InFloatsCollision 碰撞数据输入接口验证实验原理

1. 文件目录

例程目录：[安装目录]\RflySimAPIs\4.RflySimModel\0.ApiExps\14.inCopterData\

文件夹/文件名称	说明
1.InFloatsCollision\Readme.pdf	基于 InFloatsCollision 接口的碰撞引擎功能验证实验步骤
2.inFloatsCollisionApi\Readme.pdf	基于 InFloatsCollision 接口四旋翼模型碰撞坠机实验步骤
3.inFloatsCollisionMotorFail\Readme.pdf	基于 InFloatsCollision 接口碰撞触发四旋翼电机故障坠毁实验步骤

2. 总体说明

2.1. 碰撞检测和响应的需求

为了保证三维场景的真实性，不仅要能从视觉上如实看到虚拟环境中的虚拟物体以及它们的表现，而且能准确模拟场景中物体之间的交互，这就首先要求虚拟环境中的固体物体是不可穿透的。这就需要仿真系统能够及时检测出这些碰撞，产生相应的碰撞反应，并及时更新场景输出，否则就会发生穿透现象。碰撞问题一般分为碰撞检测与碰撞响应两个部分，碰撞检测的任务是检测到有碰撞的发生及发生碰撞的位置。碰撞响应是在碰撞发生后，根据碰撞点和其他参数促使发生碰撞的对象做出正确的动作，以符合真实世界中的动态效果。

在三维仿真场景中，通常有很多静止的环境对象与运动的活动物体，每一个虚拟物体的几何模型往往都是由成千上万个基本几何元素组成，虚拟环境的几何复杂度使碰撞检测的计算复杂度大大提高，同时由于虚拟现实系统中有较高实时性的要求，要求碰撞检测必须在很短的时间(如 30~50ms)完成，因而碰撞检测有时会成为实时仿真系统的瓶颈。

为实现碰撞响应，需要计算接触法向力 Fn 和切向摩擦力 Ff，每个时间步都需要一个碰撞检测过程，当接触物体的几何形状不平坦时代价较高（例如使用网格指定复杂形状）。

2.2. 碰撞数据的来源

RflySim3D 中根据载具模型的物理资产包围盒，以 6 面射线相交测试获得的距离数据，快速回传给 CopterSim 中 dll 模型的 InFloatsCollision 接口

InFloatsCollision 接口数据解析如下

1 校验位 12345

2 被碰物体 id

3 质量比

4~6 被碰物体位置

7~9 被碰物体速度

10~15 六面射线返回距离

2.3. 碰撞响应模块的设计思路错误!未找到引用源。

2.3.1. 模拟与移动物体的交互

在无人系统中，最常发挥作用的物理定律是质量和动量守恒。

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} = \mathbf{f}(t, \mathbf{q}, \dot{\mathbf{q}})$$

上述方程等价于F＝m・a。由于动态系统由几个在3D空间中移动的物体组成，物体的质量 m 被替换为广义质量$M(q) \in \mathbb{R}^{N\times N}$，该质量取决于 N 个广义坐标$\boldsymbol{q} \equiv [\boldsymbol{q}(1), \dots, \boldsymbol{q}(N)]^T$。需要解决的未知量是加速度 、速度 和广义坐标 q。

通常使用半隐式欧拉积分（一种用于求解哈密顿方程的数值方法）来求解上述方程，首先需要给定一些初始条件，比如机器人的初始位置 (q_0) 和初始速度，然后用一个很小的时间间隔 Δ t 来模拟机器人的运动过程，每隔 Δ t 秒就计算一次机器人的位置和速度，直到达到最终的时间 。依此类推，求加速度等价于求解线性方程组$\boldsymbol{Ax} = \boldsymbol{b}$, 其中$\mathbf{A} = \mathbf{M}(\boldsymbol{q}_0), \boldsymbol{x} = \ddot{\boldsymbol{q}}_0$, 且$\boldsymbol{b} = \boldsymbol{f}(t_0, \boldsymbol{q}_0, \dot{\boldsymbol{q}}_0)$。注意，A 和 b 随每个时间步长而变化。

2.3.2. 模拟与场景固定物体的交互

从宏观上看，有两种主要方法可以计算接触法向力 Fn 和切向摩擦力 Ff，即惩罚方法和互补方法，RflySim 平台目前的碰撞模型实现偏向于惩罚方法。

惩罚方法

$$F_n = k_n \delta_n^{\alpha} + k_c \dot{\delta}_n$$

$$F_f = min(k_t \delta_t, \mu F_n)$$

这种方法的主要参与者是 δ n，即 A 和 B 两个物体之间相互接触的界面处的局部法向变形。其中 k n 和 k c 是材料和形状依赖的参数，且 1 ≤ α ≤2。对于任何基于惩罚的解决方案，碰撞检测都应高度准确，因为法向刚度 kn 假设值较大。对于钢，kn≈ 1 × 10^7N/m; 这种刚度乘以 1 × 10^−6m 量级的局部变形.隐含地，碰撞检测应该在小于一微米的范围内准确，这是一个很高的要求。

由于在大多数动力学模型中，物体被视为刚体以节省计算时间，惩罚方法导致了一个悖论：刚体如何产生变形 δ n? 在这时，δ n 被解释为 A 和 B 的几何形状的重叠，而不是变形。这种与惩罚法相关的手法也影响了摩擦力 Ff的计算。在这种情况下，第二个变形 δ t 与一个虚拟的硬弹簧相关，该弹簧作用在切平面上，并与 A 和 B 上的接触点相连。这个虚拟弹簧在其内部负载达到 μ Fn 时饱和。切平面随着 A 和 B 的位置和姿态的变化而变化，从而导致 δ t 的变化。这意味着对于每个接触，算法都将 δ t 作为状态传递，并更新其值以计算 Ff，其中 kt 是经验模型参数，μ 是摩擦系数，具体取决于滑动状态（静摩擦或动摩擦）。

惩罚法不会增加问题的规模：动态约束方程中的方程数不会因为有多少接触事件而增加。此外，这种方法易于理解、易于实施，并且具有良好的可扩展性。然而，这种方法在使用过程中需要大量的调整，对步长施加了严格的限制（因为存在静态的虚拟弹簧），并且

需要许多参数的值（如 kn、kt、α 等）。

互补方法

$$0 \leq \Phi \perp \lambda_{AB} \geq 0$$

$$0 \leq v_t \perp (\mu F_n - F_f) \geq 0$$

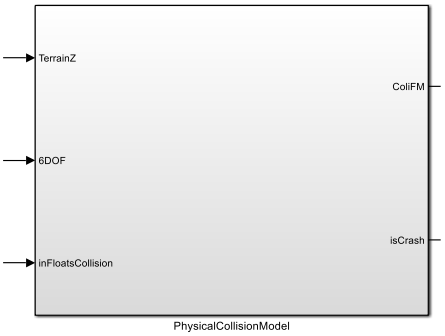
在这种方法中，会监控物体 A 和 B 之间的有符号距离 Φ，并提出物体不应穿透的条件，即 Φ ≥ 0。这是一个与接触体几何形状密切相关的运动约束。当 Φ = 0 时，问题中存在一个运动约束，与此同时，存在一个拉格朗日乘数 λ AB ≥ 0，λ AB 的正值表明接触产生了防止物体穿透的力。注意，Φ ≥ 0 和 λ AB ≥ 0，但它们的乘积始终为零。这是因为只有当 Φ = 0（物体 A 和 B 接触）时，才会出现法向力 λAB；这代表了一个互补条件（即当一个为 0 时，另一个必须大于等于 0；只有一个存在时，另一个就必须为 0），0≤Φ ⊥ λAB≥0，因此得名互补法。

摩擦力的计算也依赖于互补条件：0 ≤ vt⊥ (μFn－Ff) ≥ 0，其中 vt 是 A 和 B 之间的滑动速度。这个条件以及观察到只要 0 < vt，摩擦力 Ff 就会与相对滑动的方向相反，是数值算法计算 Fn 和 Ft 所需的要素。

相比惩罚法，除了不引入 δn 或 k n，互补法也不需要 δt 或 k t。优势在于：需要跟踪的变量更少（δn 和 δt），需要提出的模型参数更少（k n 和 k t）；由于不需要担心 δn 和 δt，碰撞检测算法可以更粗糙；此外，由于模型不需要刚性弹簧，模拟可以在大 Δt 下进行，而不会使数值积分不稳定。然而，互补法也存在挑战：首先，实现复杂；其次，计算 Fn 和 Ft 会导致一个耦合问题，将整个系统的所有摩擦力和法向力结合在一起。

3. 关键功能的实现

3.1. 四旋翼模型碰撞检测及响应的实现



碰撞检测和响应的功能主要通过 PhysicalCollisionModel 模块实现，其中的模块模拟了与移动物体和场景固定物体的交互

移动物体反作用力计算如下：

```
massOb=mass*uFloats(3)^2; 计算被碰撞物的广义质量，uFloats(3)表示质量比
veOb = uFloats(7:9); 获取被碰撞物的速度
veNew=(mass*ve+massOb*veOb)/(mass+massOb); 计算碰撞后本飞机的新速度，本飞机和被碰撞物的动量之和除以它们的质量之和。
mv0=mass*(veNew-ve); 计算冲量，本飞机的质量乘以速度的变化
mv=DCM*mv0; 将冲量旋转到体坐标系下。
mv(1)=mv(1)*(0.7+0.3*rand());：计算 xb 方向上的冲量。原始冲量乘以一个介于 0.7 和 1 之间的随机数以模拟不确定性。
mv(2)=mv(2)*(0.7+0.3*rand());：计算 yb 方向上的冲量，处理方式与 x 方向相同。
mv(3)=mv(3)*(0.7+0.3*rand());：计算 zb 方向上的冲量，处理方式与 x 方向相同。
fOut(1:3) = mv/0.05;     冲量除以时间得到反作用力
```

场景固定物体反作用力计算如下：

```
%后障碍物碰撞反馈力
        z=uFloats(11);获取与后障碍物的碰撞距离重叠部分
根据重叠距离不同调整 kn 和 kc 计算F_n = k_n δ_n^α + k_c δ̇_n
        if z>-0.01
            ddm(1)=-10*ve(1);
        elseif z>-0.04
            ddm(1)=-200*z-20*ve(1);
        else
            ddm(1)=-500*z-50*ve(1);
        end
    end
```

- **完整的碰撞效果实现和整体 Simulink 模块解析可参考[4]中的 3.1 四旋翼模型碰撞检测及响应的实现**

3.2. Python 脚本控制 2 架四旋翼相撞

\3.inFloatsCollisionMotorFail\CollisionDemo.py 中关键代码如下：

```
import time
import math
import sys
import PX4MavCtrlV4 as PX4MavCtrl
import UE4CtrlAPI
```

导入必要的库，其中：

- PX4MavCtrl 飞机控制接口的详细协议可参考[2]中的控制接口部分
- UE4CtrlAPI 场景控制接口的详细协议可参考[3]中的外部接口文件部分

```
ue = UE4CtrlAPI.UE4CtrlAPI()
```

为打开的 RflySim3D 创建 1 个通信实例

```
mav = PX4MavCtrl.PX4MavCtrler(1)
```

```
mav2 = PX4MavCtrl.PX4MavCtrler(2)
```

为打开的 2 个 CopterSim 创建 2 个通信实例

```
mav.SendPosNED(0, 0, -5, 0)
mav2.SendPosNED(2, -2, -5, 0)
```

利用位置控制接口为两架飞机设置目标航点

```
mav.SendMavArm(True)
mav2.SendMavArm(True)
```

解锁飞机，理论上此时飞机才开始起飞并向目标点运动，但这里的 dll 模型的输入没有解锁标志位，所以这里没有实际作用

```
ue.sendUE4Cmd('RflyChangeViewKeyCmd P',0)  #开启碰撞引擎
```

开启 RflySim3D 的碰撞模式

```
mav.SendVelNED(1, 0, 0, 0)
```

限制 1 号飞机的 x 方向速度最大为 1m/s

下列代码主要用于检测并处理两个飞行器（简称为 V1 和 V2）的碰撞情况。

初始化变量

```
hasV1Crashed=False
hasV2Crashed=False
startTime = time.time()
lastTime = time.time()
timeInterval = 1/30.0 # time interval of the timer
```

- hasV1Crashed 和 hasV2Crashed：这两个布尔变量用于追踪每个飞行器是否已经发生了碰撞。初始值都设置为 False，表示开始时没有碰撞发生。
- startTime 和 lastTime：这两个变量记录了时间信息，用于控制循环的执行频率。startTime 在这段代码中没有被使用。
- timeInterval：定义了时间间隔，这里设置为 1/30.0，即每秒 30 帧的频率。这意味着代码的执行间隔大约是 0.033333 秒。

循环检测飞行器的碰撞状态

```
while True:
    lastTime = lastTime + timeInterval
    sleepTime = lastTime - time.time()
    if sleepTime > 0:
        time.sleep(sleepTime)
    else:
        lastTime = time.time()

    if (not hasV1Crashed) and mav.isVehicleCrash:
        print('Vehicle #1 Crashed with vehicle #',mav.isVehicleCrashID)
        hasV1Crashed=True

    if (not hasV2Crashed) and mav2.isVehicleCrash:
        print('Vehicle #2 Crashed with vehicle #',mav2.isVehicleCrashID)
        hasV2Crashed=True

    if hasV2Crashed and hasV1Crashed:
        time.sleep(4)
        break
```

1. **时间调度：**

- 首先，通过 lastTime + timeInterval 计算下一次执行的目标时间。
- 然后，计算 sleepTime，即当前时间到下一次执行时间的时间差。如果 sleepTime 大于 0，意味着程序需要等待一段时间以达到目标频率。如果 sleepTime 小于或等于 0，表明程序已经落后于目标频率，需要立即执行下一次循环，并重置 lastTime 为当前时间。

2. **碰撞检测：**
   - 使用 mav.isVehicleCrash 和 mav2.isVehicleCrash 检测两个飞行器是否碰撞。如果 hasV1Crashed 或 hasV2Crashed 为 False，并且相应的 mav.isVehicleCrash 或 mav2.isVehicleCrash 为 True，则表示相应的飞行器发生了碰撞。
   - 发生碰撞时，打印碰撞信息，并将对应的 hasV1Crashed 或 hasV2Crashed 设置为 True。
   - 当两个飞行器都发生了碰撞（即 hasV1Crashed 和 hasV2Crashed 都为 True）时，程序会等待 4 秒钟，然后跳出循环。

4. 相关文献

[1]. The Role of Physics-Based Simulators in Robotics | Annual Reviews

[2]. ..\..\..\6.RflySimExtCtrl\API.pdf

[3]. ..\..\..\3.RflySim3DUE\API.pdf

[4]. 平台多旋翼模型控制仿真实验原理..\..\2.AdvExps\e2_MultiModelCtrl\Intro.pdf

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