# 自动驾驶仿真软件性能一致性验证与优化*

李纪伟　魏亚格　张梦蝶　王润民

（长安大学，西安 710018）

【摘要】为保证自动驾驶仿真测试的可重复性和测试结果的可靠性，针对自动驾驶仿真软件性能一致性验证与优化问题，在系统分析自动驾驶仿真软件性能一致性问题定义及潜在原因的基础上，基于Unity和虚幻引擎（Unreal Engine）搭建自动驾驶仿真测试平台，并完成了静态场景和动态场景的搭建，以及测试用例、数据处理脚本、自动化测试脚本和整体试验流程的设计，通过非一致性偏差对基于Unity和Unreal Engine的仿真测试平台的结果进行量化，受仿真系统软硬件等各方面因素影响，在多次测试中存在显著的非一致性偏差，体现在最大轨迹偏差达到15%，针对发现的偏差，提出了包括进程优化、电源模式优化等多种优化方案。试验结果表明，所提出的优化措施将偏差降低40%以上，提升了仿真测试的可靠性和一致性。

**关键词：自动驾驶技术　仿真测试　性能一致性**

**中图分类号：**U467.13　　**文献标志码：**A　　**DOI：**10.20104/j.cnki.1674-6546.20240280

# Verification and Optimization of Performance Consistency in Autonomous Driving Simulation Software

Li Jiwei, Wei Yage, Zhang Mengdie, Wang Runmin

(Chang'an University, Xi'an 710018)

**【Abstract】**In order to ensure the repeatability of simulation tests and reliability of test results, a simulation testing platform is built using Unity and Unreal Engine based on a systematic analysis of the definition and potential causes of performance consistency issues in autonomous driving simulation software, and static and dynamic scenarios are built at the same time. The study also involves the design of test cases, data processing scripts, automated testing scripts and the overall experimental process. Test results based on Unity and Unreal Engine simulation test platform are quantified through the inconsistency deviation. Influenced by various factors related to the simulation system's software and hardware, experimental results reveal significant inconsistencies in multiple tests, with the maximum deviation reaching 15%. To address the identified deviation, this paper proposes several optimization strategies, including process optimization and power mode optimization, which are validated to reduce the deviation by more than 40%, thereby enhancing the reliability and consistency of simulation tests.

**Key words: Autonomous Driving Technology, Simulation Testing, Performance Consistency**

**【引用格式】**李纪伟, 魏亚格, 张梦蝶, 等. 自动驾驶仿真软件性能一致性验证与优化[J]. 汽车工程师, 2025(5): 1-12.

LI J W, WEI Y G, ZHANG M D, et al. Verification and Optimization of Performance Consistency in Autonomous Driving Simulation Software[J]. Automotive Engineer, 2025(5): 1-12.

## 1　前言

自动驾驶汽车代表了现代汽车技术的发展方向，其在提高驾驶安全性和效率方面具有广泛应用前景。同时，自动驾驶系统的研发也越来越依赖于仿真测试，这种方法能够在虚拟环境中模拟各种道路场景，从而在降低成本的同时进行大规模测试[1]。仿真测试不仅可以评估自动驾驶系统的性能，还能

*基金项目：国家自然科学基金项目（52232015）。

进行成本效益分析，为决策提供重要数据。当前，自动驾驶技术的发展离不开仿真测试的支持，其重要性日益突出。

自动驾驶仿真测试软件是开展上述测试的核心工具。它能够模拟多种道路场景，评估自动驾驶系统在不同情况下的表现[2]。然而，即使在相同的测试条件下，仿真测试软件的测试结果也可能存在差异，即出现测试结果的不一致，直接影响测试结果的有效性和可靠性。因此，验证和优化仿真测试软件的性能一致性是亟待解决的问题。

目前，国内外对自动驾驶仿真测试软件性能一致性的研究较少，尤其是针对不同类型仿真软件的一致性研究更为稀缺。自动驾驶仿真测试软件的性能一致性验证成为提高系统性能评估准确性和安全性的关键环节。现有的研究主要集中在以下几种验证方法[3-4]：基于随机策略的测试方法通过随机选择测试用例进行性能测试，可以获得较高的覆盖率，但耗时较长；基于模型的测试方法利用模型拟合技术进行测试，能够提高测试效率，但对模型准确性有较高要求；基于优化的测试方法通过优化算法对软件进行测试，能有效提高测试效率，但实施较为复杂。这些方法在自动驾驶仿真测试软件中应用广泛，但仍面临挑战，如仿真环境的复杂性和测试用例的增加。特别是基于游戏引擎的仿真软件，虽然应用广泛[5]，但在自动驾驶仿真测试中常出现性能不一致的问题，这影响了试验的可重复性和整体可靠性[6]。

本文旨在探索一种科学的自动驾驶仿真测试软件性能一致性验证方法，并针对常见的一致性偏差进行定量评估与优化，基于Unity和虚幻引擎（Unreal Engine）平台进行试验，通过详细的数据分析对仿真过程中的非一致性偏差进行量化，并提出相应的优化方案，以期提升测试软件的性能一致性，为自动驾驶系统的性能评估提供可靠支持。

## 2　自动驾驶仿真软件性能一致性问题分析

### 2.1　自动驾驶仿真软件

随着高级自动驾驶辅助系统和自动驾驶技术的发展，自动驾驶仿真软件也在不断发展进步。为进一步提升自动驾驶仿真测试的真实性及有效性，新一代仿真测试软件都具备了直接导入路网数据、高精度地图及真实路测数据的功能，同时使用高仿真度的主流游戏引擎如Unity、Unreal Engine进行测试场景的构建和渲染[7]。目前市面上的仿真软件较为丰富，其中既有仿真软件研发公司专门研发的商业化仿真软件，也有各大科技公司为支持旗下自动驾驶技术发展而搭建的各类测试平台，传统汽车企业也在不断寻求与软件开发商及自动驾驶解决方案提供商的密切合作，开发自己的自动驾驶仿真测试软件，以满足自动驾驶车辆商业化所需的海量测试需求[8]。

目前可供选择的新一代自动驾驶仿真测试软件有很多，且各具优点，共性在于它们大多基于主流游戏引擎开发。这些游戏引擎都经历过游戏及电影行业的考验，它们在物理仿真领域所提供的真实感可以满足自动驾驶仿真测试的诸多有效性需求。极为真实的测试环境渲染，也便于试验环境调整和试验结果观察。这些面向游戏开发的引擎同时也具备易上手、易编辑和易运行的特点，有效提升了仿真测试的效率。然而，游戏引擎往往只注重玩家的沉浸式体验，在场景的一致性构建上有所疏忽。因此，本文基于两款主流游戏引擎搭建仿真环境，开展一致性的相关验证。

### 2.2　仿真软件性能一致性概述

本文讨论的仿真测试软件性能的一致性与仿真的一致性有所不同。仿真的一致性关注虚拟场景与现实道路场景之间的差异，虽然这种差异会影响仿真结果的适用性和试验有效性，但随着仿真技术和图像建模技术的进步，这种差异正逐渐减小。而仿真测试软件性能的一致性，尽管常被忽视，却是影响仿真试验有效性的重要因素。

在舒曼等人的描述下，一致性体现为一种类似因果的关系，即在满足一致性条件下，所有参与者的状态完全由上一时间点的状态决定[6]。在自动驾驶仿真中，这意味着所有参与者的状态由先前状态决定，如果仿真软件性能在每次试验中保持一致，那么每次试验在相同时间点输出的数据应当一致。自动驾驶仿真测试要求高可重复性和有效性：可重复性保证试验可以在相同条件下反复进行，从而减少工作量和加速数据积累；有效性则确保试验结果在实际道路情景中有意义。只有在满足一致性条件的环境下进行的仿真测试，才能同时具备这两个条件，从而确保测试的价值。

然而，以往的仿真常默认只要给定相同的初始条件和流程，就会产生一致的仿真输出。但在实际模拟测试中，即使保持软、硬件配置和试验环境设

置不变，游戏引擎渲染的试验情景仍可能出现不一致，这种不一致性在试验过程中最直观的表现往往是各参与者在每次试验中运动轨迹的差异。对于试验中出现的轨迹偏差，如果偏差较小，通常在低精度的仿真测试中可以忽略，或在某些城市环境标准下被允许。但较大的偏差会严重影响试验的可重复性和有效性，浪费测试时间和资源。基于上述分析，本文将重点验证仿真测试软件的性能一致性，以量化评估偏差对仿真结果的影响，并提出改进措施，以将不一致性缩小到可接受范围，降低其对仿真测试的影响。

### 2.3 性能一致性缺乏的原因分析

目前主流的自动驾驶仿真测试软件如CARLA、AirSim（基于Unreal Engine）和PreScan（基于Unity），通常依赖于游戏引擎实现仿真功能。这些引擎最初是为提升游戏体验而设计，但在自动驾驶仿真中需满足一致性等额外需求。游戏引擎的工作过程分为数据输入、数据计算和数据输出3个部分，如图1所示，时间以帧为单位积累[9]。逻辑帧（游戏引擎循环的时间）通常固定，而图像帧（图形处理器渲染时间）则受平台性能影响，可能导致超时[10]。

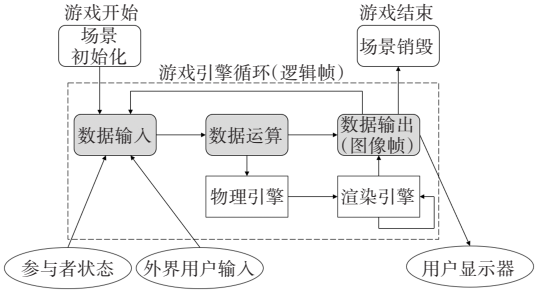



图1 游戏引擎工作过程原理

逻辑帧的帧率通常高于图像帧，以确保游戏世界的平稳运行。但当计算资源负荷高时，图像帧的渲染可能超时，导致渲染结果与逻辑时间推移产生偏差，这是游戏引擎性能不一致性的根本原因。尽管一些游戏为了流畅体验牺牲了准确性，但自动驾驶仿真需要高精度，这种不一致性使得游戏引擎不完全适用于自动驾驶仿真。因此，图像帧超时可视为引起仿真软件性能不一致的关键因素，其可能受软件配置（如浮点数运算、线程调度、引擎设置、算法随机数）和硬件配置（如中央处理器、图形处理器）影响。

此外，还有一些因素直接影响仿真一致性。测量误差源于传感器精度不足、环境干扰或标定不准确，导致输入数据不准确，进而影响仿真结果的一致性。计算误差可能因浮点运算精度限制、算法随机性或计算资源不足而引入，这些误差会逐渐积累并加剧数据偏差。数据传输延迟和网络带宽限制可能在网联环境中导致数据不同步，从而影响仿真一致性。同时，初始化误差、状态同步误差及模型不匹配也会引起仿真结果的偏差。本文将综合考虑这些因素，通过控制和优化软件、硬件及系统误差，探究有效的优化方法，以降低图像帧超时和其他因素对仿真一致性的影响，从而提升自动驾驶仿真软件的整体性能一致性。

## 3 仿真软件性能一致性验证试验设计

### 3.1 试验场景搭建

#### 3.1.1 静态场景构建

本文基于当前流行的两款游戏引擎——Unity和Unreal Engine，分别搭建自动驾驶仿真测试系统，探究主流框架下仿真测试软件的性能一致性。选取典型的城市交通场景进行构建，涵盖了真实城市交通场景中的建筑、街道、车辆、交通信号灯、行人、斑马线等要素。

静态场景的构建主要针对场景中不发生移动的元素进行建模[10]。常用的建模方法包括基于高精度地图的三维建模和利用增强现实技术完成场景建模。其中，基于高精度地图的方法更为普遍，能够快速生成精度较高的仿真测试场景，同时，高精度地图中丰富的道路元素信息便于测试人员对场景进行细粒度调整[11]。为真实还原城市交通场景中的道路状况，本文从开放街道地图（OpenStreetMap）上截取了一块现实中的城市交通道路区域，并将该区域的道路数据导出为路网文件用于道路建模。然后，将从OpenStreetMap获得的道路数据文件导入CityEngine进行具体的静态场景建模。导入后的路网如图2所示。由于格式转换过程中可能发生数据丢失，实际得到的道路模型可能与现实中的道路存在差异，需要在CityEngine中手动调整有问题的道路。在调整完成后，生成道路和建筑的三维模型，并对建筑模型的密度和高度进行调整，以保证测试程序的流畅性和稳定性，避免测试系统崩溃及试验数据丢失。

在CityEngine中基本完成城市街道场景的建模后，将模型以FBX格式导入Unity，仅需调整模型贴图及添加背景环境（如天空、地面、光照等）即可，最终在Unity中搭建的静态场景如图3所示。

相较于Unity的导入过程，Unreal Engine还支持将模型转换为glft格式进行导入。导入城市模型后，依然需要对模型贴图进行调整，并添加天空、地面和光照等背景环境。此外，根据Unreal Engine的设计经验，还需关闭引擎默认开启的动态模糊、曝光、抗锯齿等画面增强功能，以确保测试程序的流畅度和试验数据的稳定性，最终搭建的场景如图4所示。

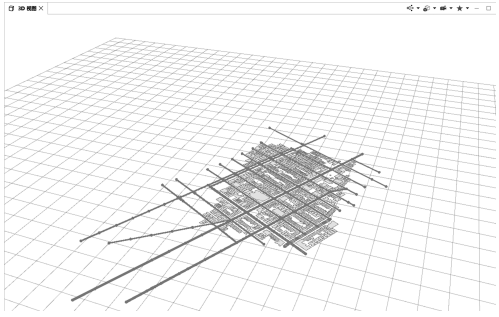


图2　CityEngine中的路网

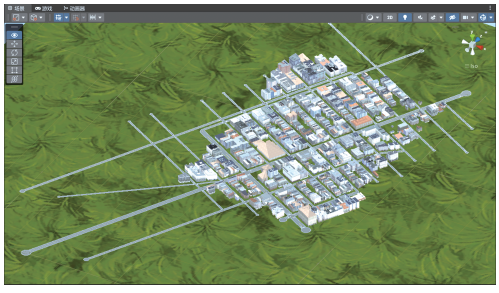


图3　Unity中的静态场景

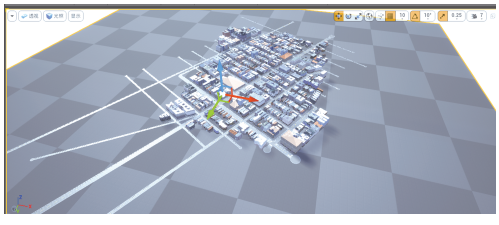


图4　Unreal Engine中的静态场景

### 3.1.2　动态场景构建

本文试验中，设定背景车辆按照规定路径行驶，过程中由脚本控制以实现前车避让和交通信号灯等候，从而反映物理引擎对车辆行驶轨迹的影响。Unity资源商店提供了多款车辆模型，本文选用其中的部分模型进行车辆的静态建模，并根据实际车辆的标准对模型的尺寸、质量和速度进行规范化调整。

车辆的逻辑行为主要由C#脚本控制，试验中车辆搭载了物理仿真、车辆移动、避让控制和轨迹记录4类脚本。物理脚本用于模拟车辆动力学、协调车轮旋转，可直接使用Unity示例中的车辆物理脚本。车辆移动脚本控制车辆按轨迹移动，需预先划定行驶轨迹，并通过循环读取控制点，控制车辆按路径行驶，匀加速至最大速度后保持匀速。避让控制脚本通过Unity中的射线（Ray）类发射射线，获取前车间距，当距离小于设定的门限值时，车辆匀减速以避免碰撞，当距离大于门限值后，车辆重新进入匀加速状态并继续沿路径行驶，算法流程如图5所示。轨迹记录脚本在每一帧中获取车辆位置数据并记录在数组中，试验结束时将数据存储至csv文件中。本文选择了6辆不同路径的背景车辆进行轨迹记录，以分析其行驶轨迹的差异，验证仿真测试软件的性能一致性。

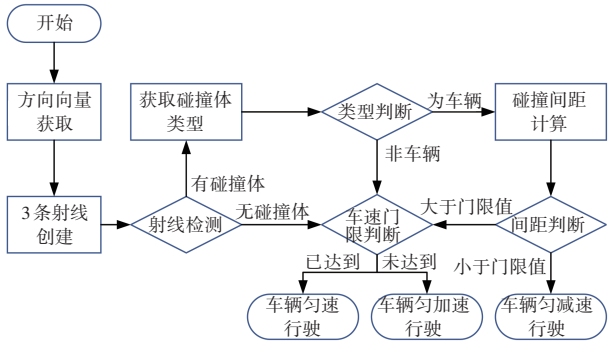



图5　车辆避让算法判断流程

行人的建模同样来源于Unity的资源库，这些模型已附带行走动画，便于直接使用。行人移动脚本的编写与车辆移动脚本相似，也按固定路径移动。交通信号灯的引入提高了交通场景的真实性，交通信号灯建模扩展为路口交通环境建模，包含交通信号灯和控制交通流的逻辑长方体。交通信号灯控制红、黄、绿光源按固定时间间隔亮起和熄灭。逻辑长方体控制区域内车辆或行人依据交通信号灯状态移动和停止，以确保遵守交通规则。

Unreal Engine同样提供了物理脚本，其他脚本的编写思路与Unity基本一致。Unreal Engine中的轨迹记录脚本需要通过根组件（RootComponent）获取位置数据并进行向量拆分，以得到车辆在仿真环境中$x$、$y$轴方向的具体位置数据。避让控制脚本使用Unreal Engine提供的对象盒体检测方法，与Unity的射线命中信息（RaycastHit）类似，沿线条扫描命中的第一个车辆对象的位置。两个引擎在脚本编写上的最大区别在于脚本形式不同。在Unreal Engine中，脚本主要以蓝图形式存在，例如轨迹记录脚本中的记录（Record）函数在蓝图中的实现如图6所示。

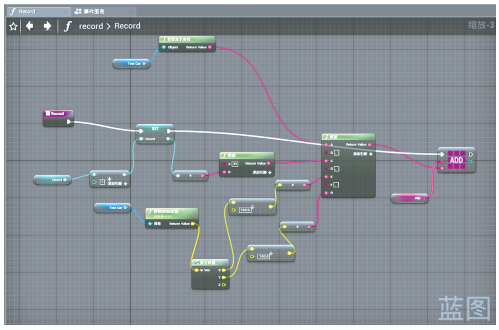



图6　Record函数蓝图

### 3.2　试验方案设计

本文的整体试验流程如图7所示。

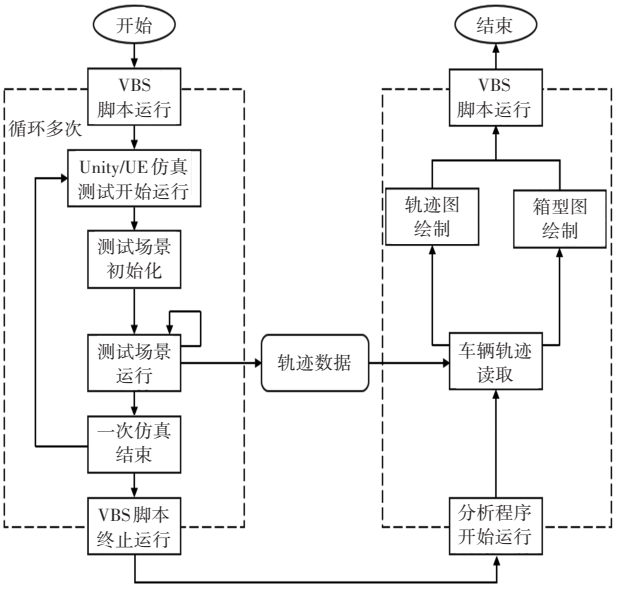



图7　自动驾驶仿真软件性能一致性验证流程

#### 3.2.1　测试用例

本文的测试用例设计主要基于前文的测试场景设计，并由动态场景的运行逻辑决定。在自动驾驶仿真测试的背景下，车辆作为测试用例的主体，每轮重复试验中选定背景车辆的行驶轨迹即为一个测试用例。

在动态场景搭建过程中，背景车辆沿预定路径行驶。为确保测试数据的多样性和全面性，这些路径包括直行和转弯，同时设计了前车避让和交通信号灯穿越等场景。不同背景车辆在试验中遇到交互事件的频率有所不同。各背景车辆在每轮重复试验中都具有相同的初始条件，并在固定路径上经历相同的交互事件，从而保证其行驶轨迹在理论上具有可重复性。

#### 3.2.2　数据处理

通过为仿真车辆挂载轨迹记录脚本，记录被测车辆在每轮测试中的行驶轨迹。采集的数据包括仿真运行时间、车辆ID、车辆的$x$坐标和$y$坐标。这些数据通过脚本获取，用于标识车辆、确定相同时间点的轨迹，并计算轨迹偏差。

轨迹记录脚本在每轮试验结束后，将车辆轨迹数据输出为csv文件。虽然记录时间间隔理论上为20 ms，但受系统资源利用率的影响，实际记录间隔可能有所偏差。每辆车的轨迹数据分别存储，最终形成轨迹数据集。

本文使用欧氏距离定义某两次测试轨迹数据间的偏差，计算参与者$a_1$在$t$时刻的第$n$次和第$m$次行驶轨迹的偏差：

$$\sigma_{a1}(t,n,m)=\sqrt{\left(x_n^2(t)-x_m^2(t)\right)+\left(y_n^2(t)-y_m^2(t)\right)} \tag{1}$$

式中：$x_n(t)$、$x_m(t)$分别为被测车辆第$n$次、第$m$次试验中$t$时刻的$x$坐标，$y_n(t)$、$y_m(t)$分别为被测车辆第$n$次、第$m$次试验中$t$时刻的$y$坐标。

在整体评价指标的获取上，先分别计算6辆被测背景车辆在全部10轮重复测试中每个时间点的平均轨迹偏差，定义为被测车辆的平均偏差。假设每次重复的仿真测试中，以20 ms的固定时间间隔进行轨迹记录，到仿真结束为止共产生$s$条位置记录，则被测车辆$a_1$的平均偏差为：

$$\overline{\sigma_{a1}}=\frac{\sum\sigma_{a1}(t,n,m)}{s\cdot\sum_{i=1}^{9}i} \tag{2}$$

式中：$\sum\sigma_{a1}(t,n,m)$为测试车辆1在所有测试中每两次测试之间轨迹偏差的总和。

再对6辆背景车各自的平均偏差求平均数来获得评价仿真软件一致性表现的最终评价指标，即观察到的所有被测车辆的整体轨迹偏离情况，将其定义为仿真软件整体的平均偏差：

$$\bar{\sigma}=\frac{\sum_{i=1}^{6}\overline{\sigma_{ai}}}{6} \tag{3}$$

最后将测试数据绘制为轨迹图和箱式图，以直观展示测试过程中轨迹的偏差和分布特征，从而帮助分析软件一致性及其影响因素。

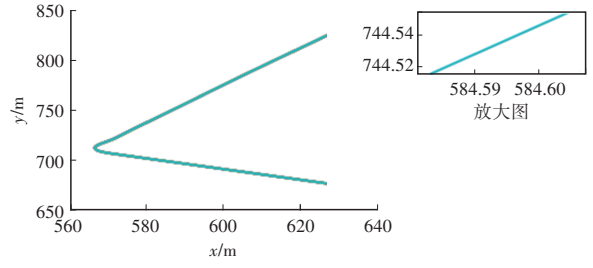
## 4　一致性验证与优化

### 4.1　一致性验证

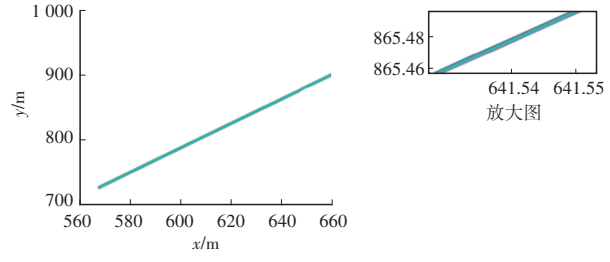
以在Unity平台上的仿真试验为例，其运行情况如图8所示。

图8　Unity平台试验

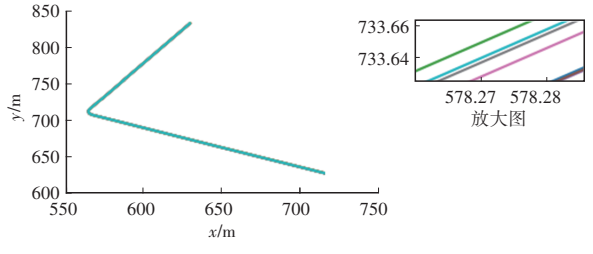
将试验所得的首批行驶轨迹数据通过Python程序进行可视化处理，生成了测试车辆的行驶轨迹如图9、图10所示。从图9、图10中可以看出，大部分测试车辆在各次测试中表现出的轨迹偏差较小，整体上各车辆的10次测试轨迹几乎重合，其中在Unity和Unreal Engine平台试验中获得的整体平均误差分别为0.32 m和0.67 m。然而，部分车辆表现出较为明显的异常，如基于Unreal Engine的测试中车辆1的轨迹整体差异显著，该车自身的平均偏差达到1.1 m。



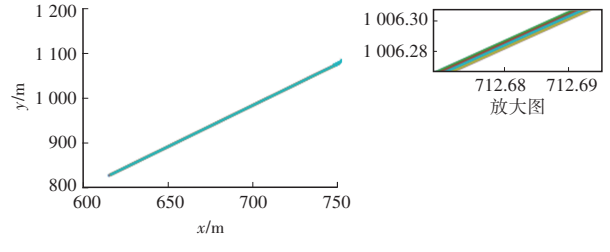

（a）测试车辆1



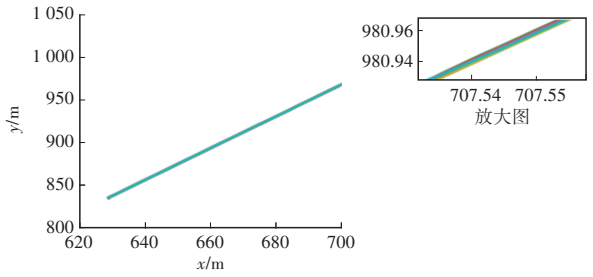

（b）测试车辆2



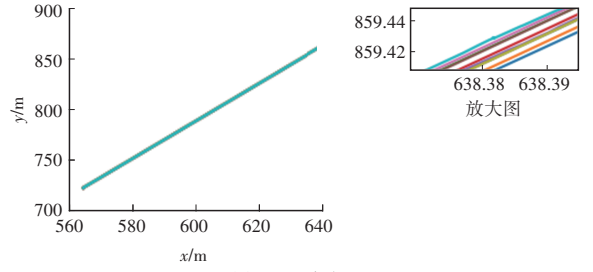

（c）测试车辆3




（d）测试车辆4

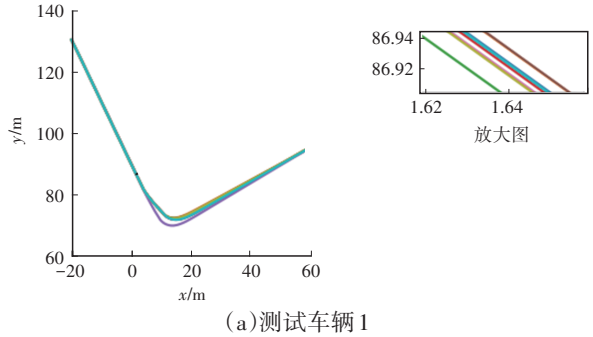



（e）测试车辆5

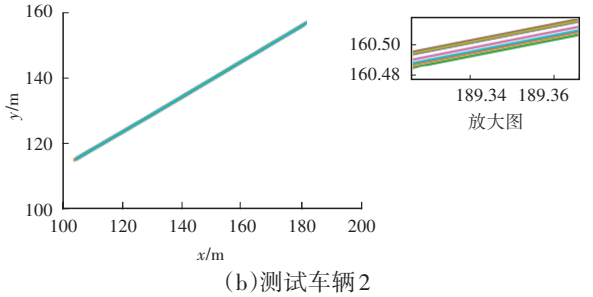



（f）测试车辆6

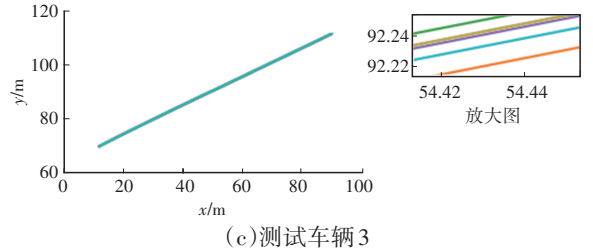
图9　Unity测试轨迹



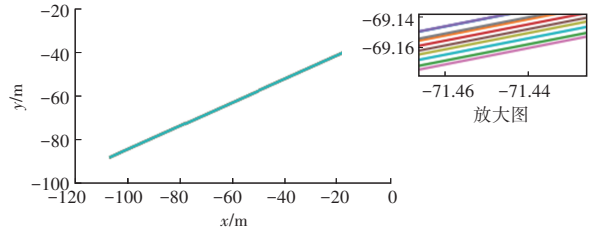

（a）测试车辆1



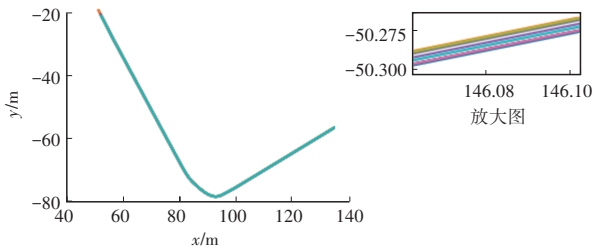

（b）测试车辆2



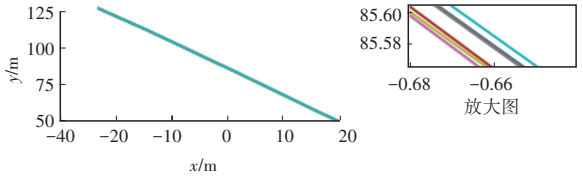

（c）测试车辆3




（d）测试车辆4




(e)测试车辆5

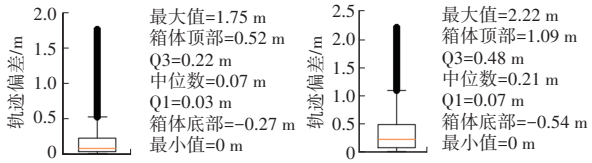



(f)测试车辆6

图10　Unreal Engine测试轨迹

进一步放大观察区域并对局部轨迹进行详细分析，可以发现所有测试车辆在每轮重复测试中都存在一定程度的行驶轨迹偏离。这一现象表明，仿真软件在测试过程中并未始终保持性能的一致性，在每次重复测试中均出现了非一致性现象。

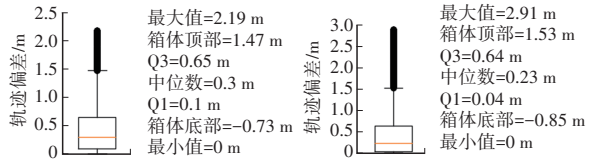
图11所示为Unity平台测试的箱式图。本文对多个被测车辆在多轮重复测试中的轨迹数据进行了偏差计算，生成了大量数据（如在Unreal Engine测试中共产生178 500条偏差数据），其中包含较多异常值。尽管异常值较多，但其在偏差数据中的占比并不高，例如Unreal Engine测试中的异常偏差数据占总偏差数据的5.15%。这些异常值表明，仿真软件在运行过程中存在极不稳定的仿真状态，导致场景参与者出现较大的状态差异，严重影响了仿真测试的稳定性和有效性，进一步证实了基于游戏引擎的仿真过程缺乏一致性保障。




(a)测试车辆1



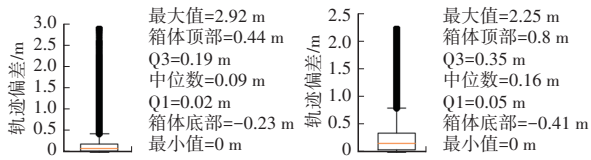

(b)测试车辆2




(c)测试车辆3



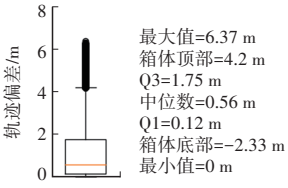

(d)测试车辆4



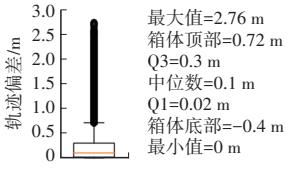

(e)测试车辆5　(f)测试车辆6

图11　Unity测试箱式图

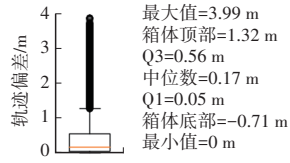
图12所示为Unreal Engine平台测试的箱式图。观察各被测参与者的极值数据，可以发现车辆间的轨迹偏差极值差异显著，同平台的测试参与者间偏差极值最大差异可达11.07 m，表明偏差受参与者不同行为的影响较大。进一步分析各测试车辆的偏差四分位值，发现部分车辆的偏差箱型较长，反映了偏差数据的波动较大。结合前述评价指标，出现较大偏差波动的车辆通常表现出较大的平均轨迹偏差。这一现象表明，仿真过程中参与者状态的不稳定性与其表现出的不一致性偏差呈正相关。



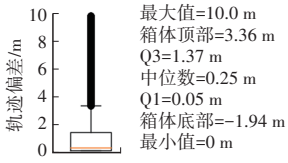

(a)测试车辆1



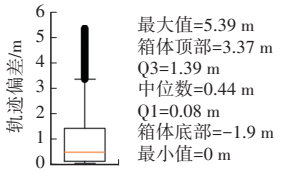

(b)测试车辆2



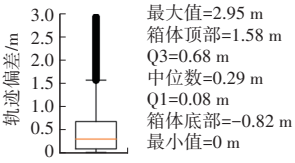

(a)测试车辆3




(b)测试车辆4




(e)测试车辆5




(f)测试车辆6

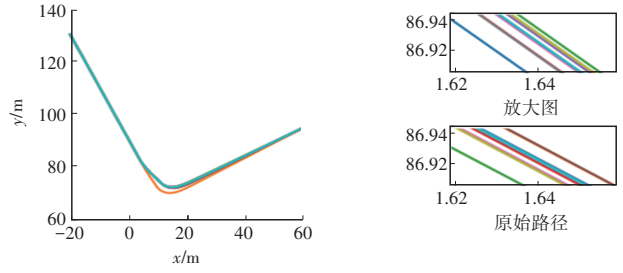
图12　Unreal Engine测试箱式图

结合测试平台中动态场景的设计可以发现，表现出不稳定性的被测车辆在行驶过程中比其他车辆更频繁地进行了前车避让和穿越交通信号灯行为，即进行了更多次的渲染状态更新。这种密集的参与者交互行为瞬间加重了游戏引擎的运算和渲染负担。结合前文的非一致性原因分析，可以推测正是这些密集的交互行为导致了渲染帧的超时，最终造成被测车辆在虚拟世界中的渲染位置和逻辑位置产生差异。

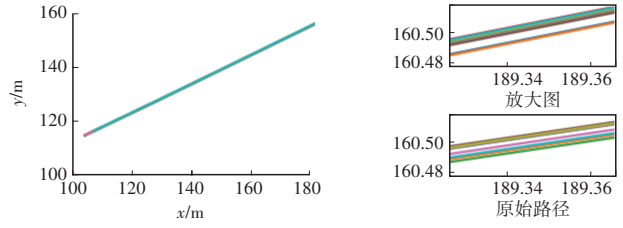
## 4.2 一致性优化

### 4.2.1 进程调度优化

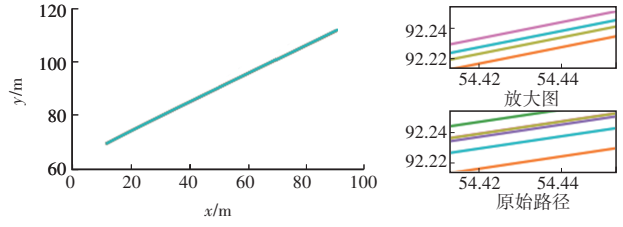
通过为仿真测试进程设置更高的优先级，理论上可以提升中央处理器（Central Processing Unit，CPU）的利用率，分配更多资源给仿真进程，以提高渲染性能并减小非一致性偏差。本文在每轮测试开始时自动为 Unity 和 Unreal Engine 程序设置最高优先级。优化后的测试结果显示，仿真程序的非一致性偏差显著降低，特别是在 Unreal Engine 上，平均轨迹偏差减小了 0.23 m，如图 13 所示。



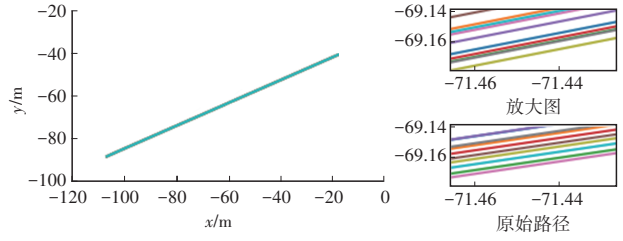

（a）测试车辆 1



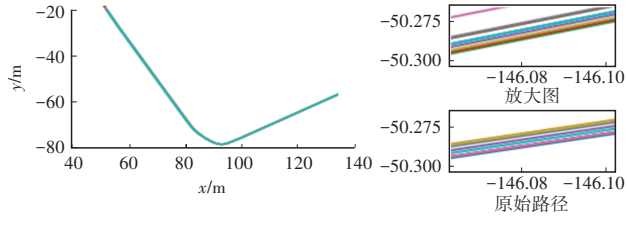

（b）测试车辆 2



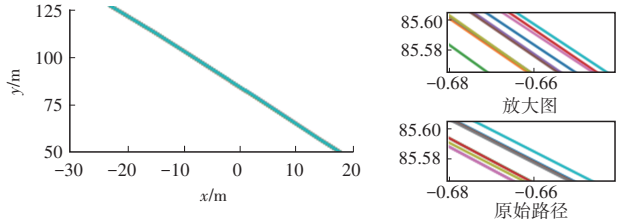

（c）测试车辆 3




（d）测试车辆 4




（e）测试车辆 5

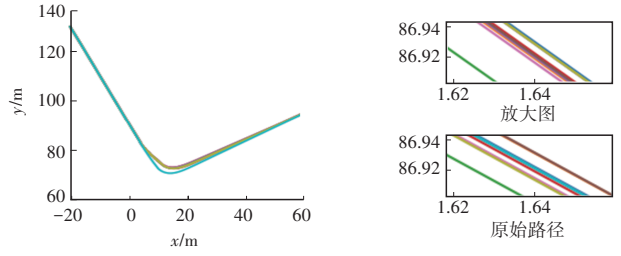



（f）测试车辆 6

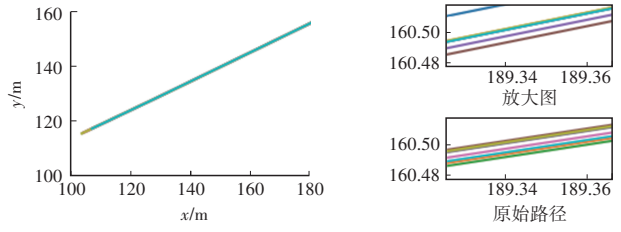
图 13 Unreal Engine 进程调度优化轨迹

### 4.2.2 电源模式优化

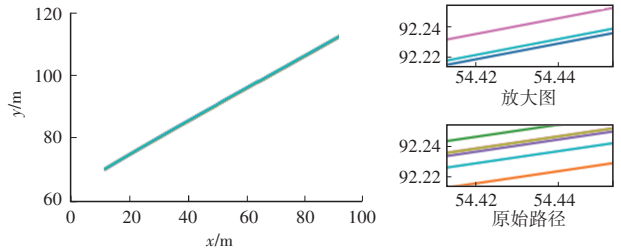
在电源模式优化中，视窗（Windows）操作系统提供 4 种模式，其中卓越性能模式通过提升 CPU 速度优化系统性能。开启该模式后，仿真软件的性能一致性得到改善，Unreal Engine 的平均轨迹偏差降低了 0.25 m，如图 14 所示。



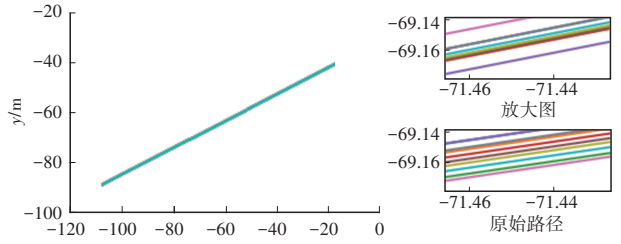

（a）测试车辆 1



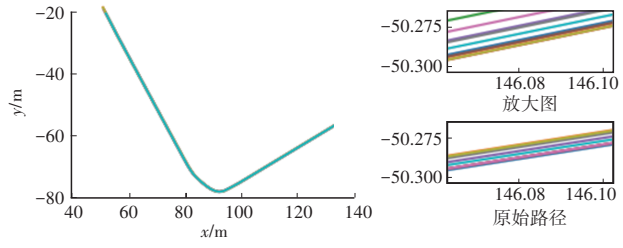

（b）测试车辆 2



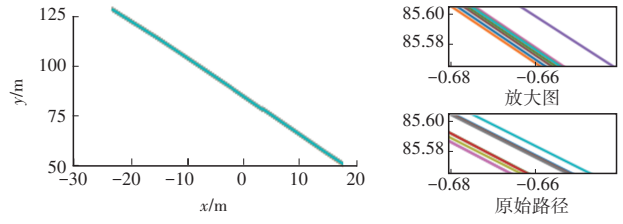

（c）测试车辆 3




（d）测试车辆 4




(e)测试车辆5

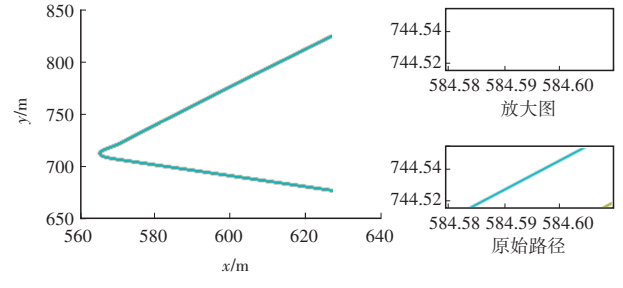



(f)测试车辆6

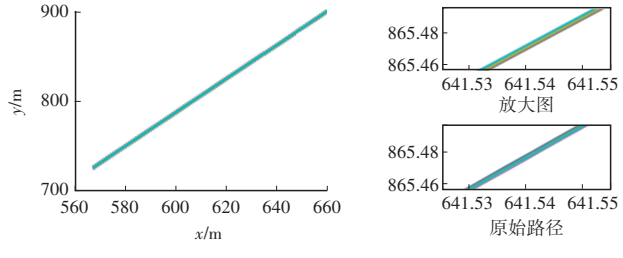
图14　Unreal Engine电源模式优化轨迹

### 4.2.3　图形处理器驱动程序优化

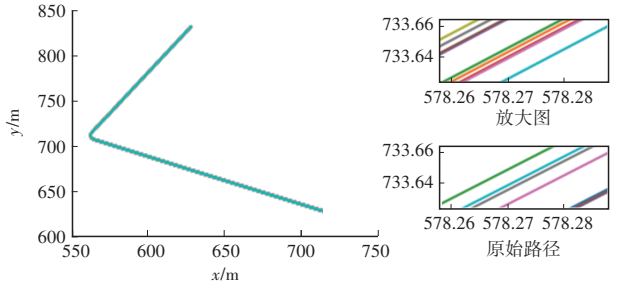
通过调整NVIDIA驱动程序设置提升图形处理器(Graphics Processing Unit，GPU)性能，减小测试中的渲染误差。本文对NVIDIA驱动程序内的图像渲染选项进行了以下优化性的调整：在图像设置中将优先选择项从“质量”调整为“性能”；在3D设置中开启“CUDA-GPUs”和“OPenGL渲染GPU”并选择平台所用的独立显卡，关闭“DSR-因数”“各向异性过滤”“平滑处理”“垂直同步”和“三重缓冲”选项；在“电源管理模式”中选择最高性能优先，开启“着色缓存器”和“纹理过滤”选项。优化后的测试结果表明，该方法有效降低了Unity和Unreal Engine的轨迹偏差，特别是在Unity平台上，平均轨迹偏差减小了0.15 m，如图15所示。



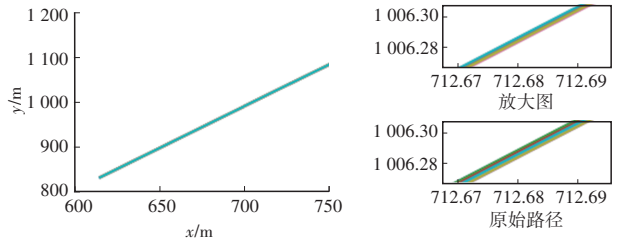

(a)测试车辆1



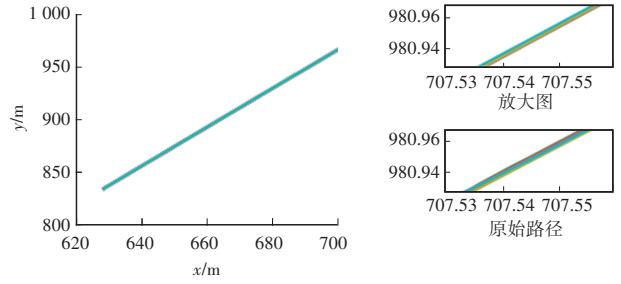

(b)测试车辆2



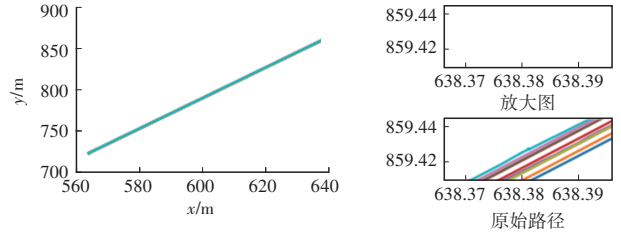

(c)测试车辆3




(d)测试车辆4




(e)测试车辆5

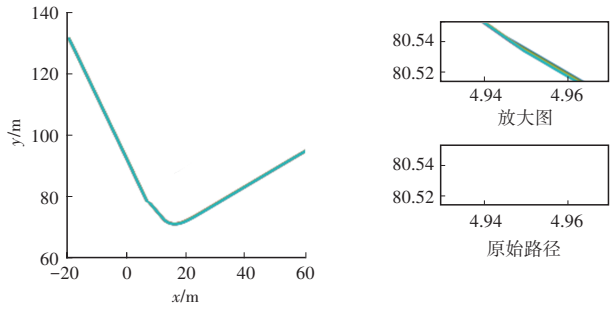



(f)测试车辆6

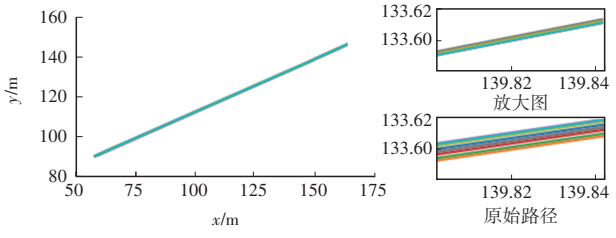
图15　Unity图形处理器驱动优化轨迹

### 4.2.4　硬件优化

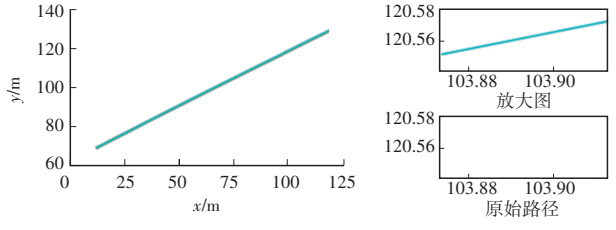
通过更换更高性能的CPU和GPU显著提高游戏引擎的渲染性能。本文首次试验所使用的测试平台的配置为i7-8700处理器搭配GTX 730显卡。对平台的CPU和GPU配置进行优化，在系统环境不变的条件下将硬件配置更换为i9-10900X处理器搭配Quadro RTX 6000显卡，二者在硬件性能上均明显优于首次试验中所使用的硬件。测试结果显示，Unreal Engine的平均轨迹偏差减小了0.58 m，如图16所示。然而，硬件优化成本高，效果可能不确定，特别是Unity平台上轨迹优化效果不明显，如图17所示。在硬件条件有限的情况下，前述的3种优化方法更易实施且具备参考价值。



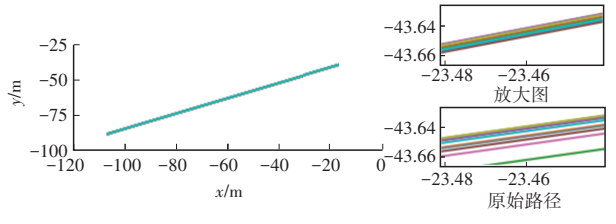

(a)测试车辆1



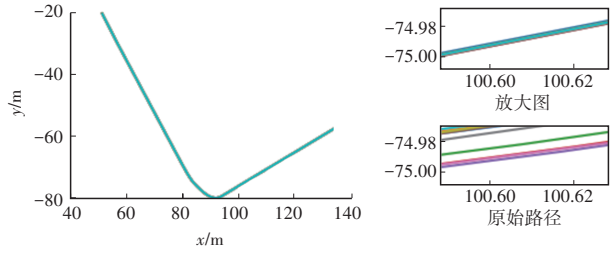

(b)测试车辆2



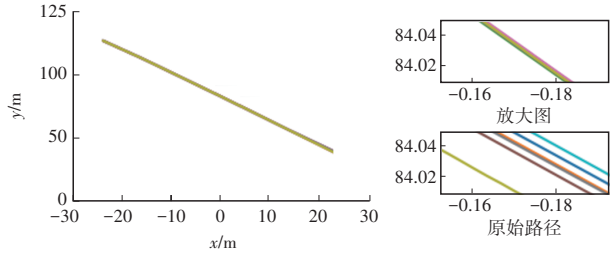

(c)测试车辆3




(d)测试车辆4

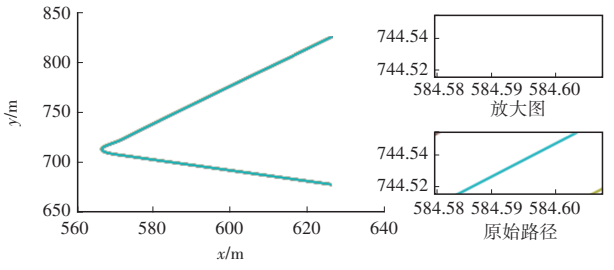



(e)测试车辆5

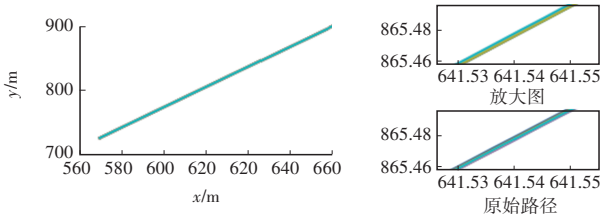



(f)测试车辆6

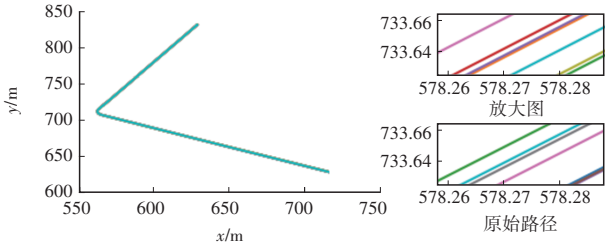
图16　Unreal Engine硬件优化轨迹



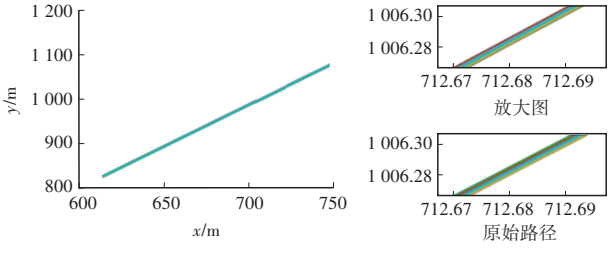

(a)测试车辆1



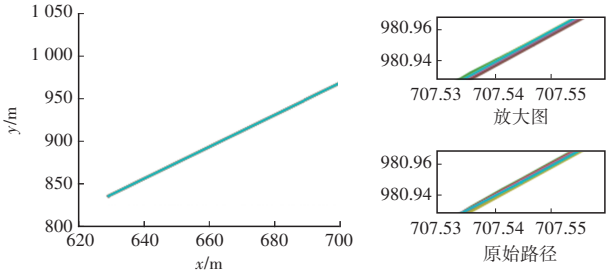

(b)测试车辆2



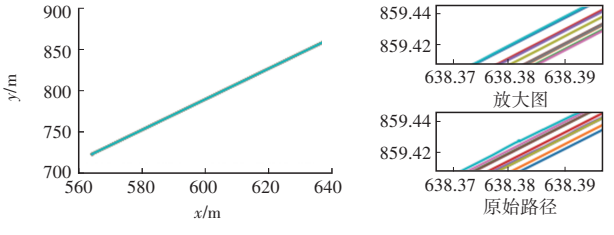

(c)测试车辆3




(d)测试车辆4




(e)测试车辆5

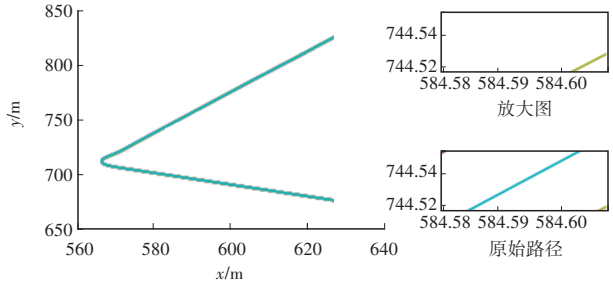



(f)测试车辆6

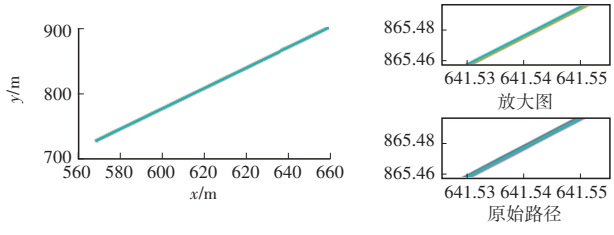
图17　Unity硬件优化轨迹

4.2.5　浮点数优化

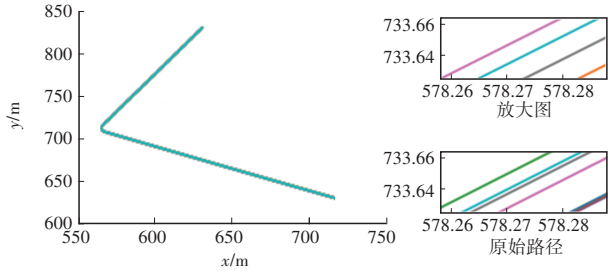
通过精确调整计算精度和改进算法，减小仿真平台中因计算误差导致的偏差。本文具体优化策略为放弃所有模拟环境属性值的小数部分，并保持每个属性值的固定精度。例如，计算前将“40.123 40”转换为“4 012 340”。经算法计算后，再将数字恢复到原始精度，用于可视化和分析。同时，在本文试验的所有算法中尽量避免浮点数加入运算的可能性。优化后测试结果表明，该方法有效降低了 Unity 和 Unreal Engine 平台上计算误差对仿真效果一致性的影响，其对于 Unity 平台的优化效果如图 18 所示，整体平均轨迹偏差下降了 0.03 m。



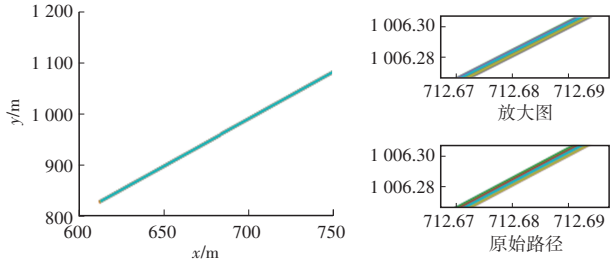

(a)测试车辆 1



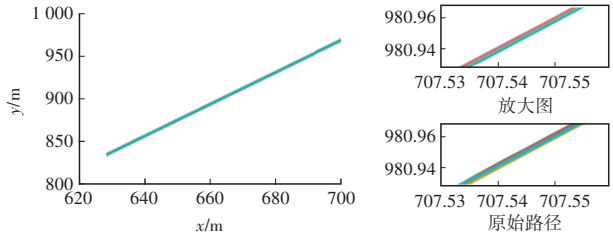

(b)测试车辆 2



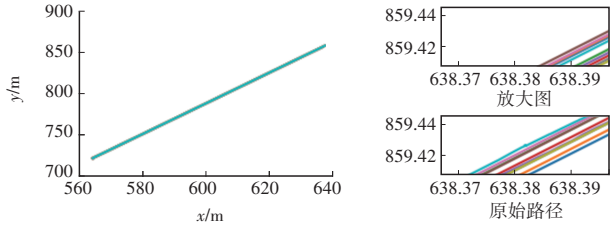

(c)测试车辆 3




(d)测试车辆 4




(e)测试车辆 5




(f)测试车辆 6

图 18　Unity 浮点数优化轨迹

综上，本文针对仿真软件性能的非一致性表现设计了 5 种优化方法，均有效减小了仿真过程中表现出的轨迹偏差，提升了一致性。

## 5　结束语

本文围绕自动驾驶仿真测试软件性能一致性进行了研究。测试结果表明，仿真测试软件在多次测试中存在性能非一致性偏差，证明了在仿真过程中进行一致性验证的必要性。

针对测试中发现的非一致性问题，本文提出了 5 种优化方案，即进程调度优化、电源模式优化、图形处理器驱动程序优化、硬件优化和浮点数优化，减小了计算误差。验证结果显示，这些优化方案均可有效减小仿真偏差，显著提升仿真测试软件的性能一致性。

尽管本文在自动驾驶仿真测试软件性能一致性研究方面取得了一定成果，但仍有诸多问题亟待进一步探讨：本文对性能一致性的定义集中在渲染更新环节，未来研究可将一致性概念扩展至自动驾驶仿真测试的更多环节，进一步挖掘仿真过程中非一致性现象的多样表现形式。当前试验场景的仿真度有限，未来研究应基于现实数据构建更为真实的场景，从而提升试验的真实性和可靠性。本文提出的优化方法虽已证明有效，但仍不足以涵盖自动驾驶仿真测试中可能出现的所有非一致性问题，未来研究应进一步扩展和总结更多优化方法，并逐步形成完善的性能一致性优化体系。

## 参 考 文 献


[1] HARPER C, CHANCE G, GHOBRIAL A, et al. Safety Validation of Autonomous Vehicles Using Assertion Checking[EB/OL]. (2022-05-13) [2024-08-28]. https://arxiv.org/abs/2111.04611.

[2] 石竖, 卓斌. 自动驾驶汽车的仿真[J]. 汽车工程, 2000(2): 97-99+80.
SHI J, ZHUO B. Simulation of Automatic Driving Vehicle [J]. Automotive Engineering, 2000(2): 97-99+80.

[3] 周干, 张嵩, 罗悦齐. 自动驾驶汽车仿真测试与评价方法进展[J]. 汽车文摘, 2019(4): 48-51.
ZHOU G, ZHANG S, LUO Y Q. Progress in Virtual Test and Evaluation of Autonomous Vehicle[J]. Automotive Digest, 2019(4): 48-51.

[4] 冯洋, 夏志龙, 郭安, 等. 自动驾驶软件测试技术研究综述[J]. 中国图象图形学报, 2021, 26(1): 13-27.
FENG Y, XIA Z L, GUO A, et al. Survey of Testing Techniques of Autonomous Driving Software[J]. Journal of Image and Graphics, 2021, 26(1): 13-27.

[5] WANG Z R, KIM B G, KOBAYASHI H, et al. Agent-Based Modeling and Simulation of Connected and Automated Vehicles Using Game Engine: A Cooperative on-Ramp Merging Study[EB/OL]. (2018-10-23)[2024-08-28]. https://arxiv.org/abs/1810.09952.

[6] CHANCE G, GHOBRIAL A, MCAREAVEY K, et al. On Determinism of Game Engines Used for Simulation-Based Autonomous Vehicle Verification[J]. IEEE Transactions on Intelligent Transportation Systems, 23(11): 20538-20552.

[7] 闻龙. 面向智能网联汽车的高性能计算仿真平台[D]. 成都: 电子科技大学, 2020.
WEN L. High Performance Computing Simulation Platform for Intelligent Connected Vehicles[D]. Chengdu: University of Electronic Science and Technology of China, 2020.

[8] 赵祥模国家重点研发计划(2021YFB2501200)团队. 自动驾驶测试与评价技术研究进展[J]. 交通运输工程学报, 2023, 23(6): 10-77.
Zhao Xiangmo's Team Supported by the National Key Research and Development Program of China (2021YFB2501200). Research Progress in Testing and Evaluation Technologies for Autonomous Driving[J]. Journal of Traffic and Transportation Engineering, 2023, 23(6): 10-77.

[9] GREGORY J. Game Engine Architecture[M]. 2nd Ed. Boca Raton, Florida, USA: CRC Press, 2017.

[10] 秦风. 无人车硬件在环的虚拟测试系统研究[D]. 西安: 长安大学, 2021.
QIN F. Research on Hardware-in-the-Loop Virtual Test System for Unmanned Vehicles[D]. Xi'an: Chang'an University, 2021.

[11] 邓伟文, 李江坤, 任秉韬, 等. 面向自动驾驶的仿真场景自动生成方法综述[J]. 中国公路学报, 2022, 35(1): 316-333.
DENG W W, LI J K, REN B T, et al. A Survey on Automatic Simulation Scenario Generation Methods for Autonomous Driving[J]. China Journal of Highway and Transport, 2022, 35(1): 316-333.


（责任编辑 斛 畔）

修改稿收到日期为2024年8月28日。