载具三维模型调整接口实验原理

3.文件目录

例程目录： [安装目录]\RflySimAPIs\3.RflySim3DUE\0.ApiExps\e4_UAVCtrl

序号	实验名称	简介	文件地址
1	三维场景交互接口模型内部组件运动层次绑定实验（8维执行器 xml配置）	通过xml脚本绑定相互关联的执行器组件。	1.ActuatorBinding\Readme.pdf
2	三维场景交互接口模型内组件运动层次绑定实验（16维执行器 xml配置）	通过修改xml文件验证超8维执行器控制。	2.ActuatorCtrl\Readme.pdf
3	三维场景交互接口模型间运动层次绑定实验（Simulink接口）	利用simulinks调整模型相对关系	3.VehicleAttachSim\Readme.pdf

序号	实验名称	简介	文件地址
4	三维场景交互接口模型间绑定实验（python接口sendUE4Attatch）	使用Python调整模型之间的相对关系	4.VehicleAttachPy\Readme.pdf

总体说明

符合正向运动学的可视化效果

正向运动学采用的基本原理如下：

- 按照父层次到子层次的链接顺序进行层次链接。
- 轴点位置定义了链接对象的连接关节。
- 按照从父层次到子层次的顺序继承位置、旋转和缩放变换。

即变换父对象将影响子对象的位置和方向，变换子对象不影响父对象。

三维模型需要具有符合运动学模型的层级结构，以便根据这个结构来驱动三维模型的网格体，从而产生逼真的动画效果。同时，利用载具模型之间的层次关系，可以使用相同的运动状态来驱动它们的三维模型，从而提高效率和可重用性。

由载具运动状态划分模型的层级结构

定义运动的层级

生成计算机动画时，最有用的工具之一是将对象链接在一起以形成层次的功能。通过将一个对象与另一个对象相链接，可以创建父子关系。应用于父对象的变换同时将传递给子对象。

因为变换的的方式是子对象从父对象继承的，所以对父对象的微小调整会导致需要对它的所有子对象进行调整。链接的典型方法是选择根对象一类的对象使它们移动得最少。与根对象相邻的对象应该移动得很少，而树叶对象应该移动得最多。

基于父对象的移动少于子对象的原则，将网格体划分出不活动的部分和可活动的部分，不变的作为机身，可变部分即为执行器组件。

利用xml文件来存储这些层级结构

XML与JSON相似，都是常用数据交换格式（用于在不同系统之间进行数据交换和存储）。XML语言有如下特点：自定义的标记语言，标签使用封闭的开始和结束标记；结构化特征，可以建立复杂的层次结构和明确的关系；高扩展性，可以通过使用命名空间和DTD（文档类型定义）来定义和约束数据结构。总而言之，XML适合处理复杂的数据结构和约束要求。

UE引擎自带的XmlParser 模块支持XML文件的读取，通过build.cs文件（在Unreal Engine的构建系统中配置构建设置和依赖项）为RflySim3D导入该模块。该模块的两个类：FXmlFile

和 FXmlNode提供了处理 XML 文件和节点的功能。在如下链接中有一个简单用例：
【UE4
C++】解析与构建 XML 数据，XmlParser 与
tinyxml 。

通常新编写一个XML是不必要的，只需要拿其他飞机的XML文件来稍作修改即可，主要需要修改模型的ClassID、Name标签、机身的isAnimationMesh、MeshPath标签以及ActuatorList中的各个执行器的相对位置和MeshPath标签。

关联不同三维模型运动状态的接口

使用Python/Simulink向RflySim3D发送定义了载具间链接关系结构体，并在RflySim3D中呈现这种关联。该结构体定义如下：

struct VehicleAttatch25 {

int checksum; // 校验值，用于验证数据完整性

int CopterIDs[25]; // 飞机的ID

int AttatchIDs[25]; // 被依附飞机的ID

int AttatchTypes[25]; // 依附的样式，包括不同的模式

}

实现原理

利用xml参数配置文件关联不同执行器运动

1.ActuatorBinding\Readme.pdf

通过XML文件的< AttatchToOtherActuator>标签绑定执行器

<Actuator>

<MeshPath>/Game/QZXY/右侧螺旋桨</MeshPath>

<MaterialPath>/Game/QZXY/v22_osprey_hull_Mat</MaterialPath>

<RelativePosToBodyCm>

<x>284.878</x>

<y>-0.403</y>

<z>40.667</z>

</RelativePosToBodyCm>

<RelativeAngEulerToBodyDeg>

<roll>0</roll>

<pitch>0</pitch>

<yaw>0</yaw>

</RelativeAngEulerToBodyDeg>

<RotationAxisVectorToBody>

<x>1</x>

<y>0</y>

<z>0</z>

</RotationAxisVectorToBody>

<RotationModeSpinOrDefect>0</RotationModeSpinOrDefect>

<AttatchToOtherActuator>2</AttatchToOtherActuator>

</Actuator>

这里将该<Actuator>标签对应的执行器绑定到排序为2 的
<Actuator>标签对应的执行器上

通过Python不断地发送控制台命令给虚幻引擎，控制仿真飞行器的动作

1.ActuatorBinding\ActuatorBindingDemo.py

import time

导入 Python 的 time 模块，用于控制程序运行时间。

import UE4CtrlAPI as UE4CtrlAPI

导入名为 UE4CtrlAPI 的自定义模块，用于与虚幻引擎进行通信和控制。

ue = UE4CtrlAPI.UE4CtrlAPI()

创建一个 UE4CtrlAPI 的实例对象 ue，用于与虚幻引擎进行通信。

i=0

lis=[0,0,0,100,0,0,0,100]

初始化变量 i 为 0，用于循环计数。初始化一个列表
lis，包含了控制动作的参数。依次对应xml文件定义的前八个Actuator标签。

while True:

i=(i+5)%360

lis[0]=i

lis[1]=i

lis[2]=i

#lis[3]=i

lis[4]=i

lis[5]=i

lis[6]=i

#lis[7]=i

进入一个无限循环，即程序会一直执行下去，直到手动终止。

每次循环更新 i 的值，使得 i 在 0 到 359 之间循环变化。

依次更新列表 lis 中的第1、2、3、5、6、7元素为当前的 i 值。

tempStr="RflySetActuatorPWMs "+"1000 "

tempStr="RflySetActuatorPWMs "+"1000 ": 初始化一个字符串
tempStr，用于存储要发送给虚幻引擎的控制台命令。这里的命令是设置CopterId为1000的飞行器的执行器控制信号。

for j in lis:

tempStr=tempStr+ str(j) +" "

ue.sendUE4Cmd(tempStr.encode('UTF-8'))

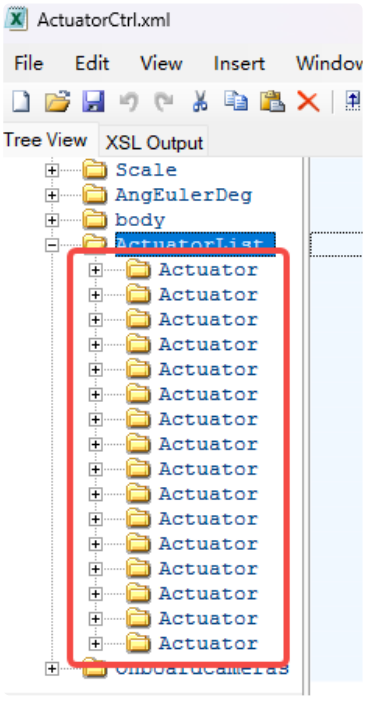
遍历列表 lis 中的每个元素。将遍历到的每个元素转换为字符串并拼接到 tempStr 中，每个参数之间用空格隔开。将拼接好的命令字符串发送给虚幻引擎，通过 UDP 方式发送。

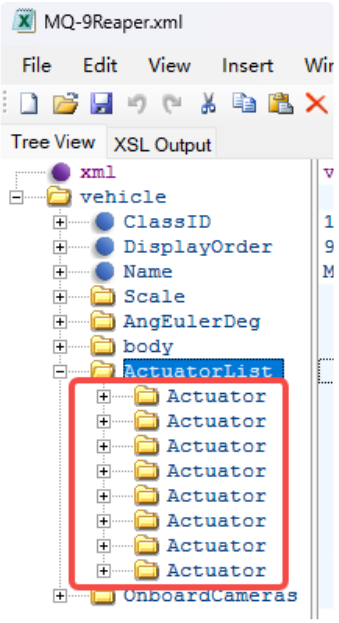
time.sleep(0.05)

让程序暂停 0.05 秒，以控制发送命令的频率，防止发送过快。

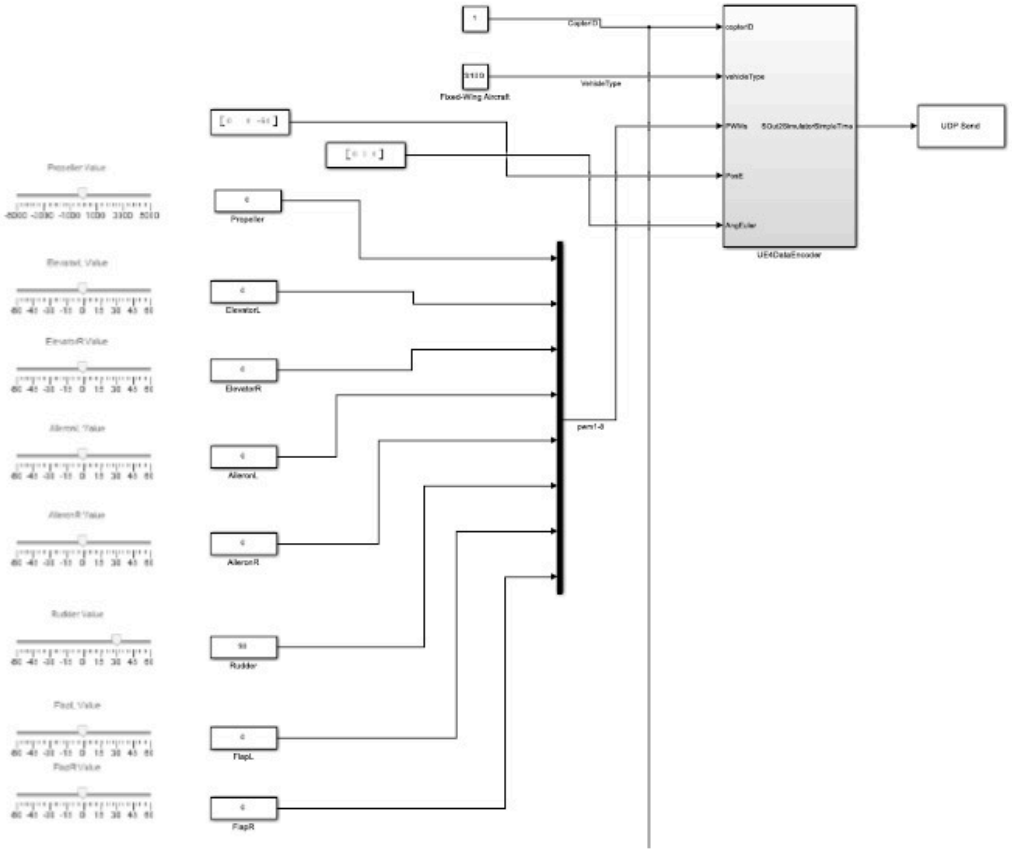
利用xml参数配置文件定义可控的执行器组件

将原来的八个Actuator标签都复制了一份，执行器总数量扩展到16





通过Simulink创建上述xml对应的三维模型并传入执行器数据



通过UE4DataEncoder模块指定位置创建上述xml对应的三维模型，并为该Copter传入前8位执行器信号，打包成UDP发送给RflySim3D，实际传输的结构体为

struct SOut2SimulatorSimpleTime {

int checkSum; //校验码：1234567891，必须设定为本值，才会认为是有效数据

int copterID; //飞机ID序号

int vehicleType; //载具样式ID，对应UE的XML中ClassID

int PosGpsInt[3];
//纬度、精度、高度：lat*10^7,lon*10^7,alt*10^3，int型发放节省空间

float MotorRPMS[8]; //执行器偏转量或转速

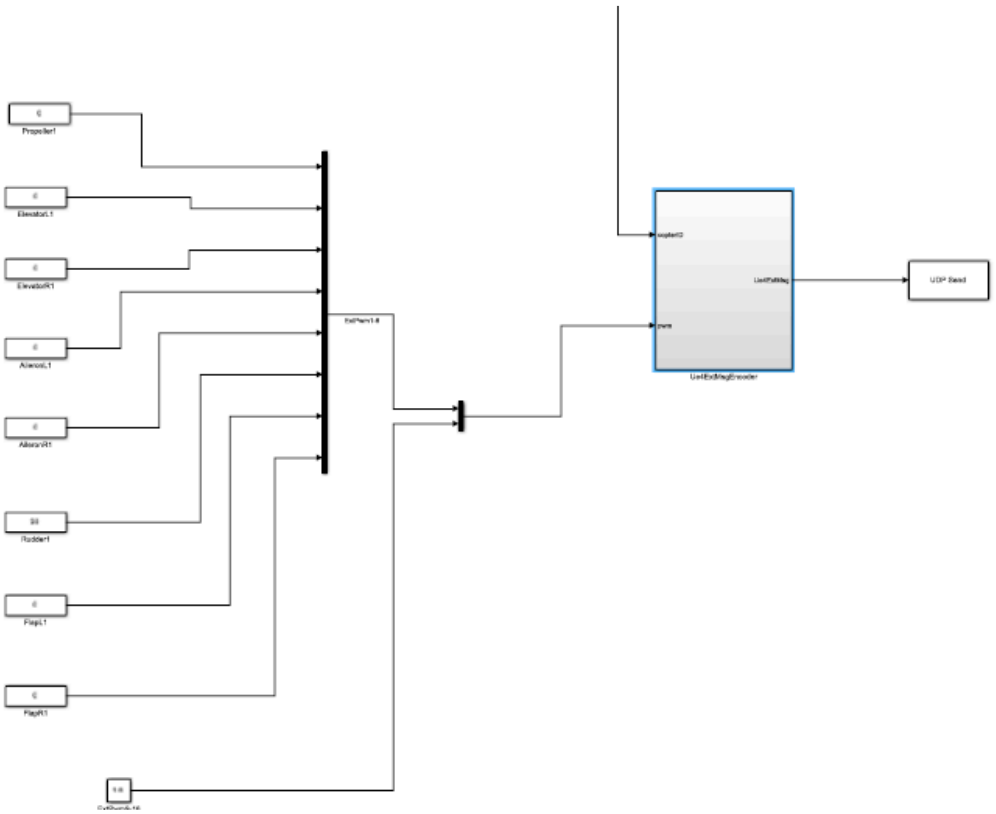
float VelE[3]; //速度，北东地，单位米/s

float AngEuler[3]; //欧拉角，滚转俯仰偏航，单位弧度

double PosE[3]; //北东地位置，单位米，z向下为正

double runnedTime; //时间戳，仿真开始为0时刻。

};



通过Ue4ExtMsgEncoder模块为该Copter传入9~16位的执行器信号，同样打包成UDP发送给RflySim3D，实际传输的结构体为

struct Ue4ExtMsg {

int checksum; 1234567894

int CopterID;

double runnedTime; //Current stamp (s)

float pwm[16];

}

利用Python接口关联不同载具运动状态

在代码编辑器中，打开VehicleAttachAPI.py，找到如下代码

ue.sendUE4PosScale(1,2030,0,[0,0,0],[0,0,0],[1,1,1])

创建1号模型

ue.sendUE4PosScale(2,3,0,[-2,0,0],[0,0,0],[1,1,1])

创建2号机

ue.sendUE4Attatch(2,1,3)

定义了2号机与1号模型之间的依附关系

ue.sendUE4Attatch([2,3,4,5],[1,1,1,1],[3,3,3,3]) # four vehicle attatch set in one command

可以定义2、3、4、5号机各自与1号机的依附关系

打开UE4CtrlAPI.py，在UE4CtrlAPI类下找到如下代码

```
1356
1357    def sendUE4Attatch(self,CopterIDs,AttatchIDs,AttatchTypes,windowID=-1):
1358        """ Send msg to UE4 to attach a vehicle to another (25 vehicles);···
1361        # change the 1D variable to 1D list
1362        if isinstance(CopterIDs,int):···
1364
1365        if isinstance(AttatchIDs,int):···
1367
1368        if isinstance(AttatchTypes,int):···
1370
1371        if not isinstance(CopterIDs,list) or not isinstance(AttatchIDs,list) or not isinstance(AttatchTypes,list):···
1374
1375        if len(CopterIDs)!=len(AttatchIDs) or len(CopterIDs)!=len(AttatchTypes) or len(CopterIDs)>25:···
1378
1379        vLen=len(CopterIDs)
1380        if vLen<25: # Extend the IDs to 25D···
1384
1385        if vLen>25:···
1389
1390        # struct VehicleAttatch25 {
1391        #   int checksum;//1234567892
1392        #   int CopterIDs[25];
1393        #   int AttatchIDs[25];
1394        #   int AttatchTypes[25];//0: 正常模式，1：相对位置不相对姿态，2：相对位置+偏航（不相对俯仰和滚转），3：相对位置+全
1395        # }i25i25i25i
1396
1397        buf = struct.pack("i25i25i25i",1234567892,*CopterIDs,*AttatchIDs,*AttatchTypes)
1398        if windowID<0:···
1404        else:···
```

该接口的目的是向 UE4
发送必要的信息，以建立飞行器之间的依附关系。通过指定飞行器的 ID、依附的 ID
和依附的类型，该方法实现了多个飞行器的依附以及同时配置不同的依附样式。可以调用
sendUE4Attatch 接口并传递飞行器的 ID、依附的 ID
列表以及相应的依附类型列表来建立飞行器之间的依附关系。最多可以指定 25 个飞行器的
ID，并为每个依附指定相应的依附类型。这将在 UE4
中创建或更新飞行器之间的依附关系，以实现所需的效果。

CopterIDs（Any）：该参数表示作为依附点的飞行器的
ID，或主动依附其他飞行器的飞行器的 ID。它可以是一个列表，最长长度为 25，指定最多
25 个飞行器的 ID。

AttatchIDs（Any）：该参数表示被依附飞行器的 ID，或被动接收依附的飞行器的 ID。与
CopterIDs 类似，它也可以是一个最长长度为 25 的列表。

AttatchTypes（Any）：该参数定义了每个依附的类型。它表示了连接被依附飞行器的方式，包括：

0：正常模式，相对位置和姿态保持不变。

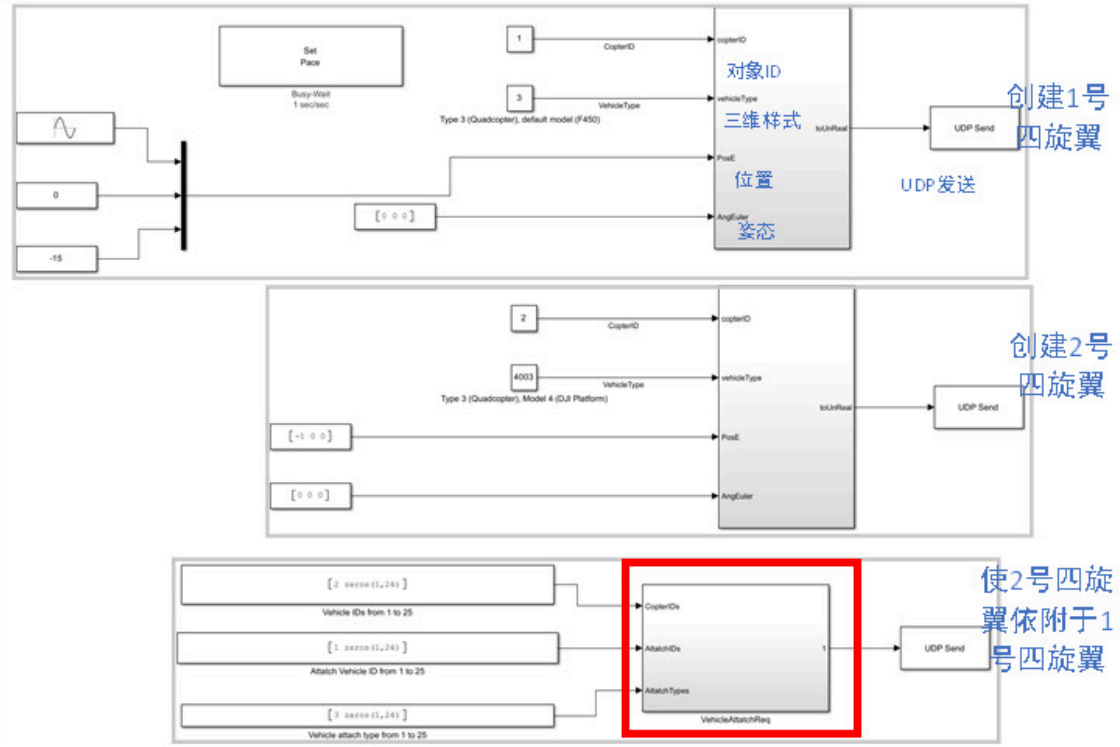
1：相对位置模式，相对位置保持不变，但姿态可以改变。

2：相对位置+偏航模式，相对位置可以改变，但俯仰和滚转保持不变，只有偏航可以改变。

3：相对位置+全姿态模式，相对位置和姿态都可以改变。

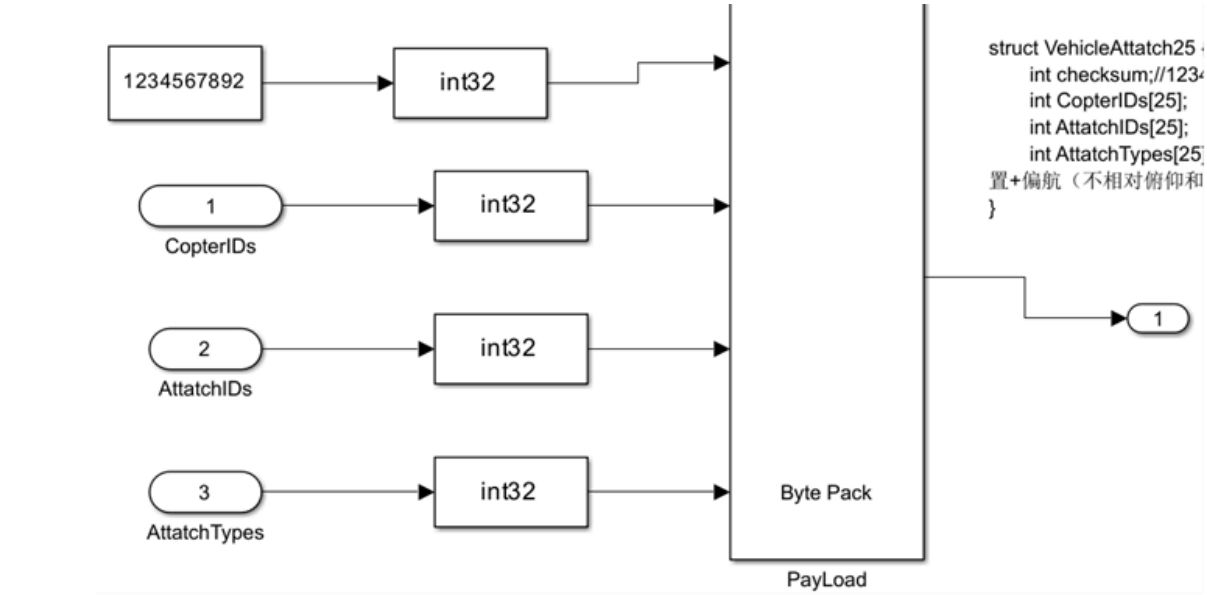
利用Simulink接口关联不同载具运动状态

在MATLAB中，打开VehicleAttachAPI.slx



可见该模块中的前两个子模块通过发送UDP包创建了两个不同样式的四旋翼飞机，第三个子模块定义了两架四旋翼飞机的依附关系。

双击打开第三个子模块VehicleAttatchReq



该模块通过UDP包发送一个特定结构体，该结构体定义如下：

struct VehicleAttatch25 {

int checksum; // 校验值，用于验证数据完整性

int CopterIDs[25]; // 飞机的ID

int AttatchIDs[25]; // 被依附飞机的ID

int AttatchTypes[25]; // 依附的样式，包括不同的模式

}

结构体中的字段包括：

checksum：校验值，可以用于验证数据的完整性。

CopterIDs：飞机的ID，表示主动依附的飞机。

AttatchIDs：被依附飞机的ID，表示被动依附的飞机。

AttatchTypes：依附的样式，有四种模式可选：

0：正常模式，表示相对位置和姿态都保持不变。

1：相对位置不相对姿态，表示相对位置保持不变，但姿态可以改变。

2：相对位置+偏航（不相对俯仰和滚转），表示相对位置可以改变，但俯仰和滚转保持不变，只能偏航改变。

3：相对位置+全姿态（俯仰滚转偏航），表示相对位置和姿态都可以改变。

相关文献

3ds Max 2021 帮助 | 层次和运动学 |

1. Autodesk
2.

附加资源

官方文档：RflySim官方文档： https://rflysim.com/doc/zh/

社区交流：加入RflySim技术交流群：951534390