软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2023,34(9):3981−4002 [doi: 10.13328/j.cnki.jos.006877]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

嵌入路网图模型的自动驾驶场景描述语言*

龚 磊[1], 孙新雨[2], 张 昱[1,2], 张燕咏[1], 吉建民[1], 华 蓓[1]

[1](中国科学技术大学 计算机科学与技术学院, 安徽 合肥 230027)

[2](中国科学技术大学 先进技术研究院, 安徽 合肥 230094)

通信作者: 张昱, E-mail: yuzhang@ustc.edu.cn

摘 要: 深度学习的快速发展带动着自动驾驶技术的迅速进步. 深度学习感知模型在识别准确率逐步提升的同时, 也存在鲁棒性和可靠性不足等隐患, 需要在大量场景下进行充分测试以确保达到可接受的安全标准. 基于场景的仿真测试是自动驾驶技术的核心和关键, 如何描述和生成多样化仿真测试场景是需要解决的关键问题之一. 场景描述语言能够描述自动驾驶场景并在虚拟环境中实例化场景获取仿真数据, 但现有的场景描述语言大都缺少对于场景道路结构的高层抽象和描述. 提出路网属性图来表示路网中抽象出的实体及他们的关系, 并设计能简洁描述场景路网结构的语言 SceneRoad. SceneRoad 可以基于描述的场景道路结构特征构建路网特征查询图. 这样, 在路网中搜索符合描述的场景道路特征的问题被抽象为路网图上的子图匹配问题, 该问题可用 VF2 算法求解. 进一步地, 将 SceneRoad 作为扩展集成到 Scenic 场景描述语言中. 使用拓展后的语言随机生成大量多样的静态场景并构建仿真数据集. 仿真数据集的统计信息表明生成的场景具有丰富的场景多样性. 不同感知模型在真实和仿真数据集上的训练测试结果表明, 模型在两个数据集上的表现呈正相关, 意味着模型在仿真数据集上的评估结果与真实情况下的表现具有一致性, 这对于感知模型的评估以及提升模型鲁棒性和安全性相关研究有重要意义.

关键词: 自动驾驶; 深度学习; 仿真测试; 场景描述语言; 子图匹配

中图法分类号: TP311

中文引用格式: 龚磊, 孙新雨, 张昱, 张燕咏, 吉建民, 华蓓. 嵌入路网图模型的自动驾驶场景描述语言. 软件学报, 2023, 34(9): 3981–4002. http://www.jos.org.cn/1000-9825/6877.htm

英文引用格式: Gong L, Sun XY, Zhang Y, Zhang YY, Ji JM, Hua B. Scenario Description Language of Autonomous Driving Embedded with Road Network Graph Model. Ruan Jian Xue Bao/Journal of Software, 2023, 34(9): 3981–4002 (in Chinese). http://www.jos.org.cn/1000-9825/6877.htm

Scenario Description Language of Autonomous Driving Embedded with Road Network Graph Model

GONG Lei[1], SUN Xin-Yu[2], ZHANG Yu[1,2], ZHANG Yan-Yong[1], JI Jian-Min[1], HUA Bei[1]

[1](School of Computer Science and Technology, University of Science and Technology of China, Hefei 230027, China)

[2](Institute of Advanced Technology, University of Science and Technology of China, Hefei 230094, China)

Abstract: The development of deep learning technology has driven the rapid progress of autonomous driving. While the accuracy of perception models based on deep learning is gradually improved, hidden dangers related to robustness and reliability still exist. Therefore, tests should be conducted thoroughly under various scenes to ensure acceptable security levels. Scene-based simulation testing is crucial in

* 基金项目: 科技创新 2030 —“新一代人工智能”重大项目 (2018AAA0100500); 国家自然科学基金 (62272434); 安徽省重点研究与开发计划标准化专项 (202104h04020039)
本文由“AI 软件系统工程化技术与规范”专题特约编辑张贺教授、夏鑫博士、蒋振鸣副教授、祝立明教授和李宣东教授推荐.
收稿时间: 2022-09-05; 修改时间: 2022-10-13; 采用时间: 2022-12-14; jos 在线出版时间: 2023-01-13
CNKI 网络首发时间: 2023-07-05

autonomous driving technology. One key challenge is to describe and generate diversified simulation testing scenes. Scenario description languages can describe autonomous driving scenes and instantiate the scenes in virtual environments to obtain simulation data. However, most existing scenario description languages cannot provide high-level abstractions and descriptions of the road structure of the scene. This study presents a road network property graph to represent the abstracted entities and their relations within a road network. It also introduces SceneRoad, a language specifically designed to provide concise and expressive descriptions of the road structure in a scene. SceneRoad can build a road network feature query graph based on the described road structure features of a scene. Thus, the problem of searching the road structures in the road network is abstracted as a subgraph matching one on the property graph, which can be solved by the VF2 algorithm. Additionally, SceneRoad is incorporated as an extension into the Scenic scenario description language. With this extended language, a diverse set of static scenes are employed to build a simulation dataset. Statistical analysis of the dataset indicates the wide variety of scenes that have been generated. The results of training and testing various perception models on both real and simulated datasets show that the model's performance on the two datasets is positively correlated, which shows that the model's evaluation on the simulated dataset aligns with its performance in real scenes. This is significant for perception model evaluation and research on improving the model's robustness and safety.

Key words: autonomous driving; deep learning; simulation testing; scenario description language; subgraph matching

自动驾驶系统 (automated driving system, ADS) 是包含感知、定位、决策和控制的复杂数字系统, 用以应对瞬息万变的道路交通场景. 随着深度学习和人工智能领域的快速发展, 自动驾驶技术取得显著进步, 并陆续走向落地应用阶段. 然而, ADS 在算法、传感器、高精地图、避免攻击、数据安全等多个层面都还有不少问题需要突破, 其安全性是自动驾驶汽车规模化面临的重要挑战之一.

针对智能驾驶道路车辆安全, 国际上已提出了一些有影响力的国际标准, 如 ISO 26262:2011/2018 功能安全标准[1]、ISO/PAS 21448:2019 预期功能安全标准[2], 以及 ISO/SAE 21434:2021 网络信息安全标准[3]. 其中, ISO/PAS 21448 定义了预期功能安全 (SOTIF), 它重点关注车辆的行为安全, 避免因自身设计不足和性能局限导致不合理的风险. 它将车辆运行场景划分成 4 个区域: 已知安全场景 (A1)、已知不安全场景 (A2)、未知不安全场景 (A3) 和未知安全场景 (A4). SOTIF 的目标是最大可能减小 A2/A3. 对 A2, 需要结合测试不断完善算法; 而 A3 则相对棘手, 需要基于用例测试、随机输入测试等发现系统设计不足. A3 是 ADS 安全风险的主要来源, 其无法定义需求且难以量化评价, 是全球自动驾驶安全开发领域的痛点.

正因如此, 基于场景的测试方法在 ADS 的研究和工程中受到极大关注. 然而, 传统的基于路测里程的测试非常昂贵, 且具有一定危险性, 这使得人们对虚拟仿真测试验证产生浓厚兴趣. 仿真测试通过构造安全关键场景对 ADS 及算法进行安全评测, 能有效缩短技术和产品开发周期, 降低研发成本. 许多自动驾驶头部企业已经在虚拟环境中对 ADS 进行了海量的测试. 例如, Google Waymo 自成立至今已经在现实世界和模拟环境中行驶的里程超过 320 亿千米[4].

仿真测试研究涉及仿真器开发以及场景描述与生成等关键技术. 在仿真器方面, 代表性开源自动驾驶仿真器有 CARLA[5]、AirSim[6]等. CARLA 模拟器基于虚幻引擎开发, 能够模拟出逼真的自动驾驶场景. 与其他仿真器相比, CARLA 拥有更丰富的传感器种类, 更多样的地图、车辆和行人模型资源, 以及更全面的 API 控制接口, 能够实现细粒度的场景创建和对实体的控制和交互, 在工业和研究领域都得到非常广泛的应用. 在场景描述与生成方面, 代表性的有 Scenic 场景描述语言[7]以及 ASAM 协会发布的 OpenX 系列自动驾驶仿真规范, 如采用 XML 表示的 OpenDRIVE 路网描述[8]、OpenSCENARIO 动态场景描述[9]. Ulbrich 等人[10]将场景分为静态场景 (Scene) 和动态场景 (Scenario). Scene 表示当前环境、实体及其间关系的快照, 而 Scenario 则表示由初始 Scene 开始的一系列随时空发展变化的 Scenes. Scenic 是为描述自动驾驶中的静态场景和动态场景等设计的领域特定语言 (DSL)[11], 允许为静态场景特征定义分布并能描述场景中智能体随时间变化的动态行为. 它实现了与仿真器的充分解耦, 其编译器实现了该 DSL 到宿主语言 (如 Python) 的翻译, 再通过仿真器控制接口将所描述的场景进行实例化, 从而最终生成虚拟环境中的具体场景实例. OpenSCENARIO 系列仅提供有关仿真场景的描述规范, 本身并不提供场景生成器.

目前仿真场景测试主要关注 ADS 规划控制模块的测试, 而对感知算法测试相对较少[12]. 仿真器能提供带标签数据的各种仿真传感器数据, 可为基于学习的感知模型提供海量且多样化的训练和测试数据. 根据 Scholtes 等

人[13]定义的场景 6 层描述模型 (6LM), 完整的场景描述包含路网结构和交通标志、路边结构、前二者的临时变化、动态物体、环境条件和数字信息 6 个层次. 路网的抽象描述以及对场景中如车辆、行人等动态物体的位置、朝向以及关系的表达是场景描述的关键. 而像 Scenic 等目前多数场景描述语言无法对 6LM 的各个层次都进行详细描述, 特别是缺少对路网的抽象和描述.

本文着眼于对场景道路层次的描述, 提出并设计了一个描述场景道路的 DSL, 称为 SceneRoad, 并将其集成到 Scenic 中作为拓展; 进一步地, 在扩展后的 Scenic 上优化场景采样过程, 提高场景生成的成功概率; 使用拓展 Scenic 生成场景并从两个方面对生成的场景数据进行评估, 包括评估生成的场景的多样性和仿真数据集质量, 以及对比 ADS 的感知模型在生成的场景数据以及真实场景数据的表现.

本文的主要贡献如下.

(1) 设计场景道路描述语言 SceneRoad, 它基于一个底层的以属性图表示的路网图. SceneRoad 以简洁可读的方式描述场景中的路网实体以及实体之间的关系, 进而能描述具有复杂道路结构的场景.

(2) SceneRoad 将在路网中搜索描述的场景道路特征的问题抽象为路网图上的子图匹配问题并用 VF2 算法求解. 测试结果表明, SceneRoad 在 CARLA 仿真器不同地图中获取候选匹配所用的平均时间均为毫秒级.

(3) 将 SceneRoad 作为拓展集成到 Scenic 场景描述语言, 并实现了在 CARLA 模拟器中生成场景以及获取仿真数据. 通过简单的场景描述生成大量随机场景, 构建一个包含 4000 场景 (可扩展) 的仿真示例数据集. 通过计算和比较仿真数据集和真实数据集中车辆和行人等相关统计量的分布以及分布的熵, 证明了生成场景的多样性.

(4) 对 MMDetection 框架[14]中 10 个视觉感知模型在真实和仿真数据集上进行训练评估. 实验表明, 被测模型在两个数据集上的精度表现呈正相关, 这表明生成的仿真数据集与对应的真实数据集具有相似性, 模型在仿真数据集上的表现对真实情况具有重要的参考价值.

1 相关工作与动机

本节主要介绍自动驾驶场景不同抽象层次的定义以及现有主流的自动驾驶场景描述语言.

1.1 场景的不同抽象层次

场景按照抽象层次不同可分为具体场景、逻辑场景、功能场景[15]等. 如图 1 所示, 功能场景是以非形式的方式对场景的整体描述. 逻辑场景是在状态空间级别上的场景空间表示, 其中包含状态空间中的参数范围. 参数范围可以用概率分布来指定. 具体场景是场景的参数化表示, 每个具体场景都是某个逻辑场景的实例, 每个参数都有一个具体的值.

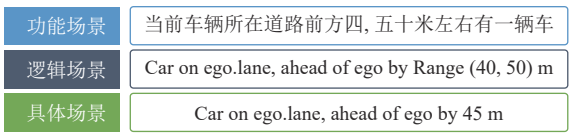


图 1 不同抽象层次的场景示例

1.2 场景描述语言

ASAM 自动化和测量系统标准化协会是世界上被广泛认可的汽车工业国际标准协会之一, 其开发的仿真相关的一系列标准 OpenX (OpenDRIVE, OpenSCENARIO 等) 已经被全球大量工具商、研发团队以及整车厂所使用. OpenSCENARIO 使用 XML 语言描述场景, 而 OpenSCENARIO2 则被设计成 DSL. 使用 DSL 的优点是它能够很好地适配特定领域的特征, 相较于通用程序语言 (general program language) 更关注于领域问题本身, 更具可读性; 并且 DSL 描述的场景与仿真器的内部实现无关, 这使得描述的场景可以在不同的虚拟环境中通用. OpenSCENARIO2 包含描述场景的 DSL 的完整定义以及定义场景实体、动作的域模型. 域模型的实体及其子类用它们的定义、状态、属性和与其他类的关系来描述. 这些实体包括物理对象、环境、行动、物理类型、道路抽

象、行为模型等. 目前该标准已发布公开发行候选版 (PRC), 相应的编译器目前也在不断开发中, 如博世 (Bosch) 的 YASE[16].

Scenic 同样是场景领域特定语言, 可描述自动驾驶场景. Scenic 除了同 OpenSCENARIO2 一样定义了语言描述框架外, 还集成了 CARLA 等多种仿真器的控制接口以及对描述语言的编译部分, 可以将场景描述转化为相应的可执行代码. 相较于 OpenSCENARIO2, 如图 2 所示, Scenic 实现了从场景描述到编译再到仿真器上实例化场景并得到仿真数据的完整过程. Scenic 的语法语义具有丰富的静态场景表达能力, 能够从位置、朝向、距离等多个角度实现对场景中物体空间分布的描述. 基于此, 本文选择 Scenic 作为描述静态场景的工具, 利用 CARLA 仿真模拟器实例化场景并获取场景感知数据.

图 2 Scenic 生成场景并获取仿真数据过程

1.3 Scenic 场景描述语言的主要特色

Scenic 的核心是概率编程语言 (PPL)[17], 它定义了场景特征的分布并通过对分布采样生成具体场景. Scenic 将对象构造过程分解成一个个语法独立的说明符 (*specifier*), 这些说明符包括 at, on, left/right of, ahead of 等位置相关的, 以及 facing, facing toward/away 等朝向相关的, 还有定义对象属性的 with 说明符, 这些说明符可以在创建对象时进行灵活的组合.

图 3 展示了一个 Scenic 场景描述语言示例, 其中 ego 表示自身车辆, 每个场景的描述和构建都围绕 ego 展开. 默认情况下, 生成的车辆将会被放置在地图中可行驶区域内的随机道路上. Scenic 内置了包括 Range, Uniform, Normal 等多种不同的分布, 程序的每次执行都会从分布中采样得到一个具体的值, 从而实现将逻辑场景实例化为不同的具体场景. 图 4 展示了 Scenic 中定义的一些不同的说明符以及相关操作符 (*operators*) 的示意图. 借助这些 *specifiers* 和 *operators*, Scenic 能够创建出多样的静态场景. Require 约束用于对生成的场景进行选择, 包括硬约束和软约束. Scenic 根据用户给出的约束对采样后生成的具体场景检查其是否符合特定要求. Scenic 还内置了一些基本约束, 用于判断场景是否符合最基本的物理规则等. 对于不符合约束的场景, Scenic 会将其视作失败场景并丢弃, 然后对场景进行重新采样.

```
1   ego = Car
2   dis = Range(2,3)
3   car = Car left of ego by dis,
4             facing roadDirection,
5             with color CarColor(1, 0, 0)
6   require distance to intersection > 50
```

图 3 Scenic 场景描述语言

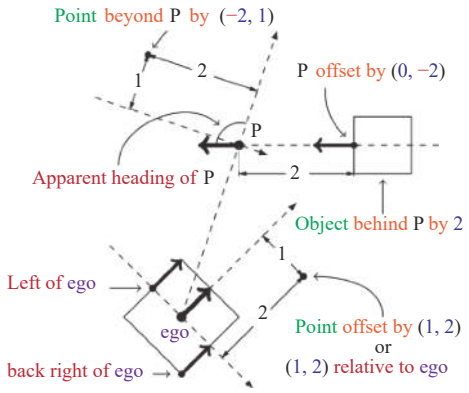


图 4 Scenic 中定义的部分 *specifiers* 和 *operators*

尽管 Scenic 实现了对静态场景中车辆行人等动态物体的空间分布描述, 然而从场景 6 层描述模型来看, 还缺少一些对其他场景层次的抽象描述, 特别是对于道路的描述. 在 Scenic 中, 场景道路采用解析过后的 OpenDRIVE 格式路网数据. Scenic 定义了 filter 对从路网中提取的道路进行过滤筛选. 这种方式使得描述语言依赖于特定的路

网解析格式, 不够简洁也缺少规范性和通用性. 对于描述多个道路实体以及不同道路实体相互之间的约束和拓扑关系十分不便. 基于此, 本文定义了一种规范的属性图模型的路网数据格式, 可以从现有路网数据中提取信息并构建路网属性图 (简称路网图). 以路网图为基础, 设计了一种描述道路的 DSL, 实现对场景道路的描述.

2 路网图模型与 SceneRoad 语言设计

2.1 路网图模型

OpenDRIVE 描述了驾驶仿真应用所需的静态路网, 如道路、车道、交叉路口等内容, 它以 XML 文件存储道路的几何形状以及影响路网逻辑的相关特征, 如车道和标志等. 这些存储路网数据的 XML 文件格式复杂, 难以直接使用. 本文提出了一种通用的以属性图为基础的路网数据格式, 相较于 OpenDRIVE 等路网数据格式, 图模型可以清晰地展示路网的拓扑结构, 更易理解和分析处理; 同时图模型具有很高的灵活性, 可以通过添加、删除和修改节点和边来表示路网的变化. 这样就可以方便地更新路网的信息, 使得路网模型能够适应不断变化的路网环境; 以图结构作为中间表示的设计能够支持对不同格式地图的抽象, 同时也方便 DSL 的扩展和修改. 图 5 展示了路网描述机制的整体设计思路, 首先对诸如 OpenDRIVE 等的路网数据进行解析, 然后为之建立描述路网中实体、属性及其之间关系的属性图, 再以路网图作为支撑设计场景道路描述 DSL.

定义 1. 属性图. 属性图是一个有向多重图, 图中的每个顶点和边都可包含多个属性键值对. 节点代表一个具有多种属性的实体对象, 边则代表两个不同实体之间存在的关系.

定义 2. 路网实体. 路网实体是基于从路网数据中提取的信息所构建的车道、道路、交叉路口等路网结构的抽象表示. 每个路网实体包括描述该实体的各种属性和值.

定义 3. 路网图. 路网图是基于原始路网数据抽取的路网实体和实体之间的关系所组成的抽象路网, 用属性图表示, 图中每个节点对应抽象路网中的一个路网实体, 各路网实体之间的关系对应为图中两个节点之间的有向边.

本文提取路网数据的各类信息并抽象出 4 类路网实体, 分别为车道 (lane)、同向车道组 (group)、道路 (road) 及交叉路口 (junction), 对应到路网图中的 4 种不同类型属性的节点. 道路实体参考了 OpenDRIVE 规范中的 <road> 元素, 道路一般由多个车道构成, 车道对应了<road>中的子元素<lane>, 每个车道各种车辆纵向排列. 车道组表示在道路中同一方向的所有车道构成的集合, 单向的道路有一个车道组, 双向的道路有两个车道组, 这对应于<road>元素中的子元素<left> 和<right>. 道路包含一个或多个车道组和车道. 交叉路口参考 OpenDRIVE 规范中的 <junction> 元素, 交叉路口是 3 条或更多道路相聚的地方, 其内部由连接不同道路的连接道路 (connecting road) 组成, 如图 6 所示.

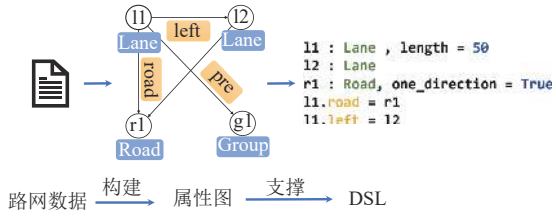


图 5 路网描述机制的整体设计

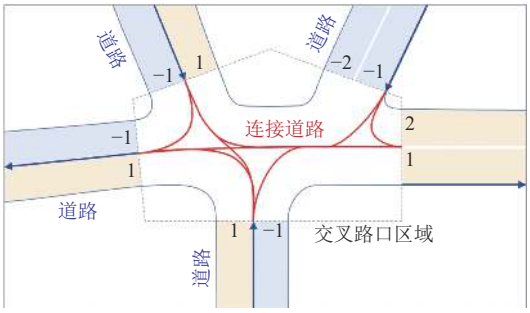


图 6 OpenDRIVE 交叉路口中的道路类型

● 路网图的节点属性和关系. 在路网图中, 每个节点有唯一的 id, 不同类型的节点分别有各自对应的属性. 例如, 道路类型节点的属性包括: 是否是单向道路、道路的长度、车道数量等; 交叉路口类型节点的属性包括: 连接道路数、是否为十字路口或 T 型路口等. 附录 A 列举了本文提取的 4 类路网实体的部分属性.

表 1 为本文中抽取的各种类型的路网实体之间的基础关系. 这些关系可以分成两大类: 一类是归属关系, 如车

道属于某个车道组, 而车道组又构成了道路, 一部分道路则组成了交叉路口; 另一类是拓扑关系, 描述了这些实体如何构建起一个道路网络. 车道和车道组是有方向的实体, 它们都有起点、终点以及连接两点的中心线, 故这两种类型的节点都有前驱 (pre) 和后继 (succ) 这两种关系类型, 是将路网中实体连通起来的两类重要关系. 以前驱为例, 车道的直接前驱是驶入该车道的车道, 前驱车道的终点是该车道的起点, 前驱车道所在车道组、道路或者是交叉路口也视作该车道的前驱. 毗邻关系包括了车道之间的左右关系以及车道组之间的相对关系. 实体间的关系越多越复杂, DSL 就能够组合更多的约束条件描述越复杂的场景道路结构.

表 1　路网图中的关系类型

关系类型	描述	Source 类型	Target 类型
pre	驶入该路网节点的前驱节点	Lane Group	Lane | Group | Road | Junction Group | Road | Junction
succ	沿当前道路前进方向的下一个路网节点	Lane Group	Lane | Group | Road | Junction Group | Road | Junction
left	沿当前车道前进方向的左侧车道	Lane	Lane
right	沿当前道路前进方向的右侧车道	Lane	Lane
group	车道所属的车道组	Lane	Group
opposite	与当前车道组所属同一道路的相对方向的车道组	Group	Group
road	所属的道路	Lane Group	Road Road
junction	所属的交叉路口	Road Lane Group	Junction Junction Junction

● 基于路网图的道路搜索. 构建了路网图后, 用户描述其希望生成的场景的路网结构等同一个搜索的问题. 路网图可以被视作一个用于查询和检索的图数据库. 用户描述场景道路特征并在数据库中搜索相应特征的道路并返回结果. 若将描述的道路特征构建为与路网图一样的属性图作为查询图, 那么在路网中搜索满足特征的道路的问题就可以抽象为一个路网图的子图匹配问题.

定义 4. 子图同构. 给定图 $G=(V,E)$ 和图 $H=(V_H,E_H)$. 子图同构指图 G 中存在子图 $G'=(V',E')$ 使得 G' 与 H 同构, 其中节点数 $|V_H|\leqslant|V'|$. 同构指存在双射 f 使得 $(u,v)\in E_H$ 当且仅当 $(f(u),f(v))\in E'$, 其中 u, v 为 H 中的节点, $f(u),f(v)$ 为 G 中的节点.

定义 5. 路网特征查询图. 路网特征查询图是基于用户所描述的场景道路特征构建的属性图, 包含用户所描述的路网实体, 以及实体之间的关系.

子图同构与子图匹配在很多情况具有类似含义. 在图 $G=(V,E)$ 中, V 为顶点的集合, E 为边的集合. 子图 $G'=(V',E')$ 被称作单态 (monomorphism), 如果有 V' 为 V 的子集, 且 E'为 E 中与 V' 诱导的所有边的集合的一个子集. 在对问题进行抽象之后, 可以使用 VF2[18]算法获取路网图中所有符合描述特征的子图.

定义 6. 路网子图匹配. 本文中, 图 G 对应为从原始路网数据中抽象出的路网图, 而图 H 则是根据用户描述的道路特征构建的路网特征查询图. 本文所述的子图匹配是在图 G 中寻找与图 H 匹配的所有单态 G'.

由于路网图和路网特征查询图都为基本属性图数据结构, 因此在子图匹配时可以直接使用原始 VF2 算法. VF2 算法是一种基于深度优先搜索和回溯 (backtracking search) 的子图匹配算法, 在算法匹配过程中, 状态 s 对应一个局部的匹配映射 $M(s)$, $M(s)$ 为双射 M (定义 4 中的 f) 的一个子集. 算法计算当前局部匹配状态 s 下所有候选匹配集合 $P(S)$ 中的匹配对 p 是否满足可行性规则 (feasibility rules), 对满足的 $p=(n,m)$, 其中 n, m 分别表示两个图中的节点, 计算后续状态 $s'=s\cup p$ 并递归的进行上述过程, 直到映射 $M(s)$ 覆盖查询图 H 中的所有节点. 本文仅需通过自定义可行性规则, 即可将 VF2 算法应用到道路匹配问题, 算法 1 为道路匹配的 VF2 算法的过程, 自定义可行性规则将在 SceneRoad 设计中进一步阐述.

算法 1. *Match*(*s*): VF2 for searching road.

输入: 状态 s, 自定义可行性规则 customFeasibilityRules; 对初始状态 s_0 有 $M(s_0)=\varnothing$;
输出: 路网图与路网特征查询图的匹配映射 M.

IF $M(s)$ 覆盖查询图 H 中的所有节点 **THEN**
  返回 $M(s)$
ELSE
  计算当前状态下候选匹配集合 $P(S)$
  FOR p in $P(S)$ **DO**
    IF customFeasibilityRules(p) is True **THEN**
      $s'=s \cup p$
      CALL *Match*(s')
    END IF
  END FOR
  保存当前数据
END IF

2.2 SceneRoad 描述语言的设计与实现

基于路网图模型, 本文围绕如何快速方便的构建路网特征查询图来设计场景道路描述 DSL, 即 SceneRoad, 其核心在于如何描述一个路网实体以及如何描述两个实体的关系. 我们将 SceneRoad 集成到 Scenic 中, 并将描述场景的过程分为两个部分. 首先, 使用 SceneRoad 语言描述场景的道路结构, 编译器对该描述进行解析并在已构建的路网图中进行搜索; 获取到匹配的道路并返回后, 使用 Scenic 原有语法在相应的道路上放置车辆, 生成描述场景. 本节通过一个具体实例来展示 SceneRoad 的相关语法.

一个自然语言描述的场景示例如下: ego 位于一个双向 4 车道的道路靠近道路中心线的车道上, 其相对方向的外侧车道有一辆汽车 carA, ego 所处道路的前方为一个十字形交叉路口; 在交叉路口中, ego 所在道路的前方左转的道路上有一辆汽车 carB. 图 7 中左侧展示了该场景中的道路结构对应的 SceneRoad 描述, 右侧展示了所描述场景经过编译并在 CARLA 中实例化之后, 获取的 ego 前视图像和俯视示意图.

```
1  r1 : Road, is2Way = True
2  g1 : Group , laneNum = 2
3  g2 : Group , laneNum = 2
4  g1.road = r1
5  g2.road  = r1
6  g1.opposite = g2

7  ego_lane : Lane, index = 1
8  l2 : Lane, index = 2
9  ego_lane.group = g1
10 l2.group = g2

11 j1 : Junction, is4Way = True
12 l3 : Lane, turn = LEFT
13 l3.junction = j1
14 ego_lane.succ = l3
```

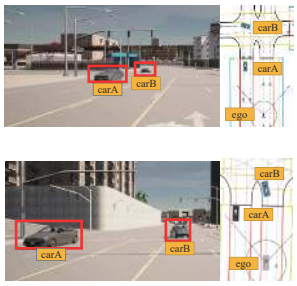


图 7　描述场景道路的 SceneRoad 程序片段示例

图 8 展示了集成 SceneRoad 的 Scenic 抽象语法, 其中的 statement 代表 Scenic 语言支持的语句 (见文献 [7] 中的表 5 等). 在 SceneRoad 中, 道路特征用若干个 graphStmt 描述, 每个 graphStmt 描述一种局部道路特征, 由 qgraph 关键词来引导, 表示在程序中将创建一个路网特征查询图, 该子图在初始情况下不包含任何节点和边. 实体声明 (entityClause) 描述了一个路网实体及其若干属性值, 编译器将据此创建一个节点并将其添加到路网特征查询图中. 实体的类型为路网图中的 4 种不同类型之一, 实体的属性值作为约束条件用于在子图匹配算法中寻找路

网图中符合描述的实体. 关系声明 (relationClause) 描述两个路网实体之间的关系, 编译器将据此在子图中创建连接两个节点的边. 关系类型以及所连接的实体类型的约束和表 1 的定义相同. 结束声明 (get) 表示场景道路描述结束并开始执行子图匹配算法, 将构建的路网特征查询图与路网图进行匹配, 返回所有符合条件的道路实体.

```
program: =(graphStmt|statement)*
graphStmt: =qgraph
            (entityClause)*
            (relationClause)*
            get result_id
entityClause: =id: entityType, (property=value)*
relationClause: =source_id.relationType=target_id
entityType: =Lane|Group|Road|Junction
relationType: =pre|succ|left|right|road|junction|group|opposite
source_id, target_id, result_id, id, property: =name
value: = number|bool|string|name
```

图 8 SceneRoad 道路描述 DSL 的语法和声明

回到图 7 中展示的场景道路描述, 该场景描述可分成 3 个部分. 第 1 部分描述了一个双向四车道的道路, r1, g1, g2 分别代表道路类型和车道组类型的实体. 道路实体的 is2Way 属性用于约束该道路的单双向, 而车道组实体的 laneNum 属性值规定该车道组包含的车道数量. 在定义了实体之后, 通过归属关系 road 和相对关系 opposite 将几个实体连接起来. 第 2 部分描述的是 ego 所在车道以及与 ego 在同一道路上的相对方向的车道. 车道实体的属性 index 定义了其在包含多个车道的车道组中的顺序位置, index 越小表明其越靠近道路中心线. 在定义了两个不同的车道实体后, 将它们分别与第 1 部分中定义的两个相反方向车道组通过归属关系 group 进行关联. 第 3 部分描述了交叉路口和交叉路口内的车道, 通过 succ 后继关系将该道路与 ego 所在的道路进行连接. 通过以上的描述, 程序构建了一个包含多个节点以及连接这些节点的多条边的路网特征查询图. 在构建路网特征查询图之后, 利用 NetWorkX[19]中实现的 VF2 算法进行与路网图的子图匹配. 使用该算法需要在子图匹配的过程中自定义可行性规则, 具体为节点的匹配以及边匹配的策略.

对于节点匹配, 若一个路网图中的节点其类型与路网特征查询图中的节点相同, 且该节点所对应的路网实体对象的属性值满足约束, 即认为两个节点匹配成功. 在 SceneRoad 中, 这些子图中描述的属性既可以是已经抽取并编码在对应类型的路网实体对象中的属性, 也可以代表一个 lambda 表达式或函数. 例如, 图 7 中描述道路实体 (road) 的属性“is2Way”既可以编码在实体对象中, 也可以对应一个如下所示的 lambda 表达式:

```
lambda_func = lambda road: len(road.groups) == 2
r1 : Road, is2Way = lamda_func.
```

若属性对应一个 lambda 表达式, 那么在节点匹配时, 算法会将待匹配的道路实体带入该表达式执行计算, 通过表达式计算的值判断是否满足该属性约束. 在已知节点匹配成功的情况下, 要求路网图中的边与路网特征查询图中对应的边满足关系类型一致以及所连接的节点类型一致, 即认为边匹配成功.

图 9 展示了上述拓展 SceneRoad 语言后的 Scenic 编译以及在路网图上查找与用户描述构建的路网特征查询图同构的子图的过程, 记为 SceneRoadQuery. 拓展后的场景描述同时包含 SceneRoad 语句 (图中的①) 以及 Scenic 语句, 该描述通过编译器进行词法分析、再统一翻译成对应的 Python 代码. 图中的②是①对应的 Python 代码, 其中为实体声明创建了节点 (node) 类对象并将其添加到路网特征查询图中, 而关系声明创建了连接 (link) 类对象并将其添加到查询图中. 生成的 Python 程序经解析编译成字节码, 然后再被解释执行. 执行期间会调用子图匹配算法 (图中的 B) 在路网图上搜索匹配路网特征查询图 (图中的 A) 的所有候选子图. 每个候选子图中的节点和边都与用户所描述的道路结构中的实体及关系一一对应. 通过对候选子图随机采样, 可以得到具有相同结构特征的不同场景道路. 最后, 使用 Scenic 原有语句在相应道路上放置车辆, 生成描述场景.

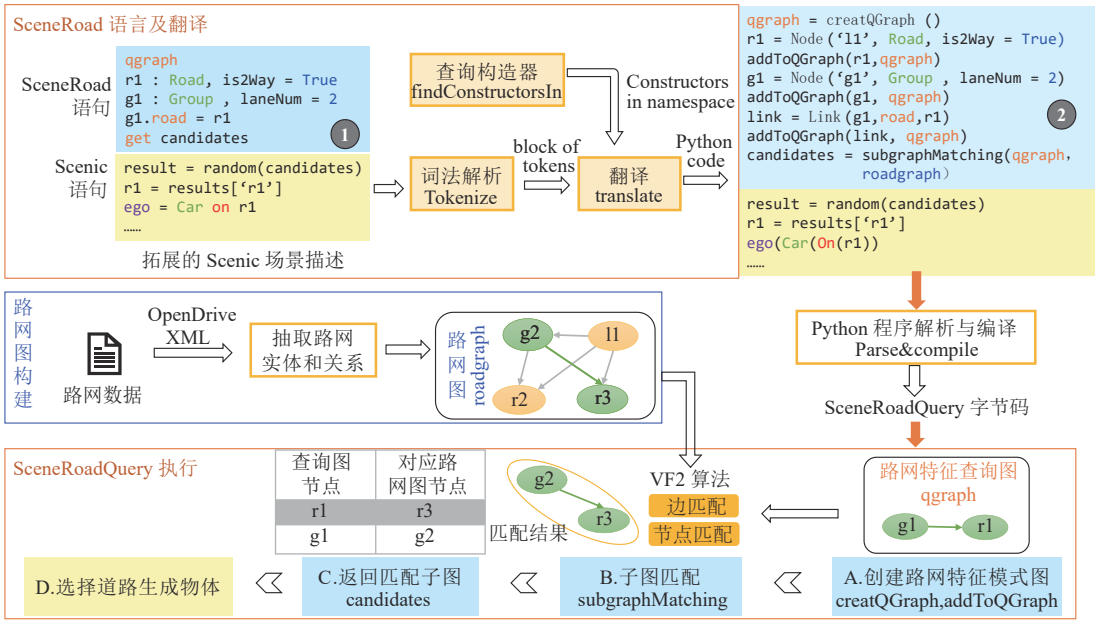


图 9 拓展 Scenic 的编译以及 SceneRoad 构建子图和子图匹配的过程

2.3 SceneRoadQuery 执行的性能测试

本文对 SceneRoadQuery 的实际使用性能进行评测. 从计算复杂性来讲, 如果对查询图不加限制, 子图匹配的判定是 NP 完全的. VF2 算法在最优情况下时间复杂度为 $O(N^2)$, 最差情况下时间复杂度为 $O(N!N^2)$, 其中 N 为节点数. 从算法角度来说, 描述的场景道路的规模没有大小限制, 小到单个车道, 大到一个局部区域内所有连通的道路, SceneRoad 都可以根据用户描述的道路结构特征构建相应的路网特征查询图. 但在实际静态场景的生成过程中, 由于自身车辆的可视范围有限 (例如, 以自身为中心半径 100 m 内), 用户通常只需考虑在自身车辆可视范围内描述场景中的道路以及场景中其他车辆、行人等实体的分布情况. 此时, 场景中的道路结构通常只涉及自身车辆所在的当前道路及其直接前驱或后继, 这使得 SceneRoad 实际构建的路网特征查询图的规模保持在较小范围内. 为了进一步定量评估 SceneRoadQuery 场景道路搜索的时间开销, 本文统计了 CARLA 模拟器中 8 个不同地图构建的路网图规模以及 5 个从简单到复杂的场景道路示例在不同路网图中的子图匹配情况以及平均耗时. 图 10 为 CARLA 模拟器中 Town03 地图的路网结构示意图. 表 2 展示了 CARLA 模拟器中 8 个不同地图构建的路网图中路网实体以及关系数.

表 3 展示了不同 SceneRoad 描述的场景道路特征在路网中的匹配情况和平均耗时, 实验 CPU 为 Intel(R) Xeon(R) Gold 5220R CPU @ 2.20 GHz. 示例对应的 SceneRoad 的描述以及在 CARLA 模拟器中生成的场景详见附录 B. 表 3 中, 候选匹配子图数表示每个示例中所描述的道路特征在所有 8 个地图中匹配到的子图总数, 每地图平均耗时则表示的是得到一个地图中所有匹配示例所描述的道路特征的子图的平均耗时. 从表中数据可以看出, 对不同的场景道路特征, SceneRoad 进行子图匹配所用的时间都为毫秒级. SceneRoad 默认获取所有候选匹配子图并进行保存, 从而在后续重新生成场景时可直接从已保存的候选匹配中随机选取而不再重复地进行子图匹配算法, 因此, 在某些示例中其平均获取每个候选子图所用的时间可以小于 1 ms. 测试结果表明, SceneRoad 在实际使用过程中有足够的场景道路搜索效率, 为大量生成仿真场景提供了保障.

对于 SceneRoad 生成的场景的正确性, 本文从场景生成前, 场景生成过程, 场景生成后 3 个方面进行考虑. 首先, 本文所定义的 4 类路网实体以及表 1 中定义的实体之间的关系参考了 OpenDRIVE 标准中定义的元素. 本文基于这些定义从原始路网数据中抽取信息构建路网图, 其正确性可以得到保证. 其次, 本文所用的 VF2 算法是解

决子图同构问题的成熟算法, 其匹配结果的准确性可以得到保证. 在生成场景的过程中, 程序定义了一些约束条件, 当生成的场景不能满足基本约束时, 场景生成失败, 程序将会采用调整策略或者重新生成场景. 最后, 对生成的场景提取关键信息, 包括场景中车辆所在的车道, 道路, 这些信息可以用于对生成的场景正确性做进一步的验证.

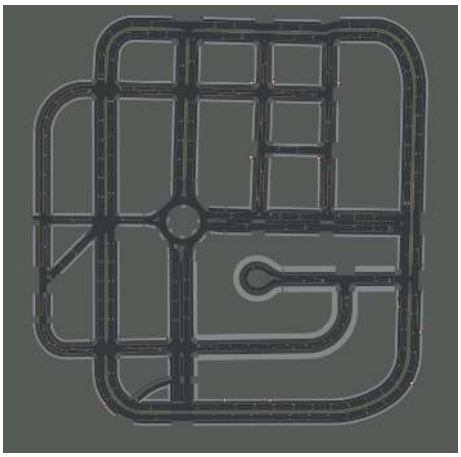


图 10 CARLA 模拟器 Town03 地图路网结构

表 2 CARLA 地图构建的路网图规模

地图	路网图节点数量	路网图边数量
Town01	358	2200
Town02	252	1560
Town03	970	6359
Town04	1011	7098
Town05	1078	7726
Town06	776	5515
Town07	818	4974
Town10HD	376	2605

表 3 不同 SceneRoad 描述的子图匹配情况和平均耗时

示例	路网特征查询图节点数量	路网特征查询图边数量	候选匹配子图数	每地图平均耗时 (s)
Case01	1	0	2420	0.1
Case02	2	1	1286	0.5
Case03	4	3	175	0.3
Case04	5	4	174	0.39
Case05	7	6	131	0.46

2.4 场景描述能力对比

SceneRoad 主要优势在于: 借助路网图模型用简单关系的组合表达多个实体之间复杂关系约束, 并且能够利用解决子图匹配问题的算法得到路网中所有符合描述的道路结构, 降低场景道路描述的复杂性, 实现从多个层次描述自动驾驶场景, 提高生成场景的多样性.

● SceneRoad 与 Scenic 的描述方式比较. 一个路网实体可能与其他多个实体之间存在关系, 因此在 Scenic 描述语言中用 filter 进行选择的时候就需要考虑到该实体关联的所有约束条件, 这其中既包含实体本身的属性约束, 还要包括与其他实体之间的关系约束.

图 11 展示了利用 Scenic 中 filter 描述的图 7 中的道路实体 r1. 由于 r1 中 ego 所在车道的前方为交叉路口, 因此该约束必须添加到筛选条件之中, 这意味着当一个场景中道路结构十分复杂, 实体之间存在非常多的关系约束时, Scenic 需要对每个实体分析与其关联的所有约束条件, 这使得构建描述过程变得复杂且程序难以理解. 相比之下, SceneRoad 能够将这种复杂的关系约束拆分成一系列简单约束的组合, 并且可以在定义实体后的程序中的任意位置描述与该实体相关的关系, 使得整个描述变得更加灵活简单, 更易理解.

```
r1 = filter( lambda r: r.is2Way ==True
    and len(r.groups[0].laneNum) == 2
    and len(r.groups[1].laneNum) == 2
    and r.groups[0].succJunction is not None
    and r.groups[0].succJunction.is4Way == True,network.roads)
```

图 11 Scenic 利用 filter 筛选道路

SceneRoad 也可用于描述一条路径. 例如, SceneRoad 可以通过组合连续道路实体获取一条车辆行驶路线, 路线中先有直行道路, 随后交叉路口右转, 接着直行一段距离后左转, SceneRoad 可以通过组合每一段的约束在路网图中搜索这样一条路径, 如图 12 所示. 此类连续多个道路实体之间存在相互约束的情况很难用 Scenic 中的 filter 描述, 一种替代的方案是如图 13 在循环中嵌套地筛选符合条件的道路, 两者对比之下, SceneRoad 的描述更加简洁和清晰.

```
l1 : Lane, turn = STRAIGHT
l2 : Lane, turn = RIGHT,in_junction = True
l3 : Lane, turn = STRAIGHT
l4 : Lane, turn = LEFT
l1.succ = l2
l2.succ = l3
l3.succ = l4
```

图 12 SceneRoad 描述一条路线中的各个车道

```
for l1 in network.lanes:
 if l1.turn == STRAIGHT:
   for l2 in l1.succLanes:
    if l2.turn == RIGHT and l2.in_junction:
     for l3 in l2.succLanes:
      if l3.turn = STARIGHT:
       for l4 in l3.succLanes:
        if l4.turn = LEFT:
         condidates.append([l1,l2,l3,l4])
```

图 13 循环嵌套地筛选符合条件的道路

● SceneRoad 与 OpenSCENARIO2 的比较. 后者同样定义了抽象路网以及对场景的道路的描述. 图 14 为 OpenScenario2 的描述示例, 它创建了一个连接了两车道道路的十字交叉路口和一条右转路线. 从该示例可以看到, OpenSCENARIO2 对实体的描述与 SceneRoad 类似, 对实体的属性值约束则通过 with, keep 进行定义. 实体之间的关系的描述由相关修饰符进行约束, 通过将一些实体作为参数, 返回相应的匹配结果. 例如, 示例中的修饰符 roads_follow_in_junction 返回的是从交叉路口的入路 r3 到出路 r4 的路线 (route). OpenSCENARIO2 定义了抽象路网 map 中的一些基础修饰符, 这些修饰符仅定义了其实现的功能, 输入以及返回的实体类型, 具体实现则需要用户或编译器去实现. 定义复合拓扑需要更复杂的自定义修饰符, 这些自定义修饰符以及其中潜在可用的道路实体可以由更小的修饰符构建, 以封装更广泛的对抽象路网的约束. 每个修饰符的内部逻辑都需要用户实现, 这要求用户需要充分理解 OpenSCENARIO2 抽象路网中定义的各种修饰符和方法, 这对用户从实际使用来说更加困难.

```
map: map
r3, r4: road with:
    keep(number_of_lanes == 2)
my_mod2: map.roads_follow_in_junction (
    in_road: r3,
    out_road: r4,
    clockwise_count: 3,
    number_of_roads: 4)
```

图 14 OpenSCENARIO2 中对道路的相关描述

3 基于场景描述语言生成仿真场景

下面通过具体场景描述实例来展示扩展了 SceneRoad 的 Scenic 语言如何通过简单的场景描述生成数以千计的多样场景.

场景可以被抽象建模为一组参数的组合, 如光照强度、天气、道路结构、车辆和行人数量、位置朝向等. 这些参数的取值范围共同构成场景参数空间. 场景描述语言考虑如何用一种简洁、方便的方式描述并生成一组特定参数的场景. 本节使用拓展的 Scenic 语言构建了一个随机生成场景的描述, 测试其生成大规模随机场景的性能. 为了加快场景生成速度, 我们改良了 Scenic 原有的场景采样策略, 通过设计的启发式微调算法对采样失败的场景中的物体进行调整, 提高了场景生成的成功率.

图 15 为生成一般随机场景的描述, 这里使用最基本的道路描述 (第 2 行) 来选择放置 ego 的车道, 子图匹配将返回地图中所有的车道构成的集合 (第 1–3 行). 从该集合中随机采样得到 lane (第 4 行) 并将 ego 放置在其中心线上随机一点 (第 5 行). 表 4 列举了该场景描述中用到的部分重要参数. 参数的取值参考了真实数据集中的相关信息. 我们设置生成车辆数量为整数区间 [1,10] 内随机整数, 行人数量为整数区间 [0,5] 内随机整数. 在 KITTI[20], nuScenes[21]等数据集中, 距离过远或像素过小的物体不会纳入训练和评估范围, 因此我们将场景中物体的生成范围限定在 ego 前方 50 m, 视角为 90°的扇形区域内. 光照, 天气等其他场景参数则采用 Scenic 中的默认值.

```
1  qgraph
2  lane : Lane
3  get matched
4  lane = getRandomlyFrom(matched)
5  ego = Car on lane.centerline,
6        with viewAngle 90 deg,
7        with visibleDistance 50,
8  generateScene(ego,c_num,p_num,lane)
```

图 15 拓展 Scenic 生成一般随机场景的描述

表 4 生成场景的重要参数

参数	含义	取值
c_{num}	车辆数量	$c_{num} \in [1,10]$
p_{num}	行人数量	$p_{num} \in [0,5]$
visibleDistance	ego的可视距离	50 m
viewAngle	ego的可视角度	$\pi/2$

场景描述中的算法 2 (generateScene) 用于在 ego 的可视区域中随机生成 c_{num} 辆车和 p_{num} 个行人. 它基于一个简单的随机取点生成车辆的方法, 并通过裁剪取点空间来降低多辆车重叠的概率. 在算法 2 中, 首先获取 ego 的可视范围内的可行驶区域 (driveable region) 以及人行道区域 (sidewalk region). 获取的可行驶区域在 Scenic 中表示为一个向量场 (VectorField), 其内部的每点都有朝向. 对道路区域来说, 此方向为车道的前进方向, 因此生成的车辆默认朝向其所在道路的前进方向. 为了减少场景生成失败的概率, 首先在每次生成新的车辆的同时对可行驶区域进行裁剪, 剔除已生成的车辆所在的区域, 使得新生成的车辆尽可能不与已有车辆重叠.

算法 2. generateScene.

输入: 自身车辆 ego, 车辆数量 c_{num}, 行人数量 p_{num}, 车道 lane;
输出: 生成场景 Scenario.

d_r←drivable region in ego.visibleRegion
s_r←sidewalk region in ego.visibleRegion
FOR i in range(c_{num}) **DO**
  car_i←Car in d_r {On a random point in d_r}
  c_r←RectangularRegion of car_i
  d_r←d_r—c_r
END FOR
FOR i in range(p_{num}) **DO**
  ped_i←Pedestrian in s_r
END FOR

对采样后仍然失败的场景, 使用启发式的微调策略对不满足两个 Scenic 内置要求 (*containment* 和 *intersection*) 的车辆的位置和朝向等进行调整 , 取代 Scenic 原有的直接丢弃失败采样的策略. 具体来说, 图 16 展示了失败场景的示例, BR1 和 BR2 分别为车辆超出道路边界以及多个车辆相互重叠的情况. 对于超出道路边界的车辆, 设该车辆所在的点为 p, 找出该车辆所在车道的中心线上与其最近的一点 c 并重新设置 c 为该车辆新的位置点. 对于相互重叠的车辆, 将它们分成不同的组, 其中同组内任意一个车辆至少与另一个车辆重叠, 而不同组之间车辆互不重叠. 对于每一个组, 随机选定一个车辆作为起始并贪心地将与其重叠的其他车辆删除, 对剩余车辆递归地执行此操

作直到该组内不存在相互重叠的车辆为止.

我们对比了使用算法 2 结合调整策略与直接在可行驶区域内随机取点放置车辆并丢弃失败场景这两种不同方法生成车辆密集型场景所用的时间. 对第 1 种方法, 我们取 c_{num} 为 13, 生成的场景中剩余车辆数量至少为 10 则视为成功. 对第 2 种方法, 我们取 c_{num} 为 10. 每种方法各生成 1000 个场景. 实验所用 CPU 为 Intel(R) Core(TM) i7-10875H CPU @ 2.30 GHz, GPU 为 NVIDI GeForce RTX 2060. 实验结果如表 5 所示. 从表中数据可以看出, 算法 2 结合调整策略能够大幅降低场景采样的次数, 其生成车辆密集型场景所用平均时间小于 1 s.

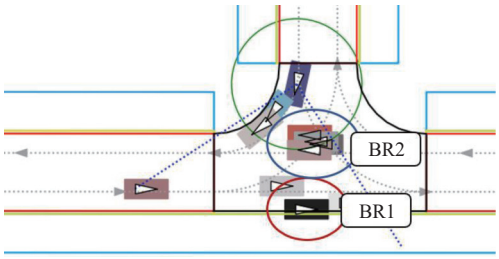
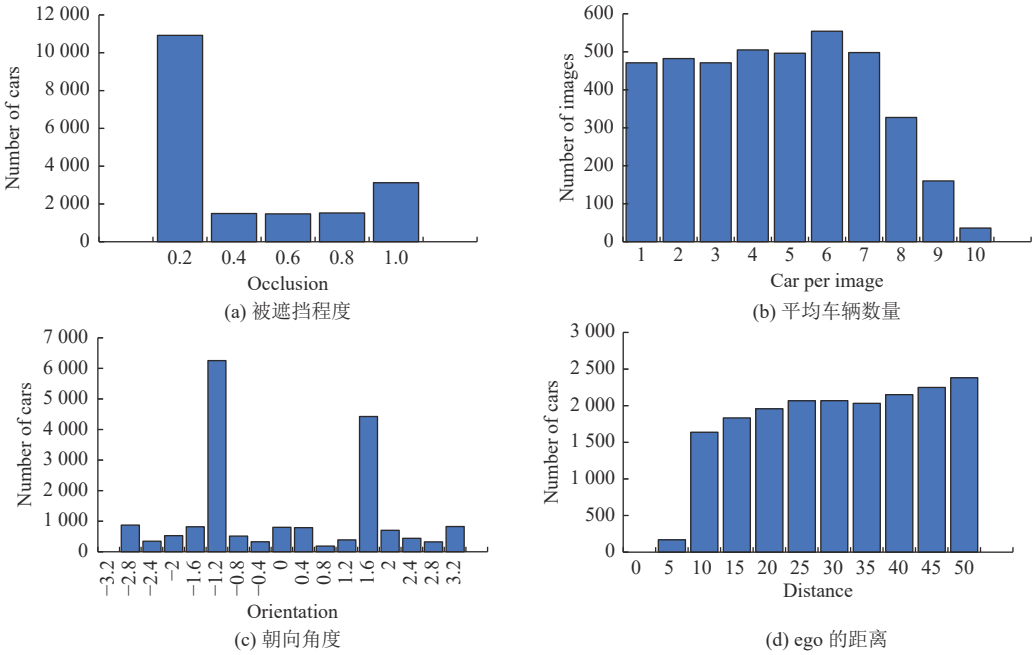


图 16　与 Scenic 内置要求 *containment* 和 *intersection* 冲突的失败场景示例

表 5　随机生成 10 辆车场景性能对比

方法	平均采样次数	平均耗时 (s)	平均车辆数量
算法2+优化采样	5	0.7	10.7
随机取点+原有采样	534	50.5	10

实验基于图 15 中的场景描述以及调整策略生成了 4 000 个场景. 实验对比了同等量级的 KITTI 数据集与仿真场景数据集的相关统计量的分布以及熵的方法评估生成场景与真实场景是否相似, 这对后续基于仿真场景的模型的测试评估工作十分重要. 图 17 展示了生成场景中车辆相关的统计信息.



(a) 被遮挡程度　(b) 平均车辆数量

(c) 朝向角度　(d) ego 的距离

图 17　生成场景的车辆相关统计信息

本文分别计算了 KITTI 数据集以及生成场景数据集中车辆和行人的数量、与 ego 之间的距离 (0 到 50 m 内每 5 m 一个间隔)、朝向角度 ($[-\pi,\pi]$ 区间 16 等分) 、被遮挡程度 (4 种遮挡状态) 这几个重要统计量分布的熵, 熵越大则表明数据集在此统计量分布上更均匀, 更具多样性, 计算的结果如表 6 所示. 从表中可以看到, 本节实验生成的随机场景数据集中, 车辆和行人的相关重要统计量分布的熵与 KITTI 数据集中对应的数值基本接近, 在车辆朝向以及遮挡情况, 行人的数量、与 ego 之间的距离等多个统计量分布上更加均匀. 下一节, 本文将利用这些生成的场景对不同的感知模型进行测试, 并通过对比多个模型在真实数据集与仿真数据集上的精度变化趋势进一步说明生成的仿真数据集与真实数据集的相似性.

表 6　数据集中车辆和行人相关统计量分布的熵

数据集	类别	H_{num}	H_{dis}	H_{ori}	H_{occ}
KITTI	Car	2.29	2.22	2.25	1.15
	Pedestrian	0.92	1.99	2.62	0.93
Ours	Car	2.17	2.22	**2.35**	**1.22**
	Pedestrian	**1.76**	**2.12**	2.34	**1.24**

4　生成的场景用于感知模型测试

在利用场景描述语言生成仿真场景后, 可以从场景中获取仿真感知数据. 图 18 展示了在 CARLA 模拟器上实例化的场景以及仿真图像数据. 仿真场景在感知模型测试中有很多应用场景, 例如, 可以验证对模型改进后的精度和鲁棒性是否提升. 现实中采集真实数据并进行标注是十分昂贵且耗时的, 在有限的数据上可能无法对模型进行系统性的评测, 对未进行充分验证的模型直接进行道路测试是存在风险的. 而在仿真环境中可以用低成本的方式生成数以百万计的测试场景, 从而实现对模型更充分的评估, 经过仿真环境充分验证之后再进行实测, 提高了系统测试验证的安全性和可靠性. 仿真环境中可以生成一些真实场景中难以测试的危险驾驶场景, 从而能够对自动驾驶系统进行模型在环, 软件在环等全方位的测试, 测试系统在面对危险场景下的应对行为, 发现潜在的安全风险, 从而提高整体的安全性.

图 18　生成场景的仿真数据

然而, 上述应用实现的前提是模型在仿真场景和真实场景上的表现具有一致性, 只有这样, 在仿真上的测试结果才具有意义. 本文已利用场景描述语言生成了与真实数据集同等量级的仿真场景, 本节将通过比较多个模型在同等规模两个数据集上检测精度是否具有一致性以验证本文仿真数据生成方法的有效性, 为后续生成关键测试场景或危险场景等提供支撑.

目前仿真场景包含车辆 (car) 和行人 (pedestrian) 两类动态实体, 因此, 为了让两个数据集保持基本一致, 所用的真实数据集也只包含这两类检测目标. 本文从 KITTI 数据集的训练集和验证集中分别筛选出只包含车辆和行人两种实体的场景, 其中从训练集中筛选得到 2 000 个场景, 从验证集中得到 1 500 个场景, 并将真值标签转为 COCO 数据集[22]的格式, 分别作为实验所用的真实数据集 KITTI_COCO 的训练集和测试集. 仿真场景则是从场景描述语言所生成的 4 000 个场景中随机选取了 3 500 个, 并同样按照训练集 2 000, 测试集 1 500 的比例进行随机划分, 构建了仿真数据集 Synthetic, 仿真图像传感器的各项参数及输出图像尺寸按照 KITTI 数据集设置. 从生成的场景中收集的信息显示, ego 及场景中的其他车辆、行人所在的道路包含了多种不同属性的 4 类路网实体, 他们相互之间构成了更复杂的道路结构. 例如, 根据车道 (lane) 实体的“turn”属性不同, ego 所处的道路包括了直道, 左转弯道, 右转弯道. 根据道路 (road) 实体的车道组 (group) 的数量的不同, ego 所在的道路包含了单向车道和双向车道. 当 ego 位于交叉路口时, 根据交叉路口 (junction) 实体的“is4Way”“is3Way”属性, 这些场景中包含了十字型路口和 T 型交叉路口等.

构造完测试的真实数据集和仿真数据集后, 我们选取了 MMDetection 框架 (2.19.1) 中支持的 10 个目标检测模型, 并分别在两个数据集上进行训练和评估. 本文在 MMDetection 官方给出的这 10 个不同模型的一些默认训练配置的基础上对学习率策略以及训练 epoch 数等进行了调整, 确保模型在训练结束时收敛, 并选取了每个模型在整个训练过程中的最优的 Bbox mAP. 每个模型的配置文件, 学习率策略以及训练 epoch 数等详细信息见附录 C, 实验 GPU 环境为 NVIDIA GeForce RTX 3090.

表 7 中列举了模型分别在两个不同数据集上的精度数据, 目标检测精度评估指标为 COCO 数据集中使用的 mAP (mean average precision). mAP 反映了不同阈值下多个类别的平均检测精度. 为了更直观和清晰的展示模型在两个不同数据集上的精度对比情况, 我们将模型按照在真实数据集上精度从低到高排序, 如图 19 所示. 实验将从纵向和横向对结果进行比较分析. 横向指 10 个不同的待测目标检测模型在同一数据集上进行训练后的目标检测精度的比较, 比较的结果体现的是不同网络模型在该数据集上的表现, 因此横向比较偏向于对模型本身的测评. 纵向是指保持这 10 个模型的配置不变, 改变训练集和测试集, 即在真实数据集与仿真数据集分别训练和测试等, 其主要目的是观察模型在不同数据集上精度的变化趋势的异同, 因此纵向的结果体现的是真实和仿真数据的特点. 为了确保模型在仿真上的评测结果能够反映其在真实情况下的表现, 我们首先对两个数据集上的结果进行纵向对比.

图 19(a) 为 6 种使用相同特征提取骨干网络 (ResNet50) 的不同模型的精度对比情况, 从图中可以看出, 模型在两个不同数据集上的检测结果具有明显的正相关性, 即在 KITTI_COCO 上表现较好的模型, 其在仿真数据集上同样取得了较好的效果, 两者呈正相关. Pearson 相关系数是用来衡量两个变量相关性的常见指标, 取值范围为 [−1, 1], 越接近 1 表示两者之间的正相关趋势越强. 我们计算两组模型的精度之间的 Pearson 相关系数为 0.92. 在图 19(b) 中, 我们选取图 19(a) 中表现最好的网络模型, 并与其他 4 种使用不同特征提取骨干网络的模型的精度进行比较, 模型在仿真和真实数据集中的表现同样基本呈正相关趋势, 两者 Pearson 相关系数为 0.966. 实验结果表明, 被测模型在真实数据集和仿真数据集中具有相同的性能差异, 模型评测的结果可以显示在真实场景中的表现情况. 实验结果验证了本文仿真数据生成方法的有效性.

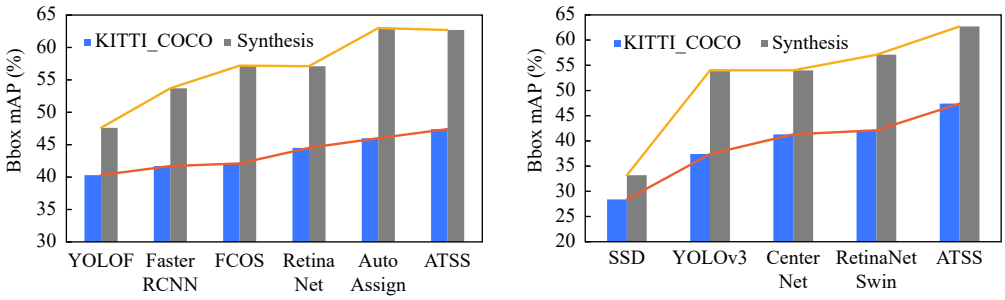
基于上述结论, 我们可以对本节实验具体应用场景下的模型进行横向比较分析. 本实验的应用场景即为通过构造与真实应用场景相似的仿真测试场景, 对多个不同的感知模型进行横向比较, 从而选择出在具体应用场景下表现最优的模型. 本文所测的 10 个模型中既有如 Faster RCNN[23]这类 anchor-based 的模型, 也有 CenterNet[24] 这类 anchor-free 的模型. 从实验数据来看, ATSS[25]和 AutoAssign[26]分别在本文所测试的真实数据集和仿真数据集上表现最优. ATSS 分析了 anchor-based 和 anchor-free 的特点并提出自动确定正负样本的方法, AutoAssign 分析 ATSS 存在的问题并完成端到端的动态 label assignment. 尽管两个数据集上表现最优的模型不同, 但两个最优模型之间精度差距相较于与其他模型之间的精度差异要小很多. 同时, 这两个模型在 COCO 数据集上的表现也优于本文所测的大多数模型, 这也从侧面体现了其方法的有效性和鲁棒性. 需要注意的是, AutoAssign 在 COCO 数据集上的精度要优于 ATSS, 但在本文所测的 KITTI_COCO 数据集上, ATSS 要优于 AutoAssign, 这也从侧面说明模型

的精度除了与模型设计相关, 也会受到实际使用场景的影响, 但这种影响要小于模型设计产生的影响, 模型的设计对模型的精度表现起主要作用.

表 7 在 KITTI 数据集和仿真数据集上的模型精度评测

模型	训练集	测试集	Bbox mAP (%)
Faster RCNN[23]	KITTI_COCO	—	41.7
	Synthetic	—	53.7
	Synthetic	KITTI_COCO	17.5
YOLOv3[27]	KITTI_COCO	—	37.4
	Synthetic	—	54.0
	Synthetic	KITTI_COCO	15.6
SSD[28]	KITTI_COCO	—	28.4
	Synthetic	—	33.2
	Synthetic	KITTI_COCO	11.9
RetinaNet[29]	KITTI_COCO	—	44.5
	Synthetic	—	57.1
	Synthetic	KITTI_COCO	25.0
FCOS[30]	KITTI_COCO	—	42.1
	Synthetic	—	57.2
	Synthetic	KITTI_COCO	21.3
ATSS[25]	KITTI_COCO	—	47.4
	Synthetic	—	62.7
	Synthetic	KITTI_COCO	25.5
YOLOF[31]	KITTI_COCO	—	40.3
	Synthetic	—	47.6
	Synthetic	KITTI_COCO	23.3
AutoAssign[26]	KITTI_COCO	—	46.0
	Synthetic	—	63.0
	Synthetic	KITTI_COCO	22.1
CenterNet[24]	KITTI_COCO	—	41.3
	Synthetic	—	54.0
	Synthetic	KITTI_COCO	21.2
RetinaNet (Swin backbone)[32]	KITTI_COCO	—	42.1
	Synthetic	—	57.1
	Synthetic	KITTI_COCO	22.8

注: 测试数据集为“—”表示测试集与训练集属于同一数据集



(a) 使用相同特征提取网络的模型精度对比 (b) 使用不同特征提取网络的模型精度对比

图 19 模型在两个不同数据集上的精度对比

从实验数据看, 模型在仿真数据集上的表现明显普遍优于其在真实数据集上的表现, 这主要与仿真器的仿真数据的真实程度以及场景的复杂程度有关, 本文所使用的 CARLA 模拟器是相关研究领域使用最为广泛的开源模拟器之一, 尽管如此, 其仿真数据相较于真实的场景来说依然具有一些差距, 复杂性也低于真实场景数据, 因此在本文设计的实验中, 模型在同等数据规模的仿真数据集上表现会更好. 但是, 得益于场景描述语言与仿真器的解耦, 该场景描述语言可以应用到其他使用 OpenDRIVE 地图等的更逼真的仿真器上, 仿真与真实之间的这种差异也会随着仿真技术的发展以及场景的复杂度的提升变得越来越小.

本文强调仿真场景在模型测试上的应用而并未将仿真数据集上训练的模型直接应用到真实数据中, 这是因为当测试集和训练集属于不同数据集时, 模型精度会发生明显下降. 表 7 中同样列举了 10 个不同模型在仿真数据集上训练并在真实数据集上测试的结果, 可以看到其结果明显低于其他两个实验结果. 这主要是因为域差异 (domain gap) 的存在, 即训练数据与测试数据来自不同的分布, 例如, 白天与夜晚, 晴天与雨雪天, 甚至传感器的高度, 角度等不同也会导致这种差异. 合成数据与真实数据通常具有较大的域差异 , 因此在合成数据上训练的模型在真实数据上检测精度往往较低. 域适应 (domain adaptation), 迁移学习 (transfer learning) 等研究工作致力于解决域差异的问题. 随着仿真技术的不断发展以及场景生成技术和深度学习技术的发展, 仿真场景也将会在自动驾驶中得到更多的应用.

5 总 结

本文设计了基于路网图模型的场景道路描述 DSL, 名为 SceneRoad, 并将其集成到场景描述语言 Scenic 中以提升其静态场景表达能力. 通过与 Scenic, OpenSCENARIO2 等场景描述语言对比, SceneRoad 能够用简洁可读的方式描述场景道路实体以及各实体之间的关系, 为 Scenic 提供了道路层次的描述, 使得其能够从多个不同场景层次创建更多样化的场景. 通过在仿真器中对场景进行实例化, 我们获取大量的场景仿真数据并且通过设计实验评估了生成的仿真数据集的质量以及其在自动驾驶深度学习感知模型评估方面的价值. 作为一门场景描述语言, SceneRoad 还存在可以进一步扩展的地方, 从原始路网数据中抽取的实体和关系可以进一步扩充, 主要有以下几个方面: 实体种类以及实体的属性值, 例如环岛实体, 人行横道的定义和抽取; 实体之间的关系, 可以通过组合已有的基础关系或者定义新的实体之间关系, 从而对路网图进行扩充, 使得 SceneRoad 拥有更丰富的场景描述能力. 未来将继续拓展场景描述语言在动态场景的描述能力, 并借助场景描述语言生成探索生成更多对自动驾驶测试有意义的测试场景, 保证自动驾驶的安全性.

References:

[1] ISO. 26262-1:2018 Road vehicles—Functional safety—Part 1: Vocabulary. 2018. https://www.iso.org/standard/68383.html

[2] ISO/PAS 21448:2019 Road vehicles—Safety of the Intended Functionality. 2019. https://www.iso.org/standard/70939.html

[3] ISO/SAE 21434:2021 Road vehicles — Cybersecurity engineering. 2021. https://www.iso.org/standard/70918.html

[4] Google Waymo. 2022. https://waymo.com/intl/zh-cn/

[5] Dosovitskiy A, Ros G, Codevilla F, Lopez A, Koltun V. CARLA: An open urban driving simulator. In: Proc. of the 1st Annual Conf. on Robot Learning. Mountain View, 2017. 1–16.

[6] Shah S, Dey D, Lovett C, Kapoor A. AirSim: High-fidelity visual and physical simulation for autonomous vehicles. In: Hutter M, Siegwart R, eds. Field and Service Robotics. Springer, 2018. 621–635. [doi: 10.1007/978-3-319-67361-5_40]

[7] Fremont DJ, Kim E, Dreossi T, Ghosh S, Yue XY, Sangiovanni-Vincentelli AL, Seshia SA. Scenic: A language for scenario specification and data generation. Machine Learning, 2022. [doi: 10.1007/s10994-021-06120-5]

[8] ASAM. OpenDRIVE 1.7.0. 2021. https://www.asam.net/standards/detail/opendrive/

[9] ASAM. OpenSCENARIO 2.0.0. 2022. https://www.asam.net/project-detail/asam-openscenario-v20-1/

[10] Ulbrich S, Menzel T, Reschka A, Schuldt F, Maurer M. Defining and substantiating the terms scene, situation, and scenario for automated driving. In: Proc.of the 18th IEEE Int'l Conf. on Intelligent Transportation Systems. Gran Canaria: IEEE, 2015. 982–988. [doi: 10.1109/ITSC.2015.164]

[11] Fowler M. Domain-specific languages. Pearson Education, 2010. https://martinfowler.com/books/dsl.html

[12] Zhang XH, Tao JB, Tan KG, Torngren M, Sanchez JMG, Ramli MR, Tao X, Gyllenhammar M, Wotawa F, Mohan N, Nica M, Felbinger H. Finding critical scenarios for automated driving systems: A systematic mapping study. IEEE Trans. on Software Engineering, 2023, 49(3): 991–1026. [doi: 10.1109/TSE.2022.3170122]

[13] Scholtes M, Westhofen L, Turner LR, Lotto K, Schuldes M, Weber H, Wagener N, Neurohr C, Bollmann MH, Körtke F, Hiller J, Hoss M, Bock J, Eckstein L. 6-layer model for a structured description and categorization of urban traffic and environment. IEEE Access, 2021, 9: 59131–59147. [doi: 10.1109/ACCESS.2021.3072739]

[14] Chen K, Wang JQ, Pang JM, Cao YH, Xiong Y, Li XX, Sun SY, Feng WS, Liu ZW, Xu JR, Zhang Z, Cheng DZ, Zhu CC, Cheng TH, Zhao QJ, Li BY, Lu X, Zhu R, Wu Y, Dai JF, Wang JD, Shi JP, Ouyang, WL, Loy CC, Lin DH. MMDetection: Open MMLab detection toolbox and benchmark. arXiv:1906.07155, 2019.

[15] Menzel T, Bagschik G, Maurer M. Scenarios for development, test and validation of automated vehicles. In: Proc. of the 2018 IEEE Intelligent Vehicles Symp. (IV). Changshu: IEEE, 2018. 1821–1827. [doi: 10.1109/IVS.2018.8500406]

[16] Bauer MP, Ngo A, Resch M. The YASE framework: Holistic scenario modeling with behavior trees. In: Proc. of the 94th IEEE Vehicular Technology Conf. Norman: IEEE, 2021. 1–7. [doi: 10.1109/VTC2021-Fall52928.2021.9625405]

[17] Gordon AD, Henzinger TA, Nori AV, Rajamani SK. Probabilistic programming. In: Proc. of the 2014 Future of Software Engineering Proc. Hyderabad: ACM, 2014. 167–181. [doi: 10.1145/2593882.2593900]

[18] Cordella LP, Foggia P, Sansone C, Vento M. A (sub) graph isomorphism algorithm for matching large graphs. IEEE Trans. on Pattern Analysis and Machine Intelligence, 2004, 26(10): 1367–1372. [doi: 10.1109/TPAMI.2004.75]

[19] Hagberg A, Swart P, Chult DS. Exploring network structure, dynamics, and function using NetworkX. Los Alamos: Los Alamos National Lab, 2008.

[20] Geiger A, Lenz P, Urtasun R. Are we ready for autonomous driving? The KITTI vision benchmark suite. In: Proc. of the 2012 IEEE Conf. on Computer Vision and Pattern Recognition. Providence: IEEE, 2012. 3354–3361. [doi: 10.1109/CVPR.2012.6248074]

[21] Caesar H, Bankiti V, Lang AH, Vora S, Liong VE, Xu Q, Krishnan A, Pan Y, Baldan G, Beijbom O. nuScenes: A multimodal dataset for autonomous driving. In: Proc. of the 2020 IEEE/CVF Conf. on Computer Vision and Pattern Recognition. Seattle: IEEE, 2020. 11618–11628 [doi: 10.1109/CVPR42600.2020.01164]

[22] Lin TY, Maire M, Belongie S, Hays J, Perona P, Ramanan D, Dollár P, Zitnick CL. Microsoft COCO: Common objects in context. In: Proc. of the 13th European Conf. on Computer Vision. Zurich: Springer, 2014. 740–755. [doi: 10.1007/978-3-319-10602-1_48]

[23] Ren SQ, He KM, Girshick R, Sun J. Faster R-CNN: Towards real-time object detection with region proposal networks. IEEE Trans. on Pattern Analysis and Machine Intelligence, 2017, 39(6): 1137–1149. [doi: 10.1109/TPAMI.2016.2577031]

[24] Zhou XY, Wang DQ, Krähenbühl P. Objects as points. arXiv:1904.07850, 2019.

[25] Zhang SF, Chi C, Yao YQ, Lei Z, Li SZ. Bridging the gap between anchor-based and anchor-free detection via adaptive training sample selection. In: Proc. of the 2020 IEEE/CVF Conf on Computer Vision and Pattern Recognition. Seattle: IEEE, 2020. 9756–9765. [doi: 10.1109/CVPR42600.2020.00978]

[26] Zhu BJ, Wang JF, Jiang ZK, Zong FH, Liu ST, Li ZM, Sun J. AutoAssign: Differentiable label assignment for dense object detection. arXiv:2007.03496, 2020.

[27] Redmon J, Farhadi A. YOLOv3: An incremental improvement. arXiv:1804.02767, 2018.

[28] Liu W, Anguelov D, Erhan D, Szegedy C, Reed S, Fu CY, Berg AC. SSD: Single shot MultiBox detector. In: Proc. of the 14th European Conf. on Computer Vision. Amsterdam: Springer, 2016. 21–37. [doi: 10.1007/978-3-319-46448-0_2]

[29] Lin TY, Goyal P, Girshick R, He KM, Dollár P. Focal loss for dense object detection. In: Proc. of the 2017 IEEE Int'l Conf. on Computer Vision. Venice: IEEE, 2017. 2999–3007. [doi: 10.1109/ICCV.2017.324]

[30] Tian Z, Shen CH, Chen H, He T. FCOS: Fully convolutional one-stage object detection. In: Proc. of the 2019 IEEE/CVF Int'l Conf. on Computer Vision. Seoul: IEEE, 2019. 9626–9635. [doi: 10.1109/ICCV.2019.00972]

[31] Chen Q, Wang YM, Yang T, Zhang XY, Cheng J, Sun J. You only look one-level feature. In: Proc. of the 2021 IEEE/CVF Conf. on Computer Vision and Pattern Recognition. Nashville: IEEE, 2021. 13034–13043. [doi: 10.1109/CVPR46437.2021.01284]

[32] Liu Z, Lin YT, Cao Y, Hu H, Wei YX, Zhang Z, Lin S, Guo BN. Swin transformer: Hierarchical vision transformer using shifted windows. In: Proc. of the 2021 IEEE/CVF Int'l Conf. on Computer Vision. Montreal: IEEE, 2021. 9992–10002. [doi: 10.1109/ICCV48922.2021.00986]

附录 A

本文所提取的 4 类路网实体的部分属性, 属性可以根据具体应用场景自定义抽取, 见表 A1.

表 A1　4 类路网实体的属性

实体	属性	描述	类型	示例
Lane	turn	车道的转向	Enumeration	LEFT | RIGHT
	index	车道在Group内的id	Int	1
	groupId	车道所属的group	String	road1_group1
Group	laneNum	包含的车道数	Int	3
	roadId	Group所属的道路	String	road1
Road	is2Way	是否为双向车道	Bool	True
	inJunction	是否在交叉路口内	Bool	True
Junction	is4Way	是否为十字路口	Bool	True
	is3Way	是否为T型路口	Bool	Fasle

附录 B

5 个从简单到复杂的场景道路示例对应的 SceneRoad 描述以及在 CARLA 模拟器中生成的场景, 其中涉及的 Scenic 描述部分对实际代码进行了简化.

● Case01: Ego 位于随机车道, 其前方有一辆车, 与 ego 在相同车道.

1. qgraph
2. lane: Lane
3. get matched
4. ego on lane
5. car in ego.visibleRegion on lane

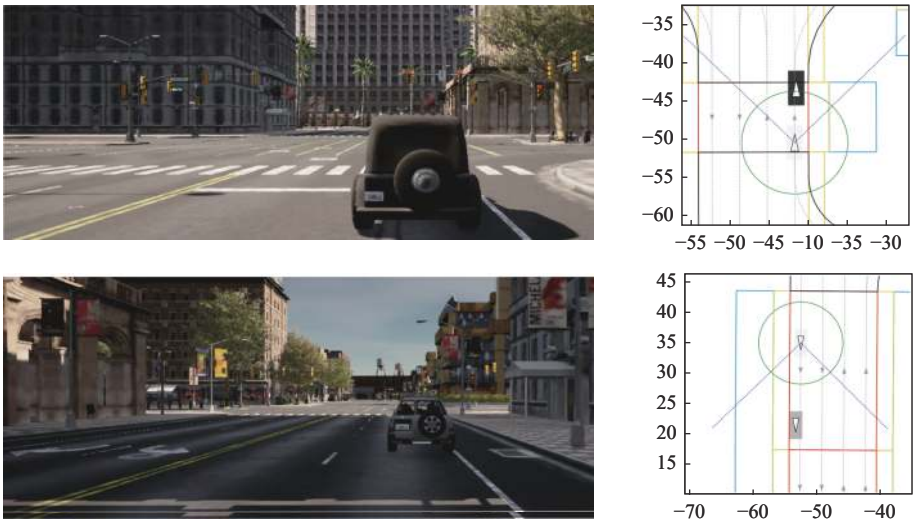


图 B1　示例 1

● Case02: Ego 位于交叉路口内的随机车道, 其前方有一辆车, 与 ego 在相同车道.

1. qgraph
2. lane: Lane
3. r1: Road, in_junction = True

4. lane.road = r1
5. get matched
6. ego on lane
7. car in ego.visibleRegion on lane

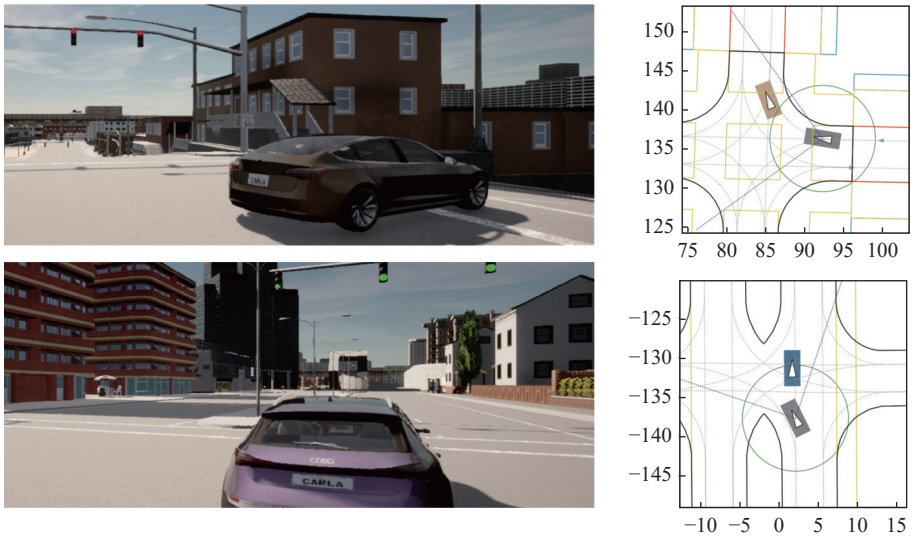


图 B2 示例 2

● Case03: Ego 位于双向 4 车道一侧靠近道路中心线的车道上, 其同向的右侧车道前方有一辆车辆.

1. qgraph
2. r1: Road, is2Way = True
3. g1: Group, laneNum = 2
4. l1 : Lane, index = 1
5. l2 : Lane, index = 2
6. g1.road = r1
7. l1.group = g1
8. l2.group = g1
9. get matched
10. ego on l1
11. car in ego.visibleRegion on l2

● Case04: Ego 位于双向 4 车道一侧靠近道路中心线的车道上, 其相对方向的外侧车道有一辆汽车.

1. qgraph
2. r1: Road, is2Way = True
3. l1: Lane, index = 1
4. l2: Lane, index = 2
5. g1: Group, laneNum = 2
6. g2: Group, laneNum = 2
7. g1.opposite = g2
8. g1.road = r1
9. l1.group = g1
10. l2.group = g2

11. get matched
12. ego on l1
13. car in ego.visibleRegion on l2

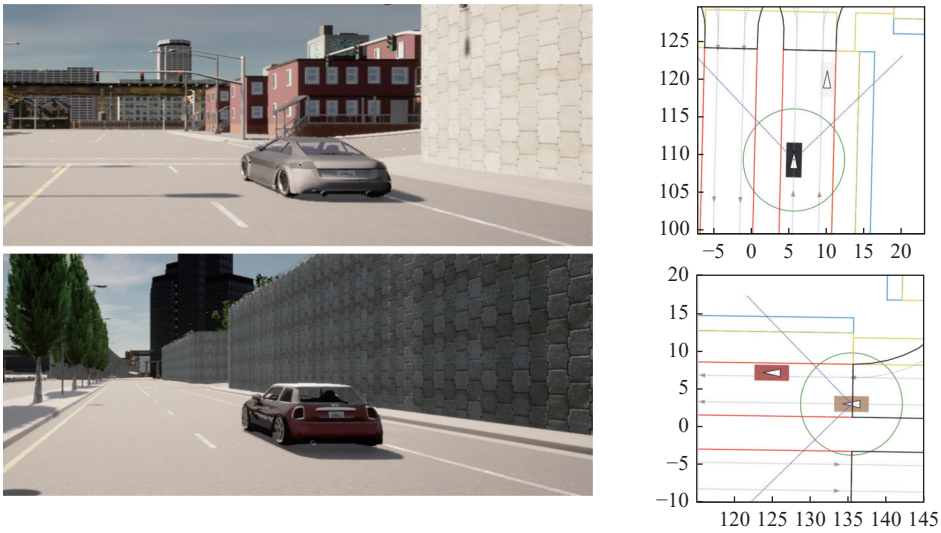


图 B3　示例 3

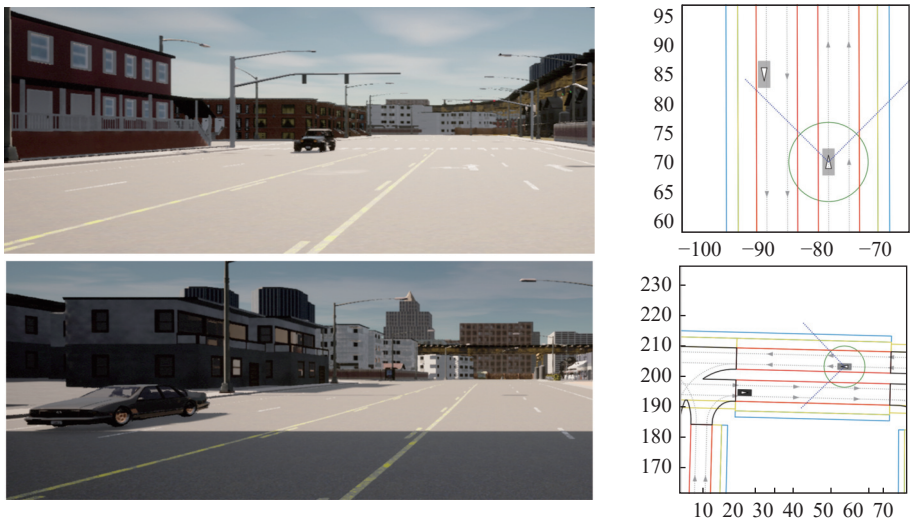


图 B4　示例 4

● Case05: Ego 位于双向 4 车道一侧靠近道路中心线的车道上, 其相对方向的外侧车道有一辆汽车 carA, ego 所处道路的前方为一个十字形交叉路口; 在交叉路口中, ego 所在道路的前方左转的道路上有一辆汽车 carB.

1. r1: Road, is2Way = True
2. g1: Group, laneNum = 2
3. g2: Group, laneNum = 2
4. g1.road = r1
5. g2.road = r1
6. g1.opposite = g2
7. ego_lane : Lane, index = 1

8. l2: Lane, index = 2
9. ego_lane.group = g1
10. l2.group = g2
11. j1: Junction, is4Way = True
12. l3: Lane, turn = LEFT
13. l3.junction = j1
14. ego_lane.succ = l3
15. ego on ego_lane
16. carA in ego.visibleRegion on l2
17. carB in ego.visibleRegion on l3

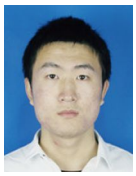
Case05 的 CARLA 模拟器中生成的场景见正文图 7.

附录 C

10 个模型训练配置相关信息, 表 C1 中 Lr schd 指本文训练模型所用的 Base config 中的学习率策略.

表 C1 模型训练配置

模型	Backbone	配置文件	Lr schd	max_epoch
Faster RCNN[23]	R-50-C4	faster_rcnn_r50_caffe_c4_1x_coco.py	2x	36
YOLOv3[27]	DarkNet-53	yolov3_d53_mstrain-608_273e_coco.py	273e	273
SSD[28]	VGG16	ssd300_coco.py	120e	36
RetinaNet	R-50-FPN	retinanet_r50_caffe_fpn_1x_coco.py	2x	36
FCOS	R-50	fcos_r50_caffe_fpn_gn-head_1x_coco.py	2x	36
ATSS	R-50	atss_r50_fpn_1x_coco.py	2x	36
YOLOF	R-50-C5	yolof_r50_c5_8x8_1x_coco.py	2x	36
AutoAssign	R-50	autoassign_r50_fpn_8x2_1x_coco.py	2x	36
CenterNet	ResNet-18	centernet_resnet18_dcnv2_140e_coco.py	2x	36
RetinaNet(Swin backbone)	Swin-T	retinanet_swin-t-p4-w7_fpn_1x_coco.py	2x	36



龚磊(1996－), 男, 博士生, CCF 学生会员, 主要研究领域为智能无人系统, 自动驾驶系统安全分析, 深度学习.

张燕咏(1975－), 女, 博士, 教授, CCF 专业会员, 主要研究领域为边缘计算, 人工智能物联网, 无人系统的感知.

孙新雨(1999－), 男, 硕士生, 主要研究领域为深度学习系统编译优化, 深度学习.

吉建民(1984－), 男, 博士, 副教授, CCF 专业会员, 主要研究领域为移动机器人, 深度强化学习.

张昱(1972－), 女, 博士, 教授, CCF 杰出会员, 主要研究领域为面向新兴领域的编程系统与优化, 智能无人系统, 量子计算.

华蓓(1966－), 女, 博士, 教授, CCF 高级会员, 主要研究领域为高性能网络系统, 算法与系统性能优化.