Closing the Gap between Sensor Inputs and Driving Properties: A Scene Graph Generator for CARLA

Trey Woodlief*, Felipe Toledo*, Sebastian Elbaum*, and Matthew B. Dwyer*
*Department of Computer Science
University of Virginia, 85 Engineer's Way, Charlottesville, Virginia 22904
Email: {adw8dm, ft8bn, selbaum, matthewbdwyer}@virginia.edu

***Abstract*—The software engineering community has increasingly taken up the task of assuring safety in autonomous driving systems by applying software engineering principles to create techniques to develop, validate, and verify these systems. However, developing and analyzing these techniques requires extensive sensor datasets and execution infrastructure with the relevant features and *known semantics* for the task at hand. While the community has invested substantial effort in gathering and cultivating large-scale datasets and developing simulation infrastructure with varying features, semantic understanding of this data has remained out of reach, relying on limited, manually-crafted datasets or bespoke simulation environments to ensure the desired semantics are met.**

To address this, we developed a plugin for the widely-used autonomous driving simulator CARLA called CARLASGG, that extracts relevant ground-truth spatial and semantic information from the simulator state at runtime in the form of *scene graphs*, enabling online and post-hoc automatic reasoning about the semantics of the scenario and associated sensor data. The tool has been successfully deployed in multiple previous software engineering approach evaluations which we describe to demonstrate the utility of the tool. The tool enables the client to adjust the precision of the semantic information captured in the scene graph to suit client application needs. We provide a detailed description of the tool's design, capabilities, and configurations, with additional documentation available accompanying the tool's online source: https://github.com/less-lab-uva/carla_scene_graphs.

***Index Terms*—autonomous systems, scene graphs, sensor abstractions, specifications**

I. INTRODUCTION

Software engineering (SE) for autonomous driving systems (ADS) is a burgeoning field of research as the community aims to develop rich and principled SE methodologies to improve the development [1], validation [2], and verification [3] of these systems. In order to develop and study these methodologies, researchers and practitioners must build custom datasets [4]–[6] or simulation environments [7]–[9] in order to obtain units of analysis *with known semantics* to serve as the basis of their evaluation. For example, validating an automated emergency braking system's (AEBS) ability to detect and avoid pedestrians requires sensor inputs with known semantics affirming the presence and location of a pedestrian; such semantic information is not present in raw sensor data.

Developing such artifacts comes at great expense, limiting their breadth and thus limiting the study's generality. Prior work has identified *scene graphs* (SGs), graphs that represent entities as nodes and spatial and semantic relationships as edges, as a rich abstraction over which to reason about the ADS-relevant semantics of a scene and accompanying sensor data [10]–[12]. To this end, we developed a first-of-its-kind tool, CARLASGG[1], a scene graph generator (SGG) plug-in for the widely-used CARLA ADS simulator [7] which generates SGs representing the simulator state, extracting the critical spatial and semantic information; this enables client applications to perform in-situ and post-hoc reasoning about the semantics of the scenario observed during simulation.

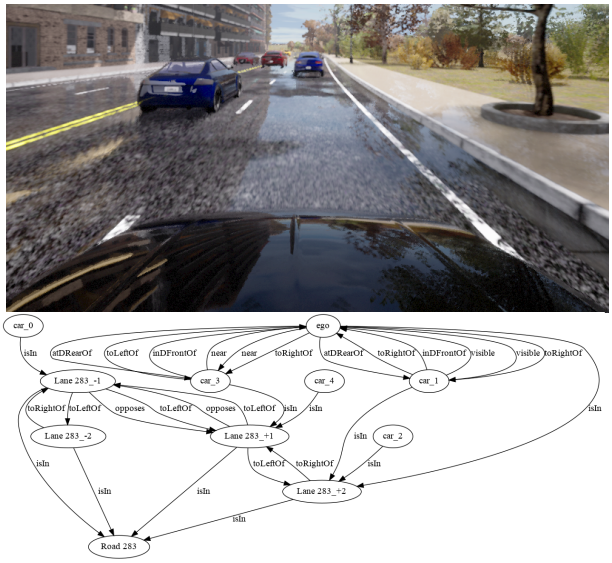

Fig. 1. An image from the CARLA simulator (top) with the corresponding scene graph (bottom) representing its high-level semantics automatically extracted by our tool, CARLASGG. Best viewed on a screen.

CARLASGG[2] has two primary usage modes: online *semantic monitoring* and offline *semantic data labeling*. In the online paradigm, the client application can use the semantic information captured by the SG to identify critical scenarios to invoke business logic for their technique and study. This allows for creation and evaluation of, e.g., runtime monitoring frameworks, online fault-localization techniques, and failure-prediction techniques. In the offline paradigm, the SGG can be paired with additional data collection to store sensor and/or

[1]Tool & video: github.com/less-lab-uva/carla_scene_graphs [Archive]

[2]This work was supported by the National Science Foundation through grants #2129824 and #2312487, the U.S. Army Research Office under grant W911NF-24-1-0089, and Lockheed Martin Advanced Technology Labs.

system trace information paired with the semantic information captured by the SG to form a rich dataset with known semantics. This enables generating sensor datasets that are searchable by their semantics with potential applications including, e.g., test selection, test adequacy metrics, and training and evaluating machine-learned components. One key insight is that the SG enables extracting semantic information at a level needed to reason about ADS requirements; Section V discusses the successful utilization of CARLASGG for a runtime monitor [12] in the online paradigm and a test adequacy tool [11] in the offline paradigm, with both uses focused on reasoning about ADS requirements.

II. Related Work

We review approaches to address the lack of ADS datasets with known semantics and provide background on SGs.

A. Other approaches to obtain semantically labeled data

Online semantic monitoring. Designing or identifying relevant scenarios and developing methods to evaluate the ADS against such scenarios poses a recurring challenge for researchers and practitioners. One of CARLA's strengths lies in its inclusion of specific scenarios, from general driving tasks [13] to stressful edge-cases [14]. These scenarios are ready-made units for analysis, but they are limited in scope, comprising only 47 distinct scenarios [14]. To develop arbitrary scenarios, recent work has proposed the SCENIC language that allows the end user to program a suite of scenarios to run in simulation [15]. Although these works provide the community simulations with known scenario-level semantics and enable reasoning about inter-scenario semantics, they lack more fine-grained information to reason about the scenario semantics at each frame. CARLASGG fills this gap, instrumenting the simulation to extract semantic information per-frame, enabling reasoning over intra-scenario semantics.

Offline semantic data labelling. Designing, training, validation, and verification of end-to-end ADS systems and individual ADS components often leverage datasets of ADS sensor data where each sensor input is paired with the ground-truth label for the relevant task. This has led to considerable effort to develop substantial and widely used datasets [4]–[6]. However, these datasets have three main limitations. First, they contain disparate sensor modalities and labeling based on their diverse goals, limiting the data available for novel tasks. Second, while expansive, they contain a finite amount of data, and obtaining additional data comes at substantial cost. Finally, extant labels generally exclude the type of rich, semantic information required for higher-order SE reasoning tasks, e.g., understanding whether a given input lies within the valid input domain of the system. CARLASGG fills this gap, enabling the collection of large-scale datasets with configurable sensors from within the CARLA simulator, pairing each sensor input with semantically-rich ground-truth labels.

B. Scene Graphs

A *scene graph* (SG) is a directed graph that encodes the semantic relationships between objects in a scene. Formally, $SG = (V, E : V \mapsto V, Ego \in V, kind : V \mapsto K, rel : E \mapsto R, att : V \cup E \mapsto M)$ where *V* is the set of nodes representing the entities in a scene, *E* is a set of directed edges, Ego is the distinguished ego node from whose perspective the SG is created, e.g. the ADS, *kind* is a function to access the entity type of a node, *rel* is a function to get the relation of an edge, and *att* is a function to retrieve the attribute values of a node or edge. A Scene Graph Generator (SGG) maps a set of sensor inputs, *I*, to an SG representation, $sgg : I \mapsto SG$. A wide variety of SGGs have been developed for different tasks, including visual question and answering, and scene understanding and reasoning [16]. A widely used benchmark for SGGs, Visual Genome [17], shows continued improvement across different techniques toward this task. Nonetheless, this benchmark, although having a wide variety of images and scenarios, lacks images describing driving scenarios. To mitigate this issue, Malawade et al. [10] presented ROADSCENE2VEC, an open source tool to extract scene graphs from images of roadway scenes for use in ADS contexts. However, as shown in previous studies [11], ROADSCENE2VEC exhibits two failure modes: 1) it uses a hard-coded road structure of 3 lanes, heedlessly allocating all entities in the left-portion of the image to the left lane, etc., which is not a sound representation in most cases; and 2) it encounters perception failures leading to inaccurate SGs. To fill this gap, CARLASGG extracts ground truth SGs using CARLA to both ensure the proper road structure is captured and that the SG entities are accurately recorded, allowing the development of downstream tasks.

III. Approach

Figure 2 illustrates the usage of our tool to generate SGs from the CARLA simulation. Throughout this section we will use *client* to refer to the program utilizing the CARLA plugin, CARLASGG, along with CARLA to collect data. The *client* instantiates CARLA with the relevant environment and sensor configurations, and controls the flow of the simulation by, e.g., operating the ego vehicle through the environment. At any point, the *client* can utilize CARLASGG to generate the scene graph representing the abstract semantics of the scene at that instant in time. Further discussion on tool setup, configuration, and performance are available in the repository. The plugin operates in three phases; we describe each phase below.

A. Pre-processing

For each simulation, CARLA first loads the static background environment, including the roadway, buildings, signage, and scenery. These environments can be vast, with the largest maps approaching 10,000 hectares. For efficiency, CARLASGG first performs a pre-processing pass over the environment, computing an initial SG containing the road structure and all static entities. The road structure is encoded as a high-definition map, with its resolution selected by the *client*; each node represents a location in the roadbed with edges describing the semantics of traffic flow at that location. The remaining static entities are extracted and their spatial and semantic relationships pre-computed to form the initial

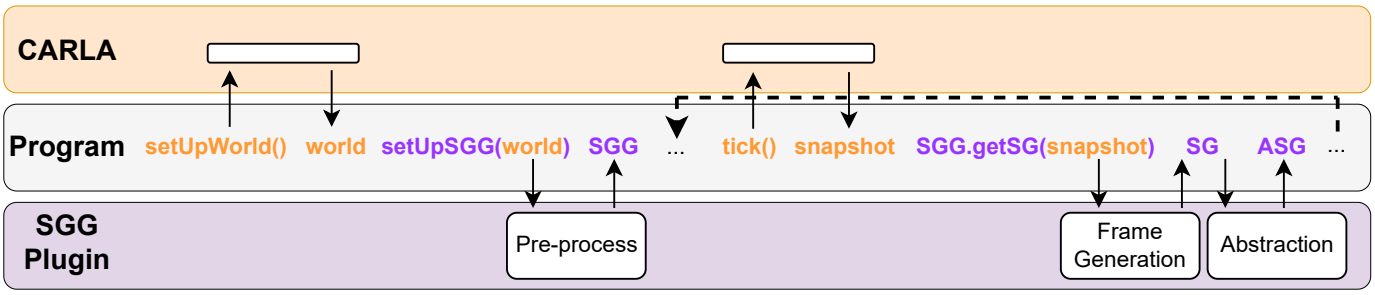

Fig. 2. CARLASGG workflow for generating SGs within the CARLA simulator.

SG. This cached static background SG will allow the tool to efficiently generate a localized SG at each frame.

B. SG Generation per Frame

At each frame, or at the desired frequency established by the *client*, CARLASGG can be invoked to generate the localized SG representing the semantics of the current environment ("world snapshot" in CARLA terms). Recall from Section II-B that a scene graph within the ADS context always contains a distinguished ego vertex identifying the vehicle from whose perspective the SG is generated.

When generating the SG, the cached static background SG is filtered to produce a subgraph retaining only the portion within some defined geometry, chosen by the *client*, of the ego vehicle. This subgraph is then enriched by adding all dynamic entities annotated with their attributes before computing the pairwise spatial and semantic relationships between all entities. The resultant SG contains all static and dynamic entities relevant to the ego vehicle's current situation.

C. Abstraction

The SG generated in the previous steps contains many fine-grained details that may not be relevant for the *client*'s particular use case. As illustrated in Figure 3, the SG on the left contains dozens of nodes representing the road structure (blue), along with two nodes representing the other entities in the roadway (magenta), and one node representing the ego vehicle (orange). While this may be suitable for tasks involving low-level motion-planning, this information may be superfluous for higher-order tasks. To this end, CARLASGG provides the functionality to *abstract* the SG to create an Abstracted Scene Graph (ASG) that lifts the semantic information contained in the original SG to a higher-level semantics. As shown in Figure 3, in the ASG on the right, CARLASGG has abstracted away the individual nodes representing the roadbed and replaced them with nodes representing the lanes and roads while preserving the semantic information that these lanes oppose each other in the flow of traffic. CARLASGG provides built-in abstractions to match the level of semantic information present in ROADSCENE2VEC [10] (shown in Figure 3), as well as higher-order abstractions to remove the lane and/or relationship information from the graph. Additionally, the abstraction functions are extensible, and client applications can design additional abstractions as needed.

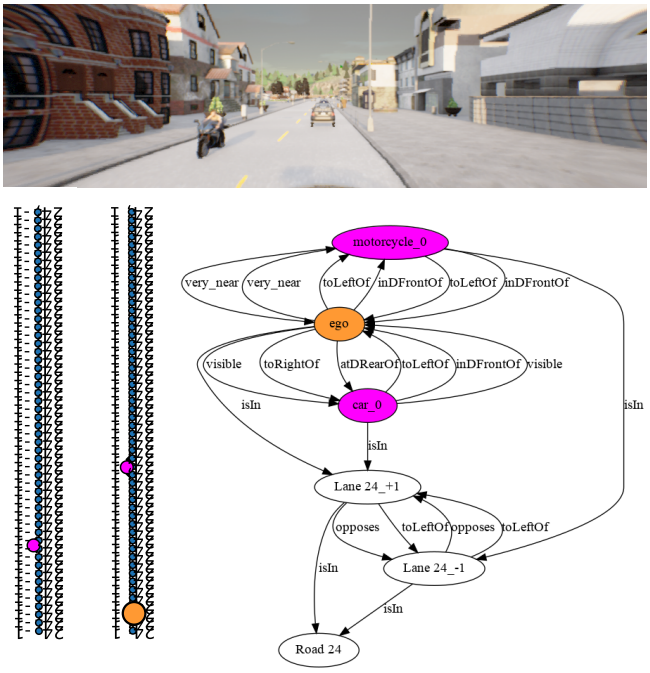

Fig. 3. A simulator image (top), its SG (left), and an abstracted SG (right).

TABLE I
COMMON PARAMETERS USED BY CARLASGG. FULL DISCUSSION ON PARAMETERS AND ABSTRACTIONS AVAILABLE ARE IN THE REPOSITORY

Parameter	Choices
Entity classes	Mapping from CARLA type; e.g., car, truck, van, bus, motorcycle
Entity special classes	Mapping from CARLA type; e.g., emergency vehicle
Traffic signal classes	Mapping from CARLA type; e.g., stop sign, traffic light, speed limit
Road Granularity	*None*, Lane, Waypoints
Distance Relationships	*None*, choice of thresholds
Angular Relationships	*None*, choice of thresholds
Entity Relationships	*None*, choice of entity pairs
SG Range	Geometry defining the range of the SG

IV. IMPLEMENTATION

CARLASGG is implemented in Python 3.7.10 and built to interface with CARLA version 0.9.14+ through its Python API with extensions to support older CARLA versions to 0.9.10 due to differing data availability. The tool is divided into

five files: `sgg.py` which contains the `SGG` singleton used to instantiate and utilize the tool as shown in Figure 2, particularly the constructor which invokes the pre-processing function, and the SG generation method; `sgg_abstractor.py` which contains helper functions for generating ASGs from a given SG; `actors.py` which contains mappings between the CARLA internal entity names and semantic labels to be utilized in the SG; `viz.py` which contains helper methods for visualizing the SGs, e.g., utilized to render Figure 3; and `utils.py` which contains helper functions. CARLASGG is highly-configurable; Table I provides a high-level overview of several parameters—an extended discussion on parameters, their defaults, and predefined abstractions is available online.

V. APPLICATIONS

We have successfully utilized our tool, CARLASGG, to implement and evaluate novel SE methodologies applied to ADS. Through these studies, we have refined the tool's infrastructure and developed a robust engineering pipeline to support its broad adoption by the community. We now discuss prior uses of CARLASGG, demonstrating its suitability for purpose.

A. Dataset Analysis for Test Coverage

In S^3C [11], CARLASGG was successfully used in the offline paradigm to generate SGs from CARLA paired with camera inputs, obtaining spatial semantics that allowed it to compute datasets' spatial coverage. By abstracting sensor inputs into interpretable scene representations, S^3C can assess how well a dataset covers different test specifications, and discriminate which inputs are more likely to cause failures. `sgg_abstractor.py` contains implementations for all abstractions evaluated by S^3C.

B. Runtime Monitoring for ADS Requirement Compliance

In SGSM [12], CARLASGG was successfully used in the online paradigm, generating SGs from CARLA at runtime while operating different ADSs, enabling the specification and monitoring of safe driving properties. The study demonstrates how developers can specify safety properties in a domain-specific language called SGL and monitor them at runtime using linear temporal logic over finite traces (LTL_f) automatically evaluated over the SGs. By bridging the gap between raw sensor data and high-level semantics using SGs extracted with CARLASGG, SGSM was able to evaluate the behavior of top performing ADSs from the CARLA leaderboard 1 [13] against safe driving properties. The results revealed that the ADSs violated 71% of the properties during at least one test, highlighting the important role of SGSM in ensuring compliance with critical driving standards and the SG abstraction as a higher-level interpretation from the sensor inputs.

VI. CONCLUSION

We present CARLASGG, a plugin for the CARLA ADS simulator that enables extracting spatial and semantic information from the simulator in the form of scene graphs. These scene graphs enable online or post-hoc reasoning about the semantics of the scenes encountered during simulation, including associating these semantics with gathered sensor data. This has myriad uses from dataset curation to monitoring for specific scenarios at runtime. We release the source for CARLASGG and its online documentation to support further adoption and usage by the community to continue advancements in software engineering for ADS.

REFERENCES

[1] J. Garcia, Y. Feng, J. Shen, S. Almanee, Y. Xia, and Q. A. Chen, "A comprehensive study of autonomous vehicle bugs," in *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, 2020, pp. 385–396.

[2] P. Koopman and M. Wagner, "Challenges in autonomous vehicle testing and validation," *SAE International Journal of Transportation Safety*, vol. 4, no. 1, pp. 15–24, 2016.

[3] N. Rajabli, F. Flammini, R. Nardone, and V. Vittorini, "Software verification and validation of safe autonomous cars: A systematic literature review," *IEEE Access*, vol. 9, pp. 4797–4819, 2020.

[4] P. Sun, H. Kretzschmar, X. Dotiwalla, A. Chouard, V. Patnaik, P. Tsui, J. Guo, Y. Zhou, Y. Chai, B. Caine, V. Vasudevan, W. Han, J. Ngiam, H. Zhao, A. Timofeev, S. Ettinger, M. Krivokon, A. Gao, A. Joshi, Y. Zhang, J. Shlens, Z. Chen, and D. Anguelov, "Scalability in perception for autonomous driving: Waymo open dataset," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[5] H. Caesar, V. Bankiti, A. H. Lang, S. Vora, V. E. Liong, Q. Xu, A. Krishnan, Y. Pan, G. Baldan, and O. Beijbom, "nuscenes: A multimodal dataset for autonomous driving," *arXiv preprint arXiv:1903.11027*, 2019.

[6] F. Yu, H. Chen, X. Wang, W. Xian, Y. Chen, F. Liu, V. Madhavan, and T. Darrell, "Bdd100k: A diverse driving dataset for heterogeneous multitask learning," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[7] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "CARLA: An open urban driving simulator," in *Proceedings of the 1st Annual Conference on Robot Learning*, 2017, pp. 1–16.

[8] P. Maul, M. Mueller, F. Enkler, E. Pigova, T. Fischer, and L. Stamatogiannakis, "Beamng.tech technical paper," https://beamng.tech/blog/2021-06-21-beamng-tech-whitepaper/bng_technical_paper.pdf, accessed: 2024-10-08.

[9] H. Abbas, M. O'Kelly, A. Rodionova, and R. Mangharam, "Safe at any speed: A simulation-based test harness for autonomous vehicles," in *Cyber Physical Systems. Design, Modeling, and Evaluation: 7th International Workshop, CyPhy 2017*. Springer, 2019, pp. 94–106.

[10] A. V. Malawade, S.-Y. Yu, B. Hsu, H. Kaeley, A. Karra, and M. A. Al Faruque, "roadscene2vec: A tool for extracting and embedding road scene-graphs," *Knowledge-Based Systems*, vol. 242, p. 108245, 2022.

[11] T. Woodlief, F. Toledo, S. Elbaum, and M. B. Dwyer, "S3c: Spatial semantic scene coverage for autonomous vehicles," in *2024 IEEE/ACM 46th International Conference on Software Engineering (ICSE '24)*. ACM, 2024.

[12] F. Toledo, T. Woodlief, S. Elbaum, and M. B. Dwyer, "Specifying and monitoring safe driving properties with scene graphs," in *2024 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2024.

[13] CarlaSimulator, "Carla leaderboard," https://leaderboard.carla.org/#leaderboard-10, accessed: 2024-07-19.

[14] ——, "Carla leaderboard 2.0," https://leaderboard.carla.org/, accessed: 2024-09-09.

[15] D. J. Fremont, T. Dreossi, S. Ghosh, X. Yue, A. L. Sangiovanni-Vincentelli, and S. A. Seshia, "Scenic: a language for scenario specification and scene generation," in *Proceedings of the 40th ACM SIGPLAN conference on programming language design and implementation*, 2019, pp. 63–78.

[16] X. Chang, P. Ren, P. Xu, Z. Li, X. Chen, and A. G. Hauptmann, "A comprehensive survey of scene graphs: Generation and application," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 1–1, 2021.

[17] R. Krishna, Y. Zhu, O. Groth, J. Johnson, K. Hata, J. Kravitz, S. Chen, Y. Kalantidis, L.-J. Li, D. A. Shamma, M. S. Bernstein, and F.-F. Li, "Visual genome: Connecting language and vision using crowdsourced dense image annotations," 2016. [Online]. Available: https://arxiv.org/abs/1602.07332