ARTICLE INFO
Article ID: 12-06-01-0007

doi:10.4271/12-06-01-0007

Procedural Generation of High-Definition Road Networks for Autonomous Vehicle Testing and Traffic Simulations

Golam Md Muktadir,[1] Abdul Jawad,[2] Ishaan Paranjape,[2] Jim Whitehead,[2] and Aleksey Shepelev[3]

[1]University of California Santa Cruz, Computer Science and Engineering Department, USA
[2]University of California Santa Cruz, Computational Media Department, USA
[3]Ford Motor Company, USA

Abstract

Effective simulation-based testing of autonomous vehicles requires the exploration of vehicle performance against a wide variety of rare and unusual road and intersection geometries. We present a high-definition road and intersection generator called JunctionArt. JunctionArt takes as input a series of control lines and generates a series of roads and intersections which conforms to them. Roads exhibit different types of lane types such as turn lanes, one-way streets, and multiple lanes, while intersections feature a range of incident roads (three to seven incident roads), leading to a variety of geometries and interior connecting lanes. These roads are output in the OpenDRIVE format and, hence, are interoperable with a wide range of tools and simulation environments. Multiple metrics are computed over generated roads—field of view (FOV), maximum turn curvature (maxCurvature), corner deviation angle (cornerDeviation), complexity, conflictArea, and the number of interior connection lanes—and are used to perform an expressive range analysis. This analysis finds that JunctionArt is capable of creating rare and unusual intersection situations, i.e., representatives of the long tail of infrequent road configurations. Interoperability with third-party simulation tools and environments RoadRunner, Carla, and esmini is demonstrated.

History

Received: 21 Aug 2021
Revised: 20 Nov 2021
Accepted: 27 Apr 2022
e-Available: 10 May 2022

Keywords

Autonomous vehicle simulation, PCG, Traffic simulation, Road network generation

Citation

Muktadir, G., Jawad, A., Paranjape, I., Whitehead, J. et al., "Procedural Generation of High-Definition Road Networks for Autonomous Vehicle Testing and Traffic Simulations," *SAE Int. J. of CAV* 6(1):99–120, 2023, doi:10.4271/12-06-01-0007.

ISSN: 2574-0741
e-ISSN: 2574-075X

This article is part of a Special Issue on Emerging Simulation Tools and Technologies for Testing and Evaluating Connected and Automated Vehicles: Part 2.

Introduction

In California's majestic Yosemite Valley, there is a granite boulder by the side of the road. It is an ordinary boulder in every respect save one: Teslas really like this one. Indeed, while on autopilot five Tesla cars have collided with this particular boulder, with the most recent accident taking place in July 2021 (see [1]). What makes this boulder so attractive? Tesla dashcam footage tells the tale. The road leading into Yosemite is a one-way road, two-lanes wide. As it enters the valley, it splits in two, one side curving off to the left, the other continuing straight. The Tesla autopilot software fails to detect the lane split until it is too late. The driver recalls the event: "Hands on wheel, eyes on road, vehicle just wanted to keep going straight, I took control, entered gravel and smashed into a boulder." Our popular boulder sits exactly where a Tesla continuing straight will hit it.

One lesson we take away from this accident is the importance of testing autonomous vehicle software in a variety of different road configurations. Out in the world, there is tremendous variety in the shapes and configurations of roads and intersections. While many of these configurations occur rarely, the Yosemite boulder demonstrates that this long tail of rare and unusual road configurations does matter, and has real consequences.

To be considered safe, autonomous vehicle software must be rigorously tested in both common and rare road situations. Multiple methods are used to perform this testing, including simulation-based testing, X-in-the-loop, real-world driving tests, and traditional software testing [2]. While real-world testing has the best environmental authenticity, it also has many drawbacks, including cost, difficulty of reproducing dangerous scenarios, and risks to other vehicles and pedestrians. Due to the need for a vehicle service and repair facility, real-world testing tends to be centered on a small number of geographic regions, limiting the overall variety of road geometries encountered. For example, Waymo, a well-funded autonomous vehicle company which is a subsidiary of Alphabet, has performed real-world testing in around two dozen cities, mostly located in California.

Simulation-based testing overcomes the drawbacks of real-world testing. Simulations can create dangerous scenarios at scale and reproduce them on demand, at a low cost. Simulation environments can create a wide range of road geometries and, hence, are capable of exploring the long tail of unusual road and intersection situations. The primary drawback is the gap between the simulated world and reality; hence, a combination of real-world and simulation testing works best. Waymo operates two simulation testing environments, called CarCraft and Simulation City, which supplement its real-world testing [3]. While Waymo has logged 20 million miles in real-world testing, in CarCraft alone it has driven over 7 *billion* miles [4].

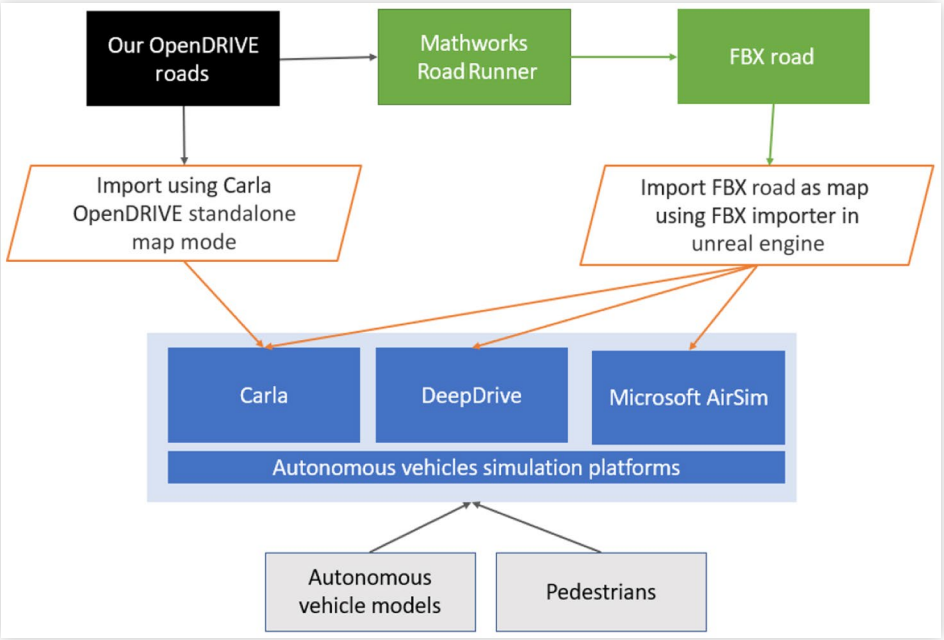
There are several sources for the road networks found in simulation environments. Some road networks are manually authored using specialized tools such as RoadRunner [5]. Hand authoring is a time-consuming activity which limits the number and range of road networks used for testing. Virtual road networks can also be obtained by importing real-world road networks from OpenStreetMap or other sources of map information. However, these formats usually lack the lane connectivity information necessary for most simulation scenarios and require hand authoring to fix dangling roads at the edges of the sample area. In contrast, procedural roads can create road networks, which are completely connected, and exhibit a range of situations which occur infrequently in the real world. They can also simulate road configurations found in different parts of the world. Ideally, road networks can be generated to focus on situations that autonomous vehicle software finds challenging, and hence fully exercise problematic aspects of the software. In the present work, we are motivated by the goal of procedurally generating large numbers of high-quality roads with diverse characteristics suitable for use in simulation software. Procedural roads should capture as much of the long tail of road and intersection configurations as possible, allowing testers to detect problematic situations like the Yosemite boulder in simulation.

We present a high-definition road and intersection generator called *JunctionArt*. JunctionArt takes as input a series of control lines and generates a series of roads and intersections which conforms to them. Roads exhibit different types of lane types such as turn lanes, one-way streets, and multiple lanes, while intersections feature a range of *incident* roads (three to seven incident roads), leading to a variety of geometries and interior connecting lanes. These roads are output in the OpenDRIVE format and, hence, are interoperable with a wide range of tools and simulation environments. For example, roads generated in the OpenDRIVE format can be imported by software such as MathWorks RoadRunner [5] for building complex road environments or Carla [6] for running simulations and training autonomous vehicles. The overall pipeline of using our generator roads in different autonomous vehicle simulations and testings is shown in Figure 1.

To evaluate these roads, we provide an *expressive range analysis* of these roads. This is accomplished by computing multiple metrics—field of view (FOV), maximum turn curvature (maxCurvature), corner deviation angle (cornerDeviation), and collision zone area—to create frequency distributions and heatmaps, which then characterize the range of generated road intersections. We also demonstrate interoperability by providing examples of roads being imported into different simulation environments.

In the remainder of the article, we provide an overview of related work before diving into a description of the JunctionArt generation algorithm (road network and then intersections). We next evaluate these generated roads via our expressive range analysis. Interoperability examples end the evaluation. A conclusion section summarizes the results.

FIGURE 1 Road networks described in OpenDRIVE can be imported into road editing tools such as MathWorks RoadRunner. RoadRunner can automatically add surface textures, and testers can edit roads further in the tool. The edited roads can be exported as FBX files, which can be imported as maps into state-of-the-art simulation tools for autonomous vehicles such as Carla, DeepDrive [7], and Microsoft AirSim [8]. Carla also has an option to import OpenDRIVE roads directly.



Related Research

Road Network Generation

Road network generation using procedural methods has primarily been used to populate virtual game worlds and is mostly discussed in the context of procedural cities [9]. Roads for procedural cities can be generated using various methods, including pattern-based approaches [10], L-systems [11], agent simulation [12], and tensor field [13]. In [9], the authors provide a broad discussion of terrain-based road network generation methods, categorizing them into user-assisted and path-finding-based methods. Road three-dimensional (3D) splines are fitted between sketch strokes or control points in user-assisted methods to minimize local elevation changes [14]. Pathfinding-based methods, such as A*, encode the desire to maintain constant elevation and curvature in a heuristic form [15]. In [16], the authors discuss road network generation in the context of traffic infrastructure creation. The survey includes [17], where network generation starts with creating a set of intersections, and [18], where road network generation is broken down into a few recurring grids or radials. Recent road network generation work employs machine learning methods, such as the use of Generative Adversarial Network variations to create road networks [19, 20, 21].

High-Definition Road Generation

High-definition road generation falls into two broad camps: (a) generating a road network on the fly by sequentially choosing and joining random segments and (b) taking a real-world road network layout as input and then creating segments along the reference lines. In [22], [23], and [18], roads are constructed by joining predefined segments. The segments are small configurable pieces of roads such as straight and curved roads, T-junctions, intersections, and roundabouts.

On-the-fly generation: PGDrive starts road generation with a set of predefined segments for which some parameters can be varied. It randomly chooses the next segment and adds to the current road [23]. Pyodrx is a Python library used for the procedural generation of OpenDRIVE roads [24]. It provides functions for generating elementary geometric shapes which can be used to create road segments. It also has some basic blocks such as three- or four-way intersections, as well as merge or split lanes. However, it lacks the ability to create entire road networks, or create diverse intersection geometries. AsFault uses a genetic algorithm to evolve roads with increasing numbers of sharp turns to as to force autonomous vehicles to make mistakes [22]. In [25], roads can be described in a simple language and the method can generate a detailed road in OpenDRIVE format given the reference

lines, coupling points, segment definitions, segment linkage, and connection roads of a road network.

In general, these algorithms lack representative variation and diversity in the segments they produce. For example, a curved connection road inside an intersection has two core properties: curvature (inverse of radius) and curve angle (which determines the length). None of the algorithms vary both properties to create realistic intersections. Furthermore, multilane features, frequently occurring in the real world, are either absent or limited.

Based on real-world data: There is a large body of work on generating digital roads based on real-world data (geographic information system (GIS), images, etc.). StreetGen [26] takes GIS data for network reference lines and rough road width estimates as inputs. While StreetGen can generate simple roads and intersections, it does not ensure smooth transitions between different geometric shapes of roads, connection roads inside an intersection, different lane types, etc. Generation of complex multilane intersections from GIS and global positioning system (GPS) navigation databases is addressed in [27]. The method uses GPS navigation data to determine lane continuity and avoid invalid turns within intersections. Wang et al. generated 3D scenes automatically using publicly available GIS data which are capable of creating simulation-ready roads and intersections [28]. Their method can build valid connection roads inside intersections. However, their generation process removes the geometric detail of the intersections and abstracts it with a simple convex polygon. Zhang et al. detect intersections from crowdsourced vehicle trajectory data [29]. All of these approaches are limited by the availability of representative real-world data which may become inconsistent with continuously evolving roads. Furthermore, human intervention is often necessary to correctly capture all of the real-world detail in generated roads.

JunctionArt: Road Network Generation

In this section, we present JunctionArt, our approach for generating high-definition road networks capable of representing real-world roads with a centimeter-level granularity of detail. These road networks are intended to be interoperable with a wide range of simulation environments, which are used to develop and validate autonomous driving features and simulate traffic flows. We achieve this interoperability by outputting roads in the OpenDRIVE format. We briefly describe OpenDRIVE below since JunctionArt directly uses its data model when generating road networks, such as by generating reference lines and then generating left and right lanes out from these reference lines.

OpenDRIVE is an extensible markup language (XML) syntax for describing road networks, standardized by the Association for Standardization of Automation and Measuring Systems (ASAM) [30]. OpenDRIVE stores data to represent a road network, including its geometry, lane information (including lane markings), and road objects (such as signs and traffic lights). Reference lines are a key abstraction in OpenDRIVE and represent an abstract road segment. Reference lines are the bare structure of the network, and other information such as roads, lanes, elevation profiles, road sign positions, and driving directions are attached to the reference line. Different road courses can be modeled using straight lines, spiral/clothoids, arcs, cubic polynomials, and parametric cubic polynomials. Users can define various attributes of these geometric elements (curvature, length) to model all kinds of complex road geometry that occur in the real world. For instance, eight coefficients are required for defining a parametric cubic polynomial.

Each road has one mandatory reference line that usually goes through the center of the road but may have a lateral offset. Lanes of a road are grouped into left, right, and center lanes. Roads contain lane information, road elevation, road types (motorways, rural roads), speed regulation, etc. Roads are usually connected via predecessor and successor roads, which allow applications to navigate through the road network. Places where three or more roads meet are called junctions. Roads leading to a junction/intersection are named incoming roads, and roads inside are called connection lanes (roads). Junctions contain lanelink data which provides information about how lanes are connected between incident roads and connecting lanes (roads).

Network Generation

JunctionArt is a procedural road network generator which takes as input a series of control lines and lane configuration parameters. Once provided with input, it operates independently without further interaction. A *control line* is a line segment which indicates a location where the final road network must have a road. Control lines are used to create the "backbone" of roads in a network, such as a series of major roads, or roads defining one direction of a Manhattan-style grid. *Connecting roads* are created between control lines, at *control points*.

The operation of JunctionArt can be viewed as a pipeline consisting of four major stages. Stage 1 defines the road network as a graph, but lacks lane information. Stage 2 then defines the number of lanes for each connecting road. In Stage 3, an intersection is created at each control point, including within-intersection connecting lanes. Creating the interior connecting lanes involves resolving a series of within-intersection constraints, which may lead to changes in the number of lanes on connecting roads. In Stage 4, any lane number mismatches are resolved, and lane markings are added.

Stage 1: Creating the Base Network

When JunctionArt begins generation, it starts with an empty space. How should it be filled? Control lines provide the answer: lay down the control lines first and then generate other

roads between them. Control lines can either be provided as input by a human designer or created automatically by JunctionArt. When provided by a human designer, the control lines allow them to specify the main roads within a network, thereby providing a high degree of control over the location of these roads and the overall shape of the road network.

For automatic generation, a designer may not require this level of control. In the automatic case, JunctionArt defines an initial control line along the x-axis, and then each subsequent line is determined by randomly generating offsets which are applied to each endpoint of the previous line. The process stops after a specified number of control lines have been generated. Automatic generation occurs within a defined spatial area (typically 2 km × 2 km in the experimental runs in this article). There is no theoretical limit on the size of the network. JunctionArt can create a 5 km × 5 km map in a few minutes which would take weeks with human effort.

Control lines are grouped into pairs, and connecting roads are only created between the lines in a pair. A single control line can participate in multiple pairs. Figure 2(a) shows four control lines which are combined into three control line pairs. In addition, control lines can be sequentially connected at endpoints, allowing a control line sequence to have a complex shape. It is also possible to cluster control line pairs into groups (called *control line groups*), which are used to make distinct subgraphs in the overall road network. Figure 3 shows three control line groups and the resulting road network.

Creating control points and connecting roads. To make connections between control line pairs, specific locations must be selected first, called *control points*. Control points are useful for overall road network consistency since they ensure a connecting road coming into a connecting point from one side will join up with a connecting road leaving from the other side [see Figure 2(b)].

The locations for control points are selected using a Markov Decision Process (MDP). In real-world city maps, a road has other roads parallel to it with a very high probability, and minor roads often tend to be orthogonal to the main road. We model this phenomenon using a simple MDP. The MDP probabilistically determines the shape of the next section of the road network (parallel, orthogonal, extension from one of the current control points, random) based on the last shape generated. In the MDP, the current state is the last generated shape between the control lines. Once a new state has been selected, additional control points are placed as needed (typically some points are new, and others come from the previously placed shape), and then connections are made to complete the outline of the shape.

This process has three design parameters: *snapDistance*, *minDistance*, and *maxDistance*. If two control points on a line are closer than snapDistance, they are merged together, preventing intersections from appearing too near to each other. minDistance and maxDistance constrain the gap between adjacent control points on a control line and are used to create more sparse or dense networks.

After the MDP has completed its processing, there are still two remaining steps:

- Create connections between adjacent control points which lie on the same control line.
- Connect some control lines from different control line groups with control points at their ends only. This connects small chunks of the network to make a fully connected one.

At the end of Stage 1, the road network contains a set of control points connected via connections, as shown in Figure 2(b). Connections will map directly to reference lines in the final OpenDRIVE representation.

FIGURE 2 Three pairs of control lines are used to control the road network generation: (red, blue), (blue, green), (green, orange). For each pair of lines, a set of control points is created (black dots) and connected.

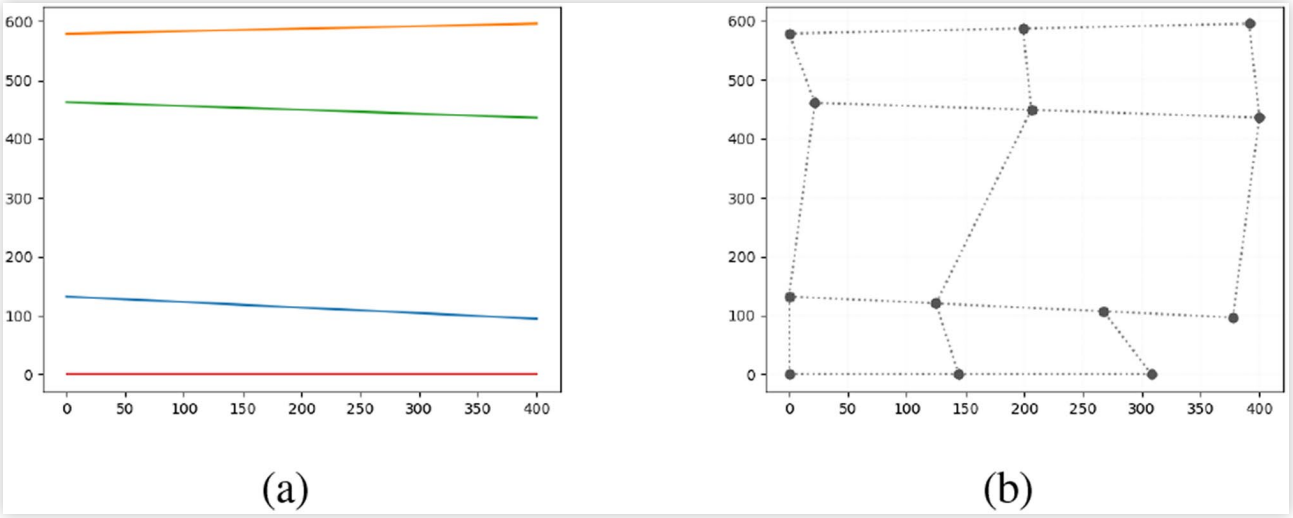
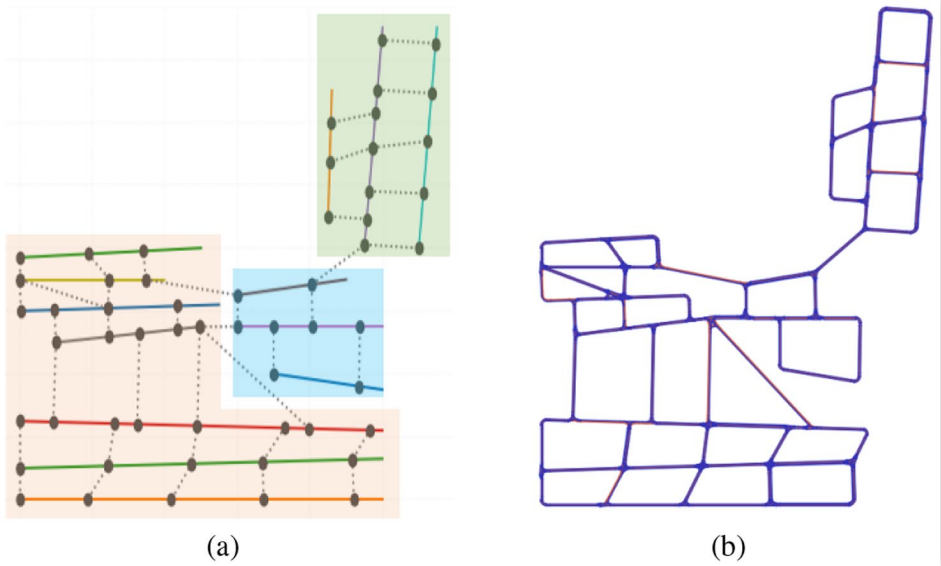


FIGURE 3 The series of control lines groups (left, represented by different colors) then result in different subgraphs of the overall road network (right).



Stage 2: Creating Lane Configurations for Connections

In this stage, for each connection, JunctionArt determines the number of lanes in each direction. There are four design parameters, configurable by the user, that determine the number of lanes in connections. *randomizeLanes* specifies whether there is a random number of lanes in the road network. If it is *False*, all the roads are generated with one left and one right lane, and if *True*, each road may have zero to three lanes on each side, but never zero lanes on both sides. *nLaneDistributionOnASide* is a vector containing the probability of selecting 0, 1, 2, or 3 lanes on a side (left or right), creating a weighted random distribution. When we determine the number of left lanes and right lanes for a connection between two control points, we also get the number of incoming lanes and outgoing lanes for each control point.

It can be useful to specify that a major road should pass through a road network. To support this, the third design parameter, *controlineLaneConfigurations*, specifies the exact number of lanes for roads that are going to be placed along a given control line. This value can vary across control lines. When this value is not set for a control line, the fourth parameter, *nLaneDistributionOnControlLines*, provides a vector of the probability of occurrence of 0, 1, 2, or 3 lanes per side (similar to *nLaneDistributionOnASide*). Since control lines typically are not single-lane roads, it is useful to have a different frequency distribution for them.

At the end of the second phase, the road network consists of a set of lane parameters defined in each direction along connections, as shown in Figure 4(b). No intersections have been defined, and the exact geometry of the lanes is still to be determined. As we do not consider the lane geometries at this step, the logical lane configurations may not have valid geometric shapes due to sharp turns. We validate the geometric shape in the next stage, and in case of an invalid configuration, we discard the whole map. However, this check can also be done at the lane configuration phase, and a valid lane configuration can be achieved (by using a constraint solver) given the same set of control points and connections. We leave this improvement opportunity for future work.

Stage 3: Intersection Generation

This stage generates intersections. In Stage 1, we generated a connection graph for the entire road network. Now in Stage 3, we convert control points to intersections and edges to roads. This is a two-step process. First, for each incident road, we determine a point (called an *incident point*) that is near the control point and thereby near the future center of the intersection [see Figure 4(a)]. This becomes the location where the incident road ends and the intersection begins. The location of the incident point needs to allow sufficient space for all of the lanes in the intersection (large numbers of lanes can lead to overlaps). In the event of an overlap, the incident point is moved outwards by up to 50 meters, and overlaps are checked again.

In the second step, we generate one intersection per control point using the incident points (above) and precomputed number of lanes (from Stage 2).

Now that the incident points have been determined, we know where a connection (incident road) ends and the intersection begins. It is now possible to convert connections into actual roads since their exact start and endpoints are known. This information, in combination with the number of left/right lanes, median type, and heading for each incident

road (from Stage 2), permits the creation of a *road definition*. A road definition is an intermediate data structure for all incident roads (Figure 5).

Next we focus on the interior of the intersection and generate the set of within-intersection connection lanes. This is accomplished by defining the centerlines of the connection lanes using parametric cubic curves (parampolys). These centerlines are then converted into lane descriptions, giving them full geometry. The complete set of connection lanes is used to create the OpenDRIVE junction specification. After creating this junction definition, the successor/predecessor relationships between incoming roads and connecting lanes are created.

When creating the connecting lanes, a series of intersection constraints must be satisfied, including the following. Every incoming lane must lead to at least one outgoing lane; otherwise, the intersection will have inconsistent lane connections (e.g., a dead-end road at the beginning of an intersection). Also, lane merging/crossing inside an intersection is avoided due to safety issues. Satisfying this constraint requires another one: the number of outgoing lanes must be greater than or equal to the number of incoming lanes. To satisfy these constraints, determine which lanes to interconnect, and determine how to connect the extra outgoing lanes, we pick one of the following heuristics as a kind of domain-specific constraint solver:

- Split-first strategy: Only the first incoming lane of the incident roads (near the center) is split to connect the extra outgoing lanes.
- Split-last strategy: Only the last incoming lane of the incident roads (near the edge) is split.
- Random Split strategy: Uses either the split-first or the split-last strategy chosen randomly for each incident road.

These strategies, accompanied by a simple algorithm to connect lanes in clockwise order, avoid merging/crossing inside an intersection.

At the end of Stage 3, for each control point, we have a series of incident points, road definitions for each of the connections (incident roads), and a complete set of interior lane connections (junction) inside the intersection.

Stage 4: Cleanup and Lane Markings

To satisfy all of the within-intersection lane constraints (Stage 3), it may be necessary to remove some outgoing lanes or add a new incoming lane. Because of this, it is possible for the number of left/right lanes to be different from one end of a connection to another [see Figure 6(a)], requiring a repair action to ensure consistency of the road. To repair this mismatch between the number of incoming and outgoing lanes in a connection, we split or merge lanes [Figure 6(b)].

As a final step, lane markings are added to all of the roads and intersections. Unnecessary lane markings inside intersections are removed. Solid yellow lines are placed along the centerline of the roads, solid lane markings are placed on the incident roads near intersections.

Road network generation is now complete, and we output the resulting network into an OpenDRIVE .xodr file.

Evaluation

When evaluating the roads generated by JunctionArt, we wish to answer the following broad questions:

- *What is an intuitive sense of the kind of variation created in road networks and intersections?* We address this by providing a selection of examples of the generated output.
- *How well does the generator do at creating challenging road and intersection configurations for autonomous vehicles?* We examine this by computing a variety of metrics and then characterize the frequency distribution of these metrics individually, and in combination. This exploration of frequency distributions of generated artifacts is known as *expressive range analysis*.

FIGURE 4 A control point, in the center, with chosen incident points in orange. Blue/green are incoming/outgoing lanes for each incident point.

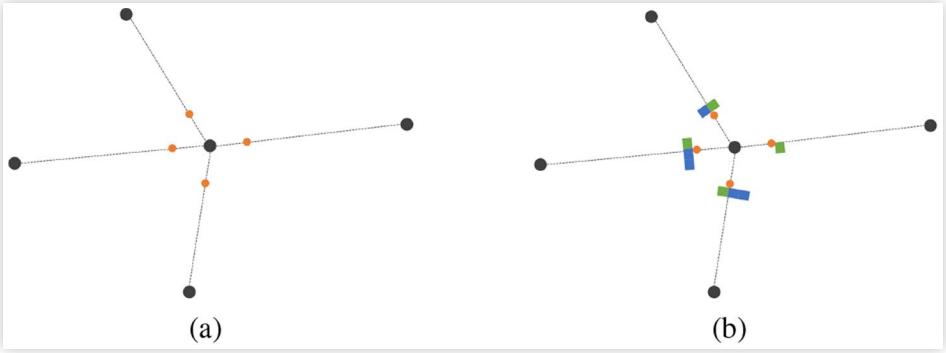
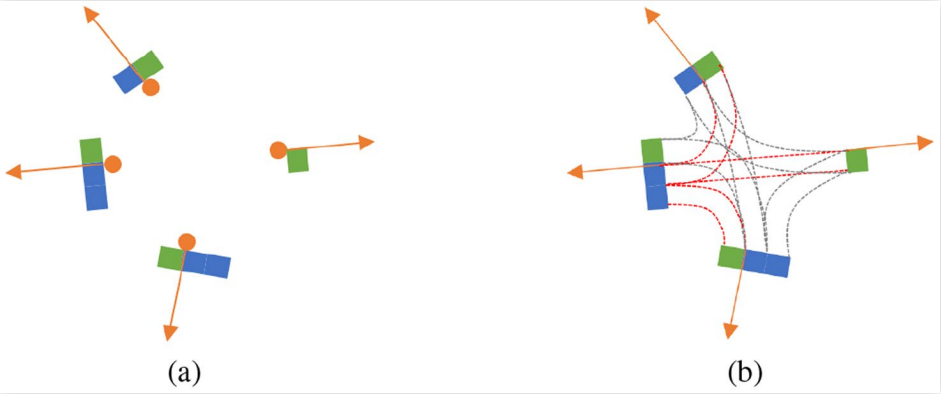


FIGURE 5 The second part of Stage 3 takes a set of incident points and the corresponding number of left and right lanes (a) and creates the corresponding intersection (b). The red lines in (b) show the connection lanes extending from the incoming lanes of the first road.



- *Are generated road networks actually usable in other tools (interoperability)?* We demonstrate the same generated road network in multiple simulation tools.
- *How long does it take to generate road networks, and how does this vary with the size and number of lanes? We present performance data for a representative sample of road network types.*

We explore these questions in the sections below.

Variation in Road Networks and Intersections

When examining a procedural generator for the first time, a natural first question is "what does the output look like?" We begin by examining the kinds of road networks which can be generated.

Variation in road networks. The input to JunctionArt is a series of control lines. The same set of control lines can result in many possible output road networks because of the randomness involved in the MDP used to create connections between control line pairs. The resulting variety can appear in the shape and number of intersections, the number of lanes, curved roads, the area covered by roads, positioning and heading of roads, etc.

Figure 7(a) provides an example of a series of *manually* created control lines which are in the shape of the letter A (an example of the kind of authorial control afforded by control lines). Figures 7(b) to 7(d) give three examples of road networks generated from the control lines. Notice that Figure 7(d) has an oddly shaped four-way intersection in the left-bottom corner of the road network, an example of the challenging situations JunctionArt can generate. To improve variety, the generator can also create curvy roads, as is shown in Figures 7(c), 8(a), and 8(b).

It is also possible for JunctionArt to *automatically* generate a set of control lines. In Figure 8, we show four road networks created from four different sets of automatically generated control lines. Notice there is a variety of road network shapes containing many different forms of intersections.

Variation in intersections. Existing procedural road generation systems generally tend to produce a limited set of intersection types and geometries, with limited numbers of lanes. Since intersections in JunctionArt emerge from the

FIGURE 6 Connecting intersections. Black circles show the control points of five intersections. Lane numbers can vary between the ends of a connection due to satisfying intersection constraints. Lane splits or merges are added to preserve consistency.

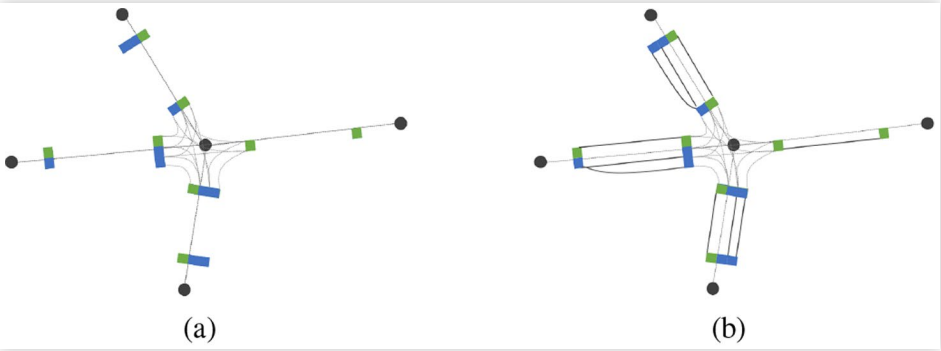
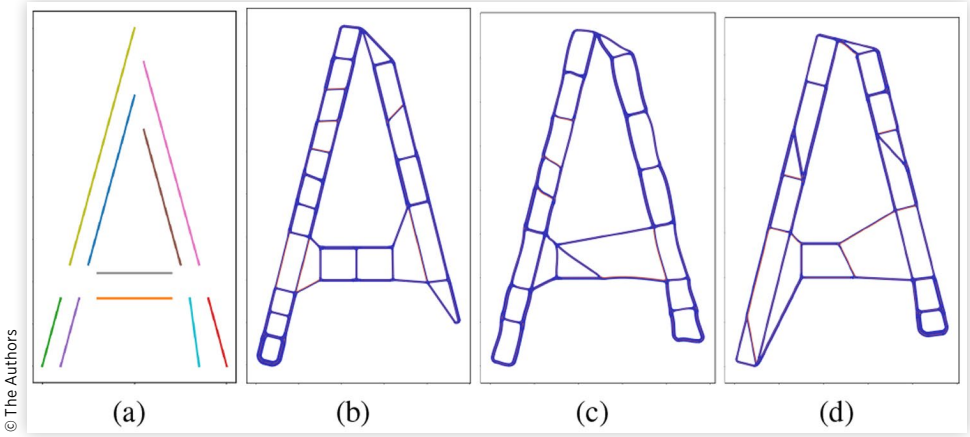
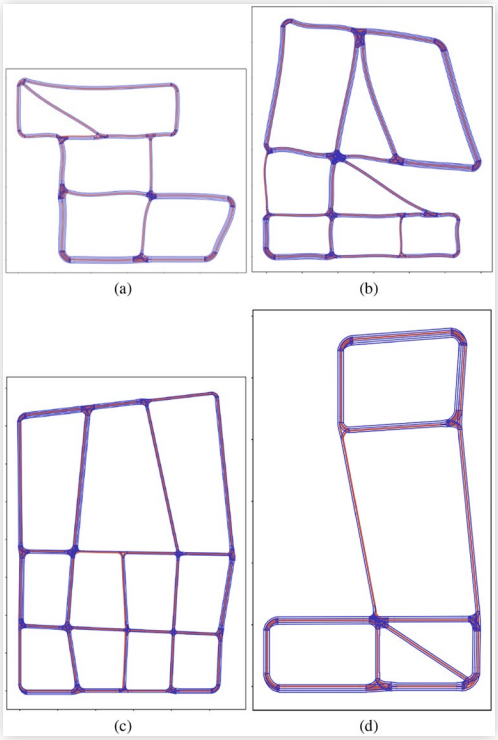


FIGURE 7 Three different detailed road networks generated from the set of control lines in (a). Thinner line segments represent fewer lanes than thicker ones.



spatial orientation of the various connecting roads, this leads to a large variation in intersections. JunctionArt creates three- to seven-way intersections (typically three- to five-way), with random skews and offsets between the incident roads. This ability for the incident roads to appear in multiple locations relative to the intersection centerpoint is what makes it possible to have such a wide range of intersection geometries. Figure 9 shows a variety of three-way intersections, including a variation in geometry (both T and Y configurations) and lane numbers. Figure 10 shows four-way intersections, with examples of cross-shaped intersections with random skew, as well as K-shaped intersections. Figure 11 shows variations in five-way intersections, including a variety of different incident lane angles.

FIGURE 8 Examples of road networks where the control lines were automatically generated. Thinner line segments have fewer lanes than thicker ones.



Expressive Range Analysis

In the Introduction, we argue that it is important to test autonomous vehicles against the long tail of unusual road configurations. This raises several questions. What kinds of unusual situations are useful for testing? How frequently does a generator create these situations? What kinds of trade-offs exist when trying to achieve multiple unusual situations simultaneously?

We answer these questions by adopting a style of analysis first introduced to examine procedural generation tools for computer game levels [31]. Expressive range analysis begins by defining a series of scalar metrics which can be computed on a generated artifact, converting a complex artifact into a single value which characterizes it with respect to some quality of interest. Once an artifact can be represented with a single value, it is then possible to compute a frequency distribution of these values across a large number of generation runs. This frequency distribution then describes the range and probability of various metric values output by the generator. In this sense, these frequency distributions characterize the generator itself, describing how its generation rules create, emergently, the resulting frequency distribution of metric values. The frequency distributions describe the *expressive range* of the generator in that the generator can express (create) the range of values seen.

Expressive range analysis is also concerned with the trade-offs between achieving certain metric outputs. Maximizing one metric may make it difficult to also maximize another. These trade-offs are explored via heatmap diagrams which show the frequency of co-occurrence of two different metrics across a large number of generation runs, creating a two-dimensional frequency distribution.

FIGURE 9 Examples of three-way intersections. (a)-(c) have two-lane incident roads, the remainder has a random number of lanes. (f) shows two incident roads having five lanes each. Light green lanes are within-intersection connecting lanes.

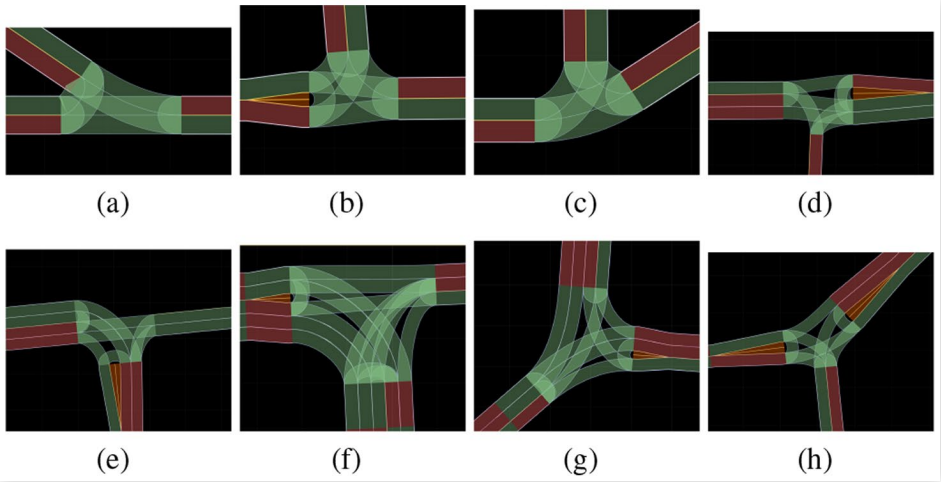


FIGURE 10 Examples of four-way intersections. Light green lanes are within-intersection connecting lanes.

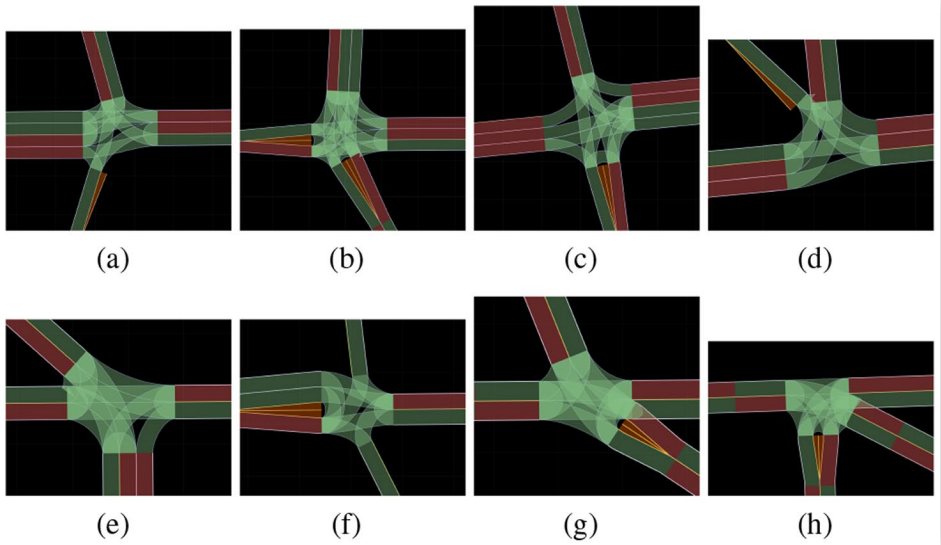
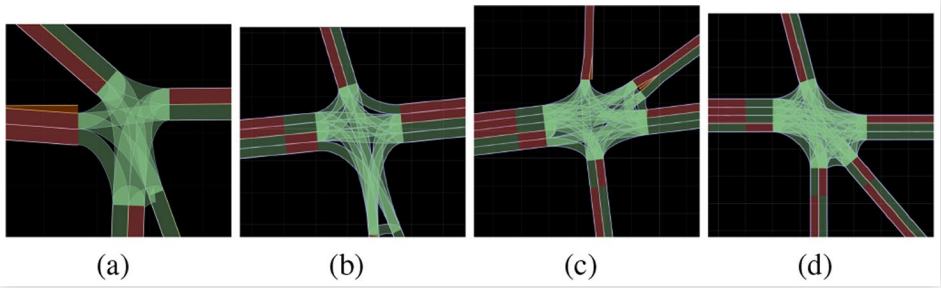


FIGURE 11 Examples of five-way intersections. Light green lanes are within-intersection connecting lanes.



Since a large proportion of accidents occur at intersections (e.g., 36 percent [787,236] of the estimated 2,188,969 NMVCSS crashes were intersection-related crashes [32]), we focus our expressive range analysis on metrics computed over intersections, or incident roads leading into intersections. As demonstrated by the Yosemite boulder crashes, our core assumption is that certain road configurations are more challenging to navigate than others, even in the absence of other vehicles. Hence, the goal of our metrics is to identify and characterize these challenging configurations.

Specifically, we aim to quantitatively characterize the form, trajectory space, and navigational and perceptional complexities of intersections in generated road networks through a set of metrics. Metrics such as FOV, maxCurvature, cornerDeviation, and complexity are used to characterize the incident roads (legs) of intersections. These try to capture the experience of a single vehicle as it enters or exits an intersection. Intersection metrics such as area, conflictArea, conflictRatio, and nConnectionLanes characterize the shape of the intersection and the interplay of all the traffic participants traveling through the intersection.

Experimental setup: In the remainder of this section, when we report experimental results in the form of frequency distributions or heatmaps, they are from generation runs with the following parameters. Our experimental setup kept generating 2 km × 2 km maps until we collected 10,000 intersections for 2-lane roads and 10,000 for random lane roads (1-6 lanes per road). The intersections were distributed over three to seven legs, where an n-leg intersection has n incident roads connected to it. During the generation runs, parameter nLaneDistributionOnASide had values [0.15, 0.6, 0.2, 0.05] and parameter nLaneDistributionOnControlLines had values [0.05, 0.45, 0.4, 0.1].

Incident Road Analysis

We begin by defining metrics which characterize the relative difficulty an autonomous vehicle would have when approaching and passing through an intersection. These metrics are diagrammed in Figure 12.

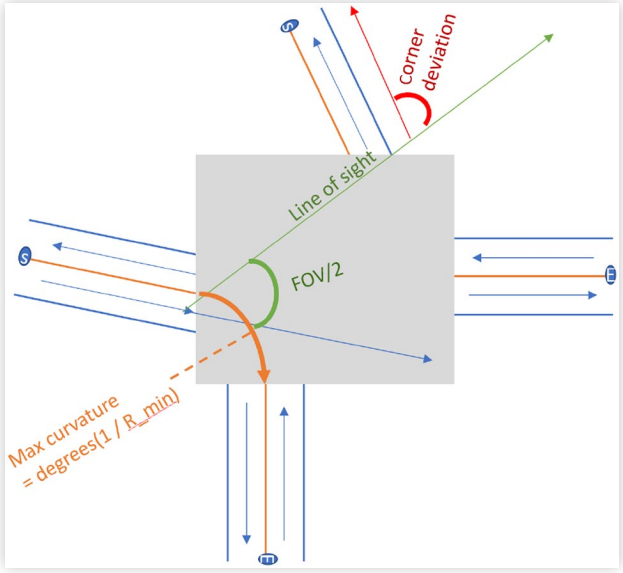
FOV is the field of view from a given incident lane and is defined as the angle a vehicle needs to see all other incident roads in an intersection. Each incident road for an intersection may have a different field of view. It is symmetric around the heading of the incident lane. This metric captures the intuition that the difficulty of traversing an intersection increases as an autonomous vehicle needs to see and process a wider volume of space (i.e., a wider field of view). A generator that constructs intersections with a variety of FOV values will test the range of input sensors such as cameras and LIDAR, and also the capability of the perception modules. FOV is defined by

$$FOV_i = max\left(\left\{x = sightAngle(i,j) | j \in I, i \neq j\right\}\right) * 2$$

where I is the set of incident roads and $sightAngle(i,j)$ is the angle between the incoming heading of the ith incident road and the sight line between ith and jth incident roads.

maxCurvature is the curvature in degrees which is required to travel from an incoming lane to the departure lane

FIGURE 12 The incident road to the west is characterized not only by its number of lanes, but also its FOV, curvature of a connecting lane, and deviation angle (cornerDeviation) from the line of sight.



requiring the maximum change in steering angle. It signifies the difficulty in keeping the vehicle inside a connecting lane. The higher this curvature the higher the chance of a vehicle to run into an adjacent lane. Higher curvatures also present a greater challenge to the driving stack of an autonomous vehicle since the region perceived by its sensors changes rapidly during a tight turn. In addition, higher curvatures present control challenges for larger vehicles or those with a large turning radius. maxCurvature is defined by

$$maxCurvature_i = max\left(\left\{x = diffCurvature(i,j) | j \in I, i \neq j\right\}\right)$$

where I is the set of incident roads and $diffCurvature(i,j)$ is the difference between the incoming heading of the ith incident road and the outgoing heading of the jth incident road.

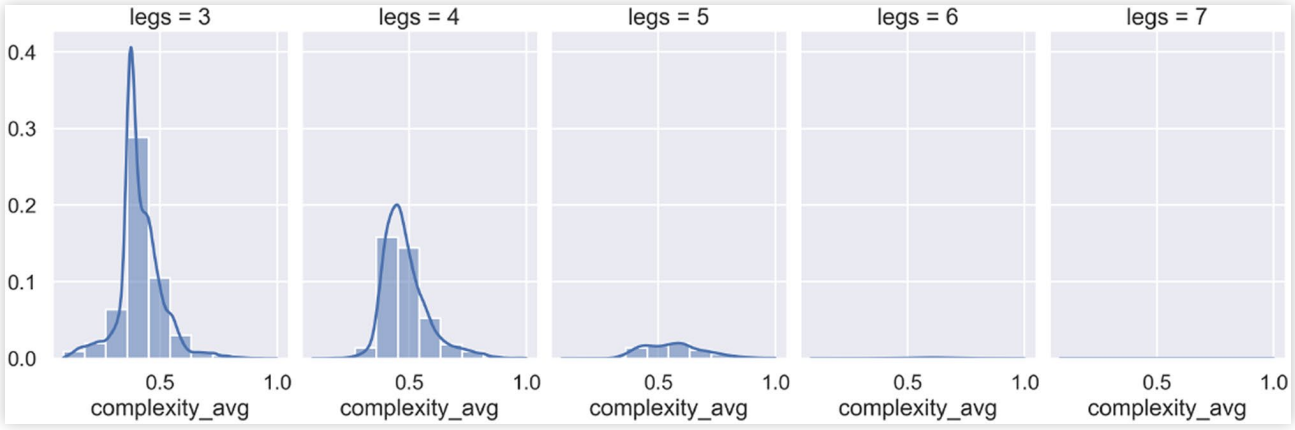
cornerDeviation is the deviation angle of other incident roads from the sight line of a given incident road. This captures the situation where, even if the starting point of an outgoing road is visible to an autonomous vehicle on an incoming road, a higher cornerDeviation angle makes it harder to see vehicles farther along the other roads. cornerDeviation is defined by

$$cornerDeviation_i = max\left(\left\{x = deviation(i,j) | j \in I, i \neq j\right\}\right)$$

where I is the set of incident roads and $deviation(i,j)$ is the angle between the outgoing heading of the jth incident road and the sight-line between ith and jth incident road.

Complexity for an incident road is the weighted sum of normalized values of FOV, maxCurvature, and cornerDeviation metrics. As a rule of thumb, we observe that if an

FIGURE 13 Probability distribution of complexity values computed over incident roads to intersections grouped by leg count. Complexity over 0.5 indicates complex intersections due to limited visibility and extreme curves.



intersection has incident roads having complexity over 0.5, it is very complex. Complexity is defined by

$$Complexity_i = 0.5 * normalized(maxCurvature_i) + 0.25 * normalized(FOV_i) + 0.25 * normalized(cornerDeviation_i) \mid i \in I$$

where I is the set of incident roads and $deviation(i,j)$ is the angle, and each metric is normalized using the min-max method.

Figures 13, 14, 15, and 16 present the frequency distributions of the above metrics. Metrics are separated by leg count (number of incident roads) to highlight how the distributions change with intersection type. The number of intersections having six or seven legs is small, and therefore, their distributions are barely visible in the graphs. Figure 13 plots complexity distribution. Almost half of the intersections are very complex. In Figure 14 showing maxCurvature, we observe that while most values are low (non-extreme curvatures), the generator is capable of generating highly curved connecting lanes (20 degrees or higher), though these are relatively infrequent. Figure 15 shows that FOV can become as large as 300 degrees, which is almost like U-turns, and the outgoing roads may not be visible at all from incoming roads. Figure 16 shows that outgoing roads can deviate greatly from the line of sight, making it hard to see lanes. In general, across all metrics, we observe that as the number of legs increases, the median complexity, FOV, and cornerDeviation are higher, indicating more challenging intersections.

In Figures 17, 18, and 19, we explore the interplay between two metric values. In Figure 16 we examine FOV versus maxCurvature using a heatmap, where brighter values indicate a higher likelihood of occurrence. The banding indicates that maxCurvature values have frequency peaks around 5, 10, and 15 degrees. This figure shows that high maxCurvature values are generally associated with wider FOV values, and it is possible to achieve higher maxCurvatures and FOV values

FIGURE 14 Probability distribution of maxCurvature from incident roads (in degrees, grouped by leg count).

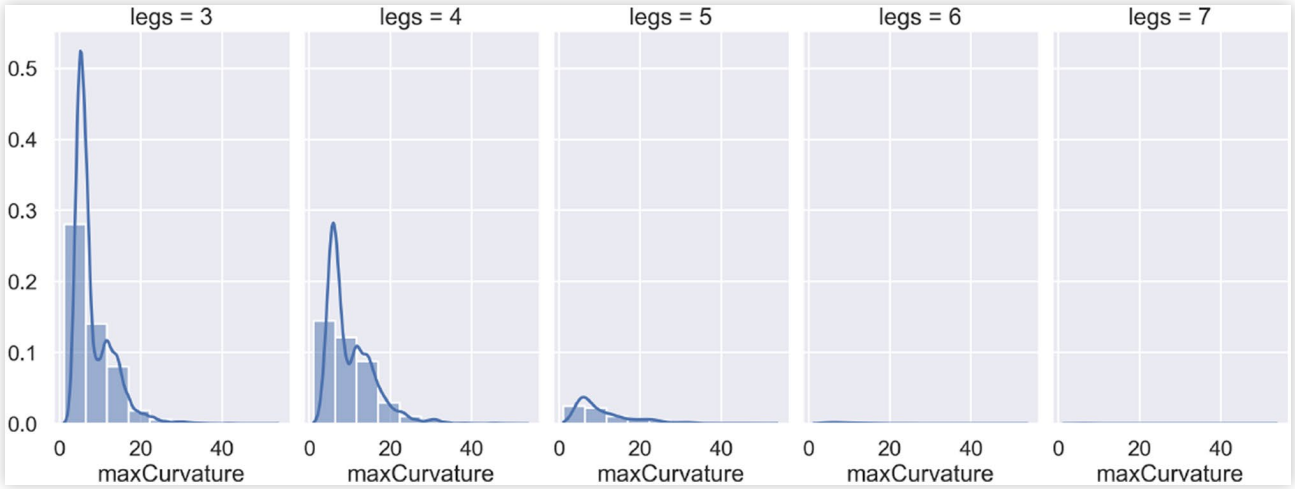
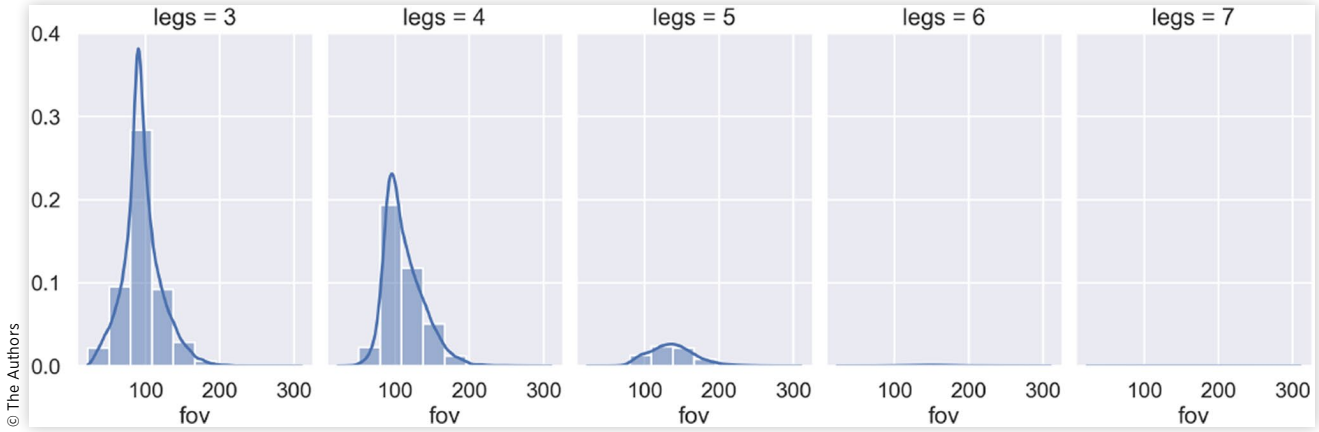


FIGURE 15 Probability distribution of FOV from incident roads (in degrees, grouped by leg count).



simultaneously. Figure 18 shows FOV versus cornerDeviation, and has a striking co-occurrence line, indicating that, for many instances, they are linearly correlated. However, there are also many values below the line, indicating there are some incident lanes where you can achieve high cornerDeviation and FOV simultaneously. Figure 19 shows cornerDeviation versus maxCurvature, where we again see banding from the frequency spikes of maxCurvature and an overall frequency distribution similar to Figure 16.

Intersection Area Analysis

In this section we explore metrics which characterize an entire intersection rather than a specific incident road.

Area: The total space taken up by all interior connecting lanes within an intersection. There can be many variations in shape for the same area.

conflictArea: Locations in an intersection where two interior connection lanes overlap, thereby creating the zones where there is the greatest likelihood of collision. Conflicts can happen when connection lanes diverge, merge, and cross. conflictArea is computed by summing up all of the areas of connection lane overlap inside an intersection. A similar notion to conflictArea is conflict points, which represent a single point where two connection lanes intersect. Since the same number of conflict points can be associated with varying amounts of conflictArea, we prefer to do an area analysis since it is more descriptive.

Figure 20 provides examples of intersection area (green and red combined) and conflict zones (in red).

conflictRatio: The ratio of conflictArea to total intersection area. Intersections with a higher conflictRatio have more opportunities for vehicles to collide, and hence are more interesting for testing autonomous vehicles.

nConnectionLanes: An interior connection lane is a one-lane road that connects an incoming lane from one incident road to an outgoing lane in another. This metric counts the number of these lanes inside an intersection. As the number of incident road lanes increases, this metric increases combinatorially. It can be viewed as a measure of possible traffic trajectories inside an intersection.

Figures 21, 22, 23, and 24 present the frequency distributions of these metrics. Figure 21 shows intersection area distributions. As expected, the median area value increases with leg count, but perhaps more slowly than expected. Figure 22 shows conflictArea distributions, which also increase in median size with leg count and are strongly correlated with

FIGURE 16 Probability distribution of cornerDeviation from incident roads (in degrees, grouped by leg count).

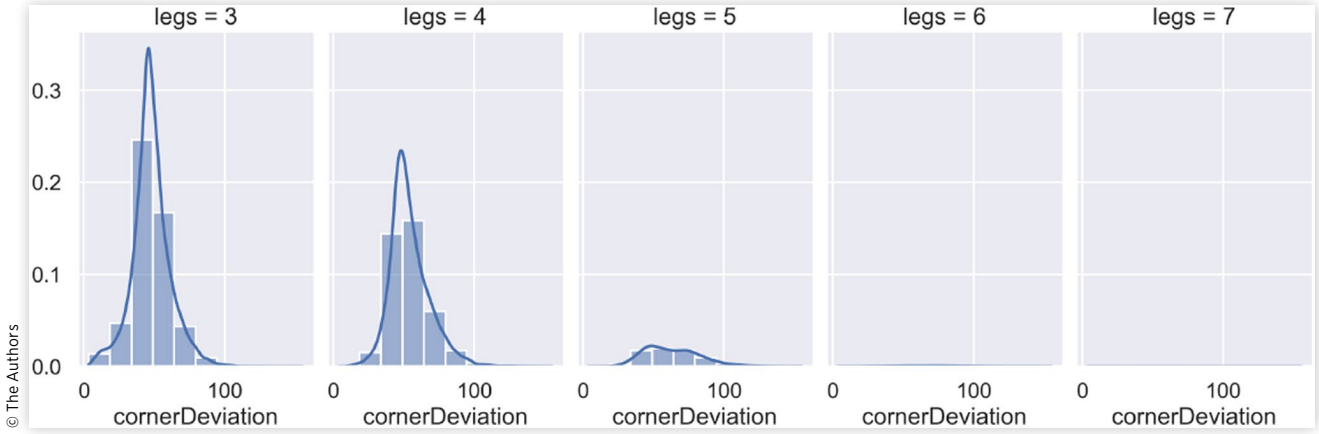


FIGURE 17 Correlation between FOV and maxCurvature from incident roads.

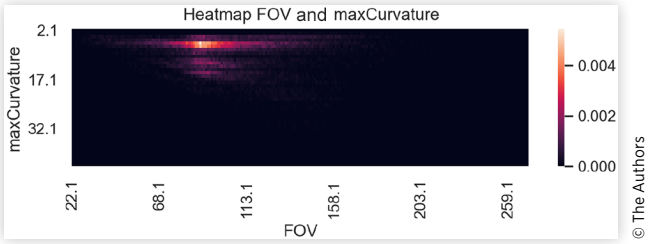
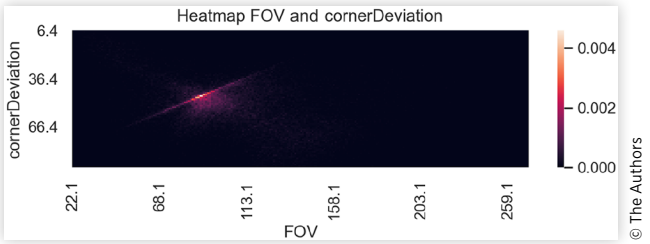


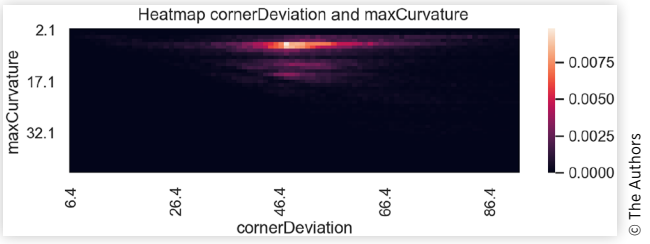
FIGURE 18 Correlation between FOV and cornerDeviation from incident roads.



intersection area. The most interesting metric, conflictRatio, is depicted in Figure 23. We can see that for an n-way intersection, the amount of conflictArea can vary greatly, and it is possible to create intersections with a high proportion of conflictArea. A higher percentage of conflictArea should create more complicated intersections for testing purposes. Figure 24 shows the distribution of nConnectionLanes and highlights the range of interior complexity for a given leg count. As the number of interior connection roads increases, it should be more challenging for an autonomous vehicle to navigate the intersection.

Figures 25, 26, 27, and 28 explore the co-occurrence of conflictArea and conflictRatio with FOV and maxCurvature. For testing, we would ideally like to have intersections which have both high conflictArea or conflictRatio, and high FOV or maxCurvature since this maximizes multiple metrics which create challenging intersections. While the heatmaps show that achieving high conflictArea and FOV

FIGURE 19 Correlation between cornerDeviation and maxCurvature from incident roads.



simultaneously is challenging, it is possible to achieve high conflictRatio and FOV. Similarly, while high conflictArea and maxCurvature are rare, high conflictRatio and maxCurvature are possible to achieve.

Interoperability and Usage

We end our evaluation of JunctionArt by examining whether the desired goal of interoperability actually does result in the ability to load generated road networks into a variety of simulation environments. We imported a road network generated by JunctionArt (from Figure 3) into three road and vehicle simulation tools: Carla [6], RoadRunner [5], and esmini [33]. Figures 29 to 31 provide screenshots showing this road network after importation. The road network retains the same shape and lane configurations as the original in all tools. Usually, tools that can import OpenDRIVE files automatically create the road meshes and waypoints used by different planning and perception algorithms. Both Carla [6] and esmini [33] can generate road meshes and waypoints using the data from the OpenDRIVE file JunctionArt creates. RoadRunner can build 3D models of the road network, including different lane markings. Moreover, esmini can automatically create simulated traffic on an OpenDRIVE road network. We show an example of this traffic in Figure 31(a).

Performance

We explore the performance of JunctionArt by executing a series of generation runs to create road networks in regions

FIGURE 20 Variations in area and conflictArea for three-way intersections. Red zones have lanes overlapping, and green zones have no lanes overlapping.

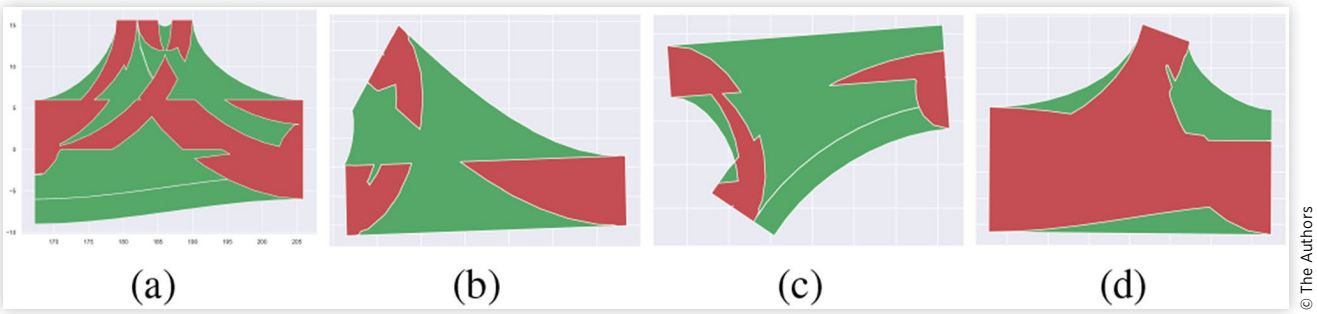


FIGURE 21 Probability distribution of intersection area (in square meters).

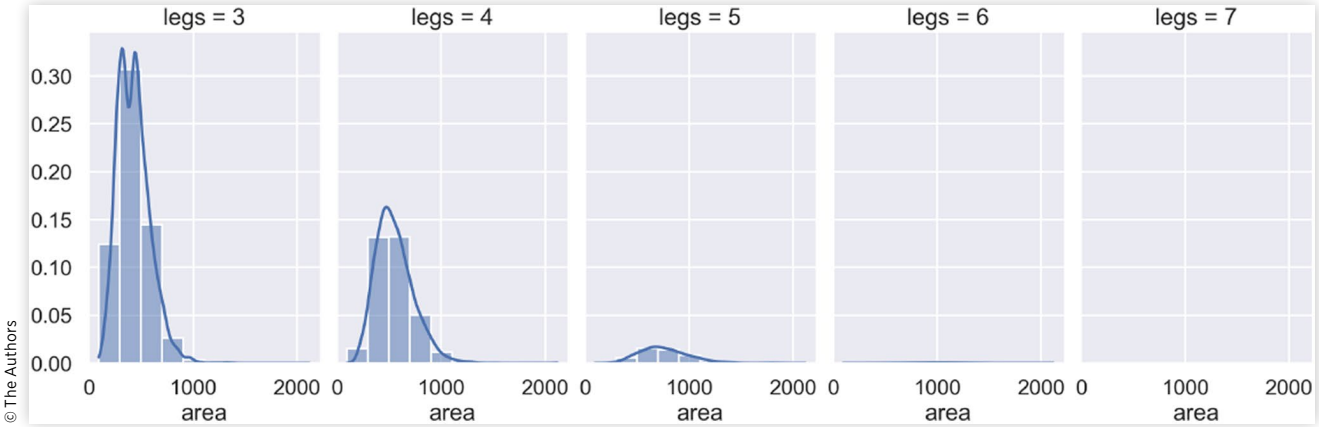


FIGURE 22 Probability distribution of conflictArea (in square meters).

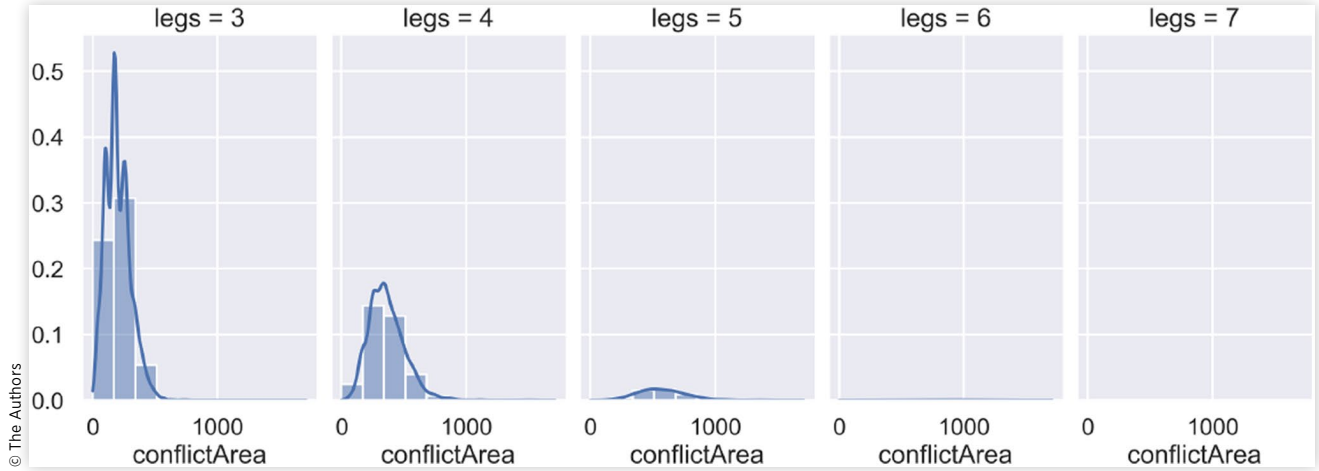


FIGURE 23 Probability distribution of conflictRatio.

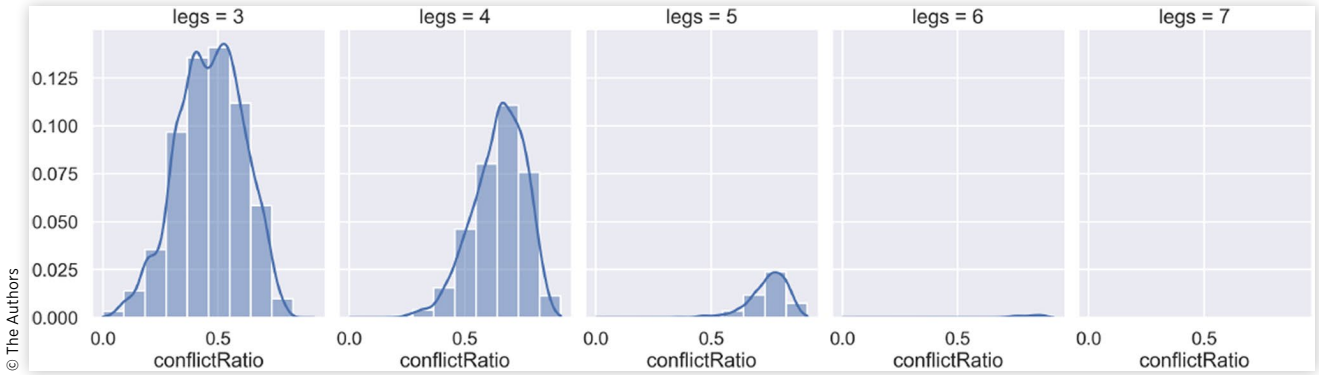
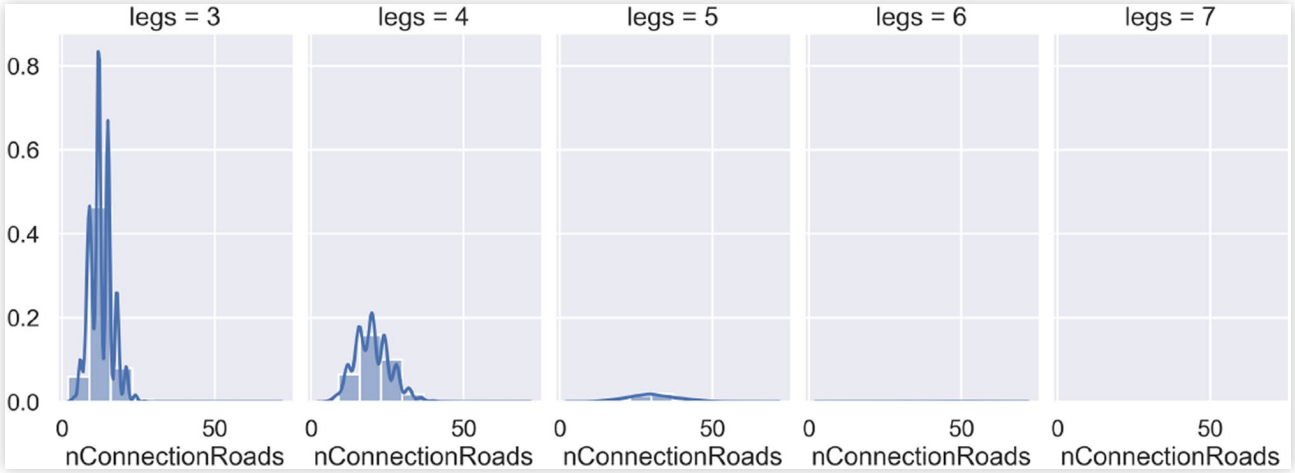


FIGURE 24 Probability distribution of nConnectionLanes (nConnectionRoads).



of varying size, and with roads of only two lanes or random numbers of lanes. Table 1 presents the results of these generation runs.

Generation runs were performed using a single process in Python on a machine with an Intel Core i7-9700K CPU and 32 GB RAM. In Table 1, the test configuration column describes how many maps we want to create for a given size. The road generator then tries to generate the same number of maps. However, sometimes JunctionArt is unable to successfully create a road network because some intersections become unrealistic due to extremely sharp turns, and thus the intersection validator discards those outputs. To account for this, we increased the number of generation attempts to achieve a specific number of generated networks. As maps get larger in size, more variations are generated, leading to more invalid intersections. The #try column reflects the number of times the generator tried to create a valid map, and the #gen column shows the number of valid ones. It takes more time to generate networks with a random number of lanes instead of standard two-lane roads. We can see the running times and memory usage are minimal due to the constructive approach of our generator instead of a search approach. In the worst case, it takes less than 3 minutes to generate a 5×5 km^2 road network with random lanes and automatic control lines.

FIGURE 25 Co-occurrence of FOV and conflictArea. y-axis scaled downed by 10.

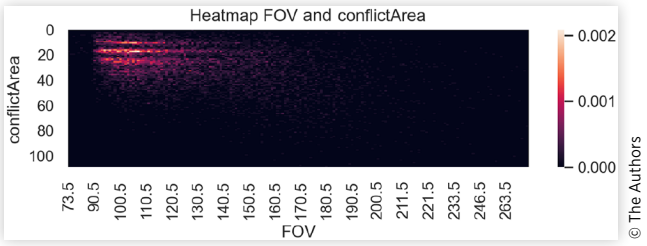


FIGURE 26 Co-occurrence of maxCurvature and conflictArea. y-axis scaled downed by 10.

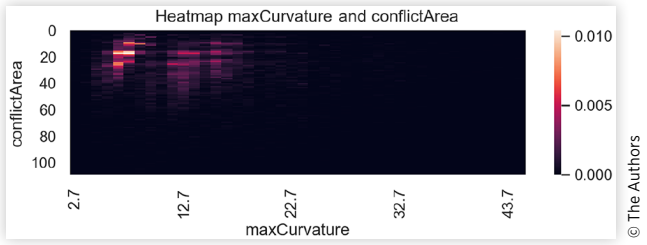


FIGURE 27 Co-occurrence of FOV and conflictRatio.

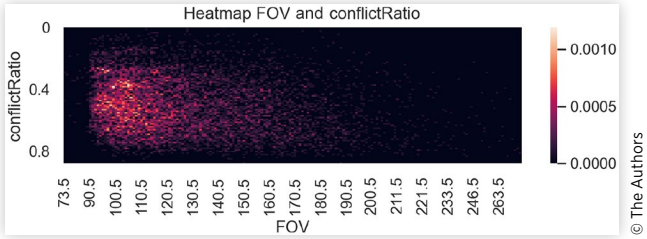


FIGURE 28 Co-occurrence of maxCurvature and conflictRatio.

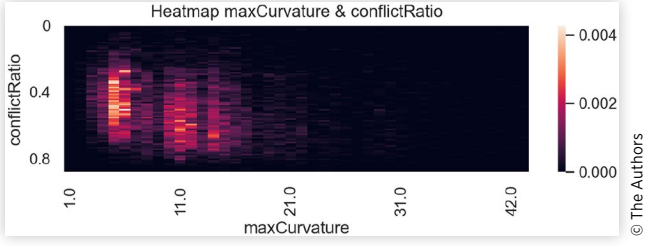
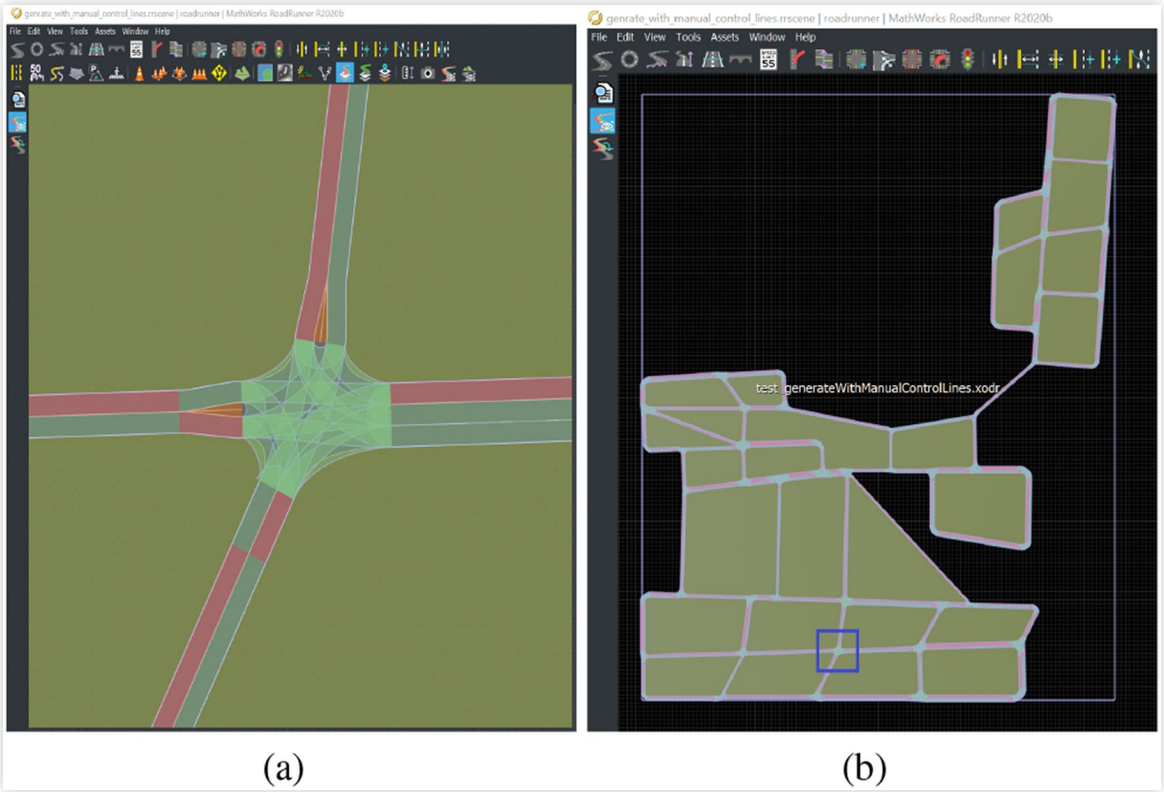
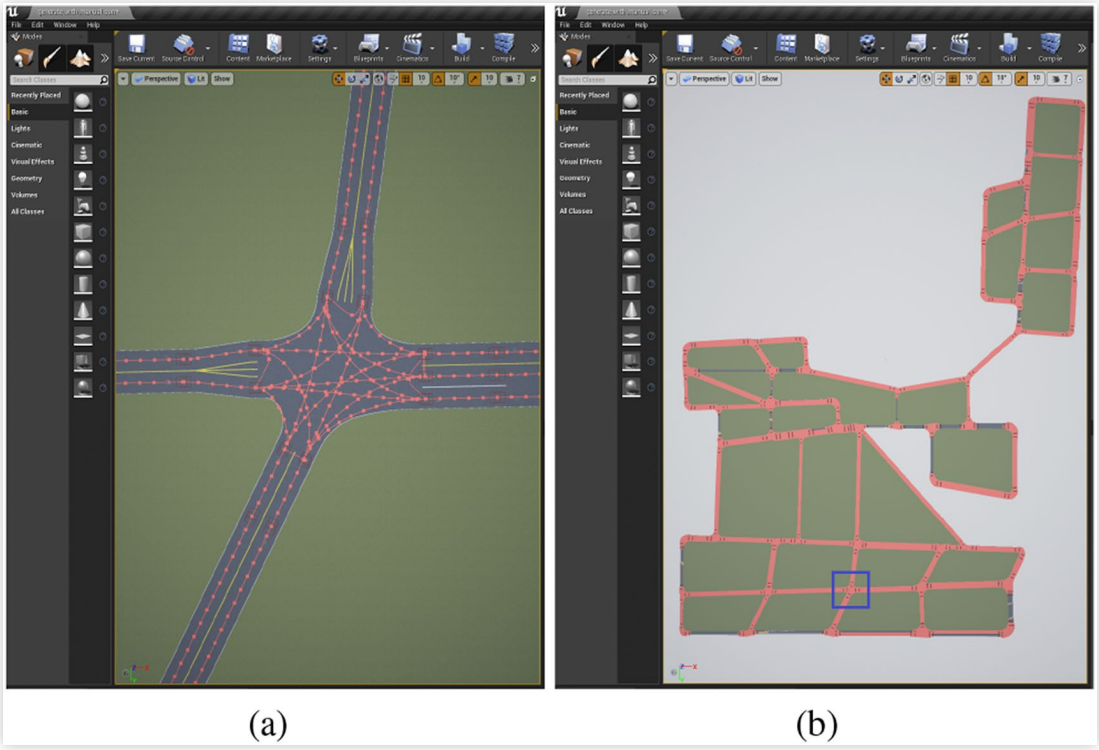


FIGURE 29 The road network from Figure 3 imported into RoadRunner [5]. A representative intersection [blue rectangle in (b)] is shown in (a).



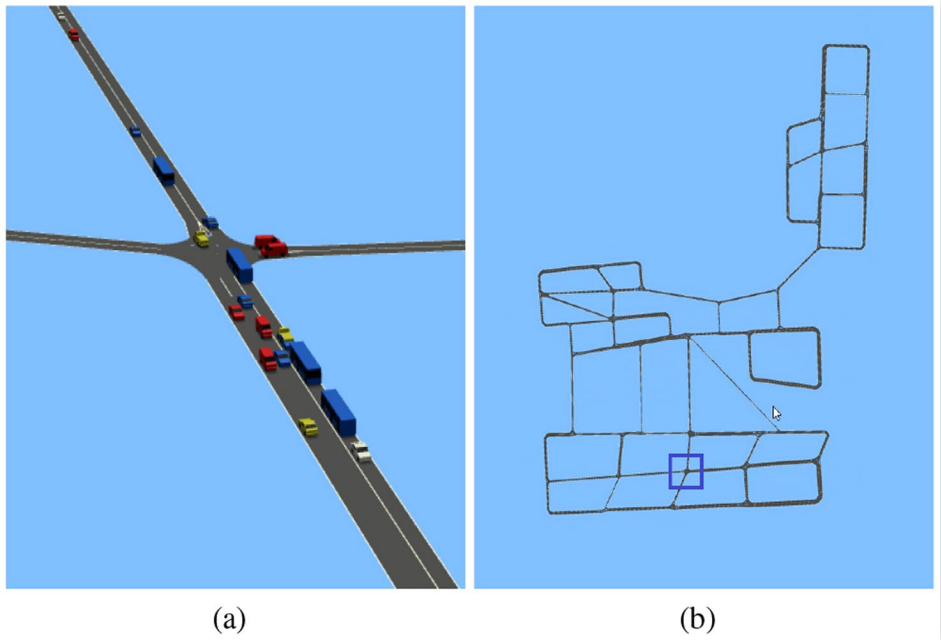
Data taken from Ref. [5]. © The Authors

FIGURE 30 The road network from Figure 3 imported into Carla [6]. In (a) waypoints were automatically generated along the center of lanes (orange dots) for a representative intersection [blue rectangle in (b)].



Data taken from Ref. [6]. © The Authors

FIGURE 31 The road network from Figure 3 imported into esmini [33]. In (a) traffic is automatically generated for a representative intersection [blue rectangle in (b)].



Threats to Validity

In this section we discuss the limitations of the research and threats to its ability to generate realistic road networks.

In creating JunctionArt, we did not train or learn from the frequency distributions of real-world road networks, such as via a machine learning or neural network approach. Instead we designed JunctionArt to generate, by construction, the variation in intersection shapes and number of lanes that is common in real-world maps (see the Expressive Range Analysis, 5.2). While generating the curves, we ensured that vehicles can smoothly navigate through them. We note that the goal of JunctionArt was not to replicate the frequency distributions of qualities of existing road networks but instead to create realistically plausible road networks which have greater numbers of unusual features than normal. In this way, JunctionArt can generate road networks which are useful for testing autonomous vehicles, precisely because they are more unusual than typical roads.

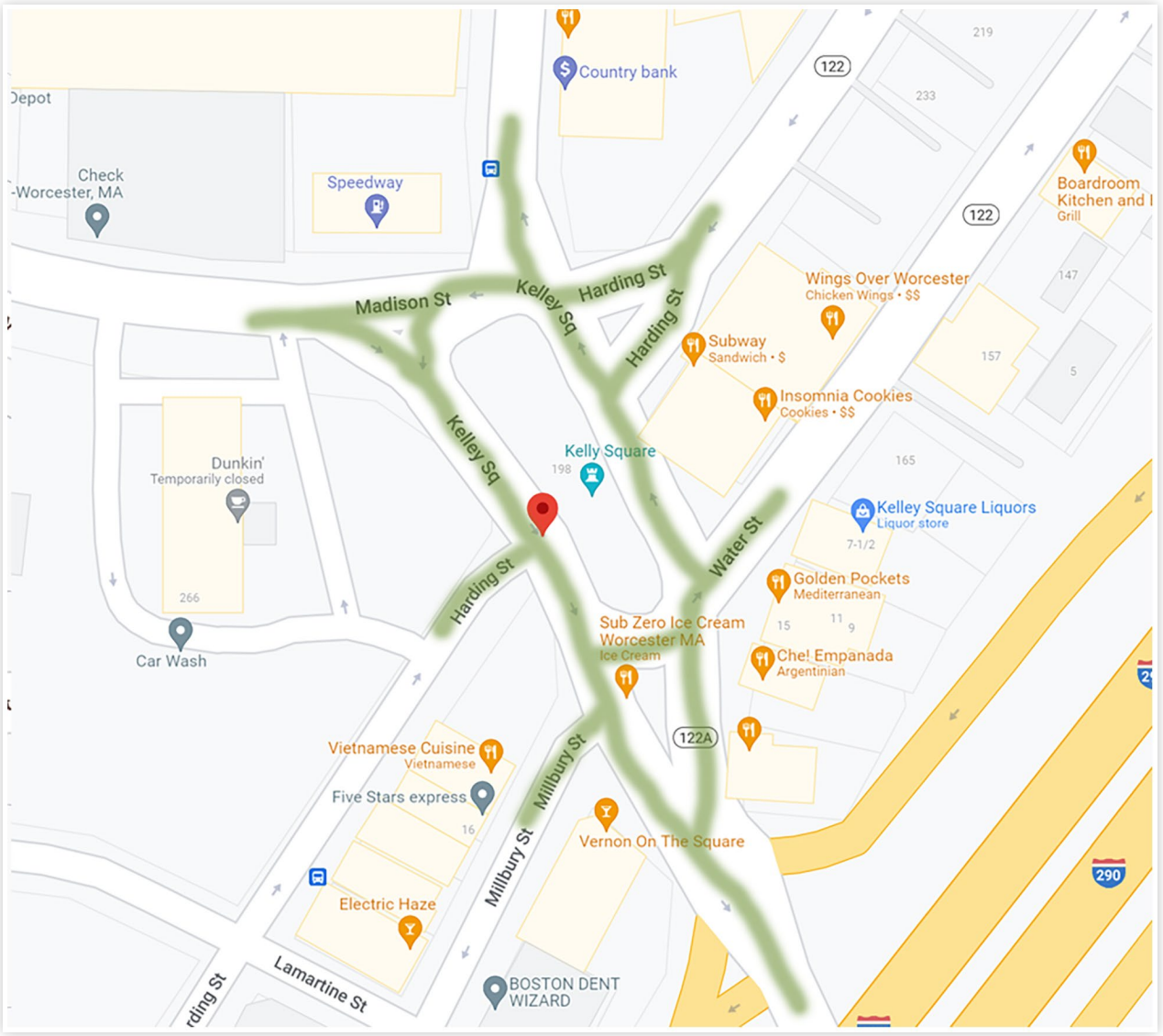
Real-world roads and intersections develop their shapes organically, over time, and often do not conform to road design guidelines. Our generator can generate some roads of this kind (see Figure 32). However, there are some intersections, such as those in Figure 33, that cannot be generated with the current network generation algorithm, that is, there are still unusual aspects of the real world that are outside the expressive range of JunctionArt.

TABLE 1 Performance analysis of road generation.

Test configuration	#try	#gen	Total time (minutes)	Time per map (seconds)	mem (MB)
50 (500 × 500) m^2 maps with 2 lanes	50	50	0.28	0.34	84
50 (500 × 500) m^2 maps with random lanes (Figure 8)	50	50	0.33	0.4	89
50 (1000 × 1000) m^2 maps with 2 lanes	50	50	1.22	1.46	94
50 (1000 × 1000) m^2 maps with random lanes	50	50	1.4	1.68	99
50 (5000 × 5000) m^2 maps with 2 lanes	163	50	74.22	89.07	437
50 (5000 × 5000) m^2 maps with random lanes	302	50	122.1	146.6	477
50 (1200 × 1400) m^2 manual control-line maps with 2 lanes	54	50	1.66	1.99	103
50 (1200 × 1400) m^2 manual control-line maps with random lanes (Figure 3)	60	50	2	2.4	107
50 (1000 × 1600) m^2 A-shaped maps with 2 lanes	53	50	1.14	1.37	98
50 (1000 × 1600) m^2 A-shaped maps with random lanes (Figure 7)	60	50	1.37	1.65	102

FIGURE 32 A K-shaped intersection at Arlington Street and Wilder Street, SF, USA. Our generator generates such intersections.

FIGURE 33 Kelley Square, Worcester, MA, USA. Our generator cannot generate such intersections.



Conclusion

JunctionArt is a procedural road generator capable of generating high-definition road networks with a wide variety of lane counts and intersection types. A series of control lines and optional control line groups provides human authorial control over the shape of the resulting road network, or JunctionArt can create these automatically. By building on the OpenDRIVE standard, test engineers can use JunctionArt road networks in existing tools which support OpenDRIVE.

An expressive range analysis characterizes the space of possible intersections JunctionArt can create. Multiple metrics, including FOV, maxCurvature, cornerDeviation, complexity, conflictArea, and nConnectionLane, provide measures of the difficulty an autonomous vehicle will have when traversing an intersection. While intersection examples that maximize these values are rare, JunctionArt is capable of generating them. Further, it is possible in many cases to generate intersections which maximize multiple metrics simultaneously. For engineers testing autonomous vehicles in simulation environments, JunctionArt produces road networks and intersections which clearly explore the long tail of possible road and intersection geometries.

JunctionArt is available on GitHub at https://github.com/AugmentedDesignLab/junction-art.

Contact Information

Golam Md Muktadir
muktadir@ucsc.edu

Jim Whitehead
ejw@ucsc.edu

Abdul Jawad
abjawad@ucsc.edu

References

1. Bethel, B., "5 Tesla Accidents in Same Location in Yosemite," https://www.reddit.com/r/SelfDrivingCars/comments/oxhbit/5_tesla_accidents_in_same_location_in_yosemite/, last accessed on November 11, 2021.
2. Huang, W., Wang, K., Lv, Y., and Zhu, F., "Autonomous Vehicles Testing Methods Review," in *Proceedings of 2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*, Rio de Janeiro, Brazil, 163-168, 2016.
3. The Verge, "Welcome to Simulation City, the Virtual World Where Waymo Tests Its Autonomous Vehicles," https://www.theverge.com/2021/7/6/22565448/waymo_simulation_city_autonomous_vehicle_testing_virtual, last accessed on November 11, 2021.
4. The Atlantic, "Inside Waymo's Secret World for Training Self-Driving Cars," https://www.theatlantic.com/technology/archive/2017/08/inside_waymos_secret_testing_and_simulation_facilities/537648/, last accessed on November 11, 2021.
5. RoadRunner, "Design 3D Scenes for Automated Driving Simulation," https://www.mathworks.com/products/roadrunner.html.
6. Dosovitskiy, A., Ros, G., Codevilla, F., Lopez, A. et al., "Carla: An Open Urban Driving Simulator," in *Conference on Robot Learning*, Mountain View, CA, 1-16, PMLR, 2017.
7. Quiter, C., DeepDrive, https://deepdrive.io/, last accessed on November 11, 2021.
8. AirSim, "Microsoft Airsim," https://microsoft.github.io/AirSim/, last accessed on November 11, 2021.
9. Smelik, R.M., Tutenel, T., Bidarra, R., and Benes, B., "A Survey on Procedural Modelling for Virtual Worlds," *Computer Graphics Forum* 33 (2014): 31-50.
10. Greuter, S., Parker, J., Stewart, N., and Leach, G., "Real-Time Procedural Generation of Pseudo Infinite' Cities," in *Proceedings of the 1st international Conference on Computer Graphics and Interactive Techniques in Australasia and South East Asia*, Dunedin, New Zealand, 87, 2003.
11. Parish, Y.I. and Müller, P., "Procedural Modeling of Cities," in *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, Boulder, CO, 301-308, 2001.
12. Lechner, T., Ren, P., Watson, B., Brozefski, C. et al., "Procedural Modeling of Urban Land Use," in *ACM SIGGRAPH 2006 Research Posters*, Boston, MA, 135, 2006.
13. Chen, G., Esch, G., Wonka, P., Müller, P. et al., "Interactive Procedural Street Modeling," in *ACM SIGGRAPH 2008 Papers*, New York, 1-10, 2008.
14. McCrae, J.P. and Singh, K., "Sketch-Based Path Design," *Proceedings of Graphics Interface 2009*, Kelowna, BC, Canada, 95-102, 2009.
15. Galin, E., Peytavie, A., Guérin, E., and Beneš, B., "Authoring Hierarchical Road Networks," *Computer Graphics Forum* 30 (2011): 2021-2030.
16. Freiknecht, J. and Effelsberg, W., "A Survey on the Procedural Generation of Virtual Worlds," *Multimodal Technologies and Interaction* 1, no. 4 (2017): 27.
17. Ilangovan, P.K., "Procedural City Generator," Master's thesis, National Centre for Computer Animation, Bourne Mouth, 2009.
18. Kelly, G. and McCabe, H., "Citygen: An Interactive System for Procedural City Generation," in *Fifth International Conference on Game Design and Technology*, Liverpool, UK, 8-16, 2007.
19. Kelvin, L.Z. and Anand, B., "Procedural Generation of Roads with Conditional Generative Adversarial Networks," in *2020 IEEE Sixth International Conference on Multimedia Big Data (BigMM)*, New Delhi, India, 277-281, IEEE, 2020.
20. Hartmann, S., Weinmann, M., Wessel, R., and Klein, R., "Streetgan: Towards Road Network Synthesis with Generative Adversarial Networks," in *International Conference on Computer Graphics, Visualization and Computer Vision*, Primavera Congress Center, Plzen, Czech Republic, 2017.
21. Guérin, É., Digne, J., Galin, E., Peytavie, A. et al., "Interactive Example-Based Terrain Authoring with Conditional Generative Adversarial Networks," *ACM Transactions on Graphics (TOG)* 36, no. 6 (2017): 1-13.
22. Gambi, A., Mueller, M., and Fraser, G., "Automatically Testing Self-Driving Cars with Search-Based Procedural Content Generation," *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis,* Beijing, China, 318-328, 2019.
23. Li, Q., Peng, Z., Zhang, Q., Qiu, C. et al., "Improving the Generalization of End-to-End Driving through Procedural Generation," arXiv preprint arXiv:2012.13681, 2020.
24. Andersson, M., Natale, I., and Tingberg, A., "Pyodrx," https://github.com/pyoscx/pyodrx, last accessed on May 12, 2022.
25. Becker, D., Ruß, F., Geller, C., and Eckstein, L., "Generation of Complex Road Networks Using a Simplified Logical Description for the Validation of Automated Vehicles," in *2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*, Rhodes, Greece, 1-7, IEEE, 2020.

26. Cura, R., Perret, J., and Paparoditis, N., "Streetgen: In-Base Procedural-Based Road Generation," in *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences*, La Grande Motte, France, vol. II-3/W5, 409-416, August 2015.
27. Despine, G. and Baillard, C., "Realistic Road Modelling for Driving Simulators Using GIS Data," in *Advances in Cartography and GIScience*, Volume 2 (Berlin: Springer, 2011), 431-448.
28. Wang, H., Wu, Y., Han, X., Xu, M. et al., "Automatic Generation of Large-Scale 3d Road Networks Based on GIS Data," *Computers & Graphics* 96 (2021): 71-81.
29. Zhang, C., Li, Y., Xiang, L., Jiao, F. et al., "Generating Road Networks for Old Downtown Areas Based on Crowd-Sourced Vehicle Trajectories," *Sensors* 21, no. 1 (2021): 235.
30. ASAM, "ASAM OpenDRIVE 1.6," https://www.asam.net/standards/detail/opendrive/, last accessed on November 11, 2021.
31. Smith, G. and Whitehead, J., "Analyzing the Expressive Range of a Level Generator," in *Proceedings of the 2010 Workshop on Procedural Content Generation in Games*, Monterey, CA, 1-7, 2010.
32. Choi, E.-H., "Crash Factors in Intersection-Related Crashes: An On-Scene Perspective (NHTSA Technical Report DOT HS 811 366)," U.S. Department of Transportation, National Highway Traffic Safety Administration, 2010.
33. Knabe, E."Environment Simulator Minimalistic (esmini)," https://github.com/esmini/esmini, last accessed on May 12, 2022.