;
persistent PosScaleY;
if isempty(binmap)

%     if ~exist([mapname,'.mat'],'file')
%         LoadPngData(mapname);
%     end

    SS=load([mapname,'.mat']);
    binmap=SS.binmap;
    PosOffsetX=SS.PosOffsetX;
    PosScaleX=SS.PosScaleX;
    PosOffsetY=SS.PosOffsetY;
    PosScaleY=SS.PosScaleY;

end


intCol = (xin*100-PosOffsetX)/PosScaleX+1;
intRow = (yin*100-PosOffsetY)/PosScaleY+1;

intColInt=floor(intCol);
intRowInt = floor(intRow);
a=intCol-intColInt;
b=intRow-intRowInt;

intRowInt1=intRowInt+1;
intColInt1=intColInt+1;

[m,n]=size(binmap);
if intColInt<1
    intColInt=1;
    intColInt1=1;
    a=0;
end


if intColInt>=n
    intColInt=n;
    intColInt1=intColInt;
    a=0;
end


if intRowInt<1
    intRowInt=1;
    intRowInt1=1;
```

```
        b=0;
    end


    if intRowInt>=m
        intRowInt=m;
        intRowInt1=intRowInt;
        b=0;
    end

    zz=binmap(intRowInt,intColInt)*(1-b)*(1-a)+binmap(intRowInt1,intColInt)*b*(1-a)+binmap(in
tRowInt,intColInt1)*(1-b)*a+binmap(intRowInt1,intColInt1)*b*a;
```

Other Interface

GenSwarmPos12.m

GenSwarmPos 30 .m

GenSwarmPos 100 .m

GenSwarmPos 2PC200 .m

10.2 Python Interface Library file UE4C trl API .py

10.2.1 Instructions for use

Examples are as follows:
```
# Import required libraries
import time
import math
import sys

# Import Rfly Sim APIs.
import UE4CtrlAPI as UE4CtrlAPI
```

```
       # Create a new MAVLink communication instance. The UDP sending port (the receiving p
ort of CopterSim) is the 20100.
    ue = UE4CtrlAPI.UE4CtrlAPI()

    # sendUE4Cmd: RflySim3D API for modifying scene display style
    # Format: ue.sendUE4Cmd (cmd, windowID = -1), where cmd is a command string, windowI
D is the received window number (assuming multiple RflySim3D windows are open at the same ti
me), and windowID = -1 means send to all windows
    # Example: The RflyChangeMapbyName command means to switch the map (scene). The fol
lowing string is the map name. Here, all open windows will be switched to the grass map.
    ue.sendUE4Cmd(b"RflyChangeMapbyName Grasslands")
    time.sleep(2)
```

After importing the necessary dependent libraries and the interface function UE4CtrlAPI.py o

f RflySim3D, it is also necessary to create a communication instance corresponding to the class to

be called, such as ue here, which corresponds to the instance corresponding to the UE4CtrlAPI cla

ss in the UE4CTrlAPI.py file.

10.2.2 Class definition

The initialization method of all classes has two constructors. One is the default constructor, w

hich initializes the property to zero. The other is constructed with a given argument IV, which is a l

ist containing data that sets the values of the attributes in order.

10.2.2 .1 PX4SILIntFloat Class (Initialization control mode)

PX4SILIntFloat is a class for outputting data to the CopterSim DLL model. It contains the fol

lowing methods:

__init__ Method

```
def __init__(self):
self.checksum = 0
self.CopterID = 0
self.inSILInts = [0, 0, 0, 0, 0, 0, 0, 0]
self.inSILFLoats = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

def __init__(self, iv):
self.checksum = iv[0]
self.CopterID = iv[1]
self.inSILInts = iv[2:10]
self.inSILFLoats = iv[10:30]
```

Parameters:

- Check sum: check digit

- CopterID: Copter ID corresponding to the message

- InSILInts: SILInts data output to the CopterSim DLL model

- The first integer of inSILInts is used to characterize and modify the state, and a 1 in the corresponding bit indicates that the system is in the corresponding state. For example, the first bit indicates emulation mode, and when the first bit of the received inSILInts [0] is 1, the system enters emulation mode. The state is set once only when the first bit of inSILInts [0] is 1, otherwise the integrated model continues to use the original state. The old state can come from an external set value or from an automatic internal state transition. For example, after receiving the take-off command, it will switch to the take-off mode first, and then automatically switch to the fixed-point mode after the completion of take-off.

- InSILFLoats: SILFoats data output to the CopterSim DLL model. InSILFloats is used to store actual data, and its specific meaning can be changed according to the setting of inSILInts. For example, the first three floating-point numbers represent the position, but the position in which coordinate system is controlled by inSILInts. The specific agreement is as follows:

  InSILFloats [0-2]: position

  InSILFloats [3-5]: pitch, roll and yaw signals of speed or remote controller. Where inSILFloats [3] can be used as the rate.

  InSILFloats [6-8]: Acceleration

  InSILFloats [9-11]: Euler angles used for attitude control, more intuitive for the user.

  InSILFloats [12-14]: attitude rate

  InSILFloats [15]: Throttle

10.2.2.2 reqVeCrashDat Class （U E Returns collision-related data)

The reqVeCrashDat class is the struct sent by RflySim3D. It is the struct that will be sent when RflySim3D enables the data return mode (using the "RflyReqVehicleData 1" console command). It mainly contains data related to collision, which will be sent to multicast IP "224.0.0.10: 20006". It contains the following methods:

__init__ Method

```
def __init__(self):
self.checksum = 1234567897
self.copterID = 0
```

```
self.vehicleType = 0
self.CrashType = 0
self.runnedTime = 0
self.VelE = [0, 0, 0]
self.PosE = [0, 0, 0]
self.CrashPos = [0, 0, 0]
self.targetPos = [0, 0, 0]
self.AngEuler = [0, 0, 0]
self.MotorRPMS = [0, 0, 0, 0, 0, 0, 0, 0]
self.ray = [0, 0, 0, 0, 0, 0]
self.CrashedName = ""
```

Parameter

- CopterID: indicates the data of which copter the struct is.

- VehicleType: Indicates the style of this Copter

- CrashType: indicates the collision object type, -2 indicates the ground, -1 indicates the scene static object, 0 indicates no collision, and above 1 indicates the ID number of the collided aircraft

- RunnedTime: Time stamp of the current aircraft

- VelE: Current aircraft speed (m/s)

- PosE: Current aircraft position (NE, in meters)

- CrashPos: Coordinates of the impact point (NE, in meters)

- TargetPos: coordinates of the center of the impacted object (northeast, unit: m)

- AngEuler: Euler angle of the current aircraft (Roll, Pitch, Yaw, radians)

- MotorRPMS: Current aircraft motor speed

- Ray: Front, back, left, right, up and down scan lines of the aircraft

- CrashedName: The name of the touched object

10.2.2.3 CoptReqData （U E Return to model data)

__init__ (Initial configuration)

```
def __init__(self):
Self. Checksum = 0 # 1234567891 as check
Self. CopterID = 0 # Aircraft ID
    self.PosUE = [0,0,0]
    self.angEuler = [0,0,0]
    self.boxOrigin = [0,0,0]
    self.BoxExtent = [0,0,0]
    self.timestmp = 0
    self.hasUpdate=True
```

For the aircraft data returned by RflySim3D, RflySim3D sends the information of a Copter according to the parameters of the command after calling the ” RflyReqObjData ” c

command of RflySIM3D (the Copter ID is attached when sending the command to RflySIM3D).

Parameter

- Check sum: is the check bit of the data, which is used to check whether an exception occurs during data transmission.

- CopterID: indicates the copter ID corresponding to the message.

- PosUE: indicates the location of the Copter (m, NE)

- AngEuler: represents the angle of this Copter (radians, Roll, Pitch, Yaw)

- BoxOrigin: Geometric center of bounding box (m, NE)

- Box Extent: Half of the side length of the bounding box (meters, X axis (forward), Y axis (right), Z axis (up))

- Times TMP: timestamp

- HasUpdate: This value is not sent by RflySim3D

10.2.2.4 ObjReqData （U E Return target object data)

__init__ (Initial configuration)

```
def __init__(self):
Self. Checksum = 0 # 1234567891 as check
    self.seqID = 0
    self.PosUE = [0,0,0]
    self.angEuler = [0,0,0]
    self.boxOrigin = [0,0,0]
    self.BoxExtent = [0,0,0]
    self.timestmp = 0
    self.ObjName="
    self.hasUpdate=True
```

For the data returned by RflySim3D, after calling the ” RflyReqObjData ” command of RflySim3D, the information of a certain object is sent by RflySim3D according to the parameters of the command (the Name of the object is attached when sending the command to RflySim3D).

Parameter

- Check sum: is the check bit of the data, which is used to check whether an exception occurs during data transmission.

- SeqID: always 0. (Obj is not a Copter, it has no ID to return)

- PosUE: indicates the position of the object (m, NE)

- AngEuler: represents the angle of this Obj (radians, Roll, Pitch, Yaw)

- BoxOrigin: Geometric center of bounding box (m, NE)
- Box Extent: Half of the side length of the bounding box (meters, X axis (forward), Y axis (right), Z axis (up))
- Timestmp: Time the current scene has been in existence (seconds, the time the scene has been in existence (because it is not a Copter, there is no timestamp))
- ObjName: The name of the object
- HasUpdate: This value is not sent by RflySim3D

10.2.2.5 CameraData （U E Back to camera data)

__init__ (Initial configuration)

```
def __init__(self):
Self. Checksum = 0 # 1234567891 as check
        self.SeqID = 0
        self.TypeID = 0
        self.DataHeight = 0
        self.DataWidth = 0
        self.CameraFOV = 0
        self.PosUE = [0,0,0]
        self.angEuler = [0,0,0]
        self.timestmp = 0
        self.hasUpdate=True
```

For the data returned by RflySim3D, after calling the ” RflyReqObjData ” command of RflySim3D, the information of a camera (visual sensor) sent by RflySim3d according to the parameters of the command (the SeqID of the camera (visual sensor) is attached when the command is sent to RFlySim3D).

Parameter

- Check sum: is the check bit of the data, which is used to check whether an exception occurs during data transmission.
- SeqID: ID of the vision sensor
- TypeID: The type of visual sensor (RPG, depth map, etc.)
- DataHeight: Data height (pixels)
- DataWidth: Data width (pixels)
- CameraFOV: horizontal field of view of camera (unit: degree)
- PosUE: indicates the position of the object (m, NE)
- AngEuler: indicates the angle of the Camera (radian, Roll, Pitch, Yaw)
- Times TMP: timestamp

- HasUpdate: This value is not sent by RflySim3D

10.2.2.6 UE4CtrlAPI Class （U E Scene Control Command)

_ _ init _ _ (initial configuration)

```
def __init__(self, ip='127.0.0.1'):
        self.ip = ip

        self.udp_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)  # Create sock
et
        self.udp_socket.setsockopt(socket.SOL_SOCKET, socket.SO_BROADCAST, 1)
        self.udp_socketUE4 = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)  # Create s
ocket
        self.udp_socketUE4.setsockopt(socket.SOL_SOCKET,socket.SO_REUSEADDR,1)
        self.inSilVect = []
        self.inReqVect = []

        self.stopFlagUE4=True
        self.CoptDataVect=[]
        self.ObjDataVect=[]
        self.CamDataVect=[]
```

This is the constructor for this class

Parameter

- IP: This is a string variable that represents the IP address to communicate with RflySim3D. The default value is ' 127.0.0.1 ' , which indicates the local host.

- UDP _ socket and UDP _ socketUE4: These are two UDP socket variables used for network communication. Where the UDP _ socket is a socket set to broadcast mode and UDP _ socketUE4 has a SO _ REUSEADDR flag. These sockets are used to communicate with RflySim3D and are the primary way to send and receive network data.

- InSilVect, inReqVect, CoptDatavect, ObjDataVect and CamDataVect: These are list variables and are used to store aircraft related data. Include aircraft status, static object data, and camera data.

- StopFlagUE4: This is a Boolean variable that controls whether RflySim3D is in a stopped state.

SendUE4Cmd (console command)

```
def sendUE4Cmd(self,cmd,windowID=-1):
```

And is use for sending RflySim3D Console commands and control that display of the th
ree-dimensional scene. See the structure Ue4CMD

Parameter

- Cmd: The command string to send. This parameter may be a display style cont
rol command of RflySim3D, a map switching command, a camera movement com
mand, a vehicle model change command, etc.

- WindowID: An optional parameter that represents the ID of the RflySim3D wi
ndow. The default value is -1. The window ID can be used to send commands to a s
pecific window to control a specific instance of RflySim3D. If the windowID is les
s than 0, the command is sent to all window instances. If the windowID is greater t
han or equal to 0, the command is sent to the specified window ID.

sendUE4Cmd N et （ **In the local area network Broadcast** Cons ole commands **. This feature is only supported above the full version.** ）

```
def sendUE4CmdNet(self,cmd):
```

Used to send RflySim3D Console commands all RflySim3D instances within the contro
l LAN. See the structure Ue4CMDNet

Parameter

- Cmd: The command string to send. This parameter may be a display style cont
rol command of RflySim3D, a map switching command, a camera movement com
mand, a vehicle model change command, etc.

SendUE4LabelID (set Copter label content)

```
def sendUE4LabelID(self,CopterID=0,Txt='',fontSize=30,RGB=[255,0,0],windowID=-1):
```

Used to set the overhead display label for the specified copter. See the structure Ue4Cop
terMsg

Parameter

- CopterID (default is 0): Indicates the ID of the drone or aircraft. This paramete
r is used to specify the aircraft to display the label.

- Txt: Indicates the text to be displayed. This is what is actually displayed in the
label.

- Font Size (default 30): Indicates the size of the font. Lets you specify the font s
ize of the displayed text.

- RGB (default is [255,0,0]): Indicates the color of the label. This is a list of three integers representing the values of red, green, and blue.
- WindowID (default -1): Indicates the ID of the window. If a window ID is specified, the label appears only on that window. If the windowID is negative, the label appears on all windows.

SendUE4LabelMsg (set the content below the Copter label)

```
    def sendUE4LabelMsg(self,CopterID=0,Txt='',fontSize=30,RGB=[255,0,0],dispTime=0,dispFlag=
-1,windowID=-1):
```

Used to create a new line of display content at the top of the specified copter. See the structure Ue4CopterMsg

Parameter

- CopterID (default 0): The ID of the aircraft that specifies the aircraft for which the message is to be displayed. If the CopterID is less than or equal to 0, the message is displayed for all aircraft; if the CopterID is greater than 0, the message is displayed only for the corresponding aircraft.
- Txt (default is empty string): Text content to be displayed.
- Font Size (default 30): The size of the font, which specifies the size of the font in which the text is displayed.
- RGB (default is [255,0,0]): The color of the label, in RGB format. The value of each component ranges from 0 to 255, representing the color components of red, green, and blue, respectively.
- DispTime (default is 0): The time (in seconds) for the message to disappear. If the value is less than 0, the message disappears immediately; if the value is equal to 0, the message remains displayed; if the value is greater than 0, it disappears after the specified number of seconds.
- DispFlag (default -1): a display flag that controls how the message is displayed. Different dispFlag values correspond to different display behaviors, including layer-by-layer accumulation, clearing all messages, updating specified lines (ID lines), and updating 1-5 lines of messages.

DispFlag <0 and dispTime> = 0 means that the message is added in a sequential accumulation manner

DispFlag < 0 and dispTime < 0 means all messages are cleared

DispFlag = 0 and dispTime > = 0 means update ID row (row 0 data)

DispFlag > = 1 and < 5 means to update the message of lines 1 to 5. Note th

at dispTime <0 will delete this message, and if> = 0, the message will be updated

DispFlag > number of current messages, switch to accumulation mode

- WindowID (default -1): ID of the window. If a window ID is specified, label m essages are displayed only on the specified window. If the windowID is negative, t he label message is displayed on all windows.

sendUE4Attatch （ Copter Attached relationship ）

```
def sendUE4Attatch(
    self: Self@UE4CtrlAPI,
    CopterIDs: Any,
    AttatchIDs: Any,
    AttatchTypes: Any,
    windowID: int = -1
)
```

Attaches the specified copter to the other copters, up to a maximum of 25 copters. See th e structure VehicleAttatch25

Parameter

- CopterIDs: a list of IDs for the aircraft that specify the aircraft to be attached t o other aircraft.

- AttatchIDs: a list of IDs for attaching target aircraft, used to specify which airc raft to attach to.

- AttatchTypes: a list of attachment types to control how they are attached. The value range of each element is 0-3, which indicates different attachment modes:

  0: Normal mode

  1: Relative position and non-relative attitude

  2: Relative position plus yaw (no relative pitch and roll)

  3: Relative position plus full attitude (pitch, roll, and yaw)

- WindowID (default -1): ID of the window. If a window ID is specified, label m essages are displayed only on the specified window. If the windowID is negative, t he label message is displayed on all windows.

SendUE4Pos (create/update model)

```
(method) def sendUE4Pos(
    self: Self@UE4CtrlAPI,
    copterID: int = 1,
    vehicleType: int = 3,
    MotorRPMSMean: int = 0,
    PosE: Any = [0, 0, 0],
```

```
        AngEuler: Any = [0, 0, 0],
        windowID: int = −1
    ) -> None
```

Send position and angle information to RflySim3D to create a new 3D model or update the status of an old one. Transmitting up to 8 bits of motor data to the blueprint model ActuatorInputs_Interface

See the structure SOut2SimulatorSimple（Stand-alone data 2）

Parameter

- CopterID (default 1): ID of the aircraft that specifies the aircraft for which you want to create or update the status.

- VehicleType (default 3): The type of vehicle. Depending on the situation, you can specify different types to create or update different types of aircraft models.

- MotorRPMSMean (default value is 0): the mean value of the motor speed. This parameter is used to control the speed of the motor, which can affect the motion state of the model.

- PosE (default value is [0,0,0]): position information, indicating the current position of the aircraft. This is a list of three elements representing the position of the aircraft on the X, y, and Z axes.

- AngEuler (default value is [0,0,0]): Euler angle information, indicating the current attitude of the aircraft. This is a list of three elements representing the pitch, roll, and yaw angles of the vehicle.

- WindowID (default -1): ID of the window. If a window ID is specified, the position and angle information applies only to the specified window. If the windowID is negative, the position and angle information applies to all windows.

sendUE4PosNew （Create / Update Model)

```
(method) def sendUE4PosNew(
    self: Self@UE4CtrlAPI,
    copterID: int = 1,
    vehicleType: int = 3,
    PosE: Any = [0, 0, 0],
    AngEuler: Any = [0, 0, 0],
    VelE: Any = [0, 0, 0],
    PWMs: Any = [0] * 8,
    runnedTime: int = −1,
    windowID: int = −1
) -> None
```

Similar to sendUE4Pos（Create / Update Model）, the structure is shown in SOut2SimulatorSimpleTime（Stand-alone data 4）

Parameter

- Self: Represents an instance object of a class, used to access member variables and other methods of the class.
- CopterID (default 1): The ID of the aircraft that specifies the aircraft for which you want to create or update the status.
- VehicleType (the default value is 3): The type of the aircraft. Different types can be specified to create or update different types of aircraft models according to the actual situation.
- PosE (default value is [0,0,0]): position information, indicating the current position of the aircraft. This is a list of three elements representing the position of the aircraft on the X, y, and Z axes.
- AngEuler (default value is [0,0,0]): Euler angle information, indicating the current attitude of the aircraft. This is a list of three elements representing the pitch, roll, and yaw angles of the vehicle.
- VelE (default value is [0,0,0]): speed information, indicating the current speed of the aircraft. This is a list of three elements representing the velocity of the vehicle on the X, y, and Z axes.
- PWMs (default value is [0,0,0,0,0,0,0,0]): PWM (pulse width modulation) information, indicating the motor control signal of the aircraft. This is a list of eight elements that control the rotational speed or thrust of the vehicle's motors.
- RunnedTime (default -1): Runtime, in milliseconds, represents the time the aircraft has been in operation.
- WindowID (default -1): ID of the window. If a window ID is specified, information such as position, attitude, and velocity is applied only to the specified window. If the windowID is negative, this information applies to all windows.

sendUE4Pos2Ground (Create / Update Model)

```
(method) def sendUE4Pos2Ground(
    self: Self@UE4CtrlAPI,
    copterID: int = 1,
    vehicleType: int = 3,
    MotorRPMSMean: int = 0,
    PosE: Any = [0, 0, 0],
    AngEuler: Any = [0, 0, 0],
    windowID: int = -1
) -> None
```

Position and angle information is sent to RflySim3D to create a new 3D model or update the state of the old model on the ground, and checksum = 1234567891 is used to tell UE

4 to generate objects that always fit the ground structure. Transmitting up to 8 bits of motor data to the blueprint model ActuatorInputs_Interface

See the structure SOut2SimulatorSimple（Stand-alone data 2）

Parameter

- CopterID (default 1): ID of the aircraft that specifies the aircraft for which you want to create or update the status.
- VehicleType (default 3): The type of vehicle. Depending on the situation, you can specify different types to create or update different types of aircraft models.
- MotorRPMSMean (default value is 0): the mean value of the motor speed. This parameter is used to control the speed of the motor, which can affect the state of the model on the ground.
- PosE (default value is [0,0,0]): Position information, indicating the current position of the aircraft on the ground. This is a list of three elements representing the position of the aircraft on the X, y, and Z axes.
- AngEuler (default value is [0,0,0]): Euler angle information, indicating the current attitude of the aircraft. This is a list of three elements representing the pitch, roll, and yaw angles of the vehicle.
- WindowID (default -1): ID of the window. If a window ID is specified, the position and angle information applies only to the specified window. If the windowID is negative, the position and angle information applies to all windows.

sendUE4PosScale（Create / Update Model)

```
(method) def sendUE4PosScale(
    self: Self@UE4CtrlAPI,
    copterID: int = 1,
    vehicleType: int = 3,
    MotorRPMSMean: int = 0,
    PosE: Any = [0, 0, 0],
    AngEuler: Any = [0, 0, 0],
    Scale: Any = [1, 1, 1],
    windowID: int = -1
) -> None
```

Send position and angle information to RflySim3D to create a new 3D model or update the status of an old model by changing the scale. Transmitting up to 8 bits of motor data to the blueprint model ActuatorInputs_Interface

See the structure SOut2SimulatorSimple1（Stand-alone data 5）

Parameter

- Self: Represents an instance object of a class, used to access member variables

and other methods of the class.

- CopterID (default 1): The ID of the aircraft that specifies the aircraft for which you want to create or update the status.

- VehicleType (the default value is 3): The type of the aircraft. Different types can be specified to create or update different types of aircraft models according to the actual situation.

- MotorRPMSMean (the default value is 0): the mean value of the motor speed, which is used to control the motor speed and affect the motion state of the model.

- PosE (default value is [0,0,0]): position information, indicating the current position of the aircraft. This is a list of three elements representing the position of the aircraft on the X, y, and Z axes.

- AngEuler (default value is [0,0,0]): Euler angle information, indicating the current attitude of the aircraft. This is a list of three elements representing the pitch, roll, and yaw angles of the vehicle.

- Scale (default [1,1,1]): Scale information to change the scale of the model. This is a list of three elements representing the scale of the vehicle on the X, y, and Z axes.

- WindowID (default -1): ID of the window. If you specify a window ID, the position, attitude, and scale information applies only to the specified window. If the windowID is negative, the position, pose, and scale information applies to all windows.

sendUE4PosScale2Ground (Create / Update Model)

```
(method) def sendUE4PosScale2Ground(
    self: Self@UE4CtrlAPI,
    copterID: int = 1,
    vehicleType: int = 3,
    MotorRPMSMean: int = 0,
    PosE: Any = [0, 0, 0],
    AngEuler: Any = [0, 0, 0],
    Scale: Any = [1, 1, 1],
    windowID: int = -1
) -> None
```

Send the position and angle information to RflySim3D to create a new 3D model or update the state of an old model by changing the scale, using checksum = 1234567891 to tell UE4 to generate an object that always fits on the ground. Transmitting up to 8 bits of motor data to the blueprint model ActuatorInputs_Interface

See the structure SOut2SimulatorSimple1 (Stand-alone data 5）

Parameter

- Self: Represents an instance object of a class, used to access member variables and other methods of the class.
- CopterID (default 1): The ID of the aircraft that specifies the aircraft for which you want to create or update the status.
- VehicleType (the default value is 3): The type of the aircraft. Different types can be specified to create or update different types of aircraft models according to the actual situation.
- MotorRPMSMean (the default value is 0): the mean value of the motor speed, which is used to control the motor speed and affect the motion state of the model.
- PosE (default value is [0,0,0]): position information, indicating the current position of the aircraft. This is a list of three elements representing the position of the aircraft on the X, y, and Z axes.
- AngEuler (default value is [0,0,0]): Euler angle information, indicating the current attitude of the aircraft. This is a list of three elements representing the pitch, roll, and yaw angles of the vehicle.
- Scale (default [1,1,1]): Scale information to change the scale of the model. This is a list of three elements representing the scale of the vehicle on the X, y, and Z axes.
- WindowID (default -1): ID of the window. If you specify a window ID, the position, attitude, and scale information applies only to the specified window. If the windowID is negative, the position, pose, and scale information applies to all windows.

sendUE4PosFull (Create / Update Model)

```
(method) def sendUE4PosFull(
    self: Self@UE4CtrlAPI,
    copterID: Any,
    vehicleType: Any,
    MotorRPMS: Any,
    VelE: Any,
    PosE: Any,
    RateB: Any,
    AngEuler: Any,
    windowID: int = -1
) -> None
```

Send position and angle information to RflySim3D to create a new 3D model or update the status of an old one.

Transmitting up to 8 bits of motor data to the blueprint model ActuatorInputs_Interface

See the structure SOut2Simulator（Stand-alone data 1）

Parameter

- Self: Represents an instance object of a class, used to access member variables and other methods of the class.

- CopterID: The ID of the aircraft that specifies the aircraft for which to create or update the status.

- VehicleType: The type of the aircraft. Different types can be specified to create or update different types of aircraft models according to the actual situation.

- MotorRPMS (default value is [0,0,0,0,0,0,0,0]): motor speed information, which is used to control the motor speed of the aircraft.

- VelE: Speed information, indicating the speed of the aircraft in the current horizontal coordinate system (NED).

- PosE: Position information, indicating the current position of the aircraft in the horizontal coordinate system (NED).

- RateB: angular velocity information, indicating the angular velocity of the current airframe coordinate system of the aircraft.

- AngEuler: Euler angle information, indicating the current attitude of the aircraft.

- WindowID (default -1): ID of the window. If a window ID is specified, the above information applies only to the specified window. If the windowID is negative, this information applies to all windows.

SendUE4 ExtAct (Trigger Extension Blueprint Interface)

```
(method) def sendUE4ExtAct(
    self: Self@UE4CtrlAPI,
    copterID: int = 1,
    ActExt: Any = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    windowID: int = -1
) -> None
```

Extended Blueprint Interface for up to 16 bits of actuator data in addition to the 8 bits of motor data to the Blueprint Model ActuatorInputsExt

Parameter

- Self: Represents an instance object of a class, used to access member variables and other methods of the class.

- CopterID (default is 1): The ID of the aircraft that specifies the aircraft to perform the external action.

- ActExt (default value is [0, 0, 0, 1, 2, 3, 4, 5]): external action information used to specify the external action performed by the aircraft. This is a list of 16 elements, each representing the value of an external action.
- WindowID (default -1): ID of the window. If a window ID is specified, the external action information applies only to the specified window. If the windowID is negative, the external action information applies to all windows.

sendUE4PosSimple (Create / Update Model)

```
(method) def sendUE4PosSimple(
    self: Self@UE4CtrlAPI,
    copterID: Any,
    vehicleType: Any,
    PWMs: Any,
    VelE: Any,
    PosE: Any,
    AngEuler: Any,
    runnedTime: int = -1,
    windowID: int = -1
) -> None
```

Send position and angle information to RflySim3D to create a new 3D model or update the status of an old one. Transmitting up to 8 bits of motor data to the blueprint model ActuatorInputs_Interface

See the structure SOut2SimulatorSimpleTime (Stand-alone data 4）

Parameter

- Self: Represents an instance object of a class, used to access member variables and other methods of the class.
- CopterID: The ID of the aircraft that specifies the aircraft for which to create or update the status. This is an arbitrary type of parameter that indicates the identity of the aircraft.
- VehicleType: The type of the aircraft. Different types can be specified to create or update different types of aircraft models according to the actual situation.
- PWMs (default value [0,0,0,0,0,0,0,0]): PWM (pulse width modulation) information, indicating the motor control signal of the aircraft.
- VelE: Speed information, indicating the current speed of the aircraft.
- PosE: Position information, indicating the current position of the aircraft.
- AngEuler: Euler angle information, indicating the current attitude of the aircraft.
- RunnedTime (default -1): Timestamp in milliseconds

● WindowID (default -1): ID of the window. If a window ID is specified, the abo ve information applies only to the specified window. If the windowID is negative, t his information applies to all windows.

sendUE4PosScale100（Create 100 A Copter）

```
(method) def sendUE4PosScale100(
    self: Self@UE4CtrlAPI,
    copterID: Any,
    vehicleType: Any,
    PosE: Any,
    AngEuler: Any,
    MotorRPMSMean: Any,
    Scale: Any,
    isFitGround: bool = False,
    windowID: int = -1
) -> None
```

Send position and angle information to RflySim3D to create 100 aircraft at a time. See t he structure Multi3DData100New （100 Machine data 2）

Parameter

● Self: Represents an instance object of a class, used to access member variables and other methods of the class.

● CopterID: The ID of the aircraft that specifies the aircraft for which to create o r update the status. This is an arbitrary type of parameter that indicates the identity of the aircraft.

● VehicleType: The type of the aircraft. Different types can be specified to create or update different types of aircraft models according to the actual situation.

● PosE: Position information, indicating the current position of the aircraft.

● AngEuler: Euler angle information, indicating the current attitude of the aircra ft.

● MotorRPMSMean (the default value is 0): the mean value of the motor speed, which is used to control the motor speed and affect the motion state of the model.

● Scale: Scale information, used to change the scale of the model.

● IsFitGround (default is False): whether to fit the ground. If set to True, the mod el adjusts to the ground structure; if set to False, it does not.

● WindowID (default -1): ID of the window. If a window ID is specified, the abo ve information applies only to the specified window. If the windowID is negative, t his information applies to all windows.

sendUE4PosScalePwm20（Create 20 A Copter）

```
(method) def sendUE4PosScalePwm20(
    self: Self@UE4CtrlAPI,
    copterID: Any,
    vehicleType: Any,
    PosE: Any,
    AngEuler: Any,
    Scale: Any,
    PWMs: Any,
    isFitGround: bool = False,
    windowID: int = -1
) -> None
```

Position and angle information is sent to RflySim3D, creating 20 aircraft at a time. See t he structure Multi3DData2New （20 Machine data 6）

Parameter

- Self: Represents an instance object of a class, used to access member variables and other methods of the class.

- CopterID: The ID of the aircraft that specifies the aircraft for which to create o r update the status. This is an arbitrary type of parameter that indicates the identity of the aircraft.

- VehicleType: The type of the aircraft. Different types can be specified to create or update different types of aircraft models according to the actual situation.

- PosE: Position information, indicating the current position of the aircraft.

- AngEuler: Euler angle information, indicating the current attitude of the aircra ft.

- Scale: Scale information, used to change the scale of the model.

- PWMs (default value is 0): PWM (pulse width modulation) information, indic ating the motor control signal of the aircraft.

- IsFitGround (default is False): whether to fit the ground. If set to True, the mod el adjusts to the ground structure; if set to False, it does not.

- WindowID (default -1): ID of the window. If a window ID is specified, the abo ve information applies only to the specified window. If the windowID is negative, t his information applies to all windows.

getUE4Pos（Get Copter Location

```
(method) def getUE4Pos(
    self: Self@UE4CtrlAPI,
    CopterID: int = 1
```

```
) -> (list | list[int])
```
Gets the location of the specified copter

getUE4Data (Get Copter Data

```
(method) def getUE4Data(
    self: Self@UE4CtrlAPI,
    CopterID: int = 1
) -> (Any | Literal[0])
```
This function can get the data of Copter in RflySim3D, and this data is reqVeCrashData also structural.

initUE4MsgRec (Enable listening)

```
(method) def initUE4MsgRec(self: Self@UE4CtrlAPI) -> None
```
Initialize self.udp_socketUE4 and start a thread T4 (self.UE4MsgRecLoop) to start listening to 224.0.0.10: 20006 and its own 20006 port.

endUE4MsgRec (Stop listening)

```
(method) def endUE4MsgRec(self: Self@UE4CtrlAPI) -> None
```
Stop listening for thread T4 (self.UE4MsgRecLoop)

UE4MsgRecLoop (Cycle)

```
(method) def UE4MsgRecLoop(self: Self@UE4CtrlAPI) -> None
```
UE4 message listening loop

Function interpretation

It monitors 224.0.0.10: 20006 and its own 20006 port. It is used to process the messages returned by RflySim3D or CopterSim. There are 6 kinds of messages in total:

1)   CopterSimCrash with length of 12 bytes:

```
struct CopterSimCrash {
    int checksum;
    int CopterID;
    int TargetID;
}
```

For the collision data returned by RflySim3D, RflySim3D will start the collision detection mode when the P key is pressed in RflySim3D. If a collision occurs, the data will be returned. The P + number can select the sending mode (0 local sending, 1 LAN sending, 2 LAN sending only in case of collision). The default is local sending.

2) PX4SILIntFloat of length 120 bytes:

```
struct PX4SILIntFloat{
int checksum;//1234567897
int CopterID;
int inSILInts[8];
float inSILFLoats[20];
};
```

3) ReqVeCrashData of length 160 bytes:

```
struct reqVeCrashData {
    int checksum; //Packet check code 1234567897.
    int copterID; //ID number of the current aircraft
    int vehicleType; //The style of the current aircraft
    Int CrashType;//Collision object type, -2 indicates ground, -1 indicates scene static object, 0 indicates no collision, and above 1 indicates ID number of the collided aircraft
    double runnedTime; //Timestamp of the current aircraft
    float VelE[3]; //Speed of the current aircraft
    float PosE[3]; //Current aircraft position
    Float CrashPos [3];//Coordinates of the collision point
    Float targetPos [3];//the center coordinate of the touched object
    float AngEuler[3]; //Euler angle of the current aircraft
    float MotorRPMS[8]; //Current aircraft motor speed
    float ray[6]; //front, back, left, right, up and down scan lines of the aircraft
    Char CrashedName [20] = { 0 };//Name of the touched object
}
```

This class has been introduced before. When data return is enabled, RflySim3D will send reqVeCrashData for all Copters (send when data changes, once per second).

4) CameraData with length of 56 bytes:

```
struct CameraData { //56
    int checksum = 0;//1234567891
    int SeqID; //camera serial number
    Int TypeID;//camera type
    Int Data Height;//pixel height
    Int Data Width;//pixel width
    Float Camera FOV;//camera field angle
    float PosUE[3]; //Camera center position
    Float angEuler [3];//camera Euler angle
    Double times TMP;//timestamp
};
```

RflySim3D returns the structure regularly according to the command sent by the reqCam CoptObj function, and the program will store it in the self. CamDataVect after receiving it.

5) CoptReqData with length of 64 bytes:

```
struct CoptReqData { //64
    int checksum = 0; //123456 78 91 as check
    Int CopterID;//Aircraft ID
    float PosUE[3]; //Object center position (specified during artificial 3D modeling, the attitude coordinate axis is not necessarily at the geometric center)
    Float angEuler [3];//Euler angle of object
    Float boxOrigin [3];//object geometric center coordinate
    Float Box Extent [3];//half of the length, width and height of the object frame
```

```
       Double times TMP;//timestamp
};
```

RflySim3D returns the structure regularly according to the command sent by the reqCam

CoptObj function, and the program will store it in the self. CoptDataVect after receiving

it.

6) ObjReqData of length 96 bytes:

```
struct ObjReqData { //96
       int checksum = 0; //123456 78 91 as check
       int seqID = 0;
       float PosUE[3]; //Object center position (specified during artificial 3D modeling, the atti
tude coordinate axis is not necessarily at the geometric center)
       Float angEuler [3];//Euler angle of object
       Float boxOrigin [3];//object geometric center coordinate
       Float Box Extent [3];//half of the length, width and height of the object frame
       Double times TMP;//timestamp
       Char ObjName [32] = { 0 };//Name of object touched
};
```

RflySim3D returns the structure regularly according to the command sent by the reqCam

CoptObj function, and the program will store it in the self. ObjDataVect after receiving i

t.

The received data can be retrieved using the getCamCoptObj function.

getCamCoptObj （ Obtain object data ）

```
(method) def getCamCoptObj(
    self: Self@UE4CtrlAPI,
    type: int = 1,
    objName: int = 1
) -> (Any | Literal[0])
```

Obtain the specified data from RflySim3D. The UE4MsgRecLoop function enables the

monitoring of RflySim3D messages and stores the monitored three types in three lists. T

his function searches the data in these lists. Therefore, you need to use reqCamCoptObj t

he function to request the relevant data from RflySim3D first.

Parameter

- Type: 0 for camera, 1 for airplane, 2 for object

ReqCamCoptObj (specified window data return)

```
(method) def reqCamCoptObj(
    self: Self@UE4CtrlAPI,
    type: int = 1,
    objName: int = 1,
    windowID: int = 0
) -> None
```

Send a data request to RflySim3D to request data for objects in the scene (not to create new objects, but to get data for existing objects). It can be a visual sensor, a Copter, and a common object in the scene. See CoptReqData class, ObjReqData class and CameraData class for details of the obtained data

Parameter

- Type: 0 for camera, 1 for airplane, 2 for object
- ObjName: when type represents camera, seqID; when it represents airplane, objName corresponds to CopterID; when it represents object, objName corresponds to object name
- WindowID: Indicates which RflySim3D you want to send a message to. The default is window 0. (Do not send this data to all RflySim3D, because the value returned is the same, and there is no need to consume additional performance.)

10.2.2.7 UEMapServe (U E Get Terrain Interface)

__init__ (Initial configuration)

```
def __init__(self,name=''):
    if name == '':
        self.PosOffsetX=0
        self.PosScaleX=0
        self.PosOffsetY=0
        self.PosScaleY=0
        self.xMax=0
        self.yMax=0
        self.binmap= []
    else:
        self.LoadPngData(name)
```

LoadPngData (Loading height diagram)

```
(method) def LoadPngData(
    self: Self@UEMapServe,
    name: Any
) -> None
```

Invoked as follow

```
map = UE4CtrlAPI.UEMapServe()
map.LoadPngData("Grasslands")
```

Parameter

- Name: map name (also file name without extension)

Function interpretation:

When using this function, it will find and read the name. Txt and name. PNG in the scrip t directory, which is the map file generated by RflySim3D. (The txt file indicates the size and extent of the map, and the PNG file is the height map of the map.) Each map in the platform will have these two files, which can be found under "C: \ PX4PSP \ CopterSi m \ external \ map".

After calling this function, the map information is read into memory, which is equivalent to the initialization before calling the getTerrainAltData function.

getTerrainAltData (Get Terrain)

```
(method) def getTerrainAltData(
    self: Self@UEMapServe,
    xin: Any,
    yin: Any
) -> Any
```

Query the height of a point on the map in meters.

Parameter

- Xin: X coordinate of the query point (m)
- Yin: Y coordinate of the query point (m)

10.3 Redis Interface function (Under development)

10.3.1 Configuration file RedisConfig.ini

```
[CommMode]
CommunicationMode = 1
[RedisConfig]
host = "192.168.31.79"
port = 6379
db = 0
memkey = "GLOBAL_CAMERA_PARAMETERS_REDIS_KEY"
r3d_rec_key = "GLOBAL_MESSAGE_REDIS_KEY"
r3d_send_key_20006 = "GLOBAL_MESSAGE_R3D_SEND_KEY_20006"
r3d_send_key_30100 = "GLOBAL_MESSAGE_R3D_SEND_KEY_30100"
```

10.3.2 Instructions for use

11. Summary of Common Special Effects Interface (Under development)

11.1   Displays the label

Shortcut key

S（Show / Hidden CopterID ）:

Command line (limited to personal advanced version or above)

 RflySetIDLabel（Set CopterID Displayed at the label)

 RflySetMsgLabel（Set CopterID Shown below the label)

Python interface (limited to personal advanced version or above)

 sendUE4LabelID（Set Copter Label Content)

 sendUE4LabelMsg（Set Copter Below the label)

11.2   Draw a line

T (Open / Off Copter Tracklog):

 T+ Number * (Change track thickness to * Number

11.3   Weather

11.4   Small map

 L（Show / Hidden mini-map）:

11.5   Virtual pipe

O+803: O+ Number * (Generate ClassID For * "Object of"）:

11.6   Communication effects

O+802: O+ Number * (Generate ClassID For * "Object of"）:

11.7   HelicopterPadDemo（Helicopter landing site）

O+800: O+ Number * (Generate ClassID For * "Object of"）:

11.8   CirclePlaneDemo（Annular plane）

O+810: O+ Number * (Generate ClassID For * "Object of"）:

11.9   SatelliteDemo（Satellite）

O+811: O+ Number * (Generate ClassID For * "Object of"）:

11.10   SatelliteReceiveDemo（Satellite Receive)

O+812: O+ Number * (Generate ClassID For * "Object of"）:

11.11   BallonDemo (Balloon)

O+830: O+ Number * (Generate ClassID For * "Object of"）:

11.12   CycleDemo (Ring)

O+831: O+ Number * (Generate ClassID For * "Object of"):

...

Reference Information

[1]. None