Automatic Map Generation for Autonomous Driving System Testing

Yun Tang, Yuan Zhou, Kairui Yang, Ziyuan Zhong, Baishakhi Ray, Yang Liu, Ping Zhang, Junbo Chen

Abstract—High-definition (HD) maps are essential in testing autonomous driving systems (ADSs). HD maps essentially determine the potential diversity of the testing scenarios. However, the current HD maps suffer from two main limitations: lack of junction diversity in the publicly available HD maps and cost-consuming to build a new HD map. Hence, in this paper, we propose the first method, FEAT2MAP, to automatically generate concise HD maps with scenario diversity guarantees. FEAT2MAP focuses on junctions as they significantly influence scenario diversity, especially in urban road networks. FEAT2MAP first defines a set of features to characterize junctions. Then, FEAT2MAP extracts and samples concrete junction features from a list of input HD maps or user-defined requirements. Each junction feature generates a junction. Finally, FEAT2MAP builds a map by connecting the junctions in a grid layout. To demonstrate the effectiveness of FEAT2MAP, we conduct experiments with the public HD maps from SVL and the open-source ADS Apollo. The results show that FEAT2MAP can (1) generate new maps of reduced size while maintaining scenario diversity in terms of the code coverage and motion states of the ADS under test, and (2) generate new maps of increased scenario diversity by merging intersection features from multiple maps or taking user inputs.

I. INTRODUCTION

Autonomous driving systems (ADSs) have gained enormous attention and development recently. Safety is the public’s primary concern for the mass deployment of ADSs. According to ISO/PAS 21448 [1], SOTIF (Safety Of The Intended Functionality) is the key for ADS safety. However, due to the enormous number of parameter combinations to specify test scenarios, it is still a challenge to validate the SOTIF of ADSs, which aims to reduce the region of unknown and unsafe scenarios.

Among the existing methods for the validation of SOTIF, simulation-based ADS testing is a major approach to discovering unknown and unsafe scenarios for ADSs [2]. In the simulation environment, we can model different elements for an autonomous vehicle, such as HD (High-Definition) maps (e.g., road network, traffic signs, and traffic lights), vehicle dynamics, and sensors (e.g., camera, LiDAR, radar). Among all the elements, HD maps are prerequisites for ADS testing, and they essentially determine the potential diversity of the testing scenarios, directly impacting the testing efficiency.

For example, Gambi et al. demonstrated that many lane departure failures have manifested on different shapes of roads [3]. Tang et al. showed that by exploiting the topology features of junctions in HD maps, diverse testing scenarios could be generated, and different issues related to SOTIF were exposed in the Apollo ADS [4]–[6].

Despite the importance of HD maps, there are some limitations of the publicly available HD maps during ADS testing. First, the commercial HD maps, such as those provided by [7], [8] are replicas of the real world and contain a significant amount of duplicate map elements (e.g., junctions with similar shapes). Thus, scenarios from such HD maps may lack diversity. Second, although the open-source simulators, such as Carla [9] and SVL [10], often come with some built-in HD maps, they are relatively small with a limited number of junctions and roads. Consequently, the diversity of scenarios generated from these maps is limited. Third, different cities or regions may contain different junction structures. Hence, to test ADSs sufficiently, we need different HD maps from different areas and perform ADS testing within each map. It is time-consuming. Fourth, manually building diverse HD maps using tools, such as SUMO netedit [11] or CommonRoad Scenario Editor [12], is laborious, given a large number of scenario requirements. Therefore, it is essential to develop a framework to generate concise but scenario-diverse HD maps automatically.

However, there is little work on automatic HD map generation. Gambi et al. [3] proposed a search-based approach to generate lanes of changing curvatures to test the lane-keeping functionality. However, other traffic elements, such as intersections and traffic controls, are not considered. Thus, the map generated is incomplete, and the scenarios are limited. Mi et al. [13] used a trained hierarchical graph generation model to generate the road network graphs. The model is trained with existing HD maps and tries to generate maps similar to its training data. As a result, it suffers from the same aforementioned limitations (e.g., limited scenario diversity and duplicated intersections) as the training data (i.e., existing HD maps).

In this paper, we propose the first feature-based method, FEAT2MAP, to generate scenario-diverse HD maps automatically based on junction features as junctions significantly affect scenario diversity, especially in urban road networks [4]–[6]. First, we define a feature vector to characterize junctions, including the number of roads connected by a junction, orientations of the connected roads, traffic controls, and crosswalk features. Second, FEAT2MAP extracts all possible values of each feature element from a set of maps or users’ inputs. Third, FEAT2MAP applies combinatorial sampling to
generate a minimal set of concrete feature vectors without duplication while covering all possible feature combinations. Finally, FEAT2MAP builds a junction per concrete feature vector and constructs a complete HD map with connected junctions in a grid layout. To demonstrate the effectiveness of FEAT2MAP, we generated several HD maps from single and multiple input maps and performed qualitative and quantitative analysis by performing route coverage testing [4] on the latest open-source ADS Apollo 7.0 [14]. The results show that (1) FEAT2MAP can effectively extract features from an input map and generate concise HD maps while preserving scenario diversity; (2) FEAT2MAP can generate HD maps based on combined features from multiple input maps with higher perceived scenario diversity by the ADS; (3) FEAT2MAP can generate customized HD maps from user input.

II. PRELIMINARIES AND PROBLEM STATEMENT

HD map is one of the essential components for modern ADSs, especially in city-driving scenarios. It encodes the semantic features (e.g., topology and geometry) of traffic network elements such as traffic lanes, roads, intersections, traffic lights, crosswalks, and stop signs [15]. Fig. 1 shows a diagram of an HD map containing a two-junction traffic network. There are six two-way roads ($R_0 - R_5$) in the diagram. Each road (e.g., $R_0$) contains a forward lane (e.g., $L_2$ in $R_0$) and a backward lane (e.g., $L_1$ in $R_0$). $R_0 - R_5$ form a traffic light-controlled junction $J_0$, and $R_3 - R_5$ construct a stop sign-controlled junction $J_1$. Lanes are the atomic element in an HD map, and each lane is represented by its center reference line. Lanes in an HD map can be classified into road lanes (e.g., $L_1 - L_{12}$ in Fig. 1) and junction lanes (e.g., $L_{13} - L_{30}$ in Fig. 1).

Given an HD map, ADS testing aims to guarantee that the autonomous vehicle can run safely at any part of the map under any scenario. However, an HD map from part of a city or a limited area usually contains limited intersection structures. Consequently, we cannot test an ADS sufficiently on such a map. Hence, in this paper, we aim at generating an HD map automatically by eliminating duplicated intersections while maximizing the diversity of intersection structures and potential scenarios.

Before illustrating the generation method in detail, we first discuss some assumptions in the paper. (1) We currently focus on the intersections of a map, as intersections can significantly affect the map complexity and scenarios diversity. In the future, we will take roads into consideration. (2) An ADS can be roughly divided into the perception and the planning components. We mainly focus on generating maps for testing the planning component. Testing of AI models in the perception component is orthogonal to our work.

III. METHODOLOGY

A. Modeling of Lanes

Based on [16], lanes can be represented by their center reference paths. We model these reference paths using cubic Bézier curves, which are extensively used in computer graphics [17]. Given four control points $P_0, P_1, P_2$ and $P_3$, the cubic Bézier curves can be formulated as:

$$B(t) = \sum_{i=0}^{3} \binom{3}{i} (1-t)^{3-i}t^i P_i$$

where $t \in [0, 1]$, and $\binom{3}{i}$ are the binomial coefficients. Cubic Bézier curves are powerful in representing simple curves such as straight lines, circle-shaped or S-shaped, and can be combined into Bézier splines of complex shapes. For any path (e.g. reference path or boundary curve of a lane), if the start point $P_{\text{start}}$, start heading $\theta_0$, end point $P_{\text{end}}$, and end heading $\theta_1$ are known, the controls points can be calculated as:

$$P_0 = P_{\text{start}}$$
$$P_1 = P_{\text{start}} + (d_0 \cos \theta_0, d_0 \sin \theta_0)$$
$$P_2 = P_{\text{end}} - (d_1 \cos \theta_1, d_1 \sin \theta_1)$$
$$P_3 = P_{\text{end}}$$

where both $d_0$ and $d_1$ are heuristically defined as half of the distance between $P_1$ and $P_3$ for a smoother curve.

B. Feature Modeling of Junctions

In the section, we define the junction features to characterize a junction, containing the road feature, the rotation feature, the control feature, and the crosswalk feature.

First, the road feature aims to describe the number of roads connected by a junction, such as T-junctions and n-legged intersections. The more connected roads, the more complex the junction becomes, resulting in more testing efforts for ADSs. Hence, the road feature of a junction $J_j$, denoted as $F_{J_j}^{\text{road}}$, is defined as the number of roads connected by a junction. For example, the junction $J_0$ shown in Fig. 1 connects four roads, so $F_{J_0}^{\text{road}} = 4$, while $F_{J_1}^{\text{road}} = 3$.

Second, the rotation feature aims to model the relative orientations of the roads connected by a junction. It indicates the relative directions an autonomous vehicle can drive after entering the junction. Given a junction $J_j$, suppose its center is $C$, which is calculated as the geometry center of its bounding polygon, and the corresponding start point of road $R_i$ is $P_i$, which is the connecting endpoint of $R_i$’s reference line to the junction. The vector $CP_i$ is called road socket of $R_i$. Hence, the relative location of $R_i$ and $R_k$ can be measured by the angle between $CP_i$ and $CP_k$. Hence, the
where were of control manners, i.e., bare regulated in a junction. Commonly, there exist three kinds in Fig. 2, suppose after rotation normalization, road sockets are uniquely determined. For example, as shown γ direction to the road sockets, i.e., F North (resp., East, South), the rotation feature can be represented by the angles from the γ feature can be compared element-wisely after rotation normalization. Hence, the collected rotation feature for the γ junctions is a vector of length N road feature can be compared element-wisely after rotation normalization. Hence, the collected rotation feature for the γ junctions is a vector of length N. For example, in Fig. 1, Fctrl = signal and Fctrl = stop.

Finally, the crosswalk feature, denoted as F_j^xwlk, indicates whether there are crosswalks in a junction. Hence, F_j^xwlk ∈ \{True, False\}.

Therefore, we have the following definition.

**Definition 1:** The junction feature of a junction J_j, denoted as F_j, is a vector defined as:

\[
F_j = [F_j^road, F_j^ctrl, F_j^xwlk],
\]

where F_j^road ∈ \{3, 4, \ldots\} and F_j^ctrl ∈ \prod_{i=1}^{r} \{0, 2\pi\} is a vector of length F_j^road.

**C. Extraction of Feature Vectors**

According to Definition 1, we describe how to extract the features from a set of HD maps.

First, since the road, the control, and the crosswalk features are discrete variables, we can obtain all their values by checking the junctions in each map. Therefore, the sets of all possible values of the three features of a given map are denoted as F^road, F^ctrl, and F^xwlk, respectively.

Second, as the rotation feature contains continuous variables, we collect from the HD maps the minimal and the maximal values of each element in the rotation feature. Note that the rotation features from the junctions with the same road feature can be compared element-wisely after rotation normalization. Hence, the collected rotation feature for the junctions with n connected roads can be described as:

\[
F^rot_n = [[\alpha_{1 \min}, \alpha_{1 \max}], \ldots, [\alpha_{n \min}, \alpha_{n \max}]],
\]

where \(\alpha_{i \min} = \min_{J_j \in J} F_j^rot[i], \alpha_{i \max} = \max_{J_j \in J} F_j^rot[i], J^n = \{J_j : F_j^road = n\}, \text{ and } F_j^rot[i] = \alpha_i \) with \(F_j^rot = [\alpha_1, \ldots, \alpha_n]\).

Hence, the map feature of a set of maps is described as \(F = \bigcup_{n \in F^road} F^rot_n \times F^ctrl \times F^xwlk\).

**D. Map Construction Based on Feature Vectors**

During the map construction with the junctions generated by the feature vectors, we need to ensure 1) all junctions are connected, 2) the generated map emulates urban road networks, and 3) no road or junction overlaps, which makes the construction process challenging. In this paper, we propose a greedy method to construct a grid-layout map iteratively. The details are shown in Algorithm 1.

Given a set of HD maps \(M\), by checking all junctions of the maps, we can extract the features \(F^road, F^ctrl\), and \(F^xwlk\). Before constructing the map, we initialize a grid point container \(P_e = \{P_{0,0}\}\) for holding empty points, another grid point container \(P_f = \emptyset\) for filled points and a set \(J = \emptyset\) for storing sampled junction features (Line 1). For each vector \([F_j^road, F_j^ctrl, F_j^xwlk]\) in \(F^road \times F^ctrl \times F^xwlk\), we perform multivariate uniform sampling in the region...
Algorithm 1: Feature-based HD Map Generation

Input: The set of junction features: \( F = \bigcup_{n \in F_{\text{road}}} F_{\text{rot}}^n \times F_{\text{ctrl}} \times F_{\text{xwlk}} \).

Output: A new map \( M_c \).

1. Initialize the map \( M_c = \emptyset \), the sets of empty grid points \( P_n = \{ P_{0,y} \} \) and filled grid points \( P_f = \emptyset \), an empty junction feature set \( J = \emptyset \);
2. \( j = 0 \);
3. for each \( n \in F_{\text{road}} \) do
   4. for each \( \text{control} \in F_{\text{ctrl}} \) do
      5. for each \( \text{crosswalk} \in F_{\text{xwlk}} \) do
         6. \( F_{\text{road}}^j = n \);
         7. \( F_{\text{rot}}^j = \{ \alpha_i : \alpha_i \in U_{F_{\text{rot}}^n[i][0], F_{\text{rot}}^n[i][1]} \} \), \( \forall i \in \{1, \ldots, n\} \);
         8. \( F_{\text{ctrl}}^j = \text{control} \);
         9. \( F_{\text{xwlk}}^j = \text{crosswalk} \);
         10. \( J_j = [F_{\text{road}}^j, F_{\text{rot}}^j, F_{\text{ctrl}}^j, F_{\text{xwlk}}^j] \);
         11. \( J = J \cup \{ J_j \} \), \( j = j + 1 \);
   12. while \( J \neq \emptyset \) do
       13. Select a junction feature \( J_j \) and \( J = J \setminus \{ J_j \} \);
       14. Find \( F_{\text{road}}^j \)'s best match grid point in \( P_c \), say \( P_{x,y} \);
       15. \( M_c = M_c \cup \{ (P_{x,y}, J_j) \} \);
       16. \( P_c = P_c \setminus \{ P_{x,y} \} \), \( P_f = P_f \cup \{ P_{x,y} \} \);
       17. if \( P_{x+1,y} \notin P_c \cup P_f \) then \( P_c = P_c \cup \{ P_{x+1,y} \} \);
       18. if \( P_{x,y+1} \notin P_c \cup P_f \) then \( P_c = P_c \cup \{ P_{x,y+1} \} \);
       19. if \( P_{x-1,y} \notin P_c \cup P_f \) then \( P_c = P_c \cup \{ P_{x-1,y} \} \);
       20. if \( P_{x,y-1} \notin P_c \cup P_f \) then \( P_c = P_c \cup \{ P_{x,y-1} \} \);
       21. return \( M_c \).

\[
\prod_{i=1}^{n}[F_{\text{rot}}^n[i][0], F_{\text{rot}}^n[i][1]]
\]

to generate a junction feature \( F_{\text{rot}}^j = [F_{\text{road}}^j, F_{\text{rot}}^j, F_{\text{ctrl}}^j, F_{\text{xwlk}}^j] \) (Lines 3 - 10). The sampled junction features are stored in \( J \) (Line 11), each of which will be used to instantiate an individual junction in the grid layout.

Next, for each junction feature \( J_j \), we search for the empty grid points in \( P_c \) to select the grid point \( P_{x,y} \) of the highest matching score with \( J_j \). The matching score is heuristically calculated in the following way:

Let \( k \) be the maximal number of roads that \( J_j \) at \( P_{x,y} \) can connect after proper rotation with the surrounding junctions (if available) at \( P_{x+1,y}, P_{x,y+1}, P_{x-1,y}, \) and \( P_{x,y-1} \). If \( F_{\text{road}}^j > k \), then the feature and the point does not match and the score is set to \(-\infty\). Otherwise, the score is set to \( k \). Random selection is performed when a draw occurs.

After the grid point, say \( P_{x,y} \), is selected for \( F_{\text{road}}^j \), and the corresponding junction is instantiated at \( P_{x,y} \) (Line 14-15). The junction feature is removed from the feature set (Line 13) and the grid is extended in the four regular directions (i.e., East, North, West, and South) from \( P_{x,y} \) free for new junction assignments (Line 17-24). After all the junctions are instantiated, the roads connecting junctions at neighboring grid points can be constructed via road socket vectors (controlled by \( F_{\text{rot}}^j \)). The start point (resp. end point) of road lanes can then be obtained by shifting the start point (resp. end point) of the road's reference path along the perpendicular direction of the start heading (resp. end heading) of the road. After the road lanes are built, the junction lanes can be added by connecting the incoming and outgoing lanes of the junction. Next, traffic controls and crosswalks are created at appropriate locations. Lastly, we can output the HD map files based on \( M_c \).

Note that to make the generated HD map as concise as possible while preserving potential scenario diversity, we set the following default configurations for \textit{Feat2Map}:

- Given two junctions with the same junction feature, but one contains a one-way road while the other contains only two-way roads, any testing scenario generated from the former can also be generated from the latter one. Hence, all roads are two-way to keep the generated map concise.
- To minimize the number of junctions in the generated map, we assume that all roads are fully connected at junctions. It means there exists at least one navigation path (i.e., junction lane) connecting each road pair.
- The crosswalks are instantiated within the range of junctions and only overlap with junction lanes instead of road lanes. The signals are placed in front of the road it controls (facing the road), and the stop signs are placed on the right side of the road (facing the road).
- The default grid-gap is 100 meters (e.g., \( P_{x+1,y} \) is 100 meters to the east of \( P_{x,y} \), lane width is 3.5 meters (default value of selected input maps), and crosswalk width is 4.5 meters (slightly longer than a regular sedan).

TABLE I: Features extracted from San Francisco map

| Feature     | Value                  |
|-------------|------------------------|
| Road        | [-143.45, 200.92]      |
| Ctrl        | [-61.40, 126.22]       |
| XWLK        | [-5.26, 13.61]         |
| AP          | [-109.24, -87.91]      |

| Feature     | Value                  |
|-------------|------------------------|
| Road        | [-143.45, 200.92]      |
| Ctrl        | [-61.40, 126.22]       |
| XWLK        | [-5.26, 13.61]         |
| AP          | [-109.24, -87.91]      |

(a) Input map. (b) \( M_1 \). (c) \( M_2 \). (d) \( M_3 \).

Fig. 3: (a) The input San Francisco map; (b)–(d) three generated HD maps \( M_1, M_2, \) and \( M_3 \).

IV. EXPERIMENTS

We conduct comprehensive experiments in this section to answer the following research questions.

\textbf{RQ1:} Can \textit{Feat2Map} generate complete HD maps connecting all the intersections from sampled features correctly?

\textbf{RQ2:} Can \textit{Feat2Map} take a single input map and generate concise maps of the same level of scenario diversity as the input?

\textbf{RQ3:} Can \textit{Feat2Map} take multiple input maps and generate concise maps of equal or greater scenario diversity than individual input?

\textbf{RQ4:} Can \textit{Feat2Map} take user-specified features and generate customized maps?
A. Experimental Settings

To demonstrate the effectiveness of FEAT2MAP, we implement a prototype of FEAT2MAP and conduct experiments on the latest Apollo 7.0 [14] stack with three of the largest HD maps available for Apollo, i.e., SanFrancisco [18], GoMentum [19] and Shalun [20]. We define two metrics to measure scenario diversity. The first one is line coverage, which describes how many statements in the ADS have been executed by the scenarios. Two scenarios are considered different if they cover different statements. The other one is behavior coverage, which describes different behavior transitions during the execution of a scenario.

To answer RQ1, we run FEAT2MAP on SanFrancisco and perform qualitative analysis on three generated maps $M_{1-3}$. To answer RQ2, we conduct quantitative analysis by comparing the coverage results on SanFrancisco and $M_{1-3}$. To answer RQ3, we generate a new map $M_4$ from SanFrancisco [18], GoMentum [19] and Shalun [20] and conduct the same experiments as in RQ2. To answer RQ4, we illustrate the construction of customized maps from manual features.

All the experiments are conducted on a desktop computer (with i7-6850K CPU, 64GB RAM, and GTX 1080T Ti). When an HD map is generated, Dreamview (Apollo’s full-stack HMI service) [21] is restarted to load the new map. We run the route coverage testing [4] in Apollo’s built-in sim-control mode [22], which does not require any third-party simulators such as SVL [10]. The input of the planning component is published using simulated dummy data. Specifically, sim-control publishes the results of localization, chassis, and prediction, while our testing script submits routing requests and publishes traffic light information at runtime.

B. RQ1: Qualitative Analysis on Map Generation

SanFrancisco map contains 91 junctions (including 88 signal-controlled and three stop-sign controlled), 164 roads, and 1524 lanes (including 868 junction lanes and 656 road lanes); there are 13 junctions with crosswalks.

Due to randomness in the rotation sampling of road socket vectors, the extracted feature set $\mathcal{F}$ can generate unlimited

Fig. 4: The detailed junctions of the map in Fig. 3(b).

C. RQ2: Quantitative Analysis of Scenario Diversity

To collect the line coverage, we instrument Apollo’s planning component during compilation using the coverage command [23] provided by its build tool Bazel [24] as the scenario handling mechanisms are implemented in the planning component. Fig. 5 shows the coverage results of our experiments on the four maps in Fig. 3. Fig. 5(a) shows the accumulated code coverage results on different maps. The accumulated lines covered eventually are 8275 (SanFrancisco), 8232 ($M_1$), 8205 ($M_2$) and 8238 ($M_3$) respectively. As we can see, SanFrancisco has many junctions covering the same lines of code (highlighted in light blue), resulting in a significant amount of duplicated test cases. On the other hand, unique lines of code get covered in every junction in $M_{1-3}$. It only takes eight junctions in $M_{3-3}$ to reach a similar level of code coverage. Notice that SanFrancisco has slightly higher code coverage than $M_{1-3}$. Based on the source code-level investigation, the
Fig. 6: Scenarios and stage transitions covered in SanFrancisco and $M_{1-3}$.

uniquely covered codes in SanFrancisco are related to human errors in the HD map file (e.g., the id of some crosswalk elements are missing). Such human errors do not exist in maps generated by Feature2Map automatically. Besides the map error, we also discovered some potential issues of the planning component (reported at [25], [26]). Such issues, encountered in different situations (e.g., on roads or inside junctions), trigger different error handling mechanisms implemented in the planning component. This is the main reason for the slight difference in the line coverage results of $M_{1-3}$.

Apollo implements its planning component as a modularized state machine, where states align with common traffic scenarios such as intersection cruising, emergency stops, etc. and publishes its internal scenario selections (e.g., LaneFollow and TrafficLightProtected) and stage transitions (e.g., Approach and IntersectionCruise) within each scenario.

The coverage results of the stage transitions are shown in Fig. 5(b). From the results, we can find that many junctions belong to the same scenario in Apollo’s view of point. As a result, the same strategies are applied by Apollo to navigate through them (highlighted in light blue as well). On the other hand, the eight junctions in $M_{1-3}$ reach the same level of coverage as that of 91 junctions in the SanFrancisco map. Specifically, there are 16 scenario types defined in Apollo [27], out of which five scenarios and their stages are relevant to the experiments [28]:

- Lane Follow
- Stop Sign Unprotected
- Traffic Light Protected
- Traffic Light Unprotected Left Turn
- Traffic Light Unprotected Right Turn

The relevant scenarios and their stage transitions are plotted in Fig. 6, where the nodes represent the stages and arrows are possible transitions. The stages and transitions covered by SanFrancisco and $M_{1-3}$ are the same and highlighted in orange, while the uncovered stages are rendered grey.

The four paths covered are:

- $LF \rightarrow S_{PS} \rightarrow S_{S} \rightarrow S_{C} \rightarrow S_{I} \rightarrow LF$
- $LF \rightarrow L_{A} \rightarrow L_{C} \rightarrow LF$
- $LF \rightarrow L_{A} \rightarrow L_{C} \rightarrow L_{I} \rightarrow LF$
- $LF \rightarrow S_{PS} \rightarrow S_{S} \rightarrow S_{C} \rightarrow S_{I} \rightarrow LF$

The stage transitions align with our testing setup, starting and ending at road lanes (i.e., $LF$). The covered scenarios also match with the features of the generated junctions. One may notice that the right turn-related scenarios are not covered. This is because Apollo only enters the right turn scenario when the ADS is turning right under the red traffic light. When the traffic light is green, which is always the case during our sim-control simulation, Apollo treats right turning the same as driving straight across the junction (i.e., $T_{A}$ and $T_{C}$).

D. RQ3: Map Generation From Multiple Maps

We generate map $M_{4}$ with multiple input maps, i.e., SanFrancisco, GoMentum, and Shalun. The experiment results are listed in Table II. The features extracted from Shalun are not much different from SanFrancisco. However, due to the existence of the bare junctions (i.e., no controls) in GoMentum, the merged control feature is $F_{ctrl} = \{signal, stop, bare\}$, which results in 12 junctions to be generated in $M_{4}$. The code coverage of $M_{4}$ is similar to those of $M_{1-3}$. We find that Apollo 7.0 treats bare junctions as normal roads, hence no significant difference in the code coverage. However, a new path consisting of only one stage $LF$ (i.e., no junction scenarios are triggered) is covered because of the existence of bare junctions.

In summary, experiments with single and multiple input maps have demonstrated the effectiveness of Feature2Map in feature-based map generation.

E. RQ4: Map Generation from User-Defined Features

The generation capability of Feature2Map is not limited to generating junctions from the existing maps, such as those shown in Fig. 4. Feature2Map also accepts manually defined feature values if users require customization. For example, the mega junction shown in Fig. 7(a) is generated by specifying the following junction feature: $F_{road}^{permega} = 5$, $F_{ctrl}^{permega} = signal$, $F_{walk}^{permega} = True$ and $F_{rot}^{permega} = [0^\circ, 72^\circ, 144^\circ, -144^\circ, -72^\circ]$. With the cubic Bézier curve model, Feature2Map can also generate a set of roads connected end-to-end by specifying a
list of road configurations where each road’s end point (resp. end heading) is the next road’s start point (resp. start heading). Finally, in addition to the default two-way roads, one-way roads are also supported by FEAT2MAP as long as users specify the road type of each road socket. Fig. 7(c) shows a T-junction where the road types associated to the sequence of the road sockets, started from the east most road socket and listed in a counterclockwise direction, is [In, Out, InOut], where In, Out, InOut mean one-way incoming, one-way outgoing, and two-way roads, respectively.

V. DISCUSSION AND CONCLUSION

Discussion. Scenario-based testing is inevitable across the entire phase of ADS development. How diverse and representative the testing scenarios are essentially determines the testing efficiency and effectiveness. Bagchi et al. [29] proposes a 5-layer model for testing scenarios, including road level (layer 1), traffic infrastructure (layer 2), manipulation of layer 1 and 2 (layer 3), object (layer 4) and environment (layer 5). A recent survey [2] shows that most works focus on sampling critical scenarios at layer 4 and layer 5, i.e., by controlling the trajectories of other traffic participants such as vehicles or pedestrians or manipulating the simulated weather. However, the topological and geometrical features of the traffic networks (i.e., layer 1-3), which we believe are the foundation of testing scenarios, are often overlooked. As a result, HD map generation based on required features substantially lifts the restrictions induced by the fixed elements of the first three layers and creates huge potential in generating diverse scenarios.

Conclusion. In this paper, we propose an automatic feature-based HD map generation framework FEAT2MAP for simulation-based testing. The framework takes in existing maps as input, extracts essential features, and applies combinatorial sampling to generate grid-layout HD maps. FEAT2MAP also accepts manual feature configurations to allow flexible customization. Experiment results show that the generated HD map is concise and preserves comparable scenario diversity.

The topological and geometrical features of the roads and junctions can be far more complex in real life than those considered here. Our framework can be extended to more complex features such as filter lanes in mega junctions, crosswalks on road lanes, and highway traffic networks. We leave the exploration of them for future work.

REFERENCES

[1] ISO, “IS0/PAS 21448: Road vehicles – Safety of the intended functionality,” International Organization for Standardization, 2019.
[2] Z. Zhong, Y. Tang, Y. Zhou, V. d. O. Neves, Y. Liu, and B. Ray, “A survey on scenario-based testing for automated driving systems in high-fidelity simulation,” arXiv preprint arXiv:2112.00964, 2021.
[3] A. Gambi, M. Mueller, and G. Fraser, “Automatically testing self-driving cars with search-based procedural content generation,” in Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis, 2019, pp. 318–328.
[4] Y. Tang, Y. Zhou, F. Wu, Y. Liu, J. Sun, W. Huang, and G. Wang, “Route coverage testing for autonomous vehicles via map modeling,” in 2021 IEEE International Conference on Robotics and Automation (ICRA). IEEE, 2021, pp. 11450–11456.
[5] Y. Tang, Y. Zhou, Y. Liu, J. Sun, and G. Wang, “Collision avoidance testing for autonomous driving systems on complete maps,” in 2021 IEEE Intelligent Vehicles Symposium (IV). IEEE, 2021, pp. 179–185.
[6] Y. Tang, Y. Zhou, T. Zhang, F. Wu, Y. Liu, and G. Wang, “Systematic testing of autonomous driving systems using map topology-based scenario classification,” in 2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE). IEEE, 2021, pp. 1342–1346.
[7] TomTom, “HD Map,” https://www.tomtom.com/products/hd-map/, 2022, online; accessed Feb 2022.
[8] HERE Technologies, “HERE HD Live Map,” https://www.here.com/platform/HD-live-map, 2022, online; accessed Feb 2022.
[9] 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.
[10] G. Rong, B. H. Shin, H. Tabataebe, Q. Lu, S. Lemke, M. Molzko, E. Boise, G. Uhm, M. Gerow, S. Mehta, E. Agafonov, T. H. Kim, E. Sterner, K. Ushiroda, M. Reyes, D. Zelenkovsky, and S. Kim, “SVL Simulator: A High Fidelity Simulator for Autonomous Driving,” arXiv e-prints, p. arXiv:2005.03778, May 2020.
[11] SUMO, “netedit - sumo documentation,” https://sumo.dlr.de/docs/Netedit/index.html, 2022, online; accessed Feb 2022.
[12] CommonRoad, “Commonroad scenario designer,” https://commonroad.in.tum.de/scenario-designer, 2022, online; accessed Feb 2022.
[13] L. Mi, H. Zhao, C. Nash, X. Jin, J. Gao, C. Sun, C. Schmid, N. Shavit, Y. Chai, and D. Anguelov, “Hdmapgen: A hierarchical graph generative model of high definition maps,” in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2021, pp. 4227–4236.
[14] Baidu, “Apolloauto/apollo: An open autonomous driving platform,” https://github.com/ApolloAuto/apollo, 2022, [Online; accessed 21-Feb-2022].
[15] K. Chellapilla, “Rethinking maps for self-driving,” https://medium.com/lyftlevel5/https-medium-com-lyftlevel5-rethinking-maps-for-self-driving-1472c4758db6, Oct. 2018.
[16] ASAM, “Asam opendrive,” https://www.asam.net/standards/detail/opendrive, 2021, online; accessed Feb 2022.
[17] Wikipedia contributors, “Bézier curve — Wikipedia, the free encyclopedia,” https://en.wikipedia.org/wiki/index.php?title=B%C3%A9zier_curve&oldid=1072458605, 2022, [Online; accessed 21-February-2022].
[18] LG Electronics, “San francisco — gsvl simulator content store,” https://content.lgsvlsimulator.com/maps/sanfrancisco, 2020, [Online; accessed 21-February-2022].
[19] ———, “Shalun — gsvl simulator content store,” https://content.lgsvlsimulator.com/maps/gomentum, 2020, [Online; accessed 21-February-2022].
[20] ———, “Shalun — gsvl simulator content store,” https://content.lgsvlsimulator.com/maps/shalun, 2020, [Online; accessed 21-February-2022].
[21] Baidu, “Dreamview,” https://github.com/ApolloAuto/apollo/tree/master/modules/dreamview, 2022, [Online; accessed 21-February-2022].
[22] ———, “Dreamview faqs,” https://github.com/ApolloAuto/apollo/blob/266fa6b168db3f4a7a12f8b950223f5bab2c0/docs/FAQs/Dreamview_FAQs.md#how-do-i-turn-on-sim-control, 2020, [Online; accessed 21-February-2022].
[23] Bazel, “Command-line-reference,” https://bazel.build/reference/command-line-reference#coverage, 2022, [Online; accessed 21-February-2022].
[24] ———, “Bazel,” https://bazel.build/, 2022, [Online; accessed 21-February-2022].
[25] Baidu, “[planning] failed to make a driving plan during a junction turn 14327,” https://github.com/ApolloAuto/apollo/issues/14327, 2022, [Online; accessed 1-March-2022].
[26] ———, “[planning] planning error: failed to shrink routing segment 14306,” https://github.com/ApolloAuto/apollo/issues/14306, 2022, [Online; accessed 1-March-2022].
[27] ———, “apollo/planning/ontology.proto,” https://github.com/ApolloAuto/apollo/blob/master/modules/planning/proto/planning_config.proto, 2022, [Online; accessed 1-March-2022].
[28] ———, “apollo/scenario_manager.cc,” https://github.com/ApolloAuto/apollo/blob/master/modules/planning/scenarios/scenario_manager.cc, 2022, [Online; accessed 1-March-2022].
[29] G. Bagschik, T. Menzel, and M. Maurer, “Ontology based scene creation for the development of automated vehicles,” in 2018 IEEE Intelligent Vehicles Symposium (IV). IEEE, 2018, pp. 1813–1820.