Efficient Arc Spline Approximation of Large Sized Complex Lane-Level Road Maps

Jinhwan Jeon and Seibum B. Choi, *Member, IEEE*

***Abstract*—The rapid advancement of autonomous driving and ADAS technologies has increased the demand for high-definition (HD) lane-level maps that accurately preserve rich geometric information while scaling to city-wide coverage. While traditional polyline-based formats are widely used, they struggle to provide continuous representations of key geometric properties such as curvature and heading angle, which are essential for autonomous driving applications. Curve-based representations have been introduced to address these limitations, but existing methods are often restricted to simplified or sparsely connected road networks, limiting their effectiveness in large-scale, real-world environments.**

This study presents an arc-spline-based lane representation framework that efficiently models complex, large-sized lane-level maps while preserving continuous road geometry. To achieve this, we introduce a novel road network decomposition and merging method that enables structured parameterization without requiring full map-scale optimization. Instead, optimization is localized to cluster connection regions, significantly enhancing computational efficiency. Validation using lanelet maps from the nuScenes dataset demonstrates that our approach maintains an average approximation error of 4.3 cm while preserving detailed lane topology and global tangential continuity, while also achieving a significant reduction in storage requirements compared to conventional polyline formats.

***Index Terms*—Lane-level maps, Road network, Arc spline approximation, Graph partitioning, Divide and conquer**

I. Introduction

With the continuous advancement of vehicle navigation technology, HD maps have evolved to store not only large-scale road network data but also essential geometric properties for autonomous driving. These maps are critical for improving localization, navigation, and decision-making in modern autonomous systems.

As autonomous vehicles operate in increasingly large and complex environments, the ability of HD maps to represent intricate lane structures accurately and efficiently becomes essential. Leading commercial providers such as TomTom [1] and HERE [2], along with mapping standards like OpenStreetMap (OSM) [3] and Lanelet2 [4], commonly use **polyline-based representations**, where road geometry is defined by sequences of discrete points. While flexible and compatible with existing frameworks, such representations lack **continuous geometric properties** like curvature, heading angle, and tangent continuity, which are critical for trajectory planning and control [5]. Moreover, polyline maps do not inherently support arc-length parameterization, making uniform path sampling and motion smoothing difficult.

Jinhwan Jeon and Seibum B. Choi are with the Department of Mechanical Engineering, Korea Advanced Institude of Science and Technology, Daejeon 34141, e-mail: (jordan98@kaist.ac.kr, sbchoi@kaist.ac.kr).

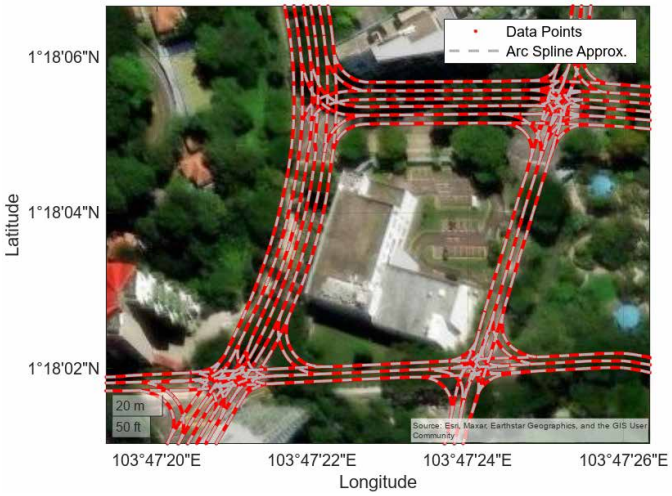


Figure 1. Exemplary arc spline approximation result (gray dash line) on the **One North** region of the nuScenes HD map. The proposed framework accurately and efficiently parameterizes large-scale HD maps, including complex urban intersections and curvy roads.

To overcome these limitations, **curve-based representations** have been proposed, offering improved geometric continuity and compactness. Various parametric curve approaches have been studied, including piecewise polynomials [6], [7], Bézier curves [8], Hermite splines [9], B-splines [10], Clothoids [11]–[14], and Arc splines [15]–[17].

Curve-based map formats have also been adopted in industry. For example, ASAM OpenDRIVE [18] supports various curve types such as arcs, spirals, and polynomials, moving beyond simple line segment representations. However, it requires users to manually provide curve parameters and lacks an automated process for converting point-based polyline data into curve representations, leaving **point-to-curve parameterization** as an open challenge. A comparison of related studies and HD map formats is presented in Table I.

Table I categorizes prior work based on **map size** and **road network complexity**. We define small maps as < 2 km² (e.g., single intersections), medium as 2–30 km² (urban district scale), and large as > 30 km² (city-scale). Similarly, we define network complexity as low (minimal intersections), medium (basic intersections), and high (dense, lane-level interconnections), as illustrated in Fig. 1.

While some studies such as [16] addressed large-scale mapping, they focused on simple highway-like networks or lacked G^1 (tangential) continuity. Others [7], [9], [11], [13] targeted complex networks but operated on small-sized regions without detailed lane-level representations.

Table I
SUMMARY AND COMPARISON OF VARIOUS LANE MARKING PARAMETERIZATION METHODS

Works	Curve Type	Map Size	Complexity
[1]–[4]	P	Small - Large	Low - High
[18]	A, C, CP	Small - Large	Low - High
[6]	CP	Medium	Low
[7]	CP[1]	Medium	Medium
[8]	BE	Small	Low
[9]	CHS	Medium	Medium
[10]	BS	Small - Medium	Low
[11]	C	Small - Medium	Medium
[12]	C	Small - Medium	Low
[13]	C	Medium	Medium
[14]	C	Small - Medium	Low
[15], [16]	A	Small - Large	Low
[17]	A	Small - Medium	Low

A stands for Arc Spline, **BE** stands for Bézier Curve, **BS** stands for B Spline,**C** stands for Clothoid, **CHS** stands for Cubic Hermite Spline, **CP** stands for Cubic Polynomial and **P** stands for Polylines.

Maintaining G^1 continuity is especially important, as discontinuities in curve segments can degrade downstream tasks like motion planning and localization. Even polyline-based formats often apply post-processing to achieve tangential continuity [5].

Building upon the limitations of prior studies, this work proposes a lane marking parameterization framework that ensures global G^1 continuity while effectively handling large-size, complex lane-level maps. This goal presents two key challenges: achieving G^1-continuous parameterization for **interconnected lane networks** and ensuring computational efficiency for **large-sized maps**.

To address the first challenge, we extend our previous work [17], which focused on arc-spline approximation for simpler road networks, by introducing modifications that support more complex lane structures with multiple connections and intersections. For the second challenge—largely overlooked in previous research—we design an efficient approach to parameterize maps containing thousands of lane segments. Instead of solving a large-scale constrained optimization problem in a single step, which is computationally expensive and difficult to maintain, we propose a scalable solution that avoids re-optimizing the entire map upon updates or extensions.

A more efficient strategy is to divide the large-scale optimization problem into smaller, manageable subproblems and then merge the results. This approach, commonly known as a **divide-and-conquer algorithm**, models the lane-level road map as a lane marking connectivity graph $G = (V, E)$, where each vertex V corresponds to a **linestring** (i.e., a sequence of lane marking points), and edges E represent valid connections between linestrings. The overall procedure is described in Algorithm 1.

After parameterizing each linestring cluster independently, G^1 continuity must still be enforced between adjacent clusters (Line 6 of Algorithm 1). However, achieving global continuity is nontrivial, as the complexity of this final merging step depends heavily on how the original graph G is partitioned. A well-designed partitioning strategy can reduce the problem to a set of simple, localized optimizations, while a poor one may lead to expensive full-map optimization.

[1]Cubic Catmull-Rom Spline was utilized for intersection modeling

Algorithm 1 Divide-and-Conquer Approach for Map Parameterization

1: Extract the lane marking connectivity graph $G = (V, E)$
2: Partition G into smaller linestring clusters $\{G'_i\}$
3: **for** each cluster G'_i **do**
4: Independently parameterize G'_i
5: **end for**
6: Merge the parameterized results to construct the final representation

Graph partitioning is a widely used technique for dividing a graph into smaller subgraphs while minimizing edge cuts and balancing node distribution. This approach is particularly effective in large-scale computational tasks, where decomposition significantly improves efficiency. Various partitioning methods have been developed, each with characteristics suited to specific applications.

In the domain of road networks, graph partitioning has been applied to enhance routing and traffic simulation performance. For example, multilevel partitioning has proven effective for managing large-scale networks. METIS [19] achieved low edge cuts and improved shortest-path computations, while recursive bisection [20] enabled hierarchical decomposition for efficient route planning. Community detection approaches have also been explored: Zhou et al. [21] divided heterogeneous networks into homogeneous regions to optimize traffic flow, and Anwar et al. [22] applied spectral clustering with recursive bipartitioning for similar purposes. Yu et al. [23] further improved partitioning compactness using travel speed data. In addition, MFMC-based methods [24] have been used to detect network vulnerabilities, and graph components [25], though not traditional partitioning methods, have been employed to identify isolated regions and simplify processing.

While these studies focus on improving traffic-related tasks, our goal differs fundamentally. We aim to identify an optimal partitioning strategy that supports efficient merging of independently parameterized linestring clusters. Ideally, adjacent clusters could be merged by optimizing only a small subset of directly connected linestrings, avoiding re-parameterization of the entire cluster. This would yield substantial computational savings. However, conventional partitioning methods are not designed to preserve global G^1 continuity—an essential requirement in curve-based lane modeling.

For example, although graph components can isolate disconnected subgraphs, they are ineffective in densely connected maps like those in the nuScenes dataset, where the entire network often forms a single large component. Similarly, the **Louvain method** [26], a popular community detection-based partitioning approach, can divide a road network G into densely connected clusters. While more scalable than component-based partitioning, it still requires re-optimizing all linestrings within merging clusters to enforce G^1 continuity—resulting in a costly full-map optimization at the final step.

To overcome these limitations, we propose a novel road net-

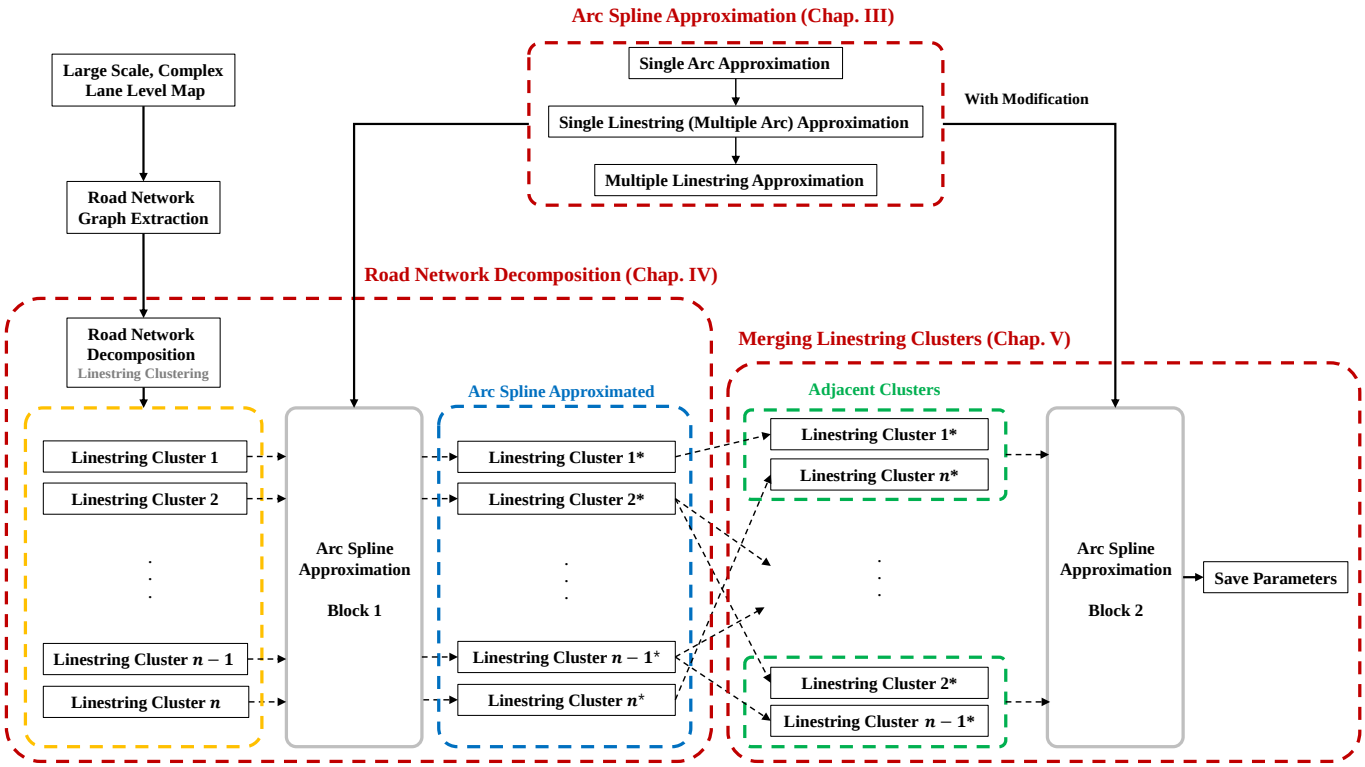


Figure 2. **Block diagram illustrating the full pipeline of the proposed arc spline approximation for large-scale, complex lane-level maps.** The process begins by decomposing the provided road network into linestring clusters, followed by arc spline approximation for each cluster(Block 1). Adjacent linestring clusters are then grouped and merged, ensuring G^1 continuity at connection points through an efficient re-approximation process(Block 2), eliminating the need for full map-scale optimization. Finally, the parameterized map is converted and saved in the conventional point-based format, with a reverse procedure for recovering arc parameters.

work decomposition method tailored to facilitate the merging process in Algorithm 1. Unlike traditional graph partitioning approaches that prioritize minimizing edge cuts, our method partitions the network in a way that supports localized merging. As a result, only a small set of adjacent linestrings must be optimized during merging, significantly reducing computation and enabling scalable parameterization for large maps, as well as easier future updates.

Our framework is primarily based on the Lanelet2 [4] format. However, since the core algorithm operates independently of map format, only the graph extraction module requires adaptation for use with alternative formats.

The contributions of our research can be summarized as follows.

1. An arc-spline approximation framework that transforms raw and noisy polyline lane data into multiple arc segments, even for complex and highly interconnected road networks.
2. A novel road network decomposition and merging strategy that ensures global G^1 continuity between adjacent arc segments, enabling scalable processing of large-size HD maps.

The structure of this paper is as follows: Section II provides an overview of the proposed parameterization pipeline for large-scale, complex lane-level maps. Section III discusses the advantages of arc-spline representations and extends our previous work [17] to support multi-linestring approximation in complex road networks. Section IV introduces a novel clustering algorithm for road network decomposition, and Section V presents an efficient merging strategy for adjacent clusters. This merging process incorporates additional constraints into the arc-spline framework from Section III, enabling localized optimization limited to directly connected linestrings, thereby improving computational efficiency. Section V also explains how parameterized maps are stored and how arc parameters can be recovered from saved data. Section VI provides experimental validation of the overall framework using real-world datasets. Finally, Section VII concludes the paper and suggests directions for future work.

II. Overview

The overall process of arc-spline approximation for large-sized, complex lane-level maps is illustrated in Fig. 2. The procedure begins with the decomposition of the road network into multiple linestring clusters (orange box) using the proposed method described in Section IV. Each cluster is then independently approximated using the arc-spline framework (Section III, blue box).

To ensure smooth transitions between adjacent clusters, physically connected clusters—those sharing common endpoints—are grouped into pairs (green boxes) for merging. Since the initial approximation is performed independently for each cluster, discontinuities may appear at the boundaries. To resolve this, we perform localized re-approximation (block 2)

only on the directly connected linestrings between adjacent clusters, as detailed in Section V. This avoids re-optimizing entire clusters and significantly improves computational efficiency. By eliminating discontinuities through this localized merging process, the framework ensures global G^1 continuity without requiring full map-scale optimization.

Finally, the arc-parameterized map is converted into a conventional point-based polyline format for storage. A reverse conversion procedure is also provided to recover arc parameters from the saved format, allowing seamless transitions between representations.

III. Arc Spline Approximation

In this section, we first explain the rationale behind selecting arc splines for lane parameterization. We then describe how our previous arc-spline approximation framework [17] has been extended in a structured, hierarchical manner—from fitting a single arc segment to approximating complex, interconnected linestrings using arc splines. Next, we discuss the limitations of this approach when applied to large-scale lane-level maps, highlighting the necessity of road network decomposition.

As illustrated in Fig. 2, the enhanced arc-spline approximation framework will be applied to the linestring clusters generated by the road network decomposition introduced in Section IV. Additionally, a constraint function will be introduced to ensure smooth transitions between adjacent linestring clusters, as discussed in Section V. Further details will be provided in the following sections.

A. Advantages and Practical Applications of Arc Splines

Among various curves considered for lane marking parameterization, arc splines offer several advantages that make them particularly well-suited for ADAS and autonomous driving applications [15]–[17]. A key property of arc segments (circular arcs) is their invariance under translation and rotation, allowing them to maintain geometric consistency across both global and local (ego vehicle) coordinate systems. In contrast, curves such as cubic splines may lose their original form when transformed, making map-based localization more challenging.

Another practical advantage is that the distance between a point and an arc segment can be computed in closed form [15], enabling efficient real-time localization using arc-spline-based maps. In addition, arc splines require only three control points for representation [17], resulting in a compact structure that integrates well with existing polyline HD map formats without major structural changes.

For these reasons, we adopt and further enhance the arc spline-based approach for lane marking parameterization in this study.

B. Single Arc Approximation

Among the various methods for parameterizing arcs, it is well-known that an arc can be defined using three control points by employing the rational Bézier curve format [27]. As shown in Fig. 3, two points ($\mathbf{A}_1$ and $\mathbf{A}_2$) are positioned at the start and end of the arc, respectively, with a third point ($\mathbf{C}$) located along the bisector of the line segment $\mathbf{A}_1\mathbf{A}_2$. To determine $\mathbf{C}$, we introduce a vector $\mathbf{v}$ orthogonal to the line segment $\mathbf{A}_1\mathbf{A}_2$, which can be computed using one additional variable k, as shown below.

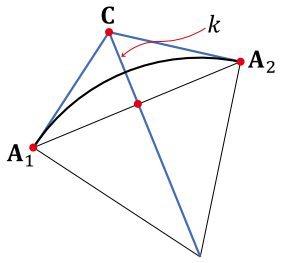


Figure 3. Single arc parameter configuration: Two arc nodes $\mathbf{A}_1$, $\mathbf{A}_2$ and a signed-distance parameter k

$$\begin{aligned} \mathbf{v} &= \frac{1}{\|\mathbf{A}_1 - \mathbf{A}_2\|_2} \cdot \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \cdot (\mathbf{A}_2 - \mathbf{A}_1) \\ \mathbf{C} &= \frac{1}{2}(\mathbf{A}_1 + \mathbf{A}_2) + k\mathbf{v} \end{aligned} \tag{1}$$

Each arc segment in this work is thus defined by two 2D points ($\mathbf{A}_1$, $\mathbf{A}_2$) and a signed-distance parameter k. While our previous work relied on three control points ($\mathbf{A}_1$, $\mathbf{A}_2$, and the midpoint of the arc) for parameterizing a single arc segment, we have reduced the dimensionality by eliminating one control point in this work.

As outlined in [17], when approximating data points with an arc segment, we optimize the variables $\mathbf{A}_1$, $\mathbf{A}_2$, and k by solving a Non-linear Least Squares (NLS) problem, which integrates two models: the **Anchor model** and the **Measurement model**.

The **Anchor model**, as its name suggests, anchors $\mathbf{A_1}$ and $\mathbf{A_2}$ to the starting and ending points, respectively, by penalizing the distance to these target points. If we define $\mathbf{P}_1$ and $\mathbf{P}_n$ as the starting and ending data points, the cost function for the Anchor model can be expressed as:

$$\mathcal{L}_{\mathrm{AC}} = \|\mathbf{P}_1 - \mathbf{A}_1\|^2_{\Sigma_{\mathrm{AC}}} + \|\mathbf{P}_n - \mathbf{A}_2\|^2_{\Sigma_{\mathrm{AC}}} \tag{2}$$

Here, the notation $\|\cdot\|^2_\Sigma = (\cdot)^\top \Sigma^{-1} (\cdot)$ represents the squared Mahalanobis distance, which can be interpreted as the square of a weighted Euclidean distance. The weight is incorporated by multiplying the inverse of the covariance matrix Σ_{AC} with the squared 2-norm of the original residual vectors $\mathbf{P}_1 - \mathbf{A}_1$ and $\mathbf{P}_n - \mathbf{A}_2$.

On the other hand, the **Measurement model** minimizes the discrepancy between data points and the arc approximation by adjusting k, thereby refining the position of the control point $\mathbf{C}$. The detailed derivation of the cost function for the Measurement model can be found in [17], so it is not repeated here. A key difference from our previous work is the introduction of a newly designed signed-distance parameter k, which is now applied to modify the original measurement model, as shown below.

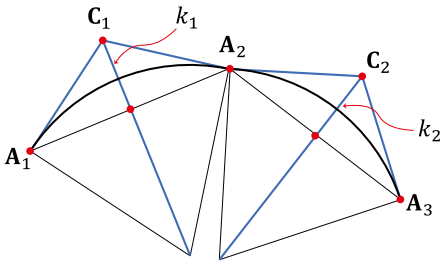


Figure 4. Parameter configuration for two arc segments: $\mathbf{A}_1$, $\mathbf{A}_2$, $\mathbf{A}_3$, k_1, k_2. While G^0 continuity is automatically achieved, an additional constraint model is required to ensure G^1 continuity.

$$\mathcal{L}_{\mathrm{ME}} = \sum_{i=1}^{n} \left\| \mathbf{r}^i_{\mathrm{ME}} \left(\mathbf{A}_1, \mathbf{A}_2, k, \mathbf{P}_i \right) \right\|^2_{\Sigma^i_{\mathrm{ME}}} \tag{3}$$

In this equation, $\mathbf{r}^i_{\mathrm{ME}}$ represents the arc approximation error for each data point $\mathbf{P}_i$, with covariance Σ^i_{ME}.

By combining the Anchor and Measurement models, the NLS problem for single arc approximation can be formulated as follows:

$$\begin{aligned} \min_{\mathbf{A}_1, \mathbf{A}_2, k} \mathcal{L} &= \mathcal{L}_{\mathrm{AC}} + \mathcal{L}_{\mathrm{ME}} \\ &= \left\| \mathbf{P}_1 - \mathbf{A}_1 \right\|^2_{\Sigma_{\mathrm{AC}}} + \left\| \mathbf{P}_n - \mathbf{A}_2 \right\|^2_{\Sigma_{\mathrm{AC}}} \\ &+ \sum_{i=1}^{n} \left\| \mathbf{r}^i_{\mathrm{ME}} \left(\mathbf{A}_1, \mathbf{A}_2, k, \mathbf{P}_i \right) \right\|^2_{\Sigma^i_{\mathrm{ME}}} \end{aligned} \tag{4}$$

Optimizing variables $\mathbf{A}_1$, $\mathbf{A}_2$, and k using MATLAB's 'lsqnonlin.m' function from the Optimization Toolbox [28], the best-fitting arc segment for the given data points can be obtained.

C. Single Linestring (Multiple Arc) Approximation

Typically, a single arc segment is insufficient to accurately approximate a sequence of data points (referred to as linestrings in Lanelet2 [4]). To improve accuracy, we represent the data using multiple arc segments.

Adjacent arc segments are constructed to share a common node, which inherently ensures point-wise (G^0) continuity. For example, as illustrated in Fig. 4, when two arc segments are connected, three arc nodes ($\mathbf{A}_1$, $\mathbf{A}_2$, $\mathbf{A}_3$) and two signed-distance parameters (k_1, k_2) are used as optimization variables.

To further ensure tangential (G^1) continuity between adjacent arc segments, we introduce a continuity constraint into the optimization formulation. In the example shown in Fig. 4, the G^1 condition is satisfied when the vectors $\overrightarrow{\mathbf{C}_1\mathbf{A}_2}$ and $\overrightarrow{\mathbf{A}_2\mathbf{C}_2}$ are parallel [29]. Extending this to a general case with m arc segments, the G^1 continuity constraint between the ith and $(i+1)$th segments can be expressed as follows.

$$\mathbf{r}_{\mathrm{Eq}}(i) = \frac{\left(\mathbf{A}_{i+1} - \mathbf{C}_i \right)^\top \left(\mathbf{C}_{i+1} - \mathbf{A}_{i+1} \right)}{\left\| \mathbf{A}_{i+1} - \mathbf{C}_i \right\| \left\| \mathbf{C}_{i+1} - \mathbf{A}_{i+1} \right\|} - 1 \ , \ \text{for } i = 1 : m \tag{5}$$

Note that control points $\mathbf{C}_i$ and $\mathbf{C}_{i+1}$ are computed using the optimization variables: $\mathbf{A}_i$, $\mathbf{A}_{i+1}$, $\mathbf{A}_{i+2}$, k_i and k_{i+1}.

By repeatedly applying the single arc approximation cost function (Equation 4) across multiple arc segments and integrating the newly introduced equality constraints from Equation 5, we can establish an optimization problem for the arc spline approximation of a single linestring consisting of multiple arc segments, as detailed below.

$$\begin{aligned} \min_{\mathbf{A}_1, \cdots \mathbf{A}_{m+1}, k_1, \cdots k_m} \mathcal{L} &= \mathcal{L}_{\mathrm{AC}} + \mathcal{L}_{\mathrm{ME}} \\ &= \left\| \mathbf{P}_1 - \mathbf{A}_1 \right\|^2_{\Sigma_{\mathrm{AC}_1}} + \left\| \mathbf{P}_n - \mathbf{A}_{m+1} \right\|^2_{\Sigma_{\mathrm{AC}_1}} \\ &+ \sum_{i=2}^{m} \left\| \mathbf{P}_{\mathrm{Idx}(i)} - \mathbf{A}_i \right\|^2_{\Sigma_{\mathrm{AC}_2}} \\ &+ \sum_{i=1}^{m} \sum_{j=\mathrm{Idx(i)}}^{\mathrm{Idx(i+1)}} \left\| \mathbf{r}_{\mathrm{ME}} \left(\mathbf{A}_i, \mathbf{A}_{i+1}, k_i, \mathbf{P}_j \right) \right\|^2_{\Sigma^j_{\mathrm{ME}}} \\ \text{s.t.} \quad & \mathbf{r}_{\mathrm{Eq}} = \mathbf{0} \end{aligned} \tag{6}$$

In Equation 6, both the **Anchor** and **Measurement models** are computed for all arc segments, while ensuring the vector function $\mathbf{r}_{\mathrm{Eq}}$ remains zero. The notation $\mathrm{Idx(i)}$ represents the index of the data point to which the first point of the ith arc segment is anchored. This is reflected in the **Anchor model** cost function: $\left\| \mathbf{P}_1 - \mathbf{A}_1 \right\|^2_{\Sigma_{\mathrm{AC}_1}} + \left\| \mathbf{P}_n - \mathbf{A}_{m+1} \right\|^2_{\Sigma_{\mathrm{AC}_1}} + \sum_{i=2}^{m} \left\| \mathbf{P}_{\mathrm{Idx}(i)} - \mathbf{A}_i \right\|^2_{\Sigma_{\mathrm{AC}_2}}$.

For the **Measurement model**, data points between indices $\mathrm{Idx(i)}$ and $\mathrm{Idx(i+1)}$ are approximated by the ith arc segment, assuming the points are well-ordered. This results in the **Measurement model** cost: $\sum_{i=1}^{m} \sum_{j=\mathrm{Idx(i)}}^{\mathrm{Idx(i+1)}} \left\| \mathbf{r}_{\mathrm{ME}} \left(\mathbf{A}_i, \mathbf{A}_{i+1}, k_i, \mathbf{P}_j \right) \right\|^2_{\Sigma^j_{\mathrm{ME}}}$.

For a fixed number of arc segments, the cost in Equation 6 is minimized with respect to the optimization variables $\mathbf{A}_1, \cdots, \mathbf{A}_{m+1}, k_1, \cdots, k_m$. If the approximation error remains high, an additional arc segment is introduced, and the optimization process is repeated until the error falls below a predefined threshold.

Based on this multiple arc spline approximation framework, a single linestring can be effectively approximated with multiple arc segments.

D. Multiple Linestring Approximation

A standard Lanelet map, as defined in [4], consists of multiple lanelets, each bounded by left and right linestrings and associated with regulatory elements. Therefore, when applying arc-spline approximation to lane-level maps, it is crucial to consider continuity across connected linestrings, rather than treating each in isolation, as discussed in Section III-C. The full process for arc-spline approximation of interconnected linestrings involves three key steps, illustrated in Fig. 5.

*1) **Individual Linestring Parameterization**:* Each linestring is first approximated independently using the method described in Section III-C. These initial results serve as the basis for enforcing global G^1 continuity across linestrings in subsequent steps.

*2) **Ensuring G^0 Continuity for Adjacent Linestrings**:* While G^1 continuity is enforced within each linestring during initial approximation, adjacent linestrings may not even satisfy

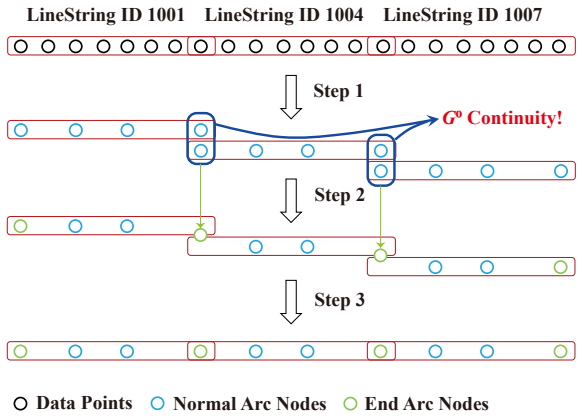


Figure 5. Arc-spline approximation process for multiple connected linestrings. Step 1: Independent approximation of each linestring (blue nodes: arc nodes). Step 2: Enforcing G^0 continuity by merging adjacent endpoints into shared end arc nodes (green nodes). Step 3: Re-optimization to ensure G^1 continuity across connected linestrings.

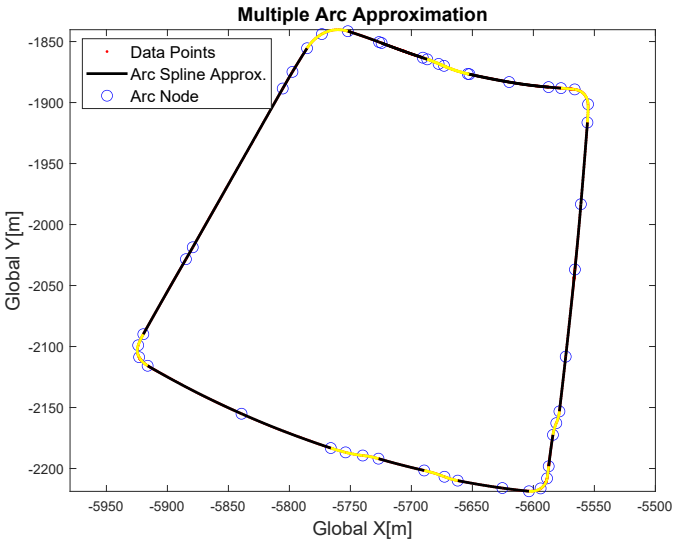


Figure 6. Example of arc spline approximation for multiple linestrings spanning approximately 1.35km. Black and yellow regions represent areas of low and high curvature, respectively, qualitatively demonstrating that our multiple linestring arc spline approximation framework (Section III-D) maintains consistent performance across varying road curvatures.

G^0 continuity due to misaligned endpoints. In this step, we merge the endpoints of connected linestrings by replacing the two separate arc nodes with a shared node, referred to as an end arc node. All other arc nodes are referred to as normal arc nodes.

*3) **Ensuring** G^1 **Continuity for Adjacent Linestrings**:* We then extend the optimization problem from Section III-C to enforce G^1 continuity not only within linestrings but also across their connections. The optimization variables include all arc node positions (both end and normal) and the signed-distance parameters k for each arc. The cost function and constraints are updated to reflect the G^1 condition across the entire connected network. Due to space limitations, the detailed formulation is omitted.

We solve this optimization using MATLAB's nonlinear least squares (NLS) solver [28], consistent with the previous sections. Fig. 6 shows an example of arc-spline approximation across multiple linestrings spanning approximately 1.35 km. The result demonstrates the framework's robustness across both low and high curvature regions.

E. Arc Spline Approximation for Large Maps

While the procedure described in Section III-D can be directly applied to large-scale lane-level maps in an offline setting, two key challenges arise in practice.

1) Time-Intensive Computation: As map size increases, the performance of the nonlinear least squares (NLS) solver degrades, resulting in significantly longer computation times—even when executed offline.

2) Inefficiency in Updates and Extensions: When road conditions change, updating the arc parameters for a single linestring requires re-solving the optimization problem for the entire network, even for unaffected or distant linestrings. This approach becomes increasingly inefficient as the map grows, complicating updates and extensions.

These limitations underscore the need for a more scalable approach. To address them, we adopt a **divide-and-conquer algorithm**, as introduced in the Introduction. By partitioning the map into manageable linestring clusters, the proposed method enables efficient parameterization while supporting faster and more localized updates.

The following section presents our road network decomposition framework, which partitions the map into smaller segments for efficient arc-spline approximation and streamlined maintenance. The multi-linestring arc-spline framework introduced in this chapter will be reused in both Sections IV and V, as illustrated in Fig. 2, with further details provided in the respective sections.

IV. Road Network Decomposition

This section introduces a novel road network decomposition algorithm designed to enable efficient arc-spline approximation for lane-level road networks. We also analyze the properties of the resulting linestring clusters, which will be used in Section V to demonstrate the scalability and efficiency of the proposed method in large and complex maps.

The decomposition algorithm utilizes a linestring connectivity graph G, constructed from the input road network, to identify appropriate clusters of lane marking linestrings. As described in Section II and illustrated in Fig. 2, the graph $G = (V, E)$ represents connectivity among linestrings, where each vertex V corresponds to a linestring and an edge E is added between two linestrings if one is reachable from the other.

A. Road Network Decomposition Strategy

As discussed in the Introduction, it is essential to avoid applying arc spline approximation to the entire map at once. Instead, the map should be divided into smaller linestring clusters, with the additional requirement that merging adjacent clusters does not necessitate re-optimizing all arc segments.

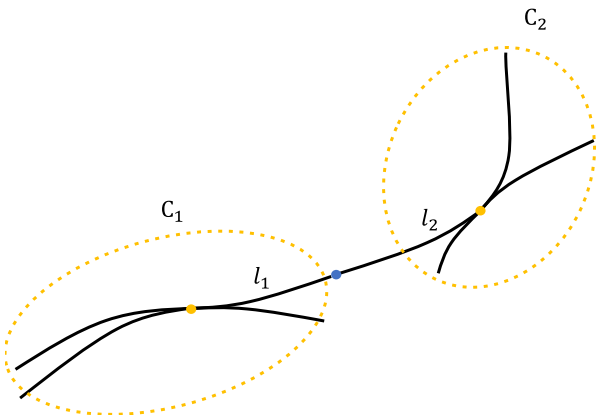


Figure 7. Road network decomposition strategy for efficient merging. When clusters $\mathbf{C}_1$ and $\mathbf{C}_2$ are connected through a simple linear region, only the boundary linestrings l_1 and l_2 require re-approximation with positional and heading constraints (orange endpoints). This avoids re-optimizing all arc segments in both clusters. Therefore, the road network should be decomposed such that cluster connections occur only at simple linear regions.

To support this, we observe that most road geometries—aside from complex regions such as intersections or roundabouts—consist of relatively simple linear segments. If the decomposition algorithm ensures that adjacent clusters are connected only through such linear segments, the merging process becomes significantly more efficient.

For example, consider two adjacent clusters $\mathbf{C}_1$ and $\mathbf{C}_2$, each independently approximated using arc splines. As illustrated in Fig. 7, ensuring G^1 continuity between them only requires re-optimizing the boundary linestrings l_1 and l_2, with position and heading constraints applied at their shared endpoints (orange nodes). All other linestrings within $\mathbf{C}_1$ and $\mathbf{C}_2$ remain unaffected, significantly reducing the overall computational cost.

To enable decomposition scenarios similar to Fig. 7, we propose **three primary clustering rules** for forming lane marking linestring clusters in road networks:

Rules for Clustering of Linestrings

1. If a node is shared by three or more linestrings, all of those linestrings belong to the same cluster.
2. If a node is shared by two linestrings that are not directly connected by a navigable vehicle path, those linestrings are assigned to the same cluster.
3. Linestrings cannot be shared between different clusters: If two clusters share a linestring, they must be merged into a single cluster.

A key distinction of our road network decomposition method lies in its objective. Unlike conventional graph partitioning approaches that cluster elements based on similarity, our method focuses on determining which linestrings *should* or *should not* belong to the same cluster. This decision is guided by the principle that adjacent clusters should connect only through simple linear regions, as illustrated in Fig. 7.

To achieve this, linestrings connected via simple linear transitions are placed into separate clusters, ensuring that cluster boundaries occur at clearly defined junctions (Fig. 7). In contrast, linestrings with complex interconnections are grouped within the same cluster.

A key classification rule addresses cases where two linestrings share a node. If the connection is linear—such as the link between l_1 and l_2 in Fig. 7—they are assigned to different clusters. However, in rare cases where two linestrings are not in the same path (i.e., represent diverging directions or unconnected sections), they are placed in the same cluster. This rule ensures that only straightforward linear transitions trigger cluster separation, while more intricate structures remain grouped.

By traversing the road network and applying these clustering rules at each linestring's endpoints, we classify linestrings into two categories. Those that follow the three rules form **Type A** clusters. Remaining linestrings that cannot be clustered using these rules, but are still connected, are grouped into **Type B** clusters. An example of this decomposition process is illustrated in Fig. 8.

B. Properties of Linestring Clusters

As discussed earlier, efficient parameterization of large-scale, complex lane-level maps requires clustering lane marking linestrings such that global map-scale optimization is avoided during the final merging step. Before analyzing how the proposed linestring clusters (Types A and B) support this, we first examine their structural characteristics.

Type A clusters are formed strictly based on the three rules introduced earlier, and thus their properties are directly derived from those rules.

In contrast, Type B clusters consist of linestrings that do not satisfy the clustering conditions for Type A and therefore remain unassigned. These remaining linestrings are grouped together into Type B clusters. The following proposition outlines the key properties that distinguish Type B clusters from Type A.

Proposition 1 (**Type B** Property)**.** *All lane marking linestrings within a **Type B** linestring cluster are connected in series, with no nested structures.*

Proof. Any nested structure necessarily includes at least one node connected to three or more linestrings. According to **Rule 1**, such a configuration satisfies the condition for forming a Type A cluster.

Therefore, if a cluster contains a nested structure, it must include at least one Type A subcluster. This contradicts the assumption that the entire structure can be classified as a Type B cluster. Hence, Type B clusters cannot contain nested structures and must consist solely of linestrings connected sequentially in a non-branching manner. □

Illustrative Example: As shown in Fig. 9, when the green linestrings are ignored, the remaining linestrings in clusters (a) and (b) form simple series connections that do not satisfy **Rules 1 and 2**, thus qualifying as Type B clusters. However, when the green linestrings are included, a nested structure is

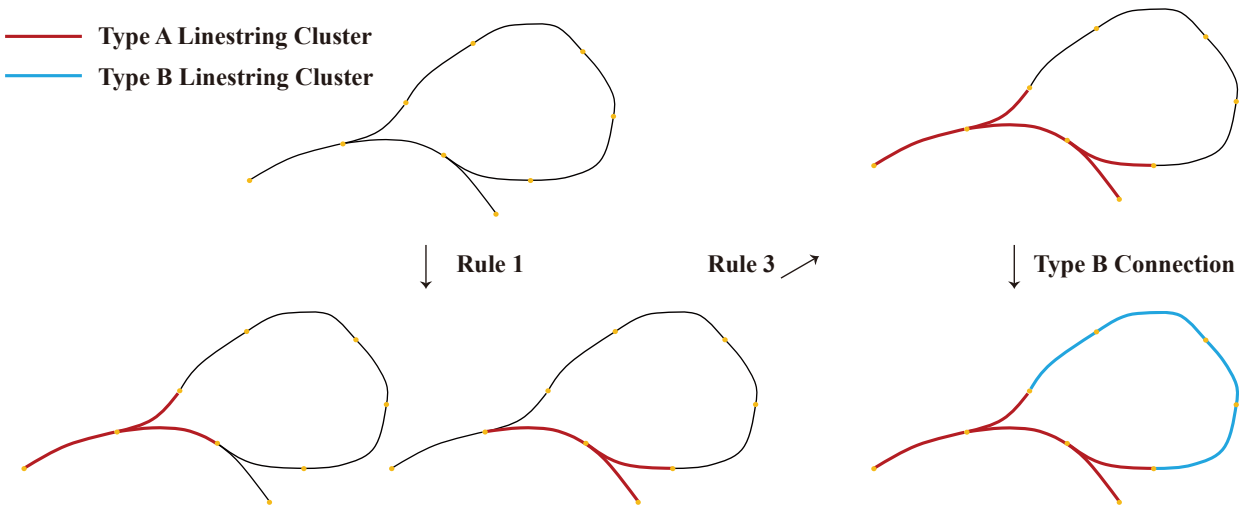


Figure 8. Example of road network decomposition based on the proposed three clustering rules. Starting from a sample road network (top left), **Rule 1** assigns the red-highlighted linestrings to a Type A cluster due to their high connectivity (bottom left). Then, **Rule 3** merges clusters sharing common linestrings (top right). Finally, remaining interconnected linestrings not covered by the rules are grouped into a Type B cluster (bottom right). This process enables systematic partitioning of lane-level road networks into multiple manageable clusters.

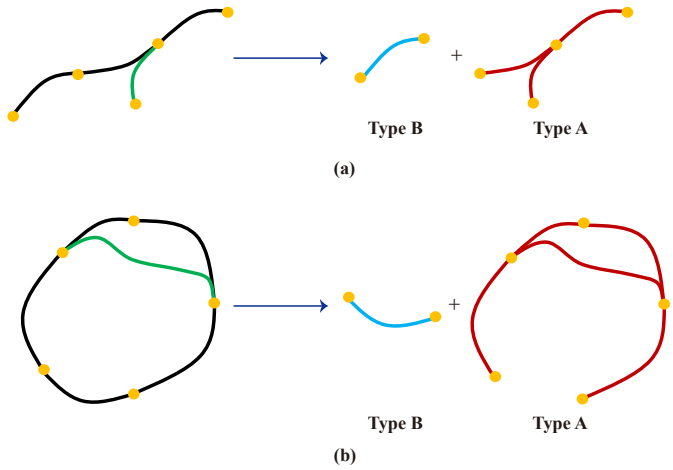


Figure 9. Illustrative example for **Proposition** 1. If a Type B linestring cluster includes a nested structure (green), it can be decomposed into a smaller Type B cluster and a Type A cluster, which contradicts the definition of a Type B cluster.

created in which nodes are shared by three or more linestrings, satisfying **Rule 1** and resulting in the formation of a Type A cluster. This illustrates that nested structures inherently lead to the presence of Type A clusters, thereby supporting the proposition.

C. Arc Spline Approximation of Individual Linestring Clusters

Referring back to the block diagram in Fig. 2, once the road network graph G has been decomposed using the clustering rules described in Section IV-A, arc-spline approximation is performed independently for each lane marking linestring cluster, regardless of whether it is Type A or Type B. At this stage, inter-cluster connections are not yet considered; the focus remains on individual cluster parameterization. Therefore, the multiple-linestring arc-spline approximation method from Section III-D is applied without modification.

This process yields arc-spline-approximated Type A and Type B clusters, highlighted by the blue dashed box in Fig. 2. Clusters that are physically adjacent and share a common node are then grouped by type, as shown in the green dashed box. In the following section, we introduce an efficient merging strategy that ensures G^1 continuity across these adjacent clusters without requiring full map-scale re-approximation.

V. Merging Adjacent Linestring Clusters

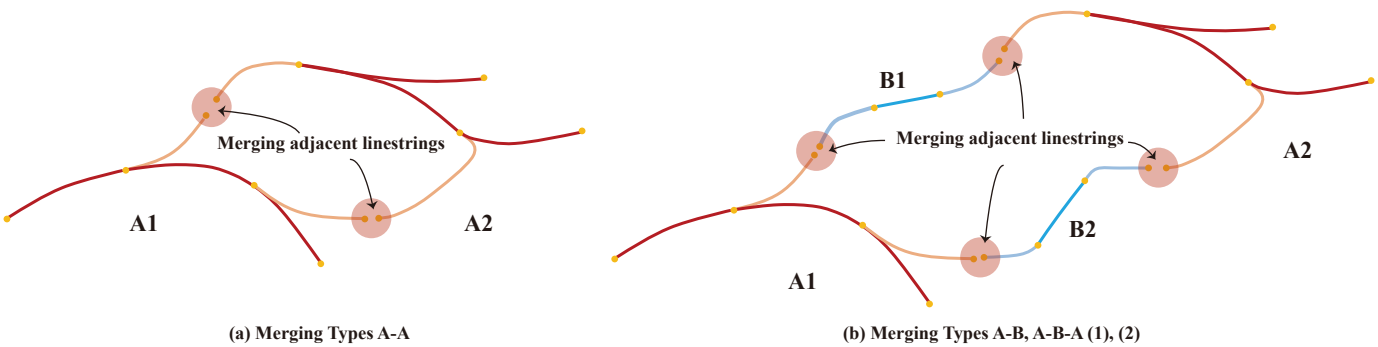
In this section, we describe the process for efficiently merging adjacent arc spline approximated linestring clusters (Types A and B) while ensuring G^1 continuity without requiring full map-scale optimization. We first present the merging strategy based on the type configuration of the clusters. Next, we detail the method for storing parameterized arc segments using conventional polyline-based map formats and outline the procedure for recovering arc parameters $(\mathbf{A}_i, \mathbf{A}_{i+1}, k_i)$ from the saved data. Finally, we discuss how the proposed framework can support efficient future updates and extensions to the map.

A. Properties of Merging Adjacent Linestring Clusters

Building on the properties of Type A and Type B linestring clusters discussed in Section IV, we now present an efficient merging strategy that avoids re-optimizing all arc segments within the merging clusters and eliminates the need for full map-scale optimization.

The merging process is illustrated in Fig. 10. For Type A–A merging, the procedure is straightforward and involves a single scenario. In contrast, merging between Type A and Type B clusters yields three distinct cases. As described in Section IV, Type B clusters—except for rare closed-loop cases—typically have two open ends.

- **A–B–A (1):** Each end of the Type B cluster connects to a different Type A cluster.



(a) Merging Types A-A

(b) Merging Types A-B, A-B-A (1), (2)

Figure 10. Merging of individually arc-spline-approximated adjacent linestring clusters based on cluster type. (a) Merging between **Type A–A** clusters involves a single scenario. Multiple connections in A–A merging are handled independently for each connected linestring pair (red circles). (b) Merging between **Type A–B** clusters includes three possible configurations: **A–B**, **A–B–A (1)**, and **A–B–A (2)**. In the A–B case, either A1 or A2 is missing (open end). In A–B–A (1), A1 and A2 belong to different clusters, while in A–B–A (2), they belong to the same cluster.

- **A–B–A (2):** Both ends of the Type B cluster connect to the same Type A cluster.
- **A–B:** Only one end of the Type B cluster connects to a Type A cluster, with the other end remaining open.

These configurations guide how local re-approximation is applied during the merging step, as described in the following subsections.

*1) **Merging Type A - Type A**:* When merging **Type A–Type A** linestring clusters, the following propositions hold:

Proposition 2 (**Type A–A Merging 1**). *Let A_1 and A_2 be two distinct **Type A** linestring clusters. If a linestring $l_1 \in A_1$ is connected to a linestring $l_2 \in A_2$, then l_1 and l_2 must be sequentially connected.*

Proof. As shown in Fig. 11, assume that l_1 and l_2 are not sequentially connected, i.e., they share a node (blue node) but diverge in direction. According to **Rule 2** in Section IV-A, such a configuration implies that l_1 and l_2 should belong to the same cluster, contradicting the assumption that they belong to different clusters A_1 and A_2. Therefore, l_1 and l_2 must be sequentially connected. □

Proposition 3 (**Type A–A Merging 2**). *Let A_1 and A_2 be two distinct **Type A** linestring clusters. If a linestring $l_1 \in A_1$ is connected to a linestring $l_2 \in A_2$, their shared connection node (blue node in Fig. 11) is not connected to any other linestrings—either from different clusters or from within A_1 or A_2.*

Proof. Suppose a third linestring l_3 also connects to the blue node. Two cases arise: (1) $l_3 \in A_3$, a different cluster, or (2) $l_3 \in A_1 \cup A_2$, i.e., within the same clusters as l_1 or l_2.

In either case, the node connects three linestrings, which—according to **Rule 1**—must be grouped into the same cluster. This contradicts the assumption that $l_1 \in A_1$ and $l_2 \in A_2$ are from different clusters.

While Fig. 11 shows l_3 connected to l_1, the same reasoning holds if it connects to l_2. Hence, no such additional connection can exist, and the proposition is proven. □

According to **Propositions** 2 and 3, and as illustrated in Fig. 11, the merging of arc spline approximated Type A linestring clusters $\mathbf{A}_1$ and $\mathbf{A}_2$ can be performed by focusing

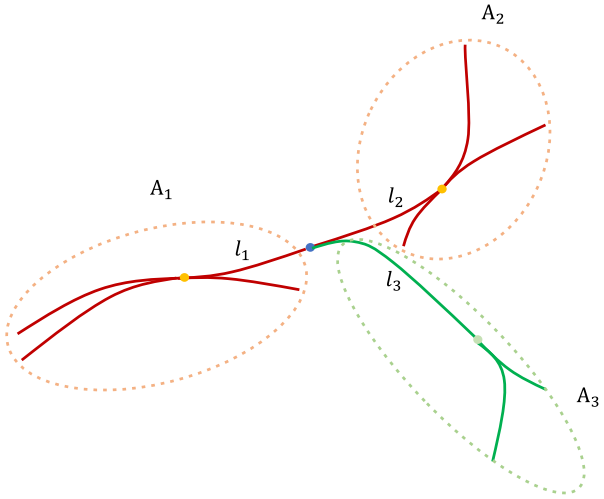


Figure 11. Illustrative example for **Propositions** 2 and 3. Two distinct Type A linestring clusters, A_1 and A_2, are connected via linestrings l_1 and l_2. (1) A contradiction arises if l_1 and l_2 are not sequentially connected. (2) A second contradiction occurs if an additional linestring l_3 connects to the blue node, violating the clustering rules.

solely on the transition between linestrings l_1 and l_2. By applying G^0 and G^1 continuity constraints at the orange nodes, the arc spline approximation module introduced in Section III-D can be applied exclusively to l_1 and l_2. Since the shared node (blue node) has no additional connections, the remaining linestrings in $\mathbf{A}_1$ and $\mathbf{A}_2$ remain unaffected and can be excluded from the optimization. This localized merging approach significantly improves computational efficiency by eliminating the need for full-cluster re-optimization.

Furthermore, multiple connections between $\mathbf{A}_1$ and $\mathbf{A}_2$ may exist. For instance, they may be connected at more than one region, as shown in Fig. 10. However, **Proposition** 3 guarantees that each such connection point involves only two linestrings, allowing each merging pair to be optimized independently. Consequently, even in cases with multiple connection points, merging can be performed in a modular and efficient manner without introducing interdependency between the connections.

*2) **Merging Type A - Type B - Type A (1) and (2)**:* Moving on to merging cases **Type A - Type B - Type A (1) and (2)**,

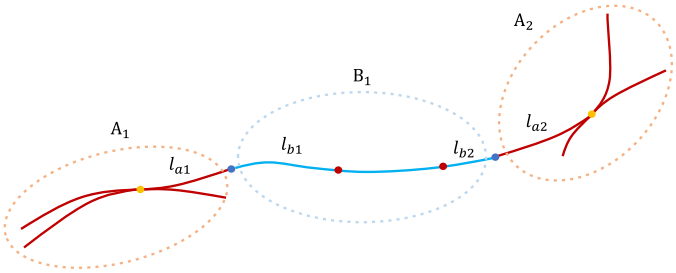


Figure 12. Illustrative example for **Proposition** 4. Two distinct Type A linestring clusters (A_1 and A_2) and a Type B cluster (B_1) are connected via linestrings l_{a1}, l_{a2}, and l_{b1}, l_{b2}. (1) A contradiction arises if l_{b1} and l_{b2} are not the two endpoints of B_1. (2) Another contradiction occurs if an additional linestring l_3 connects to either l_{a1} or l_{a2} at the blue nodes, violating the clustering rules.

the following proposition holds.

Proposition 4 (**Type A–B–A (1) and (2)** Merging)**.** *Let A_1 and A_2 be Type A linestring clusters, and B_1 be a Type B cluster. If linestrings $l_{b1}, l_{b2} \in B_1$ are connected to $l_{a1} \in A_1$ and $l_{a2} \in A_2$, respectively, then l_{b1} and l_{b2} must be the two endpoints of B_1, and all linestrings in the sequence l_{a1}, B_1, l_{a2} are sequentially connected.*

Proof. As shown in Fig. 12 and by **Proposition** 1, linestrings within B_1 are connected in a continuous, non-branching series. If l_{b1} (or l_{b2}) is not an endpoint and is connected to A_1 (or A_2), this introduces a nested structure—contradicting **Proposition** 1. Thus, both l_{b1} and l_{b2} must be the endpoints of B_1.

Furthermore, as with **Proposition** 3, if the shared blue node between l_{a1} and l_{b1} had another connecting linestring, it would violate **Rule 1**. Additionally, if l_{a1} and l_{b1} were not sequentially connected, they would be assigned to the same cluster under **Rule 2**, again contradicting the initial assumption. The same logic applies to l_{a2} and l_{b2}. Therefore, the full sequence l_{a1}, B_1, l_{a2} is sequentially connected. □

This proposition applies to both Type A–B–A (1) and A–B–A (2) configurations, as it does not require $A_1 \neq A_2$. Similar to the Type A–A merging case, the blue connection points are not linked to any other linestrings, enabling efficient merging by applying the arc spline approximation only to the adjacent linestrings at the cluster boundaries.

If B_1 contains multiple arc spline approximated linestrings, merging only requires re-optimization of the two pairs: (l_{a1}, l_{b1}) and (l_{a2}, l_{b2}). However, in the special case where B_1 contains only a single linestring ($l_{b1} = l_{b2}$), the merging is applied over the full sequence l_{a1}, B_1, l_{a2}, with additional G^0 and G^1 constraints enforced at the orange nodes, as illustrated in Fig. 12.

*3) **Merging Type A - Type B**:* The merging case of **Type A–Type B** can be viewed as a simplified version of the **Type A–B–A (1) and (2)** scenarios. For example, by removing the linestring cluster A_2 from Fig. 12, l_{b2} becomes a free end of B_1. In this case, the merging is performed by reapplying arc spline approximation to l_{a1} and all arc-spline-approximated linestrings in B_1, with additional G^0 and G^1 continuity constraints enforced at the orange node in A_1.

B. Arc Spline Approximation for Adjacent Linestring Clusters

Using the methods and properties described in Section V-A, we efficiently merge adjacent arc-spline-approximated linestring clusters based on their merge types, leveraging the arc spline approximation framework introduced in Section III-D.

1) Computation of Additional Constraints: To merge different types of clusters, additional G^0 and G^1 continuity constraints must be applied at the interface nodes, particularly within the Type A clusters. For example, in Fig. 11, the positions and tangents at the orange nodes of l_1 and l_2 must remain fixed to preserve continuity with the rest of the clusters A_1 and A_2. The process is as follows:

1. Extract the coordinates of the orange nodes from the arc spline approximated linestrings.
2. Use the arc parameters to compute tangents at these nodes.
3. Enforce the resulting positions and tangents as equality constraints in the merging step.

This approach is applicable to all merge types: for **Type A–B–A** (both (1) and (2)), constraints are applied only to the Type A clusters; for **Type A–B**, the constraint is applied only to the single Type A cluster.

2) Re-Parameterization of Adjacent Clusters with Constraints: Using the additional constraints, we reapply the arc spline approximation framework (Section III-D) exclusively to the directly connected linestrings at cluster boundaries, as illustrated in Block 2 of Fig. 2. For example, in the **Type A–A** case, only linestrings l_1 and l_2 are re-optimized. The objective function follows the original formulation but is augmented with the additional G^0 and G^1 continuity constraints derived in Section V-B1. The same strategy is applied to all **Type A–B** merging scenarios.

3) Avoiding Full Map-Scale Optimization: This local re-parameterization strategy guarantees global G^1 continuity without requiring re-optimization of the full map. Once merging is performed for all adjacent cluster pairs, the entire map satisfies the continuity condition. Consequently, the merging process becomes highly scalable, significantly reducing computational cost and enabling efficient updates and extensions of the map.

C. Saving and Recovering Parameterized Map Data

1) Saving Parameterized Map Data: After merging all linestring clusters, the resulting arc spline approximated map must be stored for visualization and future use. Rather than saving the signed-distance parameter k directly—which would require a new map format—we propose storing each arc segment using three points: the two arc nodes and the midpoint $\mathbf{N}$, computed from the rational Bézier formulation.

Remark. *One might wonder why the midpoint $\mathbf{N}$ is stored instead of the control point $\mathbf{C}$. There are two main reasons: (1) Saving $\mathbf{C}$ and converting the arcs into polylines would distort the geometry during visualization. (2) In low-curvature arcs,*

the large distance between $\mathbf{C}$ *and the arc nodes can cause numerical instability during storage or loading.*

As discussed in Section III-B, each arc segment can be represented using a rational Bézier curve. Given control point $\mathbf{C}$, arc segment endpoints $\mathbf{A}_1$, $\mathbf{A}_2$, and signed-distance parameter k, the arc is parameterized as:

$$d = \frac{1}{2}\|\mathbf{A}_1 - \mathbf{A}_2\|_2, \quad w = \frac{d}{\sqrt{d^2 + k^2}}$$
$$\mathbf{y}(s) = \frac{(1-s)^2\mathbf{A}_1 + 2s(1-s)w\mathbf{C} + s^2\mathbf{A}_2}{(1-s)^2 + 2s(1-s)w + s^2}, \quad 0 \le s \le 1 \tag{7}$$

The midpoint $\mathbf{N}$ can then be computed by evaluating $\mathbf{y}(s)$ at $s = 0.5$:

$$\mathbf{N} = \frac{\frac{1}{2}\mathbf{A}_1 + w\mathbf{C} + \frac{1}{2}\mathbf{A}_2}{w+1} \tag{8}$$

Thus, each arc is stored as an alternating sequence of arc nodes and midpoints in the conventional polyline-based format.

2) Recovering Parameterized Map Data: To recover the arc parameters from saved data, we first compute the arc center $\mathbf{X}_c$ using the three saved points: $\mathbf{A}_1$, $\mathbf{A}_2$, and the midpoint $\mathbf{N}$. Once $\mathbf{X}_c$ is obtained, we apply the inverse formulation of the center point based on the signed-distance parameter k, as derived below:

$$d = \frac{1}{2}\|\mathbf{A}_1 - \mathbf{A}_2\|_2$$
$$\mathbf{v} = \frac{1}{2d}\begin{bmatrix}0 & -1\\ 1 & 0\end{bmatrix}(\mathbf{A}_2 - \mathbf{A}_1) \tag{9}$$
$$\mathbf{X}_c = \frac{1}{2}(\mathbf{A}_1 + \mathbf{A}_2) - \frac{d^2}{k}\mathbf{v}$$

Given the computed arc center $\mathbf{X}_c$, we can rearrange the equation to solve for the signed-distance parameter k as follows:

$$\left(\frac{1}{2}(\mathbf{A}_1 + \mathbf{A}_2) - \mathbf{X}_c\right)^\top \mathbf{v} = \frac{d^2}{k}\mathbf{v}^\top\mathbf{v} = \frac{d^2}{k} \ (\because \|\mathbf{v}\|_2 = 1)$$
$$\therefore k = \frac{d^2}{\left(\frac{1}{2}(\mathbf{A}_1 + \mathbf{A}_2) - \mathbf{X}_c\right)^\top \mathbf{v}} \tag{10}$$

D. Application to Map Updates and Extensions

The proposed clustering and merging framework enables efficient arc spline parameterization not only for initial map generation but also for future updates and extensions.

For **map updates**, if a linestring requires adjustment, only the corresponding cluster and adjacent clusters are re-parameterized using the merging process in Section V-B2, eliminating the need to reprocess the entire map.

For **map extensions**, newly added regions are incorporated by clustering the affected linestrings and applying local re-parameterization. This modular approach makes the framework scalable, flexible, and robust to updates—overcoming the limitations of conventional parameterization and graph partitioning methods discussed in the Introduction.

Table II
RESULTS OF LINESTRING CLUSTERING AND MERGE TYPE CLASSIFICATION

Maps	BS	QT	ON	HV
Number of Linestrings	9273	6253	6311	4078
Number of Type A Clusters	1333	793	805	460
Number of Type B Clusters	1144	751	958	460
Number of A - A Merging	1213	802	482	411
Number of A - B - A (1) Merging	962	553	773	320
Number of A - B - A (2) Merging	2	13	11	5
Number of A - B Merging	176	185	168	135
Number of Isolated Type B Clusters	4	0	6	0

BS indicates Boston-Seaport, **QT** indicates Queenstown, **ON** indicates One North, and **HV** indicates Holland Village Lanelet Maps.

VI. EXPERIMENTAL VALIDATION

In this section, we validate our proposed road network decomposition method (Section IV) in combination with the cluster merging strategy (Section V), both of which build upon the multiple-linestring arc spline approximation framework introduced in Section III. The evaluation is conducted using several real-world, large-scale, and geometrically complex map datasets.

A. Configuration

A recent study [30] introduced a method for converting nuScenes HD maps into the Lanelet2 format [4]. The nuScenes dataset provides four distinct map regions: Queenstown, One-North, and Holland Village in Singapore, as well as Boston Seaport in the USA. Accordingly, we evaluate our framework using these four Lanelet2-converted nuScenes maps.

To emulate real-world data collection using Mobile Mapping Systems (MMS), we increased the point density along each linestring and added white Gaussian noise. Additional points were inserted to ensure an average point spacing of approximately 20 cm, in accordance with typical MMS configurations using LiDAR and camera sensors for lane detection.

All components of the proposed framework—including arc spline approximation, road network decomposition, and cluster merging—were implemented in MATLAB. Road network extraction from the Lanelet2 maps was performed using Python. All experiments were conducted on a desktop computer with an Intel i7-12700 processor and 32 GB of RAM.

B. Results

1) Road Network Decomposition and Merging: Table II presents the results of the proposed decomposition and merging methods applied to the four Lanelet2 maps. Linestrings were clustered into Type A or Type B (Section IV), and the merge types were determined based on their configurations (Section V). A representative visualization of the cluster distributions is provided in Fig. 13.

2) Arc Spline Approximation Results (Qualitative): In addition to linestring clustering and merge type classification, we present arc spline approximation results for the four Lanelet2 maps, based on the linestring cluster merge types listed in Table II.

[2]https://github.com/wjswlsghks98/Efficient-Arc-Spline-Approximation

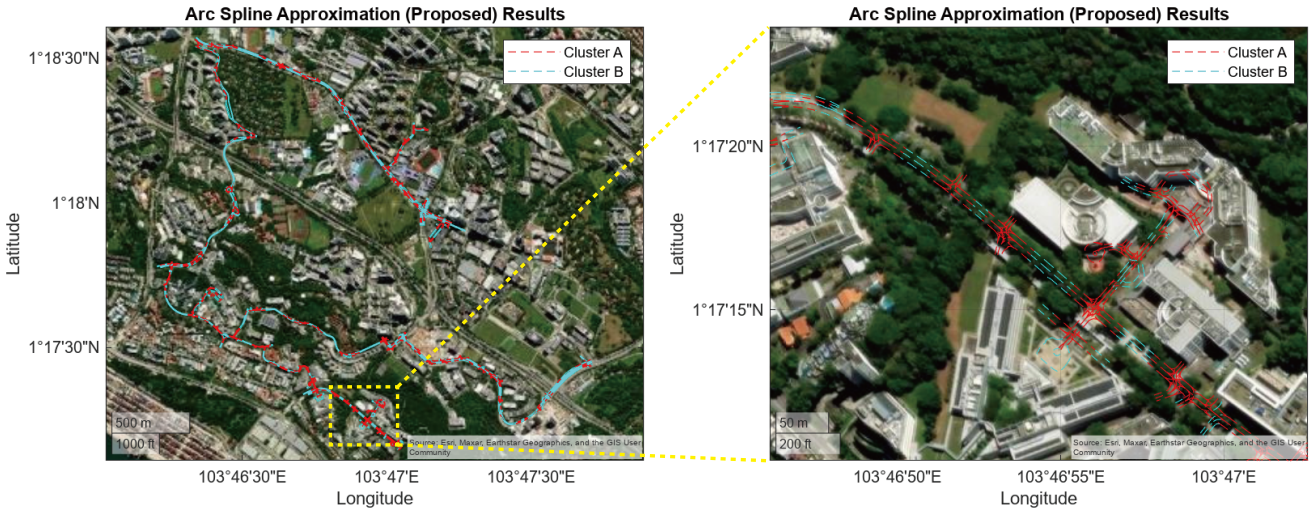


Figure 13. Distribution of Type A and Type B linestring clusters in the **Queenstown** region of the nuScenes map, based on the road network decomposition described in Section IV. All linestrings are arc spline approximated through the proposed decomposition and merging process. Type A clusters are primarily found in intersection areas, while Type B clusters correspond to simpler segments connecting those intersections. Map visualizations for other regions are available in our GitHub repository[2].

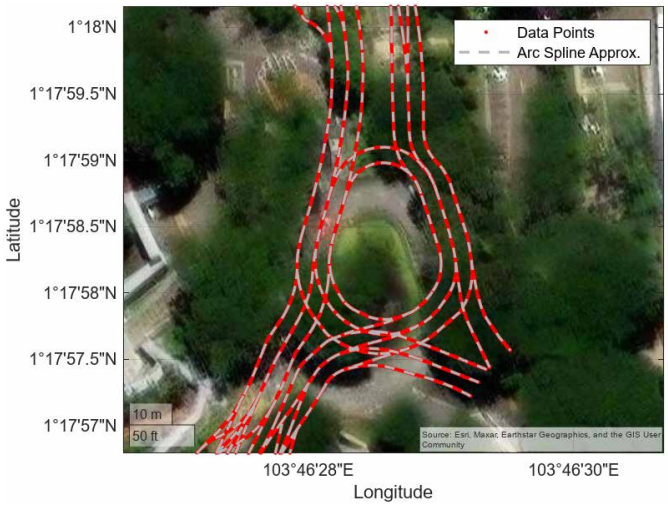


Figure 14. Exemplary arc spline approximation result (gray dash line) of complexly interconnected roundabout in the **Queenstown** region of the nuScenes HD map.

Fig. 1 and 14 illustrates the arc spline approximation applied to a section of the One-North and Queenstown lanelet map respectively, showcasing the successful parameterization of lane markings in complex intersection regions, interconnected roundabouts, as well as in high-curvature road sections. The arc spline approximation results of all four maps are available at our GitHub repository[2] and can be visualized using MATLAB's interactive figure viewer.

3) Arc Spline Approximation Results (Quantitative): For the quantitative evaluation of our proposed arc spline approximation framework, we assessed both the approximation accuracy and the reduction in data storage achieved by representing lane markings with arc splines.

To evaluate approximation accuracy, we first define the error

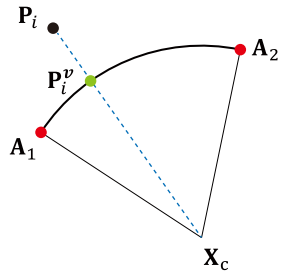


Figure 15. Measurement of arc spline approximation error. The Euclidean distance between data point $\mathbf{P}_i$ and its closest point on the arc segment $\mathbf{P}_i^v$ is used as the arc spline approximation error.

between each data point and its corresponding arc segment. As illustrated in Fig. 15, the error is computed as the Euclidean distance between a data point and its closest point on the associated arc segment. This closest point is obtained by determining the intersection between the arc segment and a line connecting the data point and the arc center.

Based on these error values, we compute the **Root Mean Square Error (RMSE)** and the **Average Precision (AP)** of the approximation. Precision is defined as the fraction of data points with approximation error below a given threshold, and AP is calculated by averaging precision over a set of predefined thresholds.

While prior works on online HD map prediction [8], [31] have used standard error thresholds of {0.5, 1.0, 1.5} meters, this study adopts more refined thresholds of {0.03, 0.05, 0.07} meters to better assess the geometric fidelity of the approximation. Table III presents the precision values at each threshold, the resulting AP, and the RMSE for all four maps. In addition, Table IV compares the data storage requirements of conventional polyline-based representations with those of our arc spline-based framework.

Analyzing the quantitative results in Table III, we observe

Table III
EVALUATION OF ARC SPLINE APPROXIMATION ACCURACY

Metrics	$P_{0.03}$	$P_{0.05}$	$P_{0.07}$	AP	RMSE(m)
BS	45.123	79.697	94.390	73.064	0.0426
QT	45.744	77.452	92.952	72.049	0.0416
ON	45.593	78.067	93.545	72.401	0.0419
HV	45.408	78.557	93.718	72.561	0.0410

P_t indicates precision value for error threshold of t meters. **BS** indicates Boston-Seaport, **QT** indicates Queenstown, **ON** indicates One North, and **HV** indicates Holland Village Lanelet Maps.

Table IV
DATA STORAGE COMPARISON: POLYLINES VS PROPOSED ARC SPLINES
BLACK: POLYLINES, BLUE: PROPOSED ARC SPLINES

Maps	BS	QT	ON	HV
Number of Linestrings	9273	6253	6311	4078
Number of Points	662523	430789	460630	297042
Storage Size (Points)	1325046	861578	921260	594084
Number of Arc Nodes	16360	11271	10658	7312
Number of Arc Segments	17439	12037	11195	7690
Storage Size (Arcs)	67598	46616	43706	30004
Storage Ratio(Points/Arcs)	19.602	18.483	21.079	19.800

BS indicates Boston-Seaport, **QT** indicates Queenstown, **ON** indicates One North, and **HV** indicates Holland Village Lanelet Maps.

that the precision values across multiple thresholds remain consistent across different maps, indicating the robustness of the proposed arc spline approximation framework. The distribution of approximation errors closely follows a normal distribution, without noticeable bias. Furthermore, the RMSE values show that our method achieves an average approximation error of approximately 4.2 cm, demonstrating high accuracy for lane-level map representation.

In Table IV, the storage size is defined as the number of parameters required to store each representation. For polyline formats, this corresponds to two parameters per point (for 2D coordinates). In our method, both 2D arc nodes and arc midpoints are stored, resulting in a total parameter count equal to twice the number of arc nodes plus twice the number of arc segments. Based on the storage ratio presented in the table, our framework achieves an average reduction in storage of approximately $\frac{1}{20}$ compared to the conventional polyline representation.

It is important to note that a direct **comparison with existing methods across multiple evaluation metrics is not feasible**, as no previous work has proposed a scalable and generalizable framework for parameterizing large-scale, complex lane-level maps.

4) Comparison of Road Network Decomposition: To evaluate the effectiveness of our proposed road network decomposition, we applied the multiple linestring arc spline approximation framework (Section III-D) to road networks partitioned by various graph-based methods. Across all four maps, our method completed the full parameterization process in under 45 minutes. In contrast, two classical partitioning algorithms—Graph Components and the Louvain Method [32]—were unable to complete within 24 hours and were terminated prematurely.

The superior performance of our approach is attributed to its ability to localize the arc spline approximation process. By ensuring that linestring clusters are connected only through simple linear regions, our method avoids the need for full map-scale optimization during the merging step (Sections IV-A and V-A).

While we do not claim that our method universally outperforms conventional graph partitioning algorithms across all domains, it is specifically designed to ensure global G^1 continuity during the parameterization of lane-level road maps with complex connectivity. In this setting, traditional partitioning approaches— which do not consider geometric continuity between connected linestrings—still require full map-scale optimization during the merging phase. As a result, their computational cost remains comparable to, or even higher than, that of the Graph Components and Louvain methods.

Therefore, our key contribution lies in designing a decomposition strategy (Section IV) that ensures cluster connectivity through linear segments only, enabling efficient merging (Section V) when integrated with our arc spline approximation algorithm for multiple linestrings (Section III-D). Although our experiments were conducted on Lanelet2-format maps, the framework can be extended to other large-scale lane-level maps by adapting the road network extraction step.

VII. CONCLUSION

This paper presents: **(1)** a novel arc spline approximation framework for multiple linestrings in complex road networks, and **(2)** a new road network decomposition and merging strategy that significantly improves the efficiency of arc spline approximation.

To the best of our knowledge, no prior work has addressed the parameterization of large-sized, complex lane-level road networks. As such, direct comparison for contribution **(1)** was not feasible due to the lack of relevant baselines. For contribution **(2)**, our decomposition and merging framework reduced computational cost by clustering linestrings such that adjacent cluster connections satisfy properties conducive to localized re-optimization. The proposed methods were validated using Lanelet2 maps derived from the nuScenes dataset, and qualitative results are available on our GitHub repository.

Future research directions include extending this framework to support online map updates and extensions, as well as exploring downstream applications such as arc spline-based localization, motion planning, and other ADAS functionalities.

ACKNOWLEDGMENT

This work was funded by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIT) (No. RS-2024-00346702); the Ministry of Trade, Industry and Energy (MOTIE, Korea) and Korea Evaluation Institute of Industrial Technology (KEIT) under Grant 20014983, 20018181, and 20023815.

REFERENCES

[1] "TomTom HD Map with RoadDNA," https://www.tomtom.com/products/hd-map/, 2018, accessed: 2024-09-09.

[2] "Here HD Live Map - On the Road Towards Autonomous Driving," https://www.here.com/platform/HD-live-map, 2018, accessed: 2024-09-09.
[3] M. Haklay and P. Weber, "OpenStreetMap: User-Generated Street Maps," *IEEE Pervasive Computing*, vol. 7, no. 4, pp. 12–18, 2008.
[4] F. Poggenhans, J.-H. Pauls, J. Janosovits, S. Orf, M. Naumann, F. Kuhnt, and M. Mayr, "Lanelet2: A high-definition map framework for the future of automated driving," in *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, 2018, pp. 1672–1679.
[5] E. Héry, S. Masi, P. Xu, and P. Bonnifait, "Map-based curvilinear coordinates for autonomous vehicles," in *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*, 2017, pp. 1–7.
[6] G.-P. Gwon, W.-S. Hur, S.-W. Kim, and S.-W. Seo, "Generation of a Precise and Efficient Lane-Level Road Map for Intelligent Vehicle Systems," *IEEE Transactions on Vehicular Technology*, vol. 66, no. 6, pp. 4517–4533, 2017.
[7] C. Guo, K. Kidono, J. Meguro, Y. Kojima, M. Ogawa, and T. Naito, "A Low-Cost Solution for Automatic Lane-Level Map Generation Using Conventional In-Car Sensors," *IEEE Transactions on Intelligent Transportation Systems*, vol. 17, no. 8, pp. 2355–2366, 2016.
[8] L. Qiao, W. Ding, X. Qiu, and C. Zhang, "End-to-End Vectorized HD-Map Construction With Piecewise Bezier Curve," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2023, pp. 13 218–13 228.
[9] T. Zhang, S. Arrigoni, M. Garozzo, D. ge Yang, and F. Cheli, "A lane-level road network model with global continuity," *Transportation Research Part C: Emerging Technologies*, vol. 71, pp. 32–50, 2016. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0968090X16301012
[10] K. Jo and M. Sunwoo, "Generation of a Precise Roadway Map for Autonomous Cars," *IEEE Transactions on Intelligent Transportation Systems*, vol. 15, no. 3, pp. 925–937, 2014.
[11] D. Bétaille and R. Toledo-Moreo, "Creating Enhanced Maps for Lane-Level Vehicle Navigation," *IEEE Transactions on Intelligent Transportation Systems*, vol. 11, no. 4, pp. 786–798, 2010.
[12] B. Gallazzi, P. Cudrano, M. Frosi, S. Mentasti, and M. Matteucci, "Clothoidal Mapping of Road Line Markings for Autonomous Driving High-Definition Maps," in *2022 IEEE Intelligent Vehicles Symposium (IV)*, 2022, pp. 1631–1638.
[13] J. A. Rodrigues da Silva, I. P. Gomes, D. F. Wolf, and V. Grassi, "Sparse road network model for autonomous navigation using clothoids," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 2, pp. 885–898, 2022.
[14] S. Zhang, R. Wang, Z. Jian, W. Zhan, N. Zheng, and M. Tomizuka, "Clothoid-based reference path reconstruction for hd map generation," *IEEE Transactions on Intelligent Transportation Systems*, vol. 25, no. 1, pp. 587–601, 2024.
[15] A. Schindler, G. Maier, and F. Janda, "Generation of high precision digital maps using circular arc splines," in *2012 IEEE Intelligent Vehicles Symposium*, 2012, pp. 246–251.
[16] S. Brummer, F. Janda, G. Maier, and A. Schindler, "Evaluation of a mapping strategy based on smooth arc splines for different road types," in *16th International IEEE Conference on Intelligent Transportation Systems (ITSC 2013)*, 2013, pp. 160–165.
[17] J. Jeon, Y. Hwang, and S. B. Choi, "Reliability-based G1 continuous arc spline approximation," *Computer Aided Geometric Design*, vol. 112, p. 102363, 2024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167839624000979
[18] Association for Standardization of Automation and Measuring Systems (ASAM), *Open Dynamic Road Information for Vehicle Environment, v1.8.0*, Nov. 2023, accessed: 9 Sept 2024. [Online]. Available: https://www.asam.net/standards/detail/opendrive/
[19] G. Karypis and V. Kumar, "A fast and high quality multilevel scheme for partitioning irregular graphs," *SIAM Journal on Scientific Computing*, vol. 20, no. 1, pp. 359–392, 1998.
[20] P. Sanders and D. Schultes, "Engineering highway hierarchies," in *European Symposium on Algorithms*. Springer, 2006, pp. 804–816.
[21] Z. Zhou, S. Lin, and Y. Xi, "A dynamic network partition method for heterogenous urban traffic networks," in *2012 15th International IEEE Conference on Intelligent Transportation Systems*, 2012, pp. 820–825.
[22] T. Anwar, C. Liu, H. Le Vu, and C. Leckie, "Spatial partitioning of large urban road networks." in *EDBT*, 2014, pp. 343–354.
[23] Q. Yu, W. Li, D. Yang, and H. Zhang, "Partitioning urban road network based on travel speed correlation," *International Journal of Transportation Science and Technology*, vol. 10, no. 2, pp. 97–109, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2046043021000034
[24] K. Otsuki, Y. Kobayashi, and K. Murota, "Improved max-flow min-cut algorithms in a circular disk failure model with application to a road network," *European Journal of Operational Research*, vol. 248, no. 2, pp. 396–403, 2016. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0377221715006694
[25] J. Clark and D. A. Holton, *A First Look at Graph Theory*. World Scientific, 1991. [Online]. Available: https://api.semanticscholar.org/CorpusID:60507276
[26] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre, "Fast unfolding of communities in large networks," *Journal of Statistical Mechanics: Theory and Experiment*, vol. 2008, no. 10, p. P10008, oct 2008. [Online]. Available: https://dx.doi.org/10.1088/1742-5468/2008/10/P10008
[27] J. Hoschek, "Circular splines," *Computer-Aided Design*, vol. 24, no. 11, pp. 611–618, 1992.
[28] The MathWorks Inc., "lsqnonlin : Nonlinear Least Squares Solver," Natick, Massachusetts, United States, 2023. [Online]. Available: https://kr.mathworks.com/help/optim/ug/lsqnonlin.html
[29] X. Song, M. Aigner, F. Chen, and B. Jüttler, "Circular spline fitting using an evolution process," *Journal of Computational and Applied Mathematics*, vol. 231, no. 1, pp. 423–433, 2009.
[30] A. Naumann, F. Hertlein, D. Grimm, M. Zipfl, S. Thoma, A. Rettinger, L. Halilaj, J. Luettin, S. Schmid, and H. Caesar, "Lanelet2 for nuScenes: Enabling Spatial Semantic Relationships and Diverse Map-Based Anchor Paths," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2023, pp. 3247–3256.
[31] B. Liao, S. Chen, Y. Zhang, B. Jiang, Q. Zhang, W. Liu, C. Huang, and X. Wang, "Maptrv2: An end-to-end framework for online vectorized hd map construction," *International Journal of Computer Vision*, pp. 1–23, 2024.
[32] L. G. S. Jeub, M. Bazzi, I. S. Jutla, and P. J. Mucha, "A generalized Louvain method for community detection implemented in MATLAB," https://github.com/GenLouvain/GenLouvain (2011-2019), July 2019.

Jinhwan Jeon received the B.S. degree in Mechanical Engineering from Korea Advanced Institute of Science and Technology(KAIST), Daejeon, Korea, and M.S.in Mechanical Engineering from the Korea Advanced Institute of Science and Technology (KAIST), in 2021 and 2023, respectively. Since 2023, he is currently pursuing the Ph.D. degree in Mechanical Engineering at KAIST. His research interests include multi-modal sensor fusion, automatic lane-level map generation and update algorithms.

Seibum B. Choi received his B.S. in Mechanical Engineering from Seoul National University, Seoul, South Korea, M.S. in Mechanical Engineering from KAIST, Daejeon, South Korea, and Ph.D. in control from the University of California, Berkeley, CA, USA, in 1993. From 1993 to 1997, he was involved in the development of automated vehicle–control systems at the Institute of Transportation Studies, University of California. Until 2006, he was with TRW, Livonia, MI, USA, where he was involved in the development of advanced vehicle control systems. Since 2006, he has been a member of the faculty of the Mechanical Engineering Department, KAIST, South Korea. His current research interests include fuel-saving technology, vehicle dynamics and control, and active safety systems. Prof. Choi is a Member of the American Society of Mechanical Engineers, Society of Automotive Engineers, and Korean Society of Automotive Engineers.