vehicles

MDPI

Article

iLDM: An Interoperable Graph-Based Local Dynamic Map

Mikel García *,†, Itziar Urbieta *,†, Marcos Nieto, Javier González de Mendibil and Oihana Otaegui

Vicomtech Foundation, Basque Research and Technology Alliance (BRTA), 20009 San Sebastián, Spain; mnieto@vicomtech.org (M.N.); jgonzalezdemendibil@vicomtech.org (J.G.d.M.); ootaegui@vicomtech.org (O.O.)

* Correspondence: mgarcia@vicomtech.org (M.G.); iurbieta@vicomtech.org (I.U.)

† These authors contributed equally to this work.

Abstract: Local dynamic map (LDM) is a key component in the future of autonomous and connected vehicles. An LDM serves as a local database with the necessary tools to have a common reference system for both static data (i.e., map information) and dynamic data (vehicles, pedestrians, etc.). The LDM should have a common and well-defined input system in order to be interoperable across multiple data sources such as sensor detections or V2X communications. In this work, we present an interoperable graph-based LDM (iLDM) using Neo4j as our database engine and OpenLABEL as a common data format. An analysis on data insertion and querying time to the iLDM is reported, including a vehicle discovery service function in order to test the capabilities of our work and a comparative analysis with other LDM implementations showing that our proposed iLDM outperformed in several relevant features, furthering its practical utilisation in advanced driver assistance system development.

Keywords: local dynamic map; LDM; iLDM; V2X; OpenLABEL; interoperability; Neo4j; ADAS; graph databases; real-time

Citation: García, M.; Urbieta, I.; Nieto, M.; González de Mendibil, J.; Otaegui, O. iLDM: An Interoperable Graph-Based Local Dynamic Map. *Vehicles* **2022**, *4*, 42–59. https://doi.org/10.3390/vehicles4010003

Academic Editor: Yongzhi Zhang, Daniel J. Auger, Chongfeng Wei and Chun Wang

Received: 20 December 2021
Accepted: 5 January 2022
Published: 8 January 2022

1. Introduction

The implementation of a standardised, interoperable, and efficient local dynamic map (LDM) component is one of the actual challenges in the context of cooperative intelligent transport systems (C-ITS). Information about the local environment is essential for automated vehicles, and with the constant developments achieved in object detection tasks for perception systems, such as cameras or LiDAR sensors, vehicles are able to perceive their environment with more precision and detail. This data must be properly structured and stored within the ego-vehicle for its posterior usage in different kinds of ADAS functions or at roadside units (RSUs) for wider-area local dynamic maps in cooperative driving contexts. LDM implementations, such as software solutions, respond to this need, aiming to provide a centralised database of road scene information including static and dynamic elements that are relevant to supporting the integration of advanced driver assistance systems (ADAS) and automated driving (AD) functions. In terms of functionality, a software component implementing the LDM specification receives information from map services, perception systems and vehicle-to-everything (V2X) communication channels in order to feed and maintain a live database with local information of the scene. Historical data can be kept and archived in files. The benefits of using an LDM implementation for C-ITS are:

- Centralisation of descriptive scene information in real time;
- Harmonisation of format and interfaces;
- Compatibility with standards.

The implementation of an LDM can support multiple C-ITS use cases, i.e., ADAS or AD functions that require receiving information from perception systems, mapping or V2X communications to produce further information about the scene. Examples of such ADAS/AD functions are:

- Time-to-collision estimation;
- Vehicle discovery service (VDS);
- Out-of-road prediction;
- Online manoeuvre/scene annotation;
- Lane change assist;
- Parking assistant.

Despite LDM being a well-known concept in C-ITS domains, standard-compliant LDM implementations have been largely ignored by the community and industry; thus, much research is yet to come to provide functional data models, real-time databases, interoperable interfaces and general-purpose applicability. Most of the few existing implementations [1–3] depend on relational database management systems (RDBMS), directly storing data into relational tables without using a scalable and flexible data model. The use of a standardised data format for storing objects into the LDM has not been taken into consideration in other works. Graph databases, instead, can be a good alternative over the predominant RDBMS approaches in the literature due to the fact of their native management of data relationships and its match with the highly connected nature of the data in driving scenarios.

In this paper, we propose the iLDM as a novel graphical database based on LDM implementation and present an analysis of the feasibility for its utilisation in real-time ADAS applications. Our approach is focused on interoperability and extensibility by means of using standard interfaces (OpenLABEL [4]) and an underlying concept ontology, automotive global ontology (AGO) [5]. The main contributions of this paper are the following:

- Proposing the use of the new OpenLABEL standard as a common data source and making the iLDM interoperable across different domains and data sources;
- Using a graph-based database against the usual RDBMS approaches;
- An LDM architecture suitable for real-time use due to the fact of its data insertion and querying response time.
- Numerical analysis of the iLDM's performance for data insertion and retrieval.

2. Related Work

The concept of an LDM has been widely adopted by the automotive sector since its introduction in 2010 with the SAFESPOT [6] project. This project defined the seminal four-layer structure (Figure 1) on which most of the LDM implementations are based on currently.

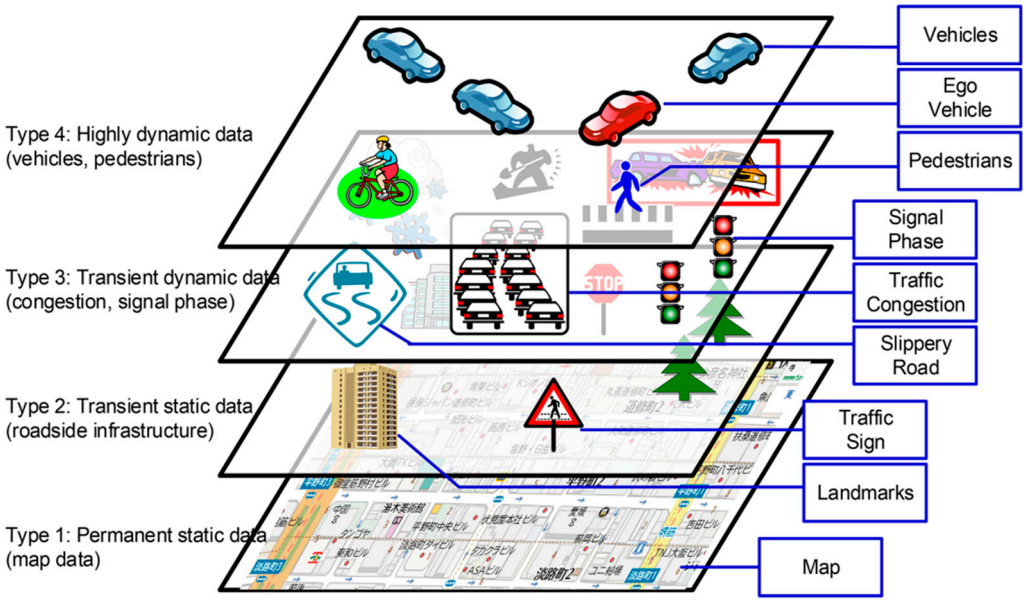

Figure 1. LDM layer structure. (Reprinted from Ref. [2]).

- Layer 1 (static): permanent static data (map data with road topology);
- Layer 2 (quasi-static): transient static data (traffic signs, buildings and roadside infrastructure);
- Layer 3 (transient): transient dynamic data (congestion information and signal phases);
- Layer 4 (dynamic): high dynamic data (position of pedestrians, vehicles and ego-vehicle).

Subsequently, standardising efforts started in 2011 with the European Telecommunications Standards Institute (ETSI)'s TR 102 863 [7] report, later updated with the definition of the interfaces to LDM data providers, consumers and to the security and management layers among the information flow in ETSI EN 302 895 [8] and the ISO [9] standard. However, these standards only specify the functional requirements and interfaces of the system. Therefore, the structuring of the information to fulfil the specifications is open to interpretation by the developer and has evolved in recent approaches towards a six-layer model [10] as defined in the PEGASUS research project, which concluded in 2019 [11]. In [12], an exhaustive analysis of the elements required for a complete description of a traffic scenario is presented based on the six-layer model introduced by PEGASUS. The additional layers included in the model are related to environmental conditions (L5) and digital information (L6). Regarding vehicular communication, relevant standards were created for the specification of V2X messages, including cooperative awareness message (CAM [13]) or, more recently, collective perception message (CPM [14]), allowing for the transmission of information regarding the ego-vehicle or the perception of its surroundings.

For the implementation of an LDM system, the data model and storage approach are crucial and will directly influence its performance. Studying the available bibliography reveals that most of the approaches make use of SQL-based database engines for storing the information. This can already be seen in [1] with the NAVTEQ-LDM implementation using SQLite or the work conducted by Tele Atlas and Bosch with the PG-LDM using PostgreSQL. Based on the SAFESPOT specifications, more LDM implementations have been suggested as seen in [2], which combined PostgreSQL and PostGIS as was performed for PG-LDM. More novel approaches have proposed different database models. In [3], the authors make use of an ontology to manage the meaning of concepts, exposing semantic query interfaces and PipelineDB [15] as a spatial stream RDBMS capable of reading ETSI messages (e.g., CAM). The work presented in [16] is the first to propose a graph-based database, introducing the term Relational-LDM (R-LDM). Currently, data obtained from V2X communications or object detection algorithms can be defined in different types, formats, represent several concepts and have differences in quality. A construction of graph databases is more suitable for these data characteristics due to the fact of their flexibility over traditional RDBMS.

On the other hand, the Dynamap implementation [17] was developed in 2013 for C-ITS, but unlike the previously mentioned works, its architecture was not defined as a data storing system but rather as an information system. The idea was to process the information in memory. In addition, [18] it clearly stated the importance of having a minimal latency for generating trustful information for the ITS stations. Hence, with the aim of working in real time, the suggested architecture is based on three servers to manage different services that will be connected to a relational database and a real-time database. However, the analysis conducted in the paper was not based on the data modelling or these storage systems. Instead, the analysis was based overall on the communication architecture and the object detection process.

In our work, an interoperable real-time graph-based LDM implementation is proposed. The approach is framed using OpenLABEL as a common data format for all inputs of the iLDM (including perception information, standard V2X messages and map nodes). Moreover, a performance analysis of the iLDM was carried out for both data insertion and querying. In Section 3, we explain the architecture and the data model of our iLDM. Section 4 is focused on the implementation details of the iLDM. Section 5 includes numerical results for data insertion and querying of the iLDM, a comparison between existing LDM implementations and an example ADAS function example using the iLDM API. Finally, Section 6 contains the conclusions of this work.

3. iLDM Architecture

3.1. Functional Architecture

The functional architecture of the iLDM is illustrated in Figure 2 and is defined based on the following main components:

- iLDM database: the database that stores the static and dynamic information;
- iLDM database interface: the interface of the database to perform CRUD (create/read/update/delete) operations;
- iLDM API: the set of exposed functions to interact with the database including input and output interfaces. Internal functions provide the necessary mechanisms to organise and convert the information into the LDM layers. Six main exported functions are considered:
    a. Configure: configures the parameters of the iLDM such as time-to-live (TTL) values and filtering routines;
    b. Add objects: introduces new or updated information about dynamic objects;
    c. Load map: introduces new or updated information about map elements;
    d. Export: retrieves sets of information for certain time intervals to store in a file system or archive;
    e. Read objects: retrieves real-time information about certain elements at certain time instants or intervals;
    f. Get info: retrieves statistics or information about the status of the database.

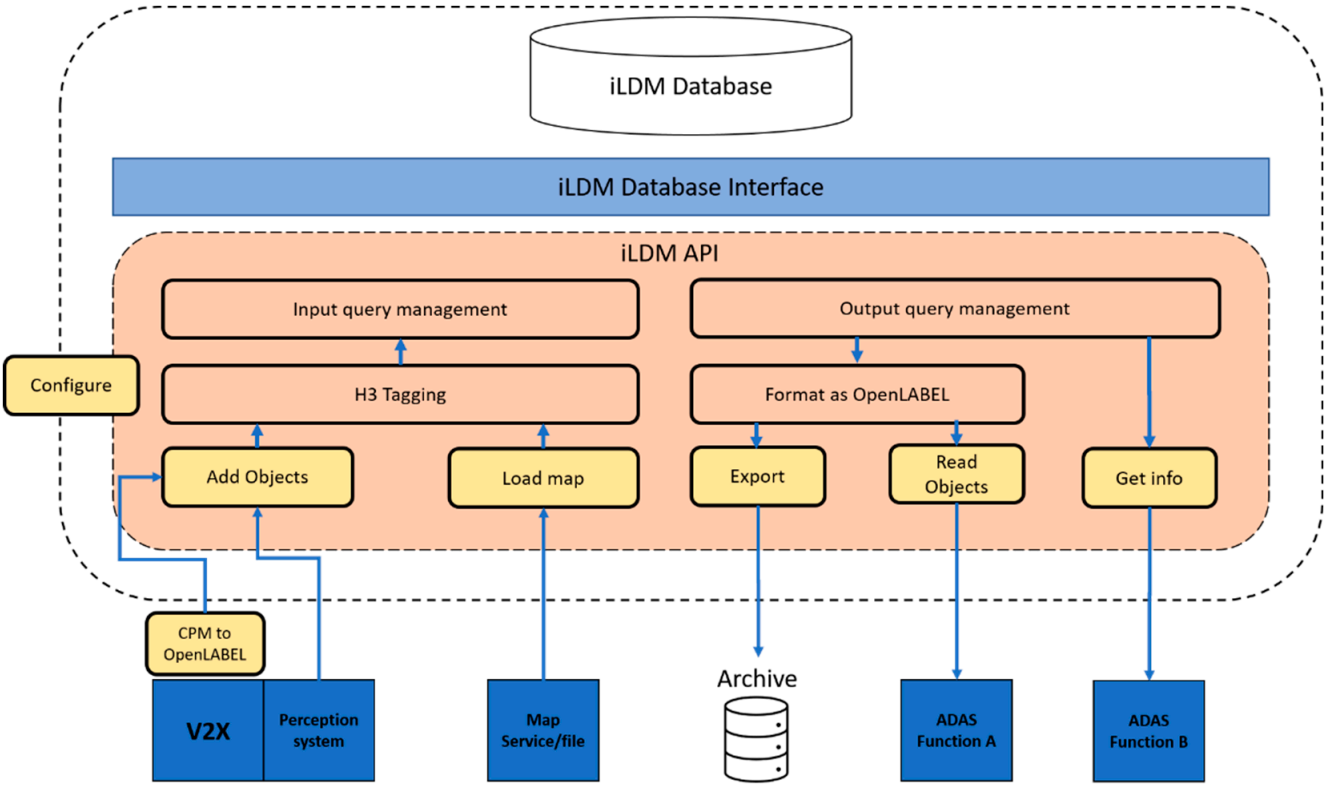

Figure 2. Functional architecture of the iLDM component.

The input interface to the iLDM can be used by three possible types of systems:

- Map service/files: digital map information received from an online service or as offline (pre-downloaded) files to fill in the lower layer (static data) of the iLDM stack;
- Perception system: local system inside the ego-vehicle that perceives information on the surroundings as a list of objects or a similar representation. Conversion to OpenLABEL format is required prior to its ingestion into the iLDM;

- V2X: communication system that receives messages (e.g., ETSI CAM and CPM messages) from exterior perception systems (e.g., other vehicles). Conversion from CPM to OpenLABEL is needed to ingest information into the iLDM.

The output interface from the iLDM shall be used by two possible types of systems:

- Archive/data logging: the iLDM component accumulates dynamic information in time. A clear-up policy, set-up during configuration, might establish that information older than a certain threshold is periodically removed. Archiving into local or remote file systems can help prevent loss of information;
- ADAS/AD functions: the second type of application that reads from iLDM is any ADAS/AD function that needs to read synchronised and calibrated information about dynamic objects and their position or relation with the underlying static layers.

3.2. Dynamic Layer (L4) Data Model

The iLDM database is structured as a graph database, and the different layers of information mentioned in Section 1 are distinguished by setting the corresponding labels (i.e., L1 to L4).

As far as the dynamic layer is concerned (L4), the information is structured following the principles of the OpenLABEL standard, where a hierarchy of elements, frames, streams, coordinate systems and metadata allows to define rich and complex static and dynamic properties of a scene including time-specific information, object geometries and attributes, transformations between coordinate systems and relationships between elements in JSON format.

Figure 3 depicts the data model defined for the dynamic layer of the iLDM. The represented element types correspond to the main superclasses defined in the domain model of the AGO ontology [5] in order to provide the semantic capabilities of the iLDM. Due to the spatio-temporal nature of the dynamic data, as represented in Figure 3, the elements are related with "frame" nodes, and the information received in a certain time instant is kept by updating the values of the element properties.

Furthermore, the main structural difference between OpenLABEL and the iLDM data model is related to the "relations". While in the standard these are subclasses of "elements", for the functional implementation of the iLDM, they are expressed as edges of the graph; that is, they are represented as relationships between nodes, since this approach allows for the construction of a semantically enriched data model. Therefore, our approach provides a mapping of the OpenLABEL information into a graph.

The benefits of this graph structure can be exploited easily using Neo4J [19], where the necessary data for the iLDM can be expressed using nodes through all the different layers.

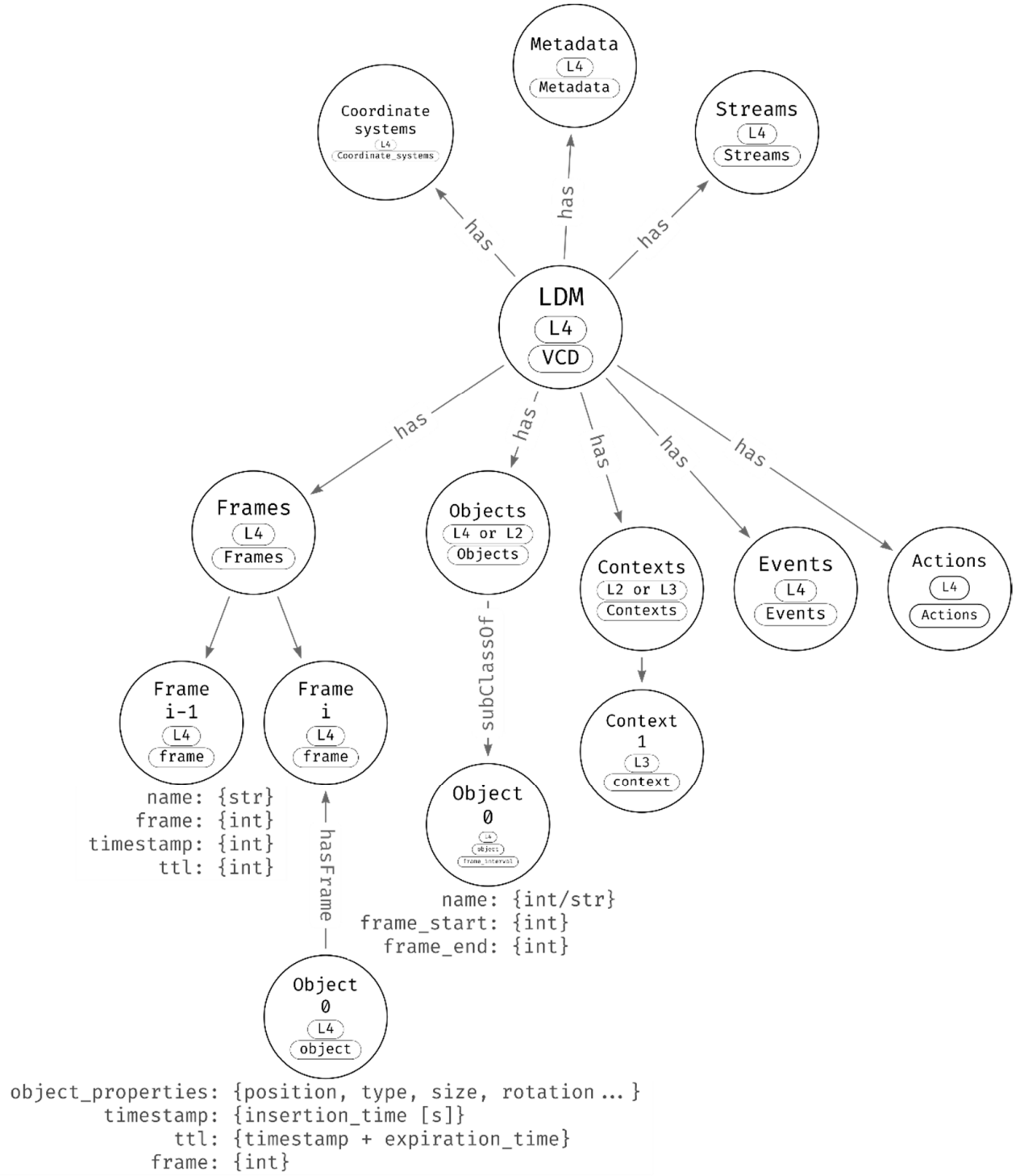

Figure 3. Dynamic layer (L4) data model graph representation.

4. iLDM Implementation

The characteristics of the LDM concept are specified in the ETSI TR 102 863 standard [8]. The proposed implementation has been defined taking into account the main features and functional requirements mentioned in the specification. Hence, the iLDM was constructed as a data store backboned with Neo4j, and the implementation of the system is performed as an application programming interface (API). The API is a Python library that exposes functions to input and output information from the database.

In this section, the emphasis is on the definition of the static layer (L1) and the dynamic layer (L4) of the iLDM. The mapping of the data is conducted considering that the information is ingested in the OpenLABEL format, which means that data with different dynamicity can be inserted in the database based on the proposed data model. Moreover, the traffic data ingestion is the most critical aspect of the implementation, as the updating frequency should be near real-time values to fulfil the requirements of several ITS applications. On the contrary, the road network is considered to be completely static and, consequently, the map definition process is disjoined.

4.1. Static Data Layer (L1)

For the static layer, OpenStreetMap (OSM [20]) data were used. OSM is an open-source collaborative project with geographic data of the entire world. One of the key features of OSM is that its database receives constant updates from users. This makes OSM a great source of data but also heavily dependent on having multiple active communities for each geographic zone. As there are still no reliable open-source HDMaps options available, this makes OSM the best open data source as other research works in the field have already stated. Other map representation formats, such as OpenDRIVE [21] or Lanelets [22], are suitable options for the LDM but lack the amount of open-source data that OSM offers.

OSM stores information in an XML formatted file using a graph structure, where the following conceptual elements of the physical world are used:

1. Nodes: define a point in space, and they can be used to define standalone point features;
2. Ways: define linear features and area boundaries using nodes;
3. Relations: used to explain how different elements are related to each other.

All of the above can have associated tag(s), and these tags can be used to add road information to the iLDM, e.g., one OSM way defining a road usually has tags related to its number of lanes, speed limit, road name, etc.

The static layer could be uploaded directly to the Neo4j database by reading an OSM file or could be continuously updated by using one of the many existing OSM APIs. For the proposed implementation, the construction of the road network layer was conducted using the OSM importer for Neo4j. This importer structures the OSM information according to its predefined data model, which distinguishes the following node types by setting different neo4j labels:

- "OSMWay": These nodes are the first node of each way represented in the database. Thus, they contain the attributes of the way, such as "way_osm_id", "name", "oneway" and "lanes", or road type under different property names such as "highway", "footway", "residential" or "cycleway". Moreover, they can also be associated with a building represented by a single point or a multipolygon.
- "OSMWayNode": Each way is divided into different nodes, the number of nodes in a way will depend on the length of it. These nodes do not provide any information, they are just for representing the way and relating the OSMNodes with it.
- "OSMNode": Each node of the way (OSMWayNode) has an associated OSMNode. These nodes provide information about the coordinates of the concrete point of the way. The properties related to these nodes are "node_osm_id", "latitude" and "longitude".
- "OSMTags": these nodes provide any extra information that the OSM file may have about the way (road or building) or concrete node (in this case, they never have coordinate information).

Figure 4 shows part of the static layer representation in Neo4j. In this case, the "OSMNode" and "OSMWay" nodes are represented with their corresponding OSM id, and they are related through "OSMWayNode" nodes. The "OSMTags" nodes display the "highway" property, which describes the road-type of that map segment. Nevertheless, these nodes contain all the properties defined in the original OSM file.

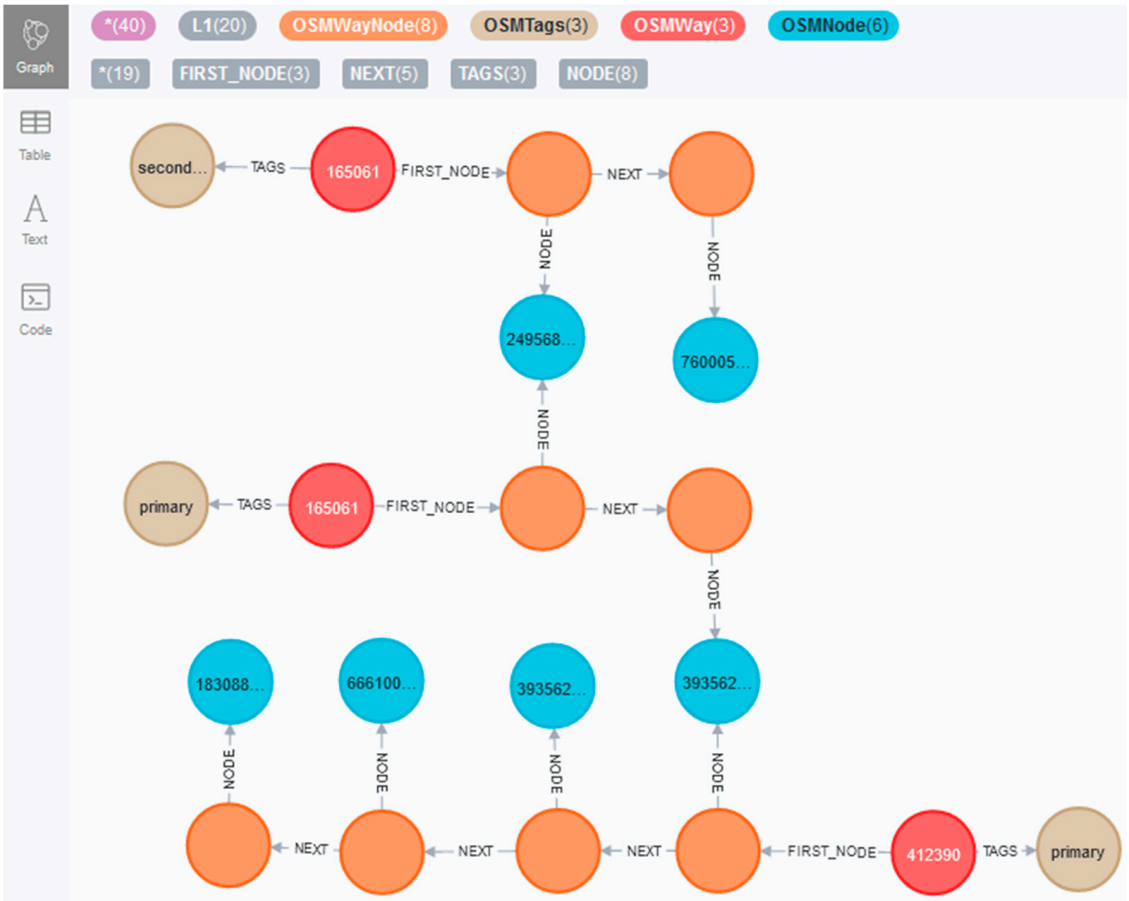

Figure 4. Static layer (L1) data model graph representation snippet in Neo4j. The colour code for the nodes is defined by the neo4j label colours on the top.

An additional post-processing stage was defined for the definition of the static layer. Once the database is initiated in Neo4j, the following steps are performed:

- Include the "L1" label to all the nodes related with the road network. This will define that the nodes under this label are used to represent the static layer of the LDM;
- Add a "position" attribute to the nodes where the latitude and longitude information exists. Hence, for each coordinate value pair, the location property is set as a point in the WGS-84 coordinate reference system (CRS) using the default spatial function included in Neo4j;
- Add a custom H3 [23] tag to the nodes with coordinates information under the "h3_address" property name. H3 is Uber's hexagonal hierarchical geospatial indexing system, allowing to index coordinates down to a square meter resolution, the hexagon size to resolution values can greatly vary from hexagon edge lengths of 1.107 km to 50 cm and areas of 4,250,546 km^2 to 0.9 m^2, respectively.

4.2. Dynamic Data Layer (L4)

The dynamic data layer must be fed and stored in the database from continuous stream data sources varying from on-board sensors to V2X communication. The database must be flexible enough to store unlimited dynamic object types obtained in real-world scenarios and must be categorised properly in the database. As OpenLABEL can store spatio-temporal annotations of objects with unlimited numeric, textual and binary fields, the proposal of the presented approach is to convert all the received data into the OpenLABEL format before feeding it to the database.

For V2X communications, converters from the CPM and CAM standard messages to OpenLABEL were developed. In the case of detections received from on-board sensors,

they can be directly outputted in the OpenLABEL format, allowing for more flexible and custom tags in the captured data.

After formatting the input data, the ingestion into Neo4j is conducted based on the data model presented in Section 3.2. As a result, each element type is represented as a node with its corresponding attributes and will lay under the "L4" label and another one will be assigned according to the element type. Moreover, the "object" nodes are related to their corresponding "frame" nodes, so that each instant of the dynamic elements are completely represented according to the semantic information given in OpenLABEL. Furthermore, with the aim of covering the temporal aspect of the data, a couple of extra properties are included for each object:

- "Frame": following the OpenLABEL terminology, each "object" node will include the corresponding "frame" number as a property;
- "Timestamp": collects the time instant when the element was created in the Neo4j database;
- "TTL": all the dynamic data of the database has set a time to live that considers the "timestamp" and a configured "expiration time".

For the spatial nature of the information, as done in the post-processing of the static layer, "position" and "h3_address" attributes are included according to the same conditions mentioned in Section 4.2. These properties are required to perform the map-matching, which means relating each instant of a vehicle with the nearest map node. The presented results exclude this operation, but the graph is constructed considering the prerequisites of this process. Therefore, the map-matching can be included in the ingestion process or post-processed to fulfil the needs of concrete applications. In case the procedure is performed during the insertion of the traffic data, it is modelled in the graph according to the following triple: "Object"—":hasLocation" → "OSMNode".

4.3. H3 Indexing

To enhance the functionality of the iLDM and avoid inefficient comparison between dynamic object positions and static nodes, both OSM nodes and L4 objects are processed before or after adding them to the database with a custom H3 tag. H3 is Uber's hexagonal hierarchical geospatial indexing system, allowing to index coordinates down to a square meter resolution.

By doing this, H3 can be used to obtain the IDs of the hexagon where a certain object is and also of the outer hexagon rings to it as seen in Figure 5. This enables fast and precise querying from the iLDM API, e.g., obtaining all vehicles in a given hexagon and detecting traffic jams in a hexagon ring.

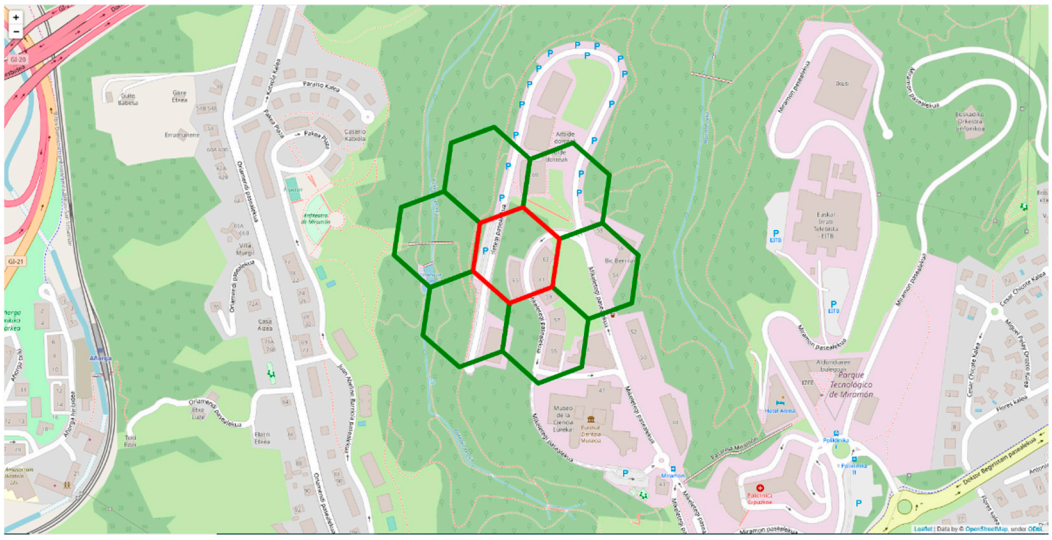

Figure 5. H3 hexagons of a given vehicle position and its outer ring for resolution = 10.

4.4. Interoperability

As the proposed LDM implementation uses OpenLABEL as a common format for every input into the database, the source of the data received by the iLDM could be anything ranging from real data, data sets or V2X messages.

Tested Data Sources

The iLDM was tested with three different types of data inputs. First, recordings from the KITTI data set [24] were used to feed the iLDM with 3D annotated objects, as they may come from a perception system. KITTI is a well-known data set in the automotive field, produced in 2012 and containing data for many tasks such as stereo, optical flow, visual odometry and 2D/3D object detection. Using 3D annotations and ego motion data from the GPS/IMU, data from both the static and dynamic layers can be inserted and visualised to test the loading process. The data were pre-processed to convert the KITTI scene into the OpenLABEL format. The usage of KITTI can allow us to test ADAS functions (Section 5.3.) against the ground truth annotated data, but it lacks customisation obtainable from simulation environments to perform more exhaustive testing of the iLDM.

Second, we selected SUMO [25] as a high-level traffic simulation tool that gives us macroscopic data about the position of vehicles in different traffic situations, something that is difficult and costly to obtain from real recordings in the field. With SUMO data, we can simulate the ingestion of a wide area of vehicle positions into a single LDM database as a way to showcase that our iLDM implementation is suitable for its utilisation in RSUs and to aggregate information from multiple sources in large areas. Thanks to this, we could generate unlimited test scenarios with a custom number of vehicles over given OSM maps. Synthetic data from SUMO traffic simulations were converted to the OpenLABEL format and used in the tests performed in Section 5.1. Processing geolocated data from SUMO simulations in combination with OSM maps, enabled 3D visualisation of the generated simulations as Figure 6 depicts.

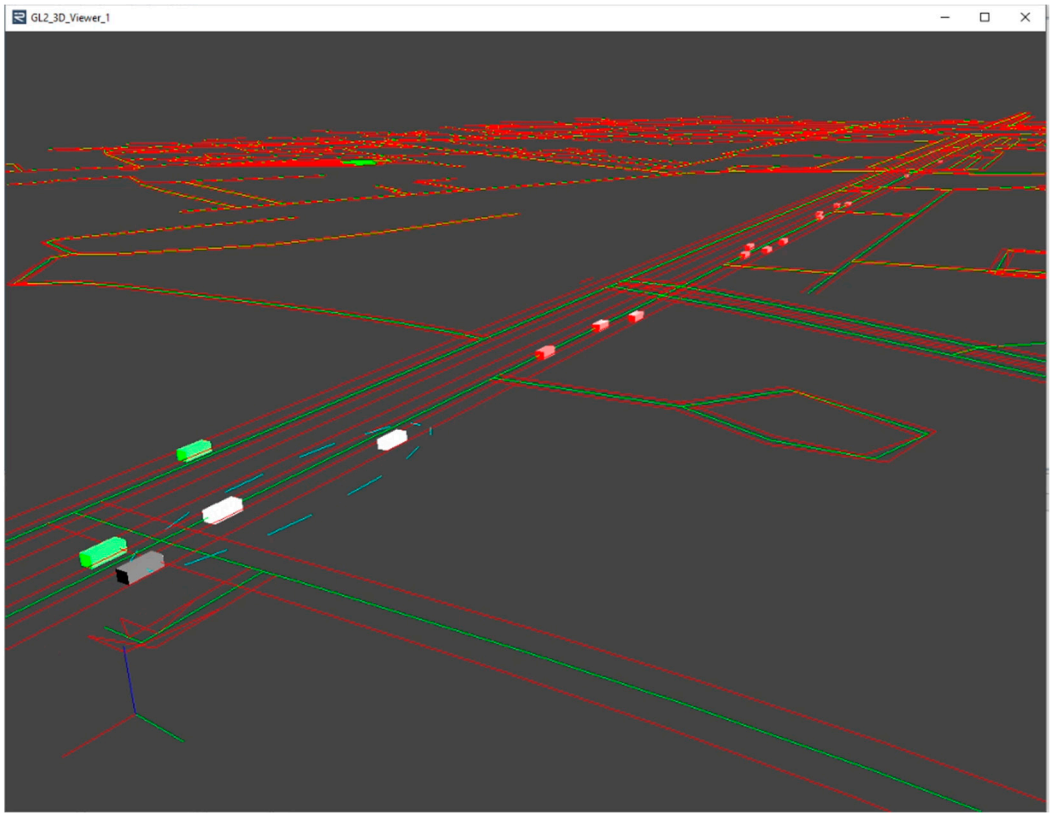

Figure 6. 3D visualisation of an SUMO simulation with its corresponding road data from OSM maps.

Third, we performed a test drive with our equipped vehicle CarLOTA (Car Learn on Road Think Autonomous), a third generation Toyota Prius as can be seen in Figure 7. CarLOTA is equipped with a variety of automotive sensors of which we used the following:

- Velodyne HDL-32 LiDAR: small, lightweight and ruggedly built, it features up to 32 lasers across a 40° field of view;
- 4× Outside cameras: automotive 180° fisheye cameras for covering 360° of the vehicle;
- XNAV 550 DGPS: precise localisation system that can be enhanced using broadcasted DGPS corrections;
- Embedded in-vehicle HW platform: car computing platform that enables processing of sensor data and applications.

Figure 7. Pictures of our test car, CarLOTA, including a picture of all the equipment installed in the trunk of the car.

The scope of the test drive was to ensure interoperability across the presented data sources, proving that detections obtained from real scenarios can be ingested into the iLDM as well. We recorded all data obtained from the different sensors for offline replaying and testing purposes. In addition, thanks to our LiDAR sensor, we executed a real-time 3D LiDAR detection module, capturing and storing all dynamic object detections surrounding our ego-vehicle in the OpenLABEL format. Our test drive consisted of a 4.15 km ride through the roads next to our facilities in San Sebastian, Spain; the route can be clearly seen in Figure 8 thanks to the JOSM [26] tool.

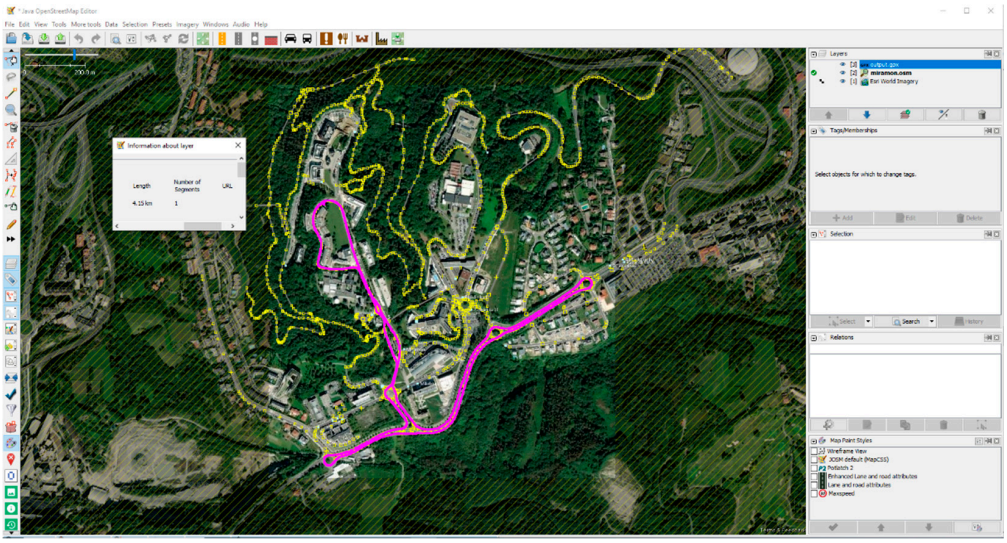

Figure 8. Satellite image visualisation using JOSM including a GPS trace of the route performed during our test drive and the associated OSM map file.

Finally, converters from CPM and CAM to our OpenLABEL data format were developed, allowing messages from V2X communications to be inserted into the database.

4.5. Real-Time Implementation

The RTMaps [27] framework was used to test the performance of our LDM implementation in a real-time environment. RTMaps stands for Real-Time Multisensor Applications and is a component-based development and execution software tool specialising in ADAS functions. RTMaps allows to record data from on-board vehicle sensors with timestamps associated to each data stream captured by the sensor, and by using these generated recordings, the same real-life scenario can be replayed offline an unlimited number of times in its framework. RTMaps also allows us to read data from data sets or simulators at different sample rates, making it perfect to test all kinds of data sources. The process flow for populating the database with dynamic data from an RTMaps diagram is summarised in Figure 9.

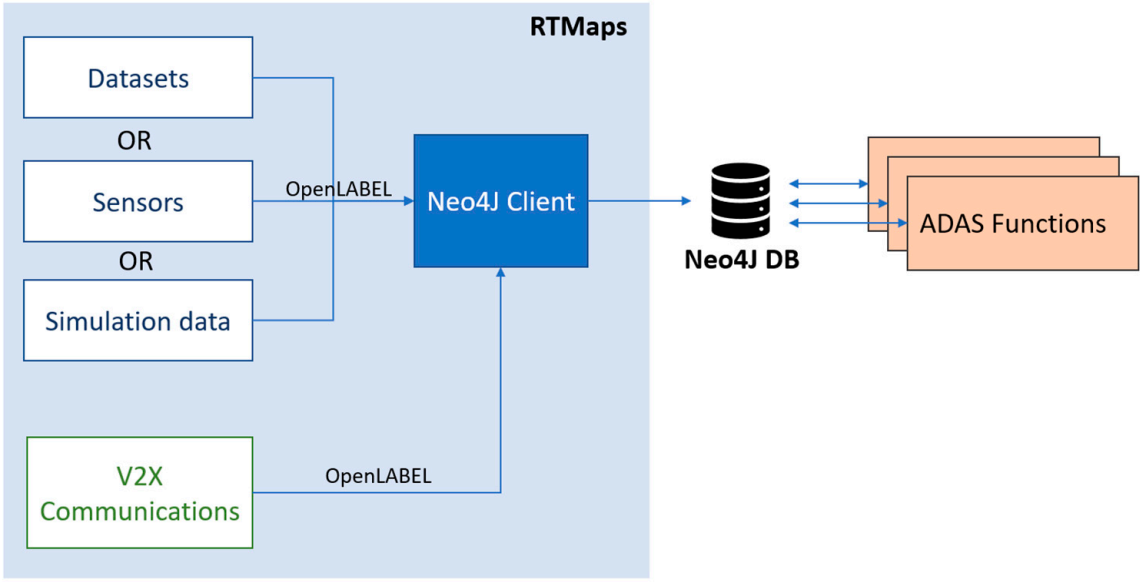

Figure 9. Dynamic data feeding process to the iLDM from RTMaps.

For a real-time performance, we took into consideration that the maximum rate that V2X messages, such as CAM [13] or CPM [14], support in their standards is 10 Hz. To fulfil the real-time requirement of the LDM, we expected that the iLDM would be able to insert and query data faster than 100 ms. Our implementation was carried out using Python components with Neo4j's API, without the need to focus on optimizing the performance of the proposed diagram.

5. Analysis/Experiments

All the tests described in this section were performed using the following setup (see Table 1).

Table 1. Computer specifications.

CPU	Intel Core I7-10750H @2.6 GHz 12 Cores
DISK	512 GB SSD
RAM	16 GB DDR4
OS	Windows 10

5.1. iLDM Performance Analysis

The analysis was divided into two different subsections. In the first one, using a test set of OpenLABEL files, insertion time into the iLDM of the different number of nodes with its corresponding relations was measured. In the second performance test, the iLDM database was populated with data from the test set, and the response time of different kinds of queries requesting information to the iLDM was measured. Finally, a VDS ADAS function was developed to showcase example applications that can be performed by using the presented iLDM implementation.

5.1.1. Adding Objects

In order to test the performance of node insertion into Neo4j, a test set of OpenLABEL files with data from SUMO traffic simulations was generated.

The performance of the data insertion was tested with different numbers of vehicles: 5, 10, 25, 50, 100, 200, 300, 400 and 500. Per each number of vehicles, 100 OpenLABEL files populated with different SUMO simulations were generated. Thereafter, the time required for the ingestion of the OpenLABEL data into Neo4j was measured for the test computer, and the results are depicted in Figure 10.

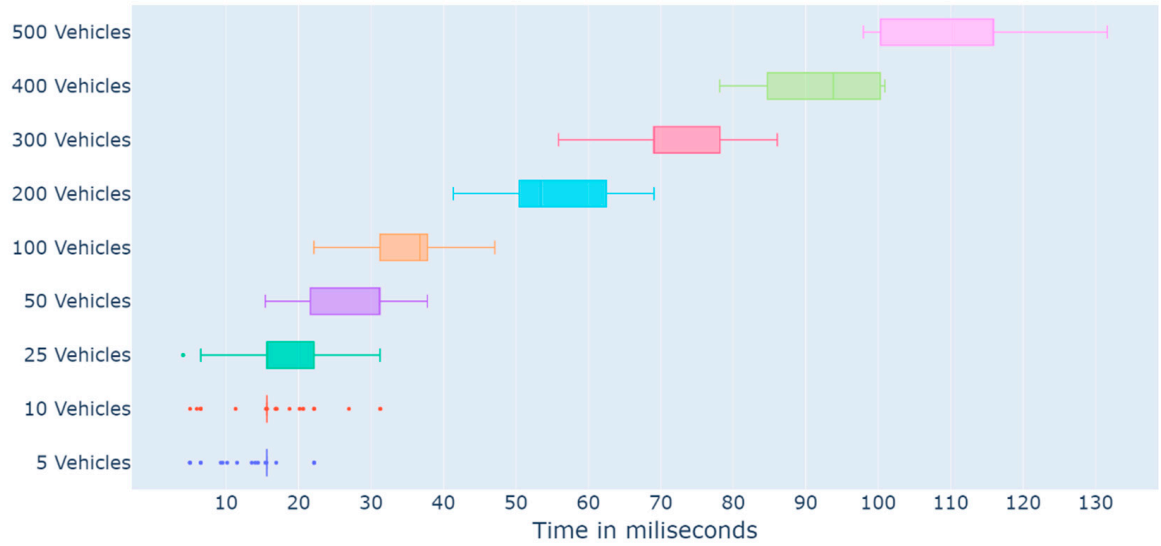

Figure 10. Box plot with insertion time into Neo4j in milliseconds of different number of vehicles.

As can be seen in the previous figure, when inserting up to 300 vehicles the information is available in the database in less than 100 ms; even for 400 vehicles, most of the time this condition will be met. Inserting up to 300 or 400 vehicles is far more than necessary for real-life scenarios, as in regular driving situations it is hard to be surrounded by even 100 vehicles or more, enabling real-time data insertion into the database.

5.1.2. Querying Time

Once the data insertion time has been analysed, it is also crucial for the iLDM to enable fast querying of the information. The previously ingested OpenLABEL test data were used to evaluate the querying performance based on the following cases:

- **Query 1**: query all L4 objects in a certain frame;
- **Query 2**: query all L4 objects in a certain frame in a 50 m radius of the ego-vehicle;
- **Query 3**: query a given vehicle in the last 10 frames;
- **Query 4**: query all L1 OSM nodes in the same h3 address as the ego-vehicle;
- **Query 5**: query all L4 objects in the same h3 address as the ego-vehicle.

The response time results for each query with different numbers of vehicles can be seen in Figure 11:

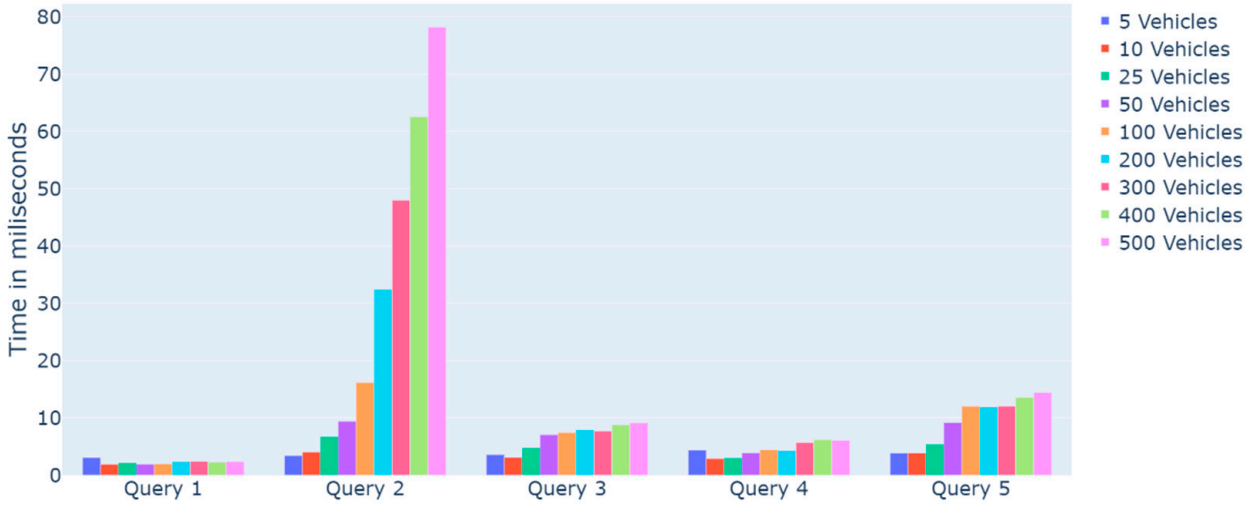

Figure 11. Bar plot with querying time in milliseconds of the different numbers of vehicles for different queries.

All the queries except for Query 2 offered performance times of less than 15 ms. The problem with Query 2 is that the iLDM compared each vehicle location against the ego-vehicle and, even in this case, we can see how the growth in time was lineal. As the proposed iLDM implementation pre-processes all geospatial data with H3 indexes, faster geospatial queries can be performed as can be seen in the results obtained for Query 5. Both Query 2 and Query 5 generated similar outputs, with the difference being that the first one had to compare each position against the ego-vehicle, while the second one used H3 indexing to improve the geospatial querying performance by almost six times in the case of 500 vehicles and did not show a lineal growth. All the other queries did not show large performance changes when the number of vehicles in the database grew.

5.2. Comparison with Existing Developments

Once we analysed the performance of the iLDM, we could elaborate a functional comparison with other existing developments (Table 2). As some of the few existing implementations do not provide numerical data for insertion and querying time, the best way to compare them is based on their key features. As previously stated, we considered that an LDM fulfils real-time performance if it can handle both insertion and querying times of less than 100 ms. We selected the three most relevant LDM implementations in this comparison. The experiments carried out in [2] only evaluated the LDM for up to 20 vehicles with an update processing time of 36 ms against the 19 ms that the iLDM took for 25 vehicles, but it did not provide the data for the more demanding scenarios. We considered their approach to working in real time for at least that number of vehicles. In the case of [16], no performance information was reported and, thus, it is reflected in Table 2 as "Not Covered" (N/C). Finally, in [3], the authors tested their LDM querying performance time with up to 5000 vehicles but without providing results for data insertion times. The authors stated that for up to 500 vehicles, their prototype could manage evaluation within 1.5 s, far more than the time obtained in the iLDM for the same number of vehicles and for what we consider real-time performance. Nevertheless, the querying performance of this work cannot be compared fairly with the iLDM due to the semantical complexity of the queries performed in their work.

Table 2. Comparison of the iLDM against similar developments.

Work	Standardised Data Format	Database	Semantic Support	Real Time	Compatible with OSM
Implementation and Evaluation of Local Dynamic Map in Safety Driving Systems	No	PostgreSQL	No	Yes *	Yes
Towards a Semantically Enriched Local Dynamic Map	No	PostgreSQL/PipelineDB	Yes	No **	Yes
R-LDM	No	Neo4j	Yes	N/C	Yes
iLDM	OpenLABEL	Neo4j	Yes	Yes	Yes

* Up to 20 vehicles; ** for the presented queries.

5.3. Use case implementation: Vehicle Discovery Service

With the aim of putting the proposed implementation into practice and testing the system in a near-real case using the KITTI data set parsed to the OpenLABEL format, a simple ADAS function named VDS was defined. The application is based on the same VDS used in [27] to retrieve vehicles in front of an ego-vehicle. Using the geospatial position of an ego-vehicle and its perception of other dynamic objects stored into the iLDM, a query was executed against the database to obtain all the stored vehicles that are in a given radius of the ego-vehicle. After this, an ellipsoidal spatial ROI is defined, as this application is more suited to having a lengthened shape. In addition, calculating if a certain point is inside an ellipse requires low computation time, for a vehicle positioned at (x, y) can be determined if it is inside of the ROI if the following equation is satisfied:

$$\frac{((x-h)\cos(\alpha)+(y-k)\sin(\alpha))^2}{m^2}+\frac{((x-h)\sin(\alpha)-(y-k)\cos(\alpha))^2}{m^2}\leq 1 \quad (1)$$

where a, b and h, k are the ellipse semi-axis and shifts in the x- and y-directions, respectively, and α is the heading of the ego-vehicle from the x-axis. Furthermore, m and n are half of the ellipse width and half of the ellipse height. Even though in [27], it has been defined as 50 m for the height and 14 m for the width, calculated as four times the standard lane width of a road, a simple GUI with sliders was created to modify this in real time. In addition, as the ellipse is oriented towards the heading of the ego-vehicle, h and k were calculated as follows.

$$h=\cos(\propto)m+x_{ego} \quad (2)$$

$$k=\sin(\propto)m+y_{ego} \quad (3)$$

In addition to these equations, a heading threshold between the ego-vehicle and the other detected vehicles was added to ensure that both vehicles were driving in the same direction.

For this test, a sequence of the KITTI data set inserted into the iLDM was used along with RTMaps as the real-time processing framework. In Figure 12, we can visualise three different windows including a 3D visualisation of objects retrieved from the iLDM in a given frame, a set of sliders for real-time adjustment of the VDS input thresholds and an image viewer of the corresponding KITTI camera image.

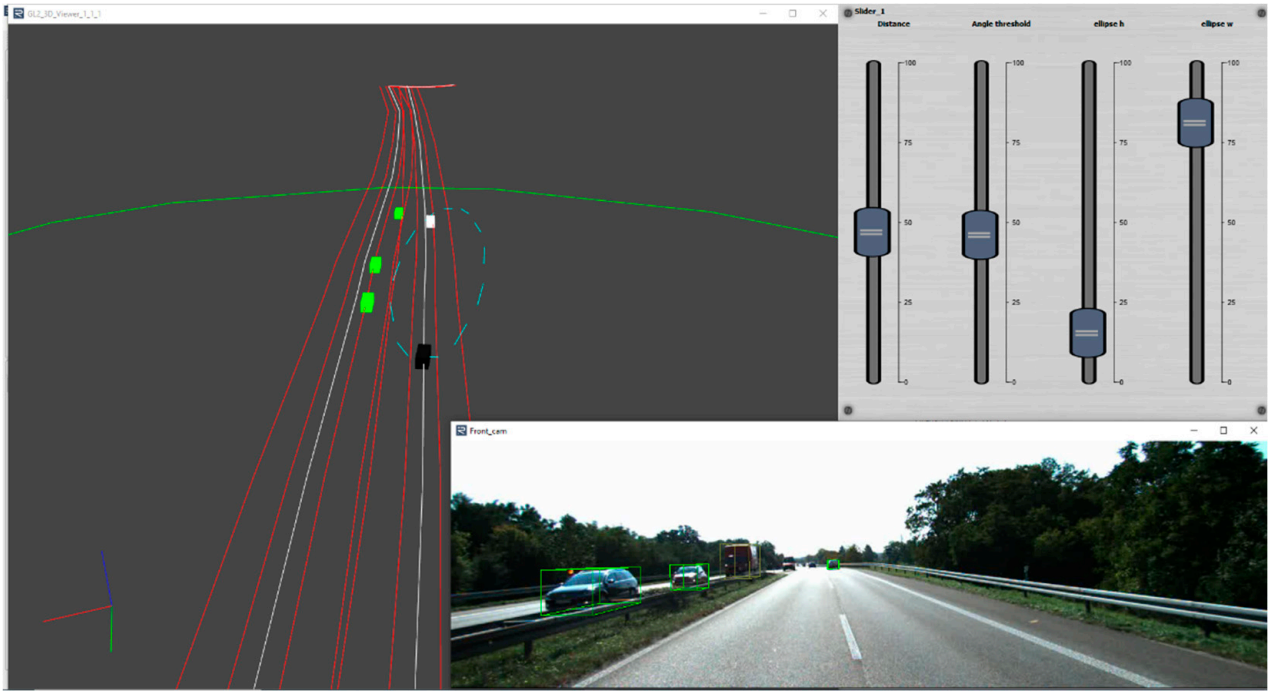

Figure 12. Visualisation in RTMaps of a given frame from a KITTI sequence while using the VDS function querying the iLDM. In the top right corner, the GUI with sliders to modify the different VDS input parameters can be seen.

6. Conclusions

A graph-based LDM using Neo4j and OpenLABEL as a common data information source was introduced in this work. The use of OpenLABEL makes the proposed LDM approach interoperable with different data sources such as sensor data detections, simulation data or V2X messages. We showcased how our approach is suitable for real-time performance by analysing both insertion and querying time to the database. Other similar graph-based approaches do not provide numerical results of the performance of their LDM implementations; however, we provide a comparison of their key features with respect to the iLDM. The novel use of OpenLABEL allows data ingestion from heterogeneous sources, making it a great tool for the constant changing pace of the industry.

Future research work may include the extension and improvement of the data model to adopt other relevant standards, particularly OpenDRIVE, to cover the static layer of the LDM structure and utilisation of multiple databases for optimised performance (real-time volatile data and persistent data logging). Evaluation of the iLDM directly from a test drive will be addressed in the future instead of logging sensor and detection data for its offline playback.

Author Contributions: Investigation, M.G., I.U., M.N.; software, M.G., I.U., J.G.d.M., and M.N.; writing—original draft preparation, M.G., I.U., and M.N.; data preparation M.G. and I.U.; supervision, M.N. and O.O.; project administration, M.N. and O.O.; funding acquisition, M.N. and O.O. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the GSA Fundamental Elements programme under grant agreement No. GSA/GRANT/03/2018 (project: ACCURATE).

Data Availability Statement: Video sample of a VDS test using KITTI in the RTMaps framework: https://drive.google.com/file/d/1NXkcjYymgOWWdB_m8S7JMfsS1HM1T_pY/view?usp=sharing (accessed on 6 July 2021).

Acknowledgments: This project received funding from the GSA Fundamental Elements programme under grant agreement No. GSA/GRANT/03/2018 (project: ACCURATE).

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

LDM	Local Dynamic Map
iLDM	Interoperable Local Dynamic Map
VDS	Vehicle Discovery Service
ADAS	Advanced Driver-Assistance Systems
C-ITS	Cooperative Intelligent Transport Systems
RSU	Roadside Unit
AD	Automated Driving
V2X	Vehicle-to-Everything
RDBMS	Relational Database Management System
AGO	Automotive Global Ontology
ETSI	European Telecommunications Standards Institute
CAM	Cooperative Awareness Message
CPM	Cooperative Perception Message
R-LDM	Relational-LDM
CRUD	Create/Read/Update/Delete
TTL	Time to Live
API	Application Programming Interface
OSM	OpenStreetMap
CRS	Coordinate Reference System
CarLOTA	Car Learn on Road Think Autonomous
N/C	Not Covered

References

1. Safespot. Safespot Integrated Project-IST-4-026963-IP Annex SP7—SCORE—SAFESPOT Core Architecture LDM API and Usage Reference. Published online 2006. pp. 1–56. Available online: http://www.safespot-eu.org/documents/SF_D7.3.1_Annex2_LDM_API_and_Usage_Reference_v0.7.pdf (accessed on 8 June 2021).
2. Shimada, H.; Yamaguchi, A.; Takada, H.; Sato, K. Implementation and Evaluation of Local Dynamic Map in Safety Driving Systems. *J. Transp. Technol.* **2015**, *5*, 102. [CrossRef]
3. Eiter, T.; Füreder, H.; Kasslatter, F.; Parreira, J.X.; Schneider, P. Towards a Semantically Enriched Local Dynamic Map. *Int. J. Intell. Transp. Syst. Res.* **2019**, *17*, 32–48. [CrossRef]
4. ASAM. ASAM OpenLABEL V1.0.0. Available online: https://www.asam.net/project-detail/asam-OpenLABEL-v100/ (accessed on 6 July 2021).
5. Urbieta, I.; Nieto, M.; García, M.; Otaegui, O. Design and Implementation of an Ontology for Semantic Labeling and Testing: Automotive Global Ontology (AGO). *Appl. Sci.* **2021**, *11*, 7782. [CrossRef]
6. Andreone, L.; Brignolo, R.; Damiani, S.; Sommariva, F.; Vivo, G.; Marco, S. D8.1.1—SAFESPOT Final Report. SAFESPOT Final Rep—Public Version. Available online: http://www.transport-research.info/sites/default/files/project/documents/20130329_130257_17414_D8.1.1_Final_Report__Public_v1.0.pdf (accessed on 6 July 2021).
7. ETSI TR 102 863 V1.1.1. Intelligent Transport Systems (ITS). Vehicular Communications. Basic Set of Applications. Local Dynamic Map (LDM); Rationale for and guidance on standardization LDM Rationale and Guidance - ETSI T R 102 863. 2011. Available online: https://www.etsi.org/deliver/etsi_tr/102800_102899/102863/01.01.01_60/tr_102863v010101p.pdf (accessed on 9 August 2021).
8. ETSI EN 302 895 V1.1.1. Intelligent Transport Systems (ITS); Vehicular Communications; Basic Set of Applications; Local Dynamic Map (LDM) Basic Set of Applications-ETSI EN 302 895. 2014. Available online: https://www.etsi.org/deliver/etsi_en/302800_302899/302895/01.01.01_60/en_302895v010101p.pdf (accessed on 16 June 2021).
9. ISO/TC 204 Systems I Transport. ISO/TS 17931:2013 Intelligent Transport Systems—Extension of Map Database Specifications for Local Dynamic Map for Applications of Cooperative ITS. 2013. Available online: https://www.iso.org/standard/62110.html (accessed on 16 July 2021).
10. PEGASUS. Scenario Description and Knowledge-Based Scenario Generation. Online. Published online 2018. Available online: https://www.pegasusprojekt.de/files/tmpl/Pegasus-Abschlussveranstaltung/05_Scenario_Description_and_Knowledge-Based_Scenario_Generation.pdf (accessed on 25 July 2021).
11. PEGASUS Project Office. Pegasus Research Project. Pegasus. Published online 2019. Available online: https://www.pegasusprojekt.de/en/ (accessed on 8 August 2021).
12. Scholtes, M.; Westhofen, L.; Turner, L.R.; Lotto, K.; Schuldes, M.; Weber, H.; Wagener, N.; Neurohr, C.; Bollmann, M.H.; Körtke, F.; et al. 6-Layer Model for a Structured Description and Categorization of Urban Traffic and Environment. *IEEE Access* **2021**, *9*, 59131–59147. [CrossRef]

13. ETSI EN 302637-2:2019. Intelligent Transport Systems (ITS); Vehicular Communications; Basic Set of Applications; Part 2: Specification of Cooperative Awareness Basic Service. 2019. Available online: https://www.etsi.org/deliver/etsi_en/302600_302699/30263702/01.04.01_60/en_30263702v010401p.pdf (accessed on 7 July 2021).
14. ETSI T R 103 562:2019. Intelligent Transport Systems (ITS); Vehicular Communications; Basic Set of Applications; Analysis of the Collective Perception Service (CPS). 2019. (accessed on 17 June 2021).
15. PipelineDB. Available online: https://github.com/pipelinedb/pipelinedb (accessed on 6 July 2021).
16. Eggert, J.; Salazar, D.A.; Puphal, T.; Flade, B. Driving Situation Analysis with Relational Local Dynamic Maps (R-LDM). International Sympposium on Future Active Safety Technology towards Zero-Traffic-Accidents (FAST-zero). 2017. Available online: https://www.researchgate.net/publication/334730164_Driving_Situation_Analysis_with_Relational_Local_Dynamic_Maps (accessed on 6 July 2021).
17. Netten, B.; Kester, L.J.H.M.; Wedemeijer, H.; Passchier, I.; Driessen, B. DynaMap: A Dynamic Map for road side ITS stations. In Proceedings of the 20th ITS World Congress Tokyo 2013, Tokyo, Japan, 14–18 October 2013; Intelligent Transportation Society of America: Washington, DC, USA, 2013.
18. Maiouak, M.; Taleb, T. Dynamic Maps for Automated Driving and UAV Geofencing. *IEEE Wirel Commun.* **2019**, *26*, 54–59. [CrossRef]
19. Neo4j. Neo4j Graph Data Platform. Available online: https://neo4j.com/ (accessed on 6 July 2021).
20. ASAM. ASAM OpenDRIVE. Available online: https://www.asam.net/standards/detail/opendrive/ (accessed on 6 July 2021).
21. Poggenhans, F.; Pauls, J.H.; Janosovits, J.; Orf, S.; Naumann, M.; Kuhnt, F.; Mayr, M. Lanelet2: A high-definition map framework for the future of automated driving. In Proceedings of the 2018 21st International Conference on Intelligent Transportation Systems (ITSC), Maui, HI, USA, 4–7 November 2018. [CrossRef]
22. Uber Technologies. Hexagonal hierarchical geospatial indexing system (H3). Available online: https://h3geo.org/ (accessed on 6 July 2021).
23. Geiger, A.; Lenz, P.; Stiller, C.; Urtasun, R. Vision meets Robotics: The KITTI Dataset. *Int. J. Robot. Res.* **2013**. published online. [CrossRef]
24. Lopez, P.A.; Behrisch, M.; Bieker-Walz, L.; Erdmann, J.; Flötteröd, Y.P.; Hilbrich, R.; Lücken, L.; Rummel, J.; Wagner, P.; Wießner, E. Microscopic Traffic Simulation using SUMO. In Proceedings of the 2018 21st International Conference on Intelligent Transportation Systems (ITSC), Maui, HI, USA, 4–7 November 2018; Available online: https://elib.dlr.de/124092/ (accessed on 26 August 2021).
25. Edgewall Software. JOSM. Available online: https://josm.openstreetmap.de/ (accessed on 6 July 2021).
26. Intempora. RTMaps. Available online: https://intempora.com/products/rtmaps/ (accessed on 6 July 2021).
27. Velez, G.; Perez, J.; Martin, A. 5G MEC-enabled vehicle discovery service for streaming-based CAM applications. *Multimed. Tools Appl.* **2021**. published online. [CrossRef]