CHALMERS
UNIVERSITY OF TECHNOLOGY

VERNE: A Spatial Data Structure Representing Railway Networks for Autonomous Robot Navigation

Downloaded from: https://research.chalmers.se, 2026-09-09 18:35 UTC

Citation for the original published paper (version of record):
Joly, L., Lacorre, V., Wolff, K. (2026). VERNE: A Spatial Data Structure Representing Railway Networks for Autonomous Robot Navigation. IEEE Open Journal of Vehicular Technology, 7: 1-14. http://dx.doi.org/10.1109/OJVT.2025.3628652

N.B. When citing this work, cite the original published paper.

© 2026 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, or reuse of any copyrighted component of this work in other works.

(article starts on next page)

IEEE Open Journal of Vehicular Technology

Received 4 October 2025; accepted 25 October 2025. Date of publication 4 November 2025;
date of current version 25 November 2025. The review of this article was coordinated by Editor Subhadip Kumar.

Digital Object Identifier 10.1109/OJVT.2025.3628652

VERNE: A Spatial Data Structure Representing Railway Networks for Autonomous Robot Navigation

LOUIS-ROMAIN JOLY [1], VIVIEN LACORRE [2] (Graduate Student Member, IEEE), AND KRISTER WOLFF [2]

[1]Technology, Innovation and Group Projects Division, Société Nationale des Chemins de fer Français, 93210 Saint-Denis, France
[2]Department of Mechanical Engineering, Chalmers University of Technology, 41296 Gothenburg, Sweden

CORRESPONDING AUTHOR: KRISTER WOLFF (e-mail: krister.wolff@chalmers.se).

This work was supported by the European Union under Project Grant HORIZON-ER-JU-2022-FA3-01 IAM4RAIL - Holistic and Integrated Asset Management for Europe's RAIL System.

ABSTRACT Efficient representation and querying of railway networks are crucial for autonomous railway systems and digital infrastructure management. This paper introduces **VEctorial Railway NEtwork (VERNE)**, an interpretable data structure and algorithm that integrates vector-based spatial partitioning with a railway-specific topological framework to enhance network representation and navigation. VERNE is designed to optimize query efficiency, reduce memory footprint, and ensure scalability for real-time applications. Its internal mechanism results from a comparative performance analysis between a k-d tree, an STRtree and two custom algorithms, highlighting trade-offs in computational efficiency and memory overhead. The proposed approach is validated using datasets from both the French and Swedish railway networks, demonstrating its effectiveness in real-world scenarios. The results indicate that VERNE provides a robust and scalable solution for railway infrastructure modeling, offering improvements in localization speed and computational efficiency. Another advantage is that it inherently manipulates atomic elements which can contain any information relevant to directly perform navigation onboard an autonomous robot. This work contributes to the advancement of railway digitalization by providing a structured methodology for spatial data processing in autonomous railway systems.

INDEX TERMS Autonomous railway systems, navigation, railway networks, spatial data structures.

I. INTRODUCTION

The traditional characteristics of the railway industry, such as large-scale machinery and complex maintenance processes similar to those found in heavy industries, are currently undergoing significant transformation. In response to emerging challenges including climate change, resource scarcity, budget constraints, increased maintenance demands with constant resources, rising performance expectations, and others, there is a growing interest in adopting lighter and more flexible tools for maintenance purposes. Recent initiatives have focused on exploring the potential of smaller, autonomous robots for railway infrastructure maintenance, both within Europe and globally. Potential applications of these technologies include railway infrastructure inspection, data collection, and actual maintenance operations [1], [2], [3].

The FP3-IAM4RAIL[1] project explores the application of inspection and intervention robots for railway infrastructure maintenance. A crucial aspect in the deployment of such robots, regardless of their tasks, is their ability to precisely localize themselves on the infrastructure [4]. Although advances in high-precision Global Navigation Satellite System (GNSS) receivers and correction processes such as real-time kinematic (RTK) may suggest that this requirement can be readily met, unfortunately the conditions necessary for ensuring centimetric geolocation accuracy cannot consistently be satisfied across every point on the railway network, even if tunnels are excluded [5]. Absolute localization can be supplemented with relative odometry sources (e.g., LiDAR

[1]https://rail-research.europa.eu/rail-projects/fp3-iam4rail/

© 2025 The Authors. This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

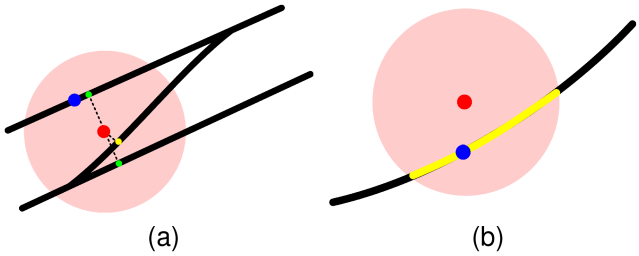

FIGURE 1. Initial localization scenario. The black lines represent the railway tracks, the blue dot indicates the true location and the red dot represents the center of the GNSS localization, with its uncertainty region indicated by a transparent red circle. (a) The yellow dot results from the projection of the GNSS position on the closest track. The green dots denote projections on two other candidate tracks. (b) The yellow line denotes all possible positions if there is only one candidate track.

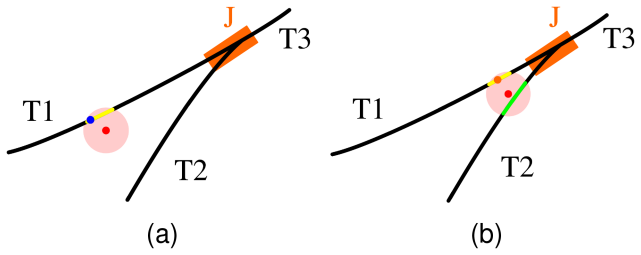

FIGURE 2. Localization hypotheses at the approach of a junction J at $t = t_0$ (a) and at $t = t_0 + \Delta t$ (b). The yellow and green lines show all possible positions along the two candidate tracks.

SLAM), but typical uncertainties cannot always guarantee unambiguous positioning, especially near parallel tracks and junctions. For the rest of this paper, the simpler example of the GNSS will be used for illustration purposes, as well as a circular uncertainty area. Regardless of the source of localization, better positioning can be achieved by leveraging knowledge of the infrastructure on which the robot operates. In off-road applications, robots can potentially localize themselves anywhere on Earth. Conversely, in rail-based scenarios, the robot's movement is guided and necessarily confined to a track. Consequently, the localization issue is effectively simplified to an almost one-dimensional problem, provided that a single track exists within a large area (exceeding the precision limitations of geolocation). However, if branching tracks are present in the vicinity, the localization challenge becomes multidimensional.

The problem can be decomposed into two distinct subproblems occurring at different stages of the robot's operation. The first sub-problem arises during the initial localization process or if a temporary loss of reference causes the need for a re-localization. Once a GNSS fix is acquired, the robot's position on the Earth's surface is determined. However, if the GNSS localization accuracy is suboptimal, it is not recommended to project the returned position on the closest railway track. As a matter of fact, it could erroneously be associated with an incorrect track, as illustrated in Fig. 1(a). At this stage, it is risky to settle on a single, potentially inaccurate position. Instead, it is essential to consider all positions that are physically possible according to the uncertainty area of the localization. In the rest of this paper, such positions will be called "candidate positions" to indicate that they are all plausible. Moreover, the assumption is made that the robot cannot be located outside of a given circular uncertainty area.

The second sub-problem concerns the accumulation and exploitation of localization hypotheses over time. Consider a simple scenario where the robot is traveling on a single track, far from any junctions. In the absence of knowledge regarding the track's geometry, localization's region of uncertainty is distributed in two dimensions over the earth's surface. However, by incorporating the geometry of the track, this uncertainty is reduced to one dimension along the track's curvilinear abscissa, as illustrated in Fig. 1(b).

As the robot approaches a junction, knowledge of the robot's previous localization and of the junction's state can be exploited. Unlike most road intersections, railway junctions are unidirectional. If approaching J from T1, the robot can only move onto T3. Thus, if the robot has previously been localized on track T1 (as shown in Fig. 2(a)), as long as J has not been crossed and no reversal of direction has been detected, the localization hypothesis on T2 (represented by the green segment) can be ruled out, even if it appears most likely based on GNSS uncertainty (Fig. 2(b)). While these considerations guided the design choices presented in this paper, it should be noted that the exploitation of these candidate positions is outside the scope of this paper and is more relevant to the field of map matching [6], [7].

Here are a few other elements taken into consideration during the exploration of potential data structures. While the operational modalities for these robots have yet to be fully determined, it is plausible that robots will deviate from traditional rail traffic patterns. Therefore, an ideal data structure should be able to model all possible changes of direction between tracks. Additionally, it would be an advantage to minimize the amount of external information required by the robot to localize itself during its missions. Indeed, waiting for such data would increase the localization latency and complicate the task of the personnel responsible for operating these robots. Likewise, it would be convenient to communicate the waypoint positions of a mission solely in geographic coordinates within a chosen terrestrial reference frame, without necessitating line, track, section references, or kilometer point designations. As a side note, stored information on the network's geometry can be exploited in various ways. For example, if the radius of curvature is known for the current robot's location, obstacle detection can be done within a portion of a toroid aligned with the track's radius, instead of using a shorter naive rectangular cuboid in front of the robot.

The primary focus of this study is directed toward factors influencing the robot's behavior during mission execution, while the preparatory stages, such as e.g., map generation, are

considered of secondary importance. This distinction is crucial for the comparative assessment of solution performance, as the time required for setting up the data structures (preprocessing of the network's descriptive information) is less significant than the time taken to determine the robot's initial location during runtime.

The main contributions of this work can be summarized as follows:

- This paper presents VERNE, a comprehensive framework including a memory-efficient data structure capable of representing railway topology and a fast query algorithm. The ultimate goal is to facilitate robust and efficient localization of robots operating on railway environments by leveraging a structured and scalable representation.
- Limitations of publicly accessible datasets, such as that of the French railway network [8] are presented (see Section III).
- Generic STRtree and k-d tree are implemented, along with two custom alternatives. Their differences are discussed from a theoretical point of view (see Section IV). The custom approaches were specifically designed to solve the problems described in this introduction.
- The custom data structures and query methods are validated with a reference tool widely used for the query of spatial data, which is used to generate the ground truth for the French and Swedish railway networks (Section V).
- Finally, the data structures are compared with each other, and different trade-offs are discussed (Section VI).

In this paper, **atoms** refer to indivisible groups of consecutive segments forming a continuous path. They are separated from other atoms by railroad junctions, or exist as isolated units with two dead ends. An atom can contain additional descriptive information related to its track. The terms **segment** and **bounding box** (oriented or non-oriented) are used in accordance with their standard geometric definitions.

II. RELATED WORK

The localization of rail vehicles has been the subject of several studies with various objectives and constraints. Most of them use data from a digital map but do not indicate the data structures involved nor the algorithms used to query them. Examples of works exploiting track maps are detailed in [9], [10], [11] and [12]. Many of them rely on multiple track hypotheses [13], [14], [15], [16], notably to handle junctions, but do not clarify how these hypotheses are fetched from the map.

A. DESCRIPTIVE DIGITAL RAILWAY MODELS

Research has been conducted to invent information-rich models representing railway networks and their features [17], [18], [19], [20], [21]. The RailTopoModel [22] is a logical object model created to standardize the representation of railway infrastructure. Relations between elements of the network are modeled via a node-edge diagram. Elements can be aggregated into larger, high-level units. A network element can be associated with different positioning systems, relative or absolute. In practice, this standard is implemented with the RailML [23] data exchange format, based on XML. Another example of a tool using RailML is the railway planning software OpenTrack [24] capable of simulating complex networks. However, it is not meant to be used for real-time navigation.

While powerful to describe railway infrastructures, these abstract models are not accompanied with instructions on how to actually implement them into a software data structure and perform queries on them. In addition, it is unclear if such models can be stored and exploited on board a robot with constrained resources. To fill these gaps, the goal of this work is to provide a well-defined data structure tailored for railway applications and guarantee a memory and speed efficiency suitable for real-time mobile robotics applications. In this study, different spatial data structures are considered to make an informed decision. The focus is not on the development of a novel modeling paradigm. VERNE is only equipped with enough descriptive capabilities to enable autonomous localization and navigation. Nevertheless, thanks to the use of "atoms", VERNE is intrinsically designed to work for railway, making it an ideal foundation for future integration to more complex models if needed.

B. SPATIAL DATA STRUCTURES

Spatial data structures facilitate the projection of georeferenced localization measurements, whose inaccuracies may place them outside of railway tracks, into realistic positions along the rails. These structures are designed to address two primary challenges: efficient storage and retrieval of spatial data, and rapid processing of spatial queries, such as range searches and nearest neighbor searches. By organizing spatial data to exploit the inherent spatial relationships between objects, these structures enable quick access and analysis. The primary focus in this context is runtime performance, rather than the transformation of the raw data into the data structure.

There are numerous spatial data structures, each possessing distinct characteristics, advantages, and limitations. Common types include: **Quadtree and Octree**, which are hierarchical tree structures employed for the partitioning of two- and three-dimensional spaces, respectively [25], [26]. A **k-d tree (k-dimensional tree)** is a binary tree structure designed to partition k-dimensional space. It is particularly effective in low-dimensional spaces [27], [28], [29]. An **R-tree** is a hierarchical tree structure optimized for indexing spatial objects in multidimensional space. R-trees efficiently handle range queries and spatial joins by organizing objects into bounding rectangles [30], [31]. An **M-tree** is a metric tree structure tailored for similarity search in metric spaces. M-trees are well-suited for applications such as content-based retrieval and multimedia databases [32], [33]. **Spatial hashing** is a grid-based method that divides a space into a grid of cells. It is particularly efficient for spatial indexing applications in computer graphics and collision detection [34], [35]. A **Voronoi**

diagram is built by partitioning a space into regions, based on proximity to a given set of points. Voronoi diagrams are particularly useful for nearest neighbor searches and proximity analysis [36], [37].

These spatial data structures present various trade-offs regarding query performance and their suitability for different types of spatial data. The selection of an appropriate spatial data structure is influenced by factors such as the characteristics of the data, the types of queries to be executed, and the performance requirements of the specific application. As spatial data become increasingly complex and voluminous, ongoing research and advancements in spatial data structures remain essential for facilitating efficient spatial data management and analysis [38].

Voronoi diagrams are commonly employed for partitioning space into regions defined by proximity to a given set of points. Similarly, the k-d tree serves as a space-partitioning data structure that is well-suited for point-based data. However, the data considered in this context consist of sets of segments (or polylines), necessitating certain adaptations to utilize these structures effectively. According to Vermeulen et al. [39], the query performance of the k-d tree generally surpasses that of the Voronoi diagram. Additionally, it is important to highlight that the k-d tree is more extensively documented compared to the Voronoi diagram, despite the latter's longer history.

M-trees and R-trees are analogous data structures. M-trees are constructed using a metric space (e.g., Euclidean distance) and rely on the triangle inequality.[2] In contrast, R-trees are built upon the minimal bounding boxes of nearby objects. Several R-trees variants have been developed (R+-tree, R*-tree, X-tree, etc.) [40], [41], notably to be more efficient for data that has to be updated often, via insertion for instance. It is essential to note that in the context of railway, network evolutions occur rarely, so the focus is on the query speed at run time.

Beyond the classic algorithms described above, the last decade has experienced a growing interest in learned index structures [42], [43], some of them are specifically designed for spatial data and can show competitive query speeds [44], [45], [46]. However, due to the high safety requirements present in the railway industry, to facilitate the deployment of VERNE this study focuses on fully interpretable and explainable approaches not based on machine learning. Nevertheless, it is a promising field that should continue to be explored. Furthermore, it should be noted that parts of the literature working on multi-dimensional indexes choose to develop hardware-specific optimizations for modern technologies [47], but a more general approach was adopted.

C. GRAPH AS A TOPOLOGICAL SPACE

To have the ability to predict future candidate positions based on the robot's previous location, the data structure needs to store the topological relationships between various elements of the network, which involves a fundamental application of graph theory.

Typically, transport networks are visualized spatially, from an overhead perspective, where intersections are represented as vertices and roads as edges. However, Gharaee and colleagues [48] suggest that an alternative representation can be beneficial: modeling roads as vertices and intersections as edges. This inversion provides distinct advantages in specific analytical contexts, see further discussion in Section IV.

III. DATA

A. DESCRIPTION

The first dataset utilized in this study is derived from the publicly available SNCF Réseau open data resource [8]. It comprehensively represents the entire French railway network. The data are provided in GeoJSON[3] format, where the network is encoded as a **FeatureCollection** comprising 122, 507 features for a total network length of 48, 625 km.

A **Feature** comprises three primary elements: its type (`Feature`), its geometry, and its properties. The geometry consists of a sequence of segments that discretize the track axis, with spacing managed to minimize chord effects. The segment endpoints are specified using the WGS84[4] coordinate system.

The GeoJSON includes 21 properties, which represent various details. These properties encompass the name and code of the track, a codification specific to SNCF Réseau's information system (referred to as the GAIA code), the code and label of the line, and the track direction (whether the track is aligned, in a left-hand curve, or in a right-hand curve). Additionally, properties include the radius of curvature, the section rank, the starting and ending PK (kilometric points), and the coordinates of these points in both the WGS84 and Lambert-93[5] reference systems. Latitude and longitude values for the starting point, end point, and midpoint of each section are also provided, grouped in pairs.

The segmented geometry defines the elementary size of each **Feature**. This segmentation becomes evident when transitioning from a straight section of track to a curved section; each change necessitates a new **Feature**. Similarly, if a left-hand curve with a radius of 500 meters changes to a left-hand curve with a different radius (e.g., 724 meters), a new **Feature** is created.

To assess the reproducibility of VERNE, a second dataset was studied. It is a representation of the Swedish railway network, stored in the GeoPackage[6] format, provided by the Swedish Transport Administration [49]. Its structure is similar to that of the French dataset, but it is using the projected coordinate system SWEREF99 TM.[7] At the time of collection, the

[2]For any triangle, the sum of the lengths of any two sides must be greater than or equal to the length of the remaining side.

[3]https://geojson.org/
[4]https://epsg.io/4326
[5]https://epsg.io/2154
[6]https://www.geopackage.org/
[7]https://epsg.io/3006

dataset is made of 48,738 features for a total network length of 18,246 km.

B. LIMITATIONS OF THE DATA

The description of paths using simple geometric elements, such as straight lines and circular arcs, offers valuable opportunities for point localization relative to these elements: the distance from a point to a straight line, as well as the distance to a circular arc, can be efficiently calculated. However, a qualitative analysis of the curve radii indicates that this piece of information cannot be reliably used. Approximately 1.5% of the database lacks radius information, or the available radius values are inconsistent with the coordinates in the **geometry** of the **Feature** (e.g., cases where the radius is less than half the distance between the entry and exit points of the curve). Consequently, only the points included in the **geometry** of each **Feature** are utilized in this study. Should the quality of this layout data improve in the future, these additional geometric elements could be further exploited.

Finally, the use of the WGS84 coordinate system (instead of Lambert-93) to describe section geometries may raise concerns regarding long-term positional stability. However, this limitation is well-recognized among users of georeferenced data.

IV. DATA STRUCTURES

To reiterate, the two operational sub-problems are defined as follows:

1) Determine on which track segments the robot may be located at startup time t_0.
2) Determine the segments on which the robot may be located at time $t = t_0 + \Delta t$, given knowledge of segments it could have previously been on.

Both sub-problems can be solved with the same functionality: the comprehensive identification of all candidate railway segments where the robot can be located on, given a geographic position and its uncertainty.

It is necessary to note a modification in the fundamental unit that will be used to represent the railway network. In the SNCF Réseau open data GeoJSON, this unit is characterized by its homogeneity in terms of curvature. The primary interest lies in identifying changes in direction, particularly at junctions. Therefore, the individual GeoJSON features can be merged as long as no junctions are encountered. Through this process, the original 122,507 features are consolidated into 22,507 groups of segments, referred to as "atoms" in this document.

In essence, an ideal spatial query has to return a minimal set of atoms in order to simplify future refinements. Importantly, that set of atoms should be a superset of the atoms truly overlapping with the uncertainty disk around the measured position, to guarantee that the real track under the robot is always considered. On the other hand, it is acceptable to have irrelevant atoms in the superset if it allows for higher query speed.

By construction, as long as the robot has not reached one of the endpoints of an atom, it is constrained to follow its current path. At each endpoint, an atom is either unconnected or connected to at least two other atoms. However, this does not necessarily imply that the robot has multiple paths available at each endpoint. For instance, in the case of a diverging junction, the robot may be limited to following only a single path (see Fig. 2 and the accompanying explanation below). To produce a consistent model of railway junction, an undirected graph representation appears appropriate, with the dual graph method documented in the literature [50], [51]. In conventional models for road transportation networks, intersections are typically represented as vertices, while the road segments connecting these intersections are represented as edges. However, due to the asymmetric nature of railway intersections—depending on whether they are approached from an incoming or outgoing direction—this model requires adaptation. In the proposed model, intersections are represented as edges, while the segments of track (referred to here as "atoms") between these intersections are treated as vertices. This adjustment captures the unidirectional flow characteristics specific to railway networks.

Take an example similar to the one shown in Fig. 2 to illustrate the modeling process. Consider now the example of the network segment illustrated in Fig. 3(a).[8] In Fig. 3(b), the vertices represent junctions, while the edges denote the atoms connecting these junctions. Each atom is traversable by the robot in both directions, which may differ from the travel directions used in commercial train operations. At this stage, it is assumed that the robots will operate under specific traffic rules, with exclusive track occupancy.

This representation implies that travel from J1 to J5 via J2 is possible using only edges A1 and A5. However, such a maneuver is feasible only by including A2. A robot starting at A1 must traverse J2 to reach A2, at which point it must stop and execute a maneuver. After adjusting the turnout, the robot can reverse, re-entering J2 and proceeding from A2 to A5. Although the edges are bidirectional, an oriented graph with mono-directional edges would not adequately represent the full range of maneuvering options.

In Fig. 3(a), the vertices correspond to atoms, and the edges denote junctions where paths intersect. This configuration accurately reflects all possible movements: transitions between A1 and A2, between A2 and A5, and vice versa, but not directly from A1 to A5, for example. Should movement restrictions be required in the future to reflect operational rules, a directed graph model could be considered.

To sum up this notion of atom, its minimum characteristics are:

- an identifier;
- an ordered list of the identifiers of the points composing its path;

[8] In this diagram, junctions J1, J3, J4, and J6 are shown incompletely to simplify the graph, with other atoms connecting these points omitted.

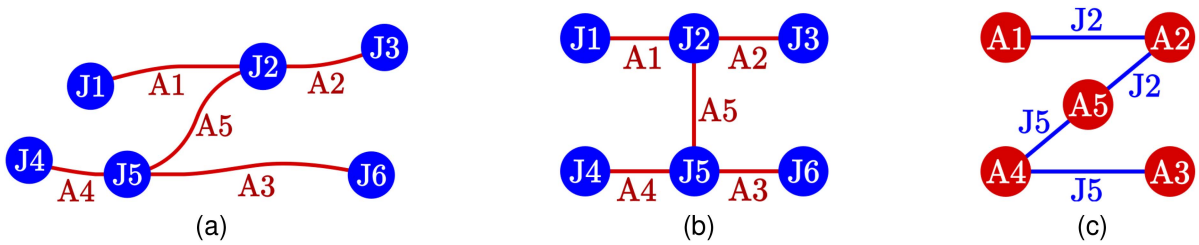

FIGURE 3. Representation of network junctions (blue) and atoms (red) as vertices in an undirected graph model. (a) Real topology of the network. (b) Graph model with junctions represented as vertices. (c) Graph model with atoms represented as vertices.

- a list of the links with neighboring atoms (this link is not purely geometrical, but reflects the possibility of a rail vehicle moving from one to the other, as discussed above). These links make the atoms the vertices of an undirected graph.

To expand the use of the data structure, it would also be possible to add properties that go far beyond pure geometric description (content of infrastructure assets such as catenary poles, bridges, level crossings, etc. or even speed limitation, semantic description of the environment, etc.).

Now that the elementary characteristics of an atom have been presented, the data structure used to store them must also be discussed, along with its query algorithm. In fact, it is only in conjunction with a search algorithm that the data structure can achieve its intended function: finding the network segments on which the robot is likely to find itself, based on its geo-referenced position and a given uncertainty.

A. THE k-D TREE WITH DISTANCE QUERY

To evaluate this approach in the context of railway spatial data, the first reference used is the k-d tree, which is a well-established data structure in the field of geospatial data management. The most relevant query type in this context is a search for points located within a circular area (Distance Query or DQ). Returned point indexes would then have to be mapped to the atoms they belong to.

The k-d tree is particularly fast for low dimensional spatial data, but a generic implementation stores geographic points and can only be queried on them. This is a strong limitation when handling segments made of points that can be far apart from each other, especially for parts of the network that are mainly straight. The intrinsic accuracy of GNSS without RTK correction generally results in a positional uncertainty of several meters. When conducting a DQ within a radius of similar magnitude, it is likely that, in many instances, no geographic points will be found within that distance. In fact, only 10% of the segments used to model the French network are shorter than 10 meters (see Fig. 4). Although 70% of the segments are 50 meters or shorter, a search radius of over 200 meters would be required to encompass 99% of the segments. The longest segment spans nearly 5700 meters (and 4100 meters for the Swedish network). Consequently, using a search radius based on the GNSS precision is inadequate. To prevent the risk of missing a potential match between the center of the GNSS-derived position and a segment, it is necessary to employ a search radius slightly longer than half the length of the longest segment.

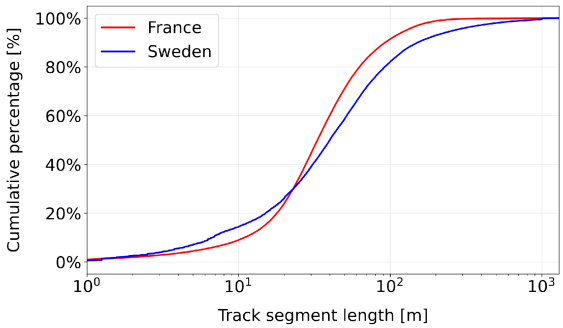

FIGURE 4. Cumulative distribution of track segment lengths (French and Swedish networks).

B. THE STRTREE WITH DISTANCE QUERY

The second reference used in this study is the STRtree, a query-only R-tree spatial index created using the Sort-Tile-Recursive (STR) [52] algorithm. As an R-tree, the STRtree is capable of indexing geometries more complex than points by delimitating them with a bounding box. This is beneficial when handling atoms whose geometries are a series of segments (**Linestring**). This avoids the need for any external mapping variable. Finally, the STRtree has a native distance query which can be applied to search for geometries overlapping with a circulars area around a point, making it a strong candidate for VERNE.

C. THE k-D TREE LR WITH CONTAINING ATOMS QUERY

Likewise, instead of naively using points, a data structure exploiting the smallest non-oriented bounding box enclosing each atom is proposed. The k-d tree LR (or "kLR") is built by iteratively dividing the geographical area, similar to the approach used in the construction of a k-d tree. This partitioning is achieved through a division based on nodes, where the plane is split alternately along the y-axis and x-axis at each consecutive node. The centers of the bounding boxes indicate if an atom ought to be associated with the left or right side of a partitioning node.

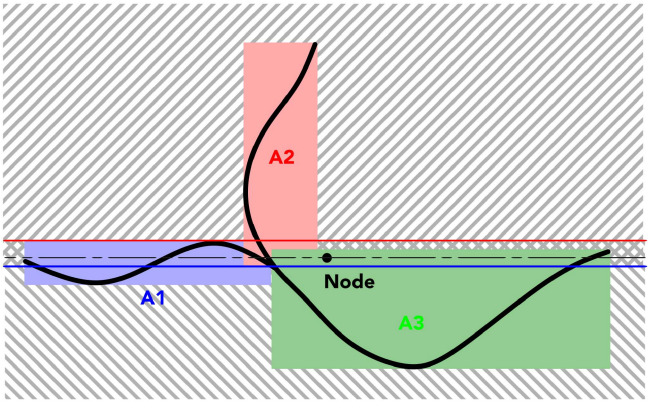

FIGURE 5. Example of network partitioning in an undirected graph using bounding boxes for atoms. This illustration shows the spatial division of atoms (A1, A2, A3) into bounding boxes based on nodes, where each box is alternately split along the x-axis and y-axis to aid in localizing the robot within the network.

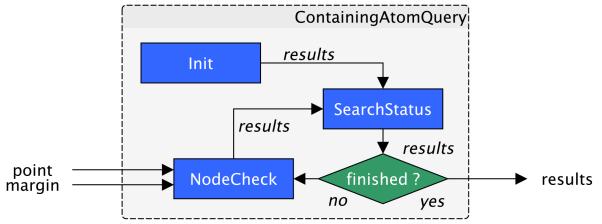

FIGURE 6. Containing Atom Query (CAQ) flowchart.

Algorithm 1: Search Status Function.

Input: *Results*
Output: **True** if the search is done, **False** otherwise
Function SEARCH_STATUS(Results):
1: **for** each (*Key*, *Value*) in *Results* **do**
2: **if** *Value* is **None then**
3: **return False**
4: **return True**

Algorithm 2: *k*LR Node Check Function.

Input: *Node*, *Point*, *Margin*, *Results*
Output: *Results*
Function *k*LR_NODE_CHECK(Node, Point, Margin, Results):
1: **if** Node is not a terminal branch **then**
2: **if** *Node.axis* is *x-axis* **then**
3: *coordinateValue* ← Point.Abscissa
4: **else** ▷axis is *y-axis*
5: *coordinateValue* ← Point.Ordinate
6: **if** *coordinateValue - Margin* ≤ *LeftLimit* **then**
7: ADD (*10* × *Node.Key* + *1*, **None**) to *Results*
8: **if** *coordinateValue + Margin* ≥ *RightLimit* **then**
9: ADD (*10* × *Node.Key* + *2*, **None**) to *Results*
10: **else** ▷Node is a terminal branch
11: B^2 ← GET Bounding Box of Node.Atom
12: **if** *Point* is in B^2 considering *Margin* **then**
13: ADD (*Node.Key*, *Node.Atom*) to *Results*

During the search process, at the level of any node, the query point may fall either to the left, to the right, or simultaneously within both left-right regions, as explained below. This makes it possible to handle the extensive overlaps that exist on the atoms' bounding boxes.

This approach represents a departure from the standard *k*-d tree construction, where partitioning is performed without overlapping uncertainty zones. The name *k*LR comes from using Left and Right limits.

In Fig. 5, three atoms (A1, A2, and A3) are shown within the region to be partitioned along the y-axis. Atom A1 and Atom A3 are primarily located below the node, whereas Atom A2 is predominantly positioned above the node. However, a single ordinate value at the node is insufficient as a strict threshold for partitioning. This can be illustrated through reductio ad absurdum: Assuming that the exact coordinates of a search point are known, it would be incorrectly excluded from A1 and A3 if its ordinate were only slightly higher than that of the node. Nevertheless, as long as the ordinate lies between the levels indicated by the mixed dot-dash line and the red line in Fig. 5, the point may indeed belong to all three atoms.

Using a single threshold is insufficient to partition the regions effectively. Instead, two thresholds are required. The first threshold can be defined as the smallest ordinate (or abscissa, if the partitioning occurs along the x-axis) of the bounding boxes whose center ordinate (or abscissa) is greater than that of the node. The second threshold can be defined as the largest ordinate (or abscissa) of the bounding boxes whose center ordinate (or abscissa) is less than or equal to the node ordinate.

This method ensures that all possible correspondences between a point and an atom are identified; however, it may lead to traversing certain branches of the tree structure unnecessarily. From a computational perspective, these branches can be traversed in parallel across multiple threads. While this parallelization does not reduce the overall computational load, it can still improve query execution speed by allowing concurrent processing.

The outcome of the query algorithm depends on whether the given point of interest can be contained in the bounding box of an atom or not. This is reflected in the name of the search method: Containing Atoms Query (CAQ). The flowchart of a CAQ is presented in fig. 6.

The pseudo-code for this query is outlined as follows. The results are stored in a stack data structure, where each element in the stack consists of a key-value pair. In Algorithm 1, a function is implemented to monitor the status of the search. It returns a Boolean value `True` if the search is complete and `False` otherwise.

Here, a node is defined by a key, an axis, left and right limits, and an index associated with a linked atom. It should be noted that instead of storing the atoms associated with the

node, it was chosen to compute beforehand and store directly the value of the limits from the bounding boxes of these atoms. If the node is not a terminal branch of the search tree, left and right limits are set appropriately and the index of its linked atom is **None**. Otherwise, the index of its linked atom is properly set and can be used to check if the point is within its bounding box. Although the axis could be inferred from other attributes of the node, such as its identifier, a minimal memory compromise was made by storing the axis information explicitly to optimize execution speed.

The root node of the tree is assigned a key of zero. For subsequent nodes, the key is represented as a sequence of 1 s and 2 s, where each step deeper in the tree increases the length of the key by one element. A value of 1 indicates movement to the left branch, while a value of 2 indicates movement to the right branch.

The node-level query function is described in Algorithm 2. Its purpose is to determines whether the progression should terminate at the current node or continue by extending to one or both of the child nodes of the current node. This function requires certain elements to be passed by reference to ensure that at least the `Results` variable is updated during execution.

If the node is a terminal node, it does not further divide the space into two new subsets but instead contains a reference to a single linked atom. This atom exists within the space "delimited by the node"; however, it does not necessarily occupy the entire region. Therefore, it is necessary to check whether the search point is contained within the atom's bounding box. While this does not confirm that the point belongs to the atom itself, it provides a necessary condition.

If the node is not a terminal node, the abscissa or ordinate of the search point (depending on the node's axis) is compared against the left and right limits. If the search point is below the left limit, the search must continue in the left child node. Conversely, if the search point is beyond the right limit, the search must proceed with the right child node. These conditions are evaluated independently, as a point may lie within the uncertainty zone between the two limits. When either or both conditions are met, the indexes of the corresponding child nodes are appended to the results list. The index of each child node is derived from the index of the parent node by multiplying the parent node index by 10, then adding 1 for the left child node and 2 for the right child node. Alternatively, Morton code could be used.

As long as the **SearchStatus** function returns a Boolean value of False upon analyzing the Results structure, this structure should be considered as "in progress" and not yet complete. After completion, the Results structure should be regarded as a list of candidate atoms, which can be further refined.

Algorithm 3 outlines the query progression throughout the entire tree. For this implementation, the atom characteristics are expanded beyond the minimal configuration previously described. Specifically, an atom includes an index, an ordered list of point indices that define its path through a sequence of segments, its bounding box (defined by the coordinates of its center, width, and height), and a list of indices for atoms connected at each extremity. Upon completion of this function, the *Results* variable will contain a list of pairs. In each pair, the second element represents the index of an atom whose bounding box encompasses the search point.

Algorithm 3: *k*LR Containing Atom Query.

Input: *Point*, *Margin*
Output: *Results*
Function *k*LR_CAQ(Point, Margin):
1: **Initialize** *Results* with (0, None)
2: **while** SEARCH_STATUS(Results) is False **do**
3: **Get** (*Node Key*, *Node Value*) from the first element of *Results* and **remove** it
4: **if** *Node Value* is **None then**
5: *k*LR_NODE_CHECK(Node Key, Point, Margin, Results)
6: **else**
7: **Add** (*Node Key, Node Value*) to *Results*

To initiate a query, the structure holding the candidate results is initialized with the pair (0, None). Here, 0 represents the key of the root node. The second element of the pair will later be assigned the identifier of the atom that may contain the search point. This identifier can only be determined if the query reaches a terminal node within a branch.

Subsequently, as long as not all candidate results have been inspected—indicated by the **SearchStatus** function returning **False**—the same sequence of operations is performed iteratively. The first pair in the *Results* structure is retrieved and removed from the structure. If this pair has not been validated in a prior stage, i.e., if no match with an atom has been found, its data is processed by the **NodeQuery** function. Conversely, if a match has been confirmed, the pair is appended back to the end of the *Results* structure.

This procedure yields a refined list of candidate atoms that may contain the search point. The query result can then be finalized by calculating the shortest distance between the segments of each candidate atom and the search point. If, for a given atom, this minimum distance exceeds the GNSS accuracy (potentially adjusted by a correction factor), the hypothesis should be rejected. Otherwise, the hypothesis can be retained as valid.

D. R-TREE OB² WITH CONTAINER ATOMS QUERY

To mitigate the overlapping effect described in the previous section, an alternative approach was developed inspired by the R-tree structure. However, rather than employing standard axis-aligned bounding boxes, oriented bounding boxes are used, hence the name R-tree OB^2 (or "rOB^2"). This modification reduces the area of each bounding element, enabling a more precise determination of whether a search point belongs to a specific atom. Despite this increased discrimination capability, the enclosing element remains sufficiently

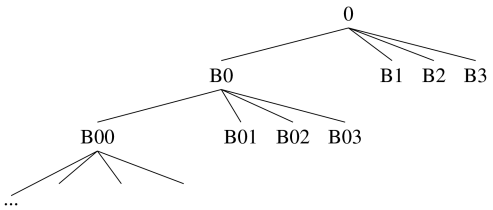

FIGURE 7. Hierarchical tree structure used to organize spatial zones. The root node (0) branches into four sub-zones, labeled B0 through B3, which continue subdividing in subsequent levels.

Algorithm 4: rOB^2 Containing Atom Query.

Input: *Point*, *Margin*
Output: *Results*
Function rOB^2_CAQ(Point, Margin):
Initialize *Results* with {}
for *n* from 1 to 4 **do**
  ADD (*n*, **None**) to *Results*
while SEARCH_STATUS(Results) is False **do**
  Get (*Node Key*, *Node Value*) from the first element of *Results* and **remove** it
  if *Node Value* is **None then**
    rOB^2_NODE_CHECK(Node Key, Point, Margin, Results)
  else
    Add (*Node Key, Node Value*) to *Results*

straightforward to allow efficient computation, ensuring that the process of verifying point containment remains both simple and fast.

To segment the space consistently, each level of the tree structure is divided into a predetermined number of zones, distributing the atoms within each parent zone accordingly. This hierarchical decomposition is not designed to create subregions with identical surface areas but rather to ensure that each region contains an equal number of atoms. For this implementation, four zones per level were selected, as illustrated in Fig. 7.

Given that there are approximately 22,500 atoms, an 8-level tree is sufficient to ensure that each terminal node contains precisely one atom. However, a shallower tree structure may also be feasible, albeit less precise. For instance, in this implementation, a 6-level tree was selected. Each zone is defined by its oriented bounding box and, at the deepest level of the tree, by a list of the atoms it contains.

The query is carried out as described in Algorithm 4. It relies on a check at node level described in Algorithm 5 The query methodology is different from the one used in a *k*LR, but the goal is the same, so the name Containing Atoms Query (CAQ) was chosen.

As implied by its designation, a terminal branch does not possess any daughter branches. Instead, it maintains a list of the atoms that it encompasses. By selecting a sufficiently deep tree structure, the size of these lists can be constrained to include only a small number of atoms. This enables a more efficient evaluation of whether the search point lies within the bounding boxes of the atoms contained in the terminal branch. If the search point is found within a bounding box, the corresponding atom identifier is appended to the list of results.

Algorithm 5: rOB^2 Node Check Function.

Input: *Node*, *Point*, *Margin*, *Results*
Output: *Results*
Function rOB^2_NODE_CHECK(Node, Point, Margin, Results):
$OB^2 \leftarrow$ GET Oriented Bounding Box of *Node*
if *Point* is in OB^2 considering *Margin* **then**
  if Node is not a terminal branch **then**
    for *n* from 0 to 3 **do**
      $ChildKey \leftarrow 10 \times Node.Key + n + 1$
      ADD (*ChildKey*, **None**) to *Results*
  else ▷Node is a terminal branch
    ADD (*Node.Key*, *Node.Atom*) to *Results*

As with the query function presented in the previous chapter, the query result can be further refined, for example by projecting the search point on the segments of the candidate atoms.

V. SIMULATIONS

A. DATA PREPROCESSING

Before it can be used to create the data structures, the raw data needs to be preprocessed. In the scope of this study, a non-exhaustive preprocessing is performed. Tracks looping on themselves, railway turntable or wheelhouses are removed from the raw dataset. Also, a minority of the raw data is encoded with a different format than the rest and is ignored for being unphysical or because they would require heavy manual work to transform into the expected format. A more advanced preprocessing should be done to cover all corner cases before deployment. At the end of this cleaning process, about 0.1% of the total network length is removed for France, and about 0.02% for Sweden.

Thereafter, the cleaned data is interpolated according to different maximum segment lengths, as will be discussed below. Finally, the interpolated data are converted into variants of the four competing data structures. Importantly, the conversion from the cleaned data to a data structure based on atoms has been verified to preserve the full length of the cleaned network data it aims to represent.

B. VALIDATION PROTOCOL

In order to validate both the data structures and the query algorithms, a well-established standard library is used as a reference, namely GeoPandas.[9] It is given the cleaned raw data for both France and Sweden.

[9]https://geopandas.org/

In this validation step, the speed of query is not considered. The goal is to verify that no information is lost when creating and exploiting the data structure. All track segments should be stored and associated with their corresponding atom.

The reference is used to check that, for a given target point, the algorithm outputs at least all atoms which have segments that overlap with the uncertainty disk around the target point. As discussed earlier, the algorithms are not designed to produce the exact set of possible atoms. They are made to quickly generate a reduced set guaranteed to include the one the robot is actually located on, but potentially including several that are irrelevant. Here is an explanation of the testing protocol for each dataset.

The first step is to generated test points on the tracks of the rail network, and to add a 2D Gaussian noise of standard deviation 1 m, such that most points end up at a maximum distance of 3 m from the rails. The choice of this distribution is based on a typical GNSS position uncertainty. All points present in the cleaned and non-interpolated data serve as the starting base for the test points. Three random test points are produced around each of them. Additionally, extra points are created every 10 meters within all segments long enough, for a total of 8,272,078 test points over the French network and 2,631,185 over the Swedish one.

The second step is as follows: For each test point, the GeoPandas function "geopandas.GeoSeries.distance" is used to get all segments that overlap with a disk of radius 3 m centered on the point. A similar query is performed for the CAQ algorithms, also configured with a radius of 3 m. Then, a check is performed such that all segments found by GeoPandas must be included in one of the candidate atoms returned for that point.

C. PERFORMANCE EVALUATION METHOD

The performance of the four proposed solutions is evaluated on the following metrics for both datasets:

- The "duration" of the query.
- The "number of superfluous atoms" included in the query output, i.e., the number of returned atoms that do not truly overlap with the uncertainty disk.
- The "memory usage", i.e., the storage of the data structure, plus any extra memory allocated during the query.

The duration of the query is a key performance indicator (KPI) to inform on the computational cost of each data structure and their associated query. A short computation time would allow an inexpensive real-time processing of incoming position estimations, whether they originate from lower-frequency sources like GNSS (typically 1–10 Hz) or from higher-rate visual-inertial SLAM systems. On the machines used for these comparisons, the operating system has complete control over process scheduling. Therefore, it was ensured that other demanding processes did not overload the CPU cores. In Python, the function time.perf_counter() is used. Only the duration of the query function was evaluated. It does not include the time taken to create the data structure. Also, no multithreading was used to explore branches of the trees. The comparison was conducted on a Unix-like operating system with an AMD EPYC 9354 processor.

The number of superfluous atoms gives an idea of how much work will be needed to further refine the generated superset of possible atoms the robot could be on. If a method were to be faster than the other, but with a significantly worst precision, it could be difficult to conclude the superiority of one over the other. Indeed, the saved computation time during the query phase could be offset by a negative effect on the speed of the refinement phase, because atoms outside the uncertainty disk will have to be unnecessarily processed.

The final metric, memory usage, ensures that the memory required to use the "map" remains within acceptable limits to prevent RAM saturation. This consideration is vital to maintain overall system stability and performance.

During evaluation, the four algorithms are compared based on the same random subset of the test points used for validation. 100,000 points are drawn, and measurements are repeated 100 times for each, in order to reduce variance on the estimated query duration.

When querying the k-d tree, the minimum required search radius for its distance query based on points (DQ) is chosen based on the currently considered data interpolation. For example, a search radius of 2850 meters is needed when no interpolation is performed on the French dataset since its longest segment is around 5700 meters, but it is possible to reduce it to 15 meters by interpolating all segments longer than 30 meters. For the other three methods, a search radius of 3 m is used to be coherent with the distribution of the test points.

The algorithms are implemented using Python 3.12.10. For the k-d tree, the `KDTree` class from the `scipy` library (version 1.16) was utilized rather than implementing the algorithm from scratch. Similarly, the STRtree from the `shapely` library is used (version 2.1.1). Importantly, the kLR data structure and its CAQ query algorithm are formatted into a Numpy array (version 1.26.4) and optimized via the Numba Just-in-Time (JIT) compiler [53] (version 0.61.2). This aims to make a fairer comparison between this custom implementation and the well-established C-optimized open source libraries. On the other hand, the rOB^2 method was not optimized and a native Python implementation was kept.

VI. RESULTS AND DISCUSSION

A. VALIDATION OF kLR AND rOB^2

All of the $8,272,078$ and $2,631,185$ test points used for the French and Swedish network respectively resulted in a comple te superset of true candidate atoms, accomplishing a recall of 100%. Therefore, both algorithms are fully validated on these cleaned datasets.

B. COMPARISON OF THE ALGORITHMS PERFORMANCE

Fig. 8 and Fig. 9 illustrate the main trade-offs that need to be done when considering the four methods proposed here. The numbers written next to all measurement points indicate

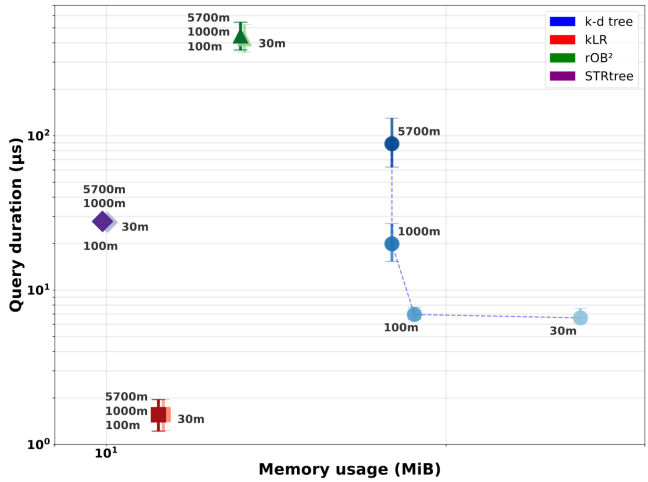

FIGURE 8. Efficiency comparison for different interpolations (France). Bars show median and quartiles.

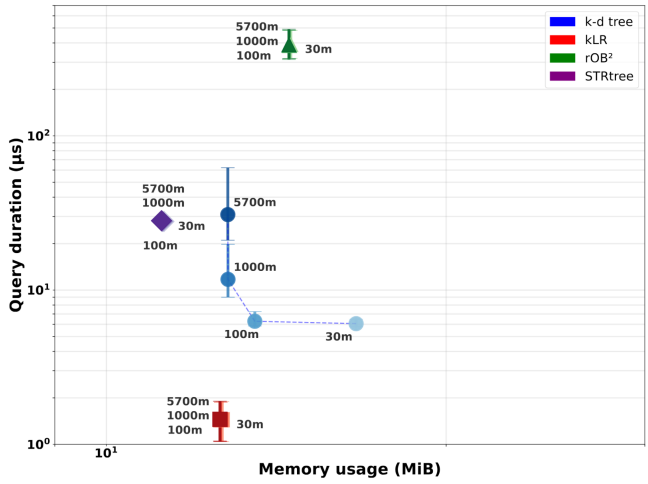

FIGURE 9. Efficiency comparison for different interpolations (Sweden). Bars show median and quartiles.

the maximum segment length in the interpolated cleaned raw data which is used to build each data structure.

For navigation purposes, data structures are queried to fetch the segments of a track, so it is always needed to store the geographic points in memory. To do so, atoms are stored with their respective **Linestring** geometries in a GeoDataFrame from the GeoPandas Python library. Interestingly, the total memory cost is overwhelmingly affected by the number of atoms rather than the number of points. This phenomenon is clearly visible when comparing Fig. 8 and Fig. 9. In fact, the French network is represented by more points than the Swedish network (1, 098, 279 against 270, 419) but because it is made of fewer atoms (22, 450 against 26, 542), its memory footprint is smaller overall.

To allow the use of a generic k-d tree, an additional variable must be stored: a mapping between points found by the query and their respective atom. This extra overhead justifies the stronger sensitivity of the k-d tree with regard to the total number of points in the dataset. On the contrary, for other methods using bounding boxes the actual resolution of the segments does not matter, and their queries are able to directly return the indexes of plausible containing atoms. This relaxed coupling between memory requirements and query speed is a valuable characteristics. It would make VERNE a scalable and memory-efficient solution ready to be deployed on embedded systems for real-world applications where data resolution can vary and disk space is a limiting factor.

The STRtree is the most memory-efficient, with a negligible overhead induced by its bounding-box indexing. On the other hand, the "kLR with CAQ" has a slightly more expensive indexing, using not only bounding boxes but also left and right limits, but achieves the shortest query duration.

The k-d tree can reach shorter query durations than the STRtree when using a high resolution interpolation, but it comes at the cost of a steep increase in memory consumption.

The "rOB^2 with CAQ" algorithm does not stand out among the evaluated methods. Its reliance on oriented bounding boxes introduces additional computational steps during each iteration of the search, as outlined below:

- Calculation of a two-component vector based on the coordinates of the search point and the center of the oriented bounding box;
- Computation of a transformation matrix (a 2×2 matrix involving trigonometric cosine and sine functions);
- Multiplication of the transformation matrix with the previously computed vector;
- Comparison of the resulting values against the width and height of the oriented bounding box.

In contrast, the other solutions require only the equivalent of the final comparison step. Potential optimizations are available. For example, instead of storing the rotation angle, the transformation matrix could be precomputed and stored, thereby reducing computational complexity. However, even with such optimizations, the "rOB^2 with CAQ" algorithm could remain inherently slower than the most efficient solution identified in this study.

To compare beyond the query speed and memory consumption, Fig. 10 and Fig. 11 provide statistics on the number of superfluous atoms produced by each approach. Algorithms using bounding boxes offer the same precision for any interpolation (except negligible variations due to some randomness during their construction from the interpolated data points). Thus, for these queries, precision statistics are calculated and plotted once from the pool of all measurements collected across different interpolations.

When considering interpolations with a resolution no better than 30 meters, the "rOB^2 with CAQ" algorithm produces the smallest number of superfluous atoms on average. In 90% of the test cases, the algorithm returns no more than two, and in 99% of the cases, at most five are generated. This superiority is explained by the use of oriented bounding boxes, able to better differentiate atoms between each other, especially when they are close to one another. Although its performance

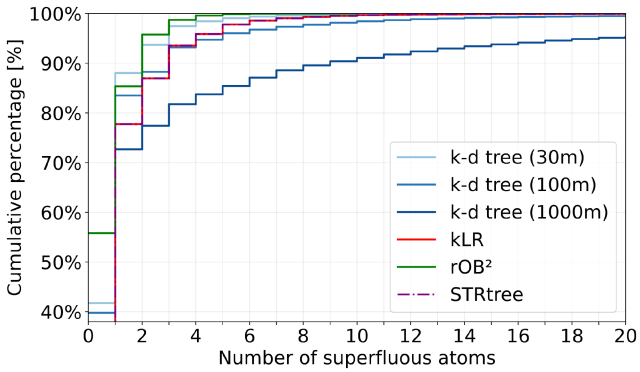

FIGURE 10. Cumulative distribution of the number of superfluous atoms returned by different methods (France).

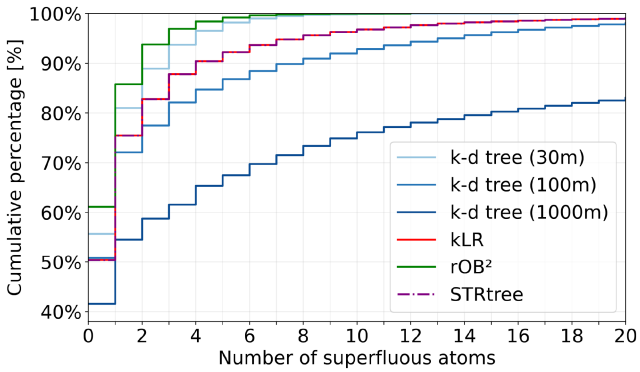

FIGURE 11. Cumulative distribution of the number of superfluous atoms returned by different methods (Sweden).

could be further enhanced by increasing the depth of the tree (e.g., from six to eight levels), this adjustment would extend query times. The "*k*LR with CAQ" and STRtree algorithms also deliver a satisfactory precision. Since they both use non-oriented bounding boxes, it is logical that they converge toward the same boxes/atoms at the end of the query process. Again, the *k*-d tree with DQ could achieve the best precision with a more memory-expensive interpolation.

Overall, the proposed *k*LR data structure and its query algorithm—dividing space alternately along the x-axis and y-axis with overlapping regions—demonstrate the best effectiveness when applied to a real-world dataset. This approach satisfies industrial requirements and highlights the potential of spatial division strategies in practical applications. Therefore, it shall be used as the inner mechanism for VERNE. Nevertheless, for applications where memory constraints prevail, the use of a traditional STRtree can be advised, at the cost of a non negligible increase in query time.

C. LIMITATIONS

The presented simulations demonstrate that for a given set of atoms and a given uncertainty disk, VERNE will consistently find a superset containing all plausible atoms. In this study, an extensive but partial preprocessing has been used to convert real track segments into atoms, and a complete one is achievable given enough resources. Despite this, precautions should be taken before real-world deployment.

First, all parts of the networks where the robot is expected to travel should be represented in VERNE. Missing track segments would be impossible to reach with an autonomous navigation system, and they could prevent the robot to travel from one atom to another, despite the existence of rails connecting them. However, this issue can be detected early, during the robot path generation phase.

Another risk to consider during operations is that junctions may not always be set in the position expected by the path planning algorithm. To catch such failure, VERNE can be used to monitor if the robot is progressing correctly along the desired atom route. Consequently, the robot's behavior can be adapted (e.g. reducing speed at times of uncertainty) and recovery behaviors can be implemented as soon as a trajectory error is detected (e.g. returning to the last known correct position within a given time limit) thus ensuring safe movement. Similar strategies are relevant if VERNE is created from an incorrect dataset containing segments that do not exist in the real track network. Such data would result in the erroneous generation of unrealistic paths. As the robot moves along the real, physical track, significant deviation from the planned trajectory will be observed, and action can be taken.

It is also necessary to take into account uncertainties in the geographic coordinates of the segments stored in any digital map used for localization. For instance, an error of two meters with a 90% confidence interval has been empirically observed by engineers working with the Swedish dataset. The optimization of the query margin would require more information on the inaccuracy of public datasets, unavailable for the time being. On another note, most odometry sources have a more complex uncertainty area than a disk, which would need specific analyses.

Lastly, when deploying robots using a digital map such as VERNE, operators should be ready to intervene if the initial localization of the robot cannot be resolved automatically, for example if the margins used or the current sensors' inaccuracies prevent a non-ambiguous localization between multiple atoms upon startup.

VII. CONCLUSIONS AND FUTURE WORK

This paper presents VERNE, a comprehensive framework aiming to format raw geographic data into a fast, memory-efficient, practical and customizable data structure capable of representing railway topology. Both its data structure and its query algorithm are provided. The ultimate goal is to facilitate robust and efficient localization of robots operating on railway environments by leveraging a structured and scalable representation.

This study demonstrates that, thanks to a spatial data structure based on a simple topological representation, VERNE can effectively address the problem of querying candidate atoms with a performance level suitable for high-frequency

onboard computation. The proposed structure is both visualizable and interpretable, ensuring comprehensibility for human operators. This includes properties such as limits in terms of abscissa and ordinate, bounding box positions, and sizes.

Through an extensive comparison between two common spatial data structures and two custom ones, the internal mechanism of VERNE results from a compromise between memory size and query speed. In any case, VERNE does not rely on fine-grained interpolation or intensive preprocessing, and delivers strong performance in terms of precision.

While the scalability of VERNE has been demonstrated between the Swedish and French networks, the non-trivial relation between total memory size and network characteristics (e.g., length, complexity and resolution) could be subject to further investigations on a wider variety of countries.

Limited efforts have been made here to optimize overlaps between bounding boxes, particularly when using the rOB^2 structure. This remains an area of interest for future studies and could provide valuable insights into improving the performance of that spatial data structures.

As mentioned above, more comprehensive description models exist but lack a deployment-ready spatial data structure. It could be investigated how VERNE can be combined with them to benefit from advanced description capabilities as well as memory-efficient and fast queries. For example, offering a conversion service (import/export) between RailML and VERNE could contribute positively to scaling up the solution. The translation of the VERNE framework into an open-source tool is a clear objective that would significantly benefit the field. Intensive work is currently being carried out on the remaining non-trivial task of creating a user-friendly build system with comprehensive documentation.

To further evaluate its robustness and performance in operational scenarios, VERNE will be integrated into a robotic platform and tested under real-world conditions.

REFERENCES

[1] G. Jing, X. Qin, H. Wang, and C. Deng, "Developments, challenges, and perspectives of railway inspection robots," *Automat. Construction*, vol. 138, 2022, Art. no. 104242.

[2] H. Liu et al., "An autonomous rail-road amphibious robotic system for railway maintenance using sensor fusion and mobile manipulator," *Comput. Elect. Eng.*, vol. 110, 2023, Art. no. 108874. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0045790623002987

[3] M. Rahman, H. Liu, M. Masri, I. Durazo-Cardenas, and A. Starr, "A railway track reconstruction method using robotic vision on a mobile manipulator: A proposed strategy," *Comput. Ind.*, vol. 148, 2023, Art. no. 103900.

[4] M. Rahman, H. Liu, I. D. Cardenas, A. Starr, A. Hall, and R. Anderson, "A review on the prospects of mobile manipulators for smart maintenance of railway track," *Appl. Sci.*, vol. 13, no. 11, 2023, Art. no. 6484.

[5] D. Mikhaylov et al., "Toward the future generation of railway localization exploiting RTK and GNSS," *IEEE Trans. Instrum. Meas.*, vol. 72, 2023, Art. no. 8502610.

[6] I. Millan-Jimenez, P. Zabalegui, G. De Miguel, J. Mendizabal, and I. Adin, "Map-matching techniques for train localisation: A taxonomic survey," *IEEE Access*, vol. 12, pp. 192328–192340, 2024.

[7] S. Taguchi, S. Koide, and T. Yoshimura, "Online map matching with route prediction," *IEEE Trans. Intell. Transp. Syst.*, vol. 20, no. 1, pp. 338–347, Jan. 2018.

[8] SNCF Réseau, "Open data," 2023. [Online]. Available: https://data.sncf.com

[9] S. S. Saab, "A map matching approach for train positioning. I. development and analysis," *IEEE Trans. Veh. Technol.*, vol. 49, no. 2, pp. 467–475, Mar. 2000.

[10] O. Heirich, P. Robertson, and T. Strang, "Railslam-localization of rail vehicles and mapping of geometric railway tracks," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2013, pp. 5212–5219.

[11] J. Liu, B.-g. Cai, and J. Wang, "A GNSS/trackmap cooperative train positioning method for satellite-based train control," in *Proc. 17th Int. IEEE Conf. Intell. Transp. Syst.*, 2014, pp. 2718–2724.

[12] W. Jiang, S. Chen, B. Cai, J. Wang, W. ShangGuan, and C. Rizos, "A multi-sensor positioning method-based train localization system for low density line," *IEEE Trans. Veh. Technol.*, vol. 67, no. 11, pp. 10425–10437, Nov. 2018.

[13] W. Löffler and M. Bengtsson, "Evaluating the impact of map inaccuracies on path discrimination behind railway turnouts," in *Proc. IEEE 95th Veh. Technol. Conference:(VTC2022-Spring)*, 2022, pp. 1–5.

[14] M. Lauer and D. Stein, "A train localization algorithm for train protection systems of the future," *IEEE Trans. Intell. Transp. Syst.*, vol. 16, no. 2, pp. 970–979, Apr. 2014.

[15] O. Heirich, "Bayesian train localization with particle filter, loosely coupled GNSS, IMU, and a track map," *J. Sensors*, vol. 2016, no. 1, 2016, Art. no. 2672640.

[16] O. Heirich, P. Robertson, A. C. Garcıá, T. Strang, and A. Lehner, "Probabilistic localization method for trains," in *Proc. IEEE Intell. Veh. Symp.*, 2012, pp. 482–487.

[17] F. Böhringer and A. Geistler, "Location in railway traffic: Generation of a digital map for secure applications," *WIT Trans. Built Environ.*, vol. 88, pp. 459–468, 2006.

[18] K. Gerlach and M. M. zu Hörste, "A precise digital map for galileo-based train positioning systems," in *Proc. 9th Int. Conf. Intell. Transport Syst. Telecommun.*, 2009, pp. 343–347.

[19] S. Wandelt, Z. Wang, and X. Sun, "Worldwide railway skeleton network: Extraction methodology and preliminary analysis," *IEEE Trans. Intell. Transp. Syst.*, vol. 18, no. 8, pp. 2206–2216, Aug. 2016.

[20] H. Winter, S. Luthardt, V. Willert, and J. Adamy, "Generating compact geometric track-maps for train positioning applications," in *Proc. IEEE Intell. Veh. Symp. (IV)*, 2019, pp. 1027–1032.

[21] H. Li et al., "Integrated representation of geospatial data, model, and knowledge for digital twin railway," *Int. J. Digit. Earth*, vol. 15, no. 1, pp. 1657–1675, 2022.

[22] A. Hlubuček, "Railtopomodel and railml 3 in overall context," *Acta Polytechnica CTU Proc.*, vol. 11, pp. 16–21, 2017.

[23] A. Nash, D. Huerlimann, J. Schütte, and V. P. Krauss, "RailMLa standard data interface for railroad applications," *WIT Trans. Built Environ.*, vol. 74, pp. 5–6, 2004.

[24] A. Nash and D. Huerlimann, "Railroad simulation using opentrack," *WIT Trans. Built Environ.*, vol. 74, pp. 5–9, 2004.

[25] R. A. Finkel and J. L. Bentley, "Quad trees a data structure for retrieval on composite keys," *Acta Informatica*, vol. 4, pp. 1–9, 1974.

[26] H. Samet, "Hierarchical spatial data structures," in *Proc. Symp. Large Spatial Databases*, 1989, pp. 191–212.

[27] J. L. Bentley, "Multidimensional binary search trees used for associative searching," *Commun. ACM*, vol. 18, no. 9, pp. 509–517, Sep. 1975. [Online]. Available: https://doi.org/10.1145/361002.361007

[28] N. Sample, M. Haines, M. Arnold, and T. Purcell, "Optimizing search strategies in kd trees," in *Proc. 5th WSES/IEEE World Multiconference Circuits, Syst., Commun. Comput.*, 2001, pp. 8–9.

[29] J. H. Friedman, J. L. Bentley, and R. A. Finkel, "An algorithm for finding best matches in logarithmic expected time," *ACM Trans. Math. Softw.*, vol. 3, no. 3, pp. 209–226, 1977.

[30] A. Guttman, "R-trees: A dynamic index structure for spatial searching," in *Proc. 1984 ACM SIGMOD Int. Conf. Manage. Data*, 1984, pp. 47–57.

[31] S. Brakatsoulas, D. Pfoser, and Y. Theodoridis, "Revisiting r-tree construction principles," in *Proc. East Eur. Conf. Adv. Databases Inf. Syst.*, 2002, pp. 149–162.

[32] P. Ciaccia et al., "M-Tree: An efficient access method for similarity search in metric spaces," in *Proc. VLDB*, 1997, vol. 97, pp. 426–435.

[33] T. Bozkaya and M. Ozsoyoglu, "Indexing large metric spaces for similarity search queries," *ACM Trans. Database Syst.*, vol. 24, no. 3, pp. 361–404, Sep. 1999. [Online]. Available: https://doi.org/10.1145/328939.328959

[34] S. Lefebvre and H. Hoppe, "Perfect spatial hashing," *ACM Trans. Graph.*, vol. 25, no. 3, pp. 579–588, 2006.
[35] M. Tang, Z. Liu, R. Tong, and D. Manocha, "PSCC: Parallel self-collision culling with spatial hashing on GPUs," in *Proc. ACM Comput. Graph. Interactive Techn.*, vol. 1, no. 1, pp. 1–18, 2018.
[36] F. Aurenhammer and R. Klein, "Voronoi diagrams," in *Handbook of Computational Geometry*. Amsterdam, The Netherlands: Elsevier, 2000, pp. 201–290.
[37] A. Andreou, C. X. Mavromoustakis, J. M. Batalla, E. K. Markakis, and G. Mastorakis, "UAV-assisted RSUs for V2X connectivity using voronoi diagrams in 6g infrastructures," *IEEE Trans. Intell. Transp. Syst.*, vol. 24, no. 12, pp. 15855–15865, Dec. 2023.
[38] M. M. Alam, L. Torgo, and A. Bifet, "A survey on spatio-temporal data analytics systems," *ACM Comput. Surv.*, vol. 54, no. 10, pp. 1–38, 2022.
[39] J. L. Vermeulen, A. Hillebrand, and R. Geraerts, "A comparative study of k-nearest neighbour techniques in crowd simulation," *Comput. Animation Virtual Worlds*, vol. 28, no. 3-4, 2017, Art. no. e1775.
[40] Y. Manolopoulos, A. Nanopoulos, A. N. Papadopoulos, and Y. Theodoridis, "R-trees have grown everywhere," Tech. Rep. 2003. [Online]. Available: http://www.rtreeportal.org
[41] N. Beckman, H.-Peter Begel, and R. Schneider, "The R*-tree: An efficient and robust access method for points and rectangles," in *Proc. 1990 ACM SIGMOD Int. Conf. Manage. Data*, vol. 19, no. 2, pp. 322–331, 1990.
[42] T. Kraska, A. Beutel, E. H. Chi, J. Dean, and N. Polyzotis, "The case for learned index structures," in *Proc. 2018 Int. Conf. Manage. Data*, 2018, pp. 489–504.
[43] J. Ding et al., "ALEX: An updatable adaptive learned index," in *Proc. 2020 ACM SIGMOD Int. Conf. Manage. Data*, 2020, pp. 969–984.
[44] V. Pandey, A. van Renen, A. Kipf, I. Sabek, J. Ding, and A. Kemper, "The case for learned spatial indexes," 2020, *arXiv:2008.10349*.
[45] V. Nathan, J. Ding, M. Alizadeh, and T. Kraska, "Learning multi-dimensional indexes," in *Proc. 2020 ACM SIGMOD Int. Conf. Manage. Data*, 2020, pp. 985–1000.
[46] P. Li, H. Lu, Q. Zheng, L. Yang, and G. Pan, "LISA: A learned index structure for spatial data," in *Proc. 2020 ACM SIGMOD Int. Conf. Manage. Data*, 2020, pp. 2119–2133.
[47] M. Li et al., "A survey of multi-dimensional indexes: Past and future trends," *IEEE Trans. Knowl. Data Eng.*, vol. 36, no. 8, pp. 3635–3655, Aug. 2024.
[48] Z. Gharaee, S. Kowshik, O. Stromann, and M. Felsberg, "Graph representation learning for road type classification," *Pattern Recognit.*, vol. 120, 2021, Art. no. 108174.
[49] Swedish Transport Administration, "Lastkajen," 2024. [Online]. Available: https://lastkajen.trafikverket.se
[50] F. Ahmadzai, K. Rao, and S. Ulfat, "Assessment and modelling of urban road networks using integrated graph of natural road network (a GIS-based approach)," *J. Urban Manage.*, vol. 8, no. 1, pp. 109–125, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2226585618301341
[51] U. Demšar, O. Špatenková, and K. Virrantaus, "Centrality measures and vulnerability of spatial networks," in *Proc. 4th Int. Conf. Inf. Syst. Crisis Response Manage. Academic Proc.*, 2007, pp. 3–4.
[52] S. T. Leutenegger, M. A. Lopez, and J. Edgington, "STR: A simple and efficient algorithm for r-tree packing," in *Proc. 13th Int. Conf. Data Eng.*, 1997, pp. 497–506.
[53] S. K. Lam, A. Pitrou, and S. Seibert, "Numba: A LLVM-based python JIT compiler," in *Proc. 2nd Workshop LLVM Compiler Infrastructure HPC*, 2015, pp. 1–6.

LOUIS-ROMAIN JOLY graduated from the Ecole Nationale Supérieure des Arts et Métiers (ENSAM), Paris, France, and from the Universitaet Karslruhe, Karlsruhe, Germany, in 2001. He is currently a Program Manager with the Research Department of SNCF, the french historical railway company. His activities are now focused on robotics for railway maintenance. His main target is to create an ecosystem that will boost robotics in the whole railway sector at least in Europe.

VIVIEN LACORRE (Graduate Student Member, IEEE) received the M.Sc. degree from the INP-ENSEEIHT engineering school of Toulouse, Toulouse, France in 2022. He is currently working toward the Ph.D. degree with the Department of Mechanical Engineering, Chalmers University of Technology, Gothenburg, Sweden. His research interests include localization, navigation and artificial intelligence applied to autonomous robots.

KRISTER WOLFF received the Ph.D. degree from the Chalmers University of Technology, Gothenburg, Sweden, in 2006. He is currently an Associate Professor of Applied AI, a Faculty Member, and Vice Head of the Department of Mechanical Engineering, with Chalmers University of Technology. His research focuses on the application of artificial intelligence in autonomous robots and self-driving vehicles, with an emphasis on machine learning and bio-inspired computational methods.