University of Tartu
Institute of Computer Science

Valentina Sagris

Semi-automatic generation of Lanelet2 maps for autonomous driving

Masters' Thesis (15 ECTS)

Supervisors: Edgar Sepp, MSc
Tambet Matiisen, Msc

Tartu 2023

Semi-automatic generation of Lanelet2 maps for autonomous driving

Abstract:

High-definition (HD) maps are essential in autonomous driving (AD) by providing data about the static driving environment for vehicle localisation, route planning, perception and manoeuvre decision-making. To conduct research in autonomous driving, the Autonomous Driving Lab (ADL) of the University of Tartu needs HD maps. This thesis presents the database structure and the effective workflow for the semi-automatic generation of HD map elements. In particular, it tests the capability of the PostGIS spatial database in helping to minimise manual work. The main goals of this thesis can be summarised into the following topics. The ADL's previous mapping efforts were overviewed, and the unified database design for storing spatial data required in various HD map formats was proposed. Then previously collected data was used to develop an algorithm for the semi-automatic generation of missing elements – spatial features needed to construct proper lanelets, the fundamental primitives of maps in Lanelet2 format. Further on, the Lanelet2 requirements for the shared bounds and automated finding of semantic relationships between primitives that constitute a lanelet were addressed. The critical step here was producing spatial elements, which we call 'relations' that establish the spatial relationships between a centreline and its bounds in the database. Finally, the data converter for HD maps in Lanelet2 format was proposed. It transforms spatial primitives of PostGIS into primitives of Lanelet2, which are nodes, ways and relations. Due to PostgreSQL's capability to store the XML datatype, elements for each primitive were created in the database. Further on, the database was accessed from the python script, where XML root was made, and the following elements were loaded from the database into the root to create a proper Lanelet2 file.

The workflow was tested on two sites: Lai-Jakobi-Kroonuaia and Narva-Roosi-Puiestee. It helped to assess the algorithm's robustness in various road network configurations and improve its performance. It turned out that the algorithm performance is very good with standard road structures where plain street stretches meet in T-shape or X-shape intersections. Also, the algorithm is capable of performing well in areas of complex street intersections, but some human assistance is needed. The statistics obtained for comparison of the proposed solution with a fully manual process of map elements digitisation demonstrated a considerable reduction in time and human efforts in HD map creation. Due to the PostGIS functionality, the data geoprocessing was impressively fast. Depending on road structure complexity, the amount of effort needed per one kilometre of lanes can drop by 45% to 65%.

Keywords: High-definition maps, autonomous driving, Lanelet2, PostGIS

CERCS: P170 (Computer science, numerical analysis, systems, control)

Osaliselt automatiseeritud isejuhtivatele sõidukitele mõeldud Lanelet2 kaartide loomine

Lühikokkuvõte:

Täppiskaardid (*high-definition maps*) on isejuhtivatele sõidukitele (*autonomous driving*) oluline andmeallikas, mis on abiks sõidukite lokaliseerimisel, marsruudi planeerimisel, ümbruse tajumisel ja manööverdamisotsuste tegemisel. Antud valdkonnas uurimistöö läbiviimiseks vajab Tartu ülikooli isejuhtivate sõidukite labor (*Autonomous Driving Lab, ADL*) täppiskaarte. Käesolev uurimistöö tegeleb täppiskaardi andmebaasi struktuuriga ja täppiskaardi elementide poolautomaatse genereerimisega. Eelkõige testiti töö käigus PostGIS ruumiandmebaasi võimalusi andmete automaatseks töötlemiseks, mis vähendaks kaardistamiseks vajaliku käsitsitööd.

Esiteks, analüüsiti ADL-i varasemaid läbiviidud kaardistusi ja koostati andmebaasi struktuur, mis võimaldab säilitada vajalikke ruumiandmeid erinevate täppiskaardiformaatide jaoks ühes andmebaasis.

Teiseks, kasutati varem kogutud andmeid niinimetatud "puuduvate" elementide poolautomaatse algoritmi loomisel. "Puuduvate" elementide all mõistetakse siin ruumilisi nähtusi, mis on vajalikud sõidulõikude (*lanelet*) ehk Lanelet2 kaardiformaadi põhiliste primitiivide loomiseks.

Kolmandaks, käsitleti Lanelet2 nõudeid sõidulõikudede jagatud piiride ja semantiliste suhete kohta ning pakuti välja automatiseeritud protseduur seoste leidmiseks sõidulõiku moodustavate elementide vahel. Kriitiliseks sammuks osutus seoseelementide (*relation*) genereerimine; need elemendid loovad ruumiseosed sõidulõigu keskjoone ja selle piiride vahel andmebaasis.

Neljandaks, loodi andmete konverter Lanelet2 formaadis täppiskaartide genereerimiseks. Konverter teisendab PostGIS-i ruumilisi primitiive Lanelet2 primitiivideks, milleks on tipud, teed ja seosed. Tänu PostgreSQL-i võimalusele hoida andmebaasis XML-andmetüüpe, oli võimalik luua igale Lanelet2 primitiivile vastav XML-element. Loodud elementidest loodi korrektne Lanelet2 fail Pythoni skripti abil.

Töövoogu testiti kahel alal Tartus: Lai-Jakobi-Kroonuaia ja Narva-Roosi-Puiestee. See aitas hinnata algoritmi jõudlust erineva struktuuriga tänavavõrgu korral ja parandada selle efektiivsust. Selgus, et algoritmi jõudlus on väga hea standardse tänavatevõrgu korral, kus tänavalõigud saavad kokku T- või X-kujulistel ristmikel. Algoritm on võimeline hästi toime tulema ka keeruliste ristmikega aladel, kuid vajab mõningast inimese poolset abi ja kontrollimist. Pakutud lahendusele kulunud aega võrreldi ajaga, mis kuluks kõikide kaardielementide käsitsi digiteerimisele ning võrdlus näitas olulist tööaja vähenemist.

PostGIS-i funktsionaalsus võimaldas muljetavaldavat kiiret geoandmete töötlust. Sõltuvalt tänavatevõrgu keerukusest võib ühe kilomeetri sõiduradade kohta vajaminev tööaeg langeda 45% kuni 65%.

Võtmesõnad: Täppiskaardid, isejuhtivad sõidukid, Lanelet2, PostGIS

CERCS: P170 Arvutiteadus, arvutusmeetodid, süsteemid, juhtimine (automaatjuhtimisteooria)

Contents

1 Introduction 6
2 Background 7
2.1 Autonomous Driving 7
2.2 High-definition maps 8
2.3 High-Definition map formats and standards 10
2.4 Mapping methods 16
3 Methodology 19
3.1 Data sources and data collection 19
3.1.1 Aerial imagery 19
3.1.2 Elevation data 20
3.1.3 AD vehicle trajectories 20
3.1.4 Data mapping campaigns in the ADL 20
3.2 PostGIS Database 21
3.3 Semi-automatic generation of HD map elements and converting to Lanelet2 format 23
4 Results 26
4.1 Database 26
4.2 Algorithm development for generation of HD map missing elements 27
4.2.1 Data extraction and pre-processing. 27
4.2.2 Find borders for straight lanes when exists 27
4.2.3 Generation of bounds for lane segments where one or both bounds are missing 28
4.2.4 Creation of lanelets topology. 29
4.2.5 Generate missing boundaries for turning lanes. 30
4.3 Converter for Lanelet2 HD map 31
4.3.1 Preparing the lanelet primitives 31
4.3.2 Creation of XML elements 31
4.3.3 Creation of XML document 32
5 Testing 33
6 Conclusions 36
Acknowledgement 37
References 38
Annexes 40
Annex A Algorithmfor generation of HD map missing elements 40
Annex B Converter for Lanelet2 HD map 42
License 43

List of acronyms and abbreviations

AD	Autonomous Driving
ADAS	Advanced Driver Assistance Systems
ADL	Autonomous Driving Lab
AOI	Area of Interest
ASAM	Association for Standardization of Automation and Measuring Systems
AV	Autonomous Vehicle
AWF	Autoware Foundation
CRS	Coordinate Reference System
DE	Driving Environment
DEM	Digital Elevation Models
FOV	Field of View
GNSS-IMU	Global Navigation Satellite Systems / Inertial Measurement Unit
GIS	Geographic Information System (also geoinformatics as a synonym)
HD	High-Definition (map)
HAD	Highly Automated Driving
LiDAR	Light Detection and Ranging
MMS	Mobile Mapping Systems
OSM	OpenStreetMap
OGC	Open Geospatial Consortium
PCD	Point Cloud Data
ROS	Robot Operating System
SQL	Structured Query Language
QGIS	Quantum GIS
QA	Quality Assurance
SAE	Society of Automotive Engineers
SLAM	Simultaneous Localisation And Mapping
XML	eXtensible Markup Language

1 Introduction

Autonomous driving (AD) technology has been attracting much attention worldwide as one of the fields of artificial intelligence. Autonomous vehicles (AV) suppose to offer a safe and comfortable ride building their behaviour on the input from several onboard sensors, a map, and a high-level route. They intend to be capable of route planning, self-localisation in the driving environment, and quick responses if the driving environment changes. Depending on the type of the AD system, AVs rely on lane-level precise maps providing information on static elements of the driving environment. That makes so-called High-Definition (HD) maps one of the critical enabling technologies for automated driving and active research topic in recent years [1], [2].

To conduct research in autonomous driving in Estonia, the Autonomous Driving Lab (ADL) of the University of Tartu was founded in 2019 in cooperation with the Estonian mobility company Bolt. It was acknowledged soon that HD maps are needed to start with self-driving car experiments [3]. On the market, HD maps are produced by data providers using fleets of mobile mapping systems (MMS) vehicles that are expensive to set up and maintain. The pipeline comprises the data registration in vehicle coordinates while driving, followed by post-processing, including the extraction of the elements of the driving environment and georeferencing into map coordinates. Furthermore, different analogue research undertakings came out with diverse HD map formats and representations, and some became a standard in the research field. Therefore, it quickly became clear that the only option was to produce maps needed for test sites in the ADL while testing different formats and mapping approaches.

An attempt to establish the ADL's own pipeline for HD maps production is covered in the [3]. Tests and analysis of existing HD map editors produced by open research projects have shown that setting up these pipelines from zero involves laborious manual work and, therefore, is costly and time-consuming. Also, the editors, created for open research projects, are not optimised for mass data insertion and therefore are slow. A robust solution was proposed by [3], which includes applying geospatial software capable of fast data digitalisation in combination with areal photoimagery, which is already in map coordinates and freely available in Estonia. Several sets of geospatial data were created, bearing in mind the requirements of different HD formats. Still, extracting driving environment elements, visible in the imagery as well as virtual elements, such as lane left and right boundaries, remained tedious and called for some semi-automated approach. Also, it was not worth creating the digital dataset underlining the HD map for each format separately. Collected data should be stored in a manner to be reusable for any map format that is employed in the ADL.

The current thesis investigates the possibility of storing data underlining the HD maps in PostGIS, the open-source spatial database built on top of PostgreSQL. The functionality of the PostGIS is not only allowing for storing but also for geoprocessing the data. That means creating new geometry features based on existing ones, finding spatial relations and joins between elements, applying spatial predicates and looking for proximities. Therefore, there is a promise that the semi-automatic approach can be built, and a less time-consuming workflow can be proposed.

The goals of this thesis are to:

- Propose the database design for storing spatial data for various HD map formats;
- Facilitate data collection with semi-automatic creation of missing data;
- Automate finding semantic relationships between map elements;
- Create a data converter for HD maps in Lanelets2 format.

2 Background

2.1 Autonomous Driving

Currently, academics and manufacturers are working to develop advanced driver-assistance systems (ADAS) to attain high-level autonomy in vehicles [2]. Autonomous vehicles (AV) intend to offer a safe and comfortable ride using the output of sensory units, a map, and a high-level route [2]. To operate safely, AVs must have accurate and reliable self-localisation and precisely predict other road actors, such as vehicles and pedestrians, future actions and trajectories [2].

SAE (Society of Automotive Engineers)[1] produced a visual chart (Figure 1) that clarifies and simplifies its J3016 "Levels of Driving Automation" standard. It defines six levels of driving automation, from level 0 (no automation) to level 5 (full vehicle autonomy). It serves as the industry's most-cited reference for automated vehicle capabilities. ADAS belongs to the levels below three, and only starting from level four, the system can be considered fully autonomous, though it does not require human intervention. Such systems are also referred to as autonomous driving (AD) systems.

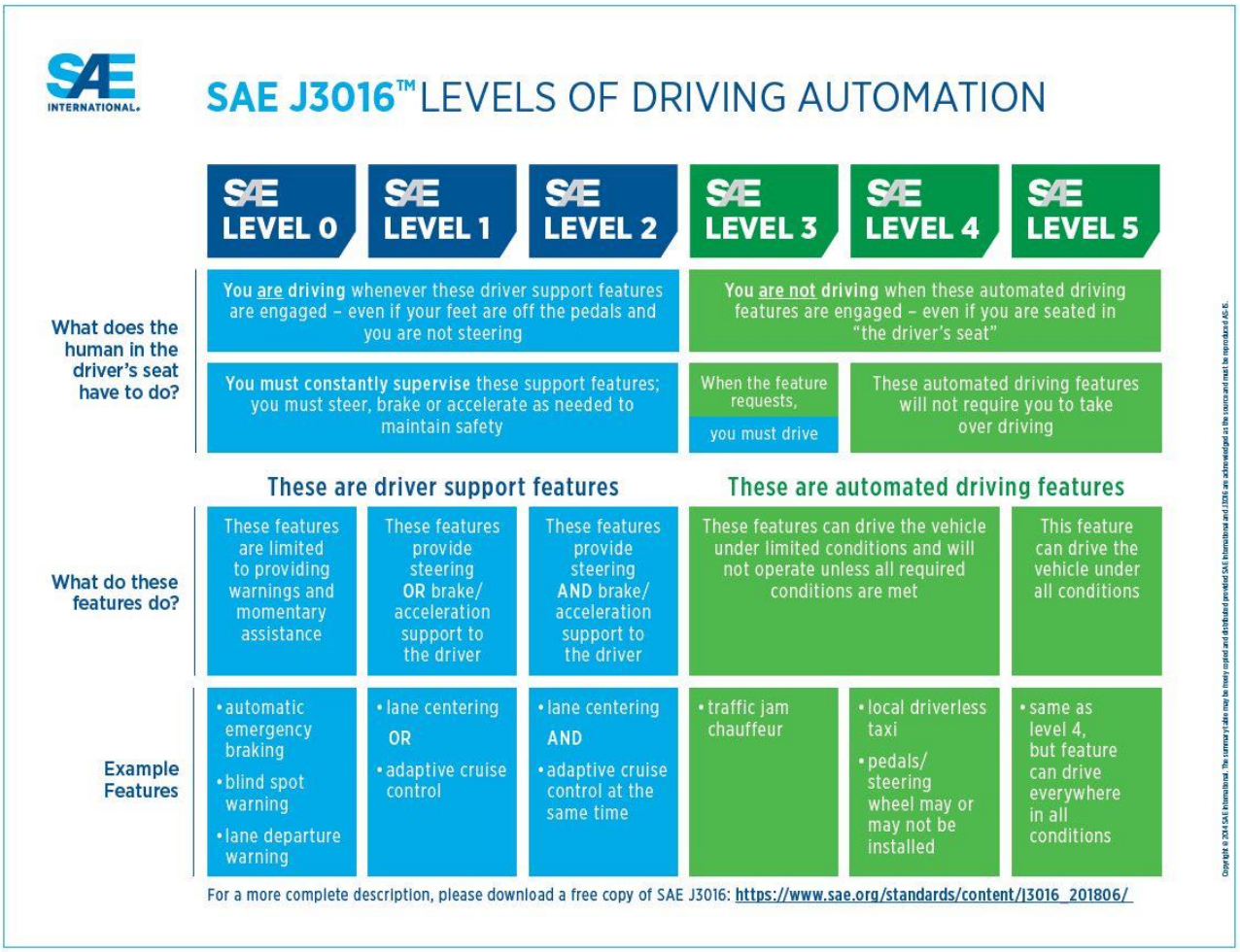


Figure 1. Levels of Driving Automation by Society of Automotive Engineers (SAE).

On a broad level, the research on autonomous driving can be divided into two main approaches: (1) *modular* and (2) *end-to-end* [2], [4]. The first one, originating from robotics, consists of interrelated modules such as

[1] https://www.sae.org/news/2019/01/sae-updates-j3016-automated-driving-graphic

perception, localisation, planning, and control [3] models. The higher levels of autonomy need high-precision localisation with the scene context, commonly provided as prior knowledge in the form of *high-definition (HD) maps* [2] [4]. Different sensors can be used for localisation; the most common solutions are based on Global Navigation Satellite Systems (GNSS), and LiDAR, both fused with Inertial Measurement Unit (IMU) and odometry data and, additionally, radars and cameras are used [3]. Furthermore, when combined with a prebuilt map, a high-precision self-localisation solution can transform the complex problem of perception and scene interpretation into a less difficult positioning problem [2]. The advantage of this solution is that the modularity allows to reliably reason about how the system arrived at specific driving decisions. Even though the modular approach is considered conventional and provides great interpretability, it is often criticised for the pipeline, which is costly to build and maintain [4]. The intricate nature of interdependencies between such modules is a well-established problem in the broader autonomous robotics literature. It has led to the development of frameworks, such as *robot operating systems (ROS)* [4].

Differently from the modular approach, which includes map prior knowledge, other solutions use the so-called *end-to-end* approach. It creates an internal-learned map representation of the driving environment [2] and optimises the entire driving pipeline from processing sensory inputs to generating steering and acceleration commands as a single machine learning task [4]. Current end-to-end solutions use simultaneous perception and prediction to provide outputs such as *object tracking* and *predicted trajectories* [2]. The system learns through imitation of human drivers (imitation learning, IL) or by improving driving policy from scratch via reinforcement learning (RL) [4]. Compared to the modular approach, the end-to-end solution has simpler architecture but a smaller number of intermediate interpretable results, which makes it difficult to trace the initial cause of an error. In addition, HD maps can be incorporated into the end-to-end pipelines if needed, which displays a compromise between the two extremes. Such approaches are sometimes referred to in the literature as mid-to-mid [4] solutions.

2.2 High-definition maps

The modular approach depends heavily on route planning and accurate localisation, especially in dense urban environments. Therefore, HD maps are crucial for AD and can provide much information about the driving scene.

According to [1], high-accuracy map representations can be loosely categorised based on their level of information into one of three categories: *digital maps*, *enhanced digital maps*, and *high-definition maps*. Traditional street maps, such as maps of commercial providers like Google Maps, include street geometry, location names, and distances, are digital maps. The applications can associate location by coordinates or addresses of points of interest with the road network and perform high-level navigational tasks; still, the positional accuracy is not enough for AD [2]. There is also a lack of information that be identified by sensors such as traffic signs and lights.

The enhanced digital maps – maps with more accurate road geometry and richer road attributes and marking added as a layer on top of the existing digital maps – were started to use in Advanced Driver Assistance Systems (ADAS) in the 2000s [1]. These maps included information necessary to compute navigational graphs and reach the destination as well as to assist in some driving tasks. Roads are represented by an imaginary centre line as used in navigation devices. The interpretation of the semantics of the map elements is left to the human driver. Typically, these maps support up to SAE level 3 driving automation.

The concept of HD map was born during the Bertha Drive Project in 2010 [5], [6]. Further, during this project, in 2013, an automated Mercedes Benz S-Class S 500 completed a 103 km journey covering urban and rural roads in fully autonomous mode. According to the proposed concept, the HD map is lane-level accurate, with a centimetre-level precision three-dimensional map, which includes several layers of information about road elements and surroundings [1], [2], [7], [8]. The idea is that after vehicle localisation on the map, not only navigational tasks can be performed, but also some decisions on manoeuvres for ego vehicle and predictions on the behaviour of other traffic participants can be completed by using map information. Therefore some authors call the HD map a powerful "sensor" or "prior perception" that provides detailed information around and further around the corner to support ego vehicle perception [1], [2], [9]. An HD map can be as simple as a collection of accurate positioning of road signs, lane markings, and guardrails in the surroundings or be as complex as a dense semantically segmented LiDAR point cloud that stores the distance to every obstacle around the agent [2]. Some

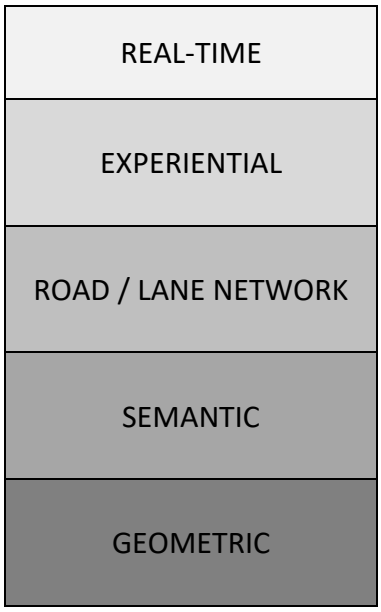
researchers refer to maps that enable safe highly-automated driving (HAD) even in complex city centre scenarios as HAD maps. In this thesis, I use the HD term.

The definition of the content of HD maps is often based on the functionality of its elements. In line with [1], the three-level subdivision is one of the most popular approaches (Table 1, Figure 3). The existing enhanced digital map is the common layer 1 in Table 1. The second layer, often called the "semantic layer", contains data such as lane marking, travel directions, and traffic sign locations [1]. The roadDNA layer in the TomTom solution refers to the detailed 3D representation of the road environment created via utilising the sensor data. There are also solutions having more layers including, for example, "real-time" and "highly-dynamic" layers for mapping traffic dynamics such as conjunctions and traffic lights phases and localisation of the actors [1], [2] or so-called experiential, map "prior" layer[2].

Table 1. Examples of the layered structure of an HD map, after [1].

Map layer	HERE Live map (here.com)	Bertha Drive lanelets	TomTom HD map (tomtom.com)
1	Road Model	Open Street Map (road network)	Navigation data
2	HD Lane Model	Lane level map	Planning data
3	HD Localisation Model	Landmarks/road markings map	Road DNA

Simon Thompson[2] divides the content of HD maps into five layers according to the level of abstraction of information: geometric, semantic, road/lane network, experiential – map prior layer, and real-time layer (Figure 2).



- Traffic information collected by AV fleet operators. E.g. traffic jams, accidents, obstacles.

- Also known as the map 'prior' layer, represents information from the past such as traffic flow, areas with high incidents, etc. Collected and updated by a fleet of AVs.

- Defines connectivity of a road network that is capable of generating routes. Defined by proximity and semantics from two lower layers. Examples of lane connectivity are: succeeding, preceding, adjusting, and conflicting

- Attaches semantic labels/meaning to geometric primitives, e.g. traffic lights to the point. Metadata about such semantic objects, e.g. a number of bulbs, height, etc., can further expand their semantics and define the AV behaviour.

- Geometric position of physical or virtual elements in points, lines, and areas. The layer directly models the environment via a point cloud or subdivide space into areas such as lanes, intersection, etc.

Figure 2. Five levels of abstraction of information in HD maps according to Simon Thompson[2]

Considering relationship semantic information, it must be attached to geometry [9]. In other words, the underlying geometric data must be structured to accommodate the semantics. If driving roles change, e.g. possibility to take lane change or limiting speed, then the geometry must be split into separate segments.

The HD maps are used in every step of AD – sensing, perception, localisation, decision-making, planning, and control (Figure 3); tasks that rely on spatial information need geometric, semantic, and temporal data. HD maps are used as a virtual sensor replacing the actual sensor to a great extent, expanding the range of perception out of the current field of view (FOV). Also, HD maps can direct sensors' attention in areas such as intersections or

[2] https://www.autoware.org/training, Lecture 14. HD maps

pedestrian crossings and aid in the perception of the driving environment. Localisation is assisted by matching current sensor data to what we expect to see. Finally, HD maps can be used in motion planning to plan ego vehicle motion inside the lane network. They are also good at predicting other actors' movements with an expectation that those actors obey the traffic rules and regulations of the lane where they are currently located.

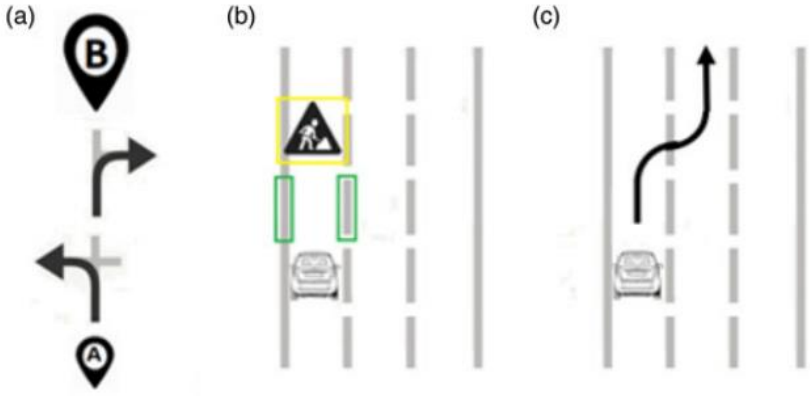


Figure 3. The functionality of the HD Map. (a) Road Model supports navigation; (b) localisation model enables perception using Lane Model: vehicle understands the presence of lane marking and an obstacle; (c) Lane Model supports tactical planning, for example, lane change manoeuvre. After [1].

There are several map solutions and formats that exist nowadays. Some of them are tailored to the proprietary ADAS solutions, while others, in contrast, are open maps fitted to the particular research project or academic publication [8]. For the success of a specific map format, documentation must be available and well-composed. It is also helpful for AD systems developers when HD maps are supplied with APIs and libraries for using and interpreting stored data.

2.3 High-Definition map formats and standards

There are some examples of HD maps already mentioned in Table 1. Two of them – TomTom HD Map[3,4] and HERE Live map[5] – are examples of **commercial proprietary formats** from global map data providers. The interoperability issues between proprietary formats and APIs are addressed by the **Navigation Data Standard** (NDS)[6].

The NDS was jointly developed by automotive manufacturers, map data providers, and navigation device/application providers[7] and is not an open standard as such. Memberships and annual fees are required. The NDS specification covers the data model, storage format, interfaces, and protocols. Association provides the members with the SQLite reference implementation for map database, online and desktop map viewers, validation tools, and OpenDrive converter. Commonly, map providers describing their products make references to the NDS and the level of compliance with the standard. Not members are offered the reduced version OpenLaneModel for free, for which documentation is available after registering on the NDS website. The visualisation requirements of the car navigation system heavily influence the content of NDS. The format classifies all data into different building blocks, such as routing building block, lane building block, and 3D objects building block, which address various functional aspects [10].

The NDS Association has developed a new generation of the standard called NDS.Live and made the specification available to everyone from April 2022. It caters to the transition of navigation from offline to a

[3] https://www.tomtom.com/newsroom/product-focus/adas-map-vs-hd-map/
[4] https://www.tomtom.com/products/hd-map/
[5] https://www.here.com/platform/HD-live-map
[6] https://nds-association.org/
[7] https://discourse.ros.org/t/autoware-maps-and-map-formats-working-group-meeting-minutes-24-july-2019/10059

hybrid/online paradigm and addresses the growth of map data in size. Data can be available as tiles, along driving paths, or just as a point-based data feature.

TomTom HD Map[3,4] has three layers and addresses the challenges of AD by utilising the concept of RoadDNA, which is a suite of localisation layers, namely *roadsides, signs, markings, road-surface-marks, radar, poles,* and *reflectivity* (Figure 4). An AV compares RoadDNA data with the real-time sensors' data to precisely position itself on the road, obtaining an accurate lateral and longitudinal position.

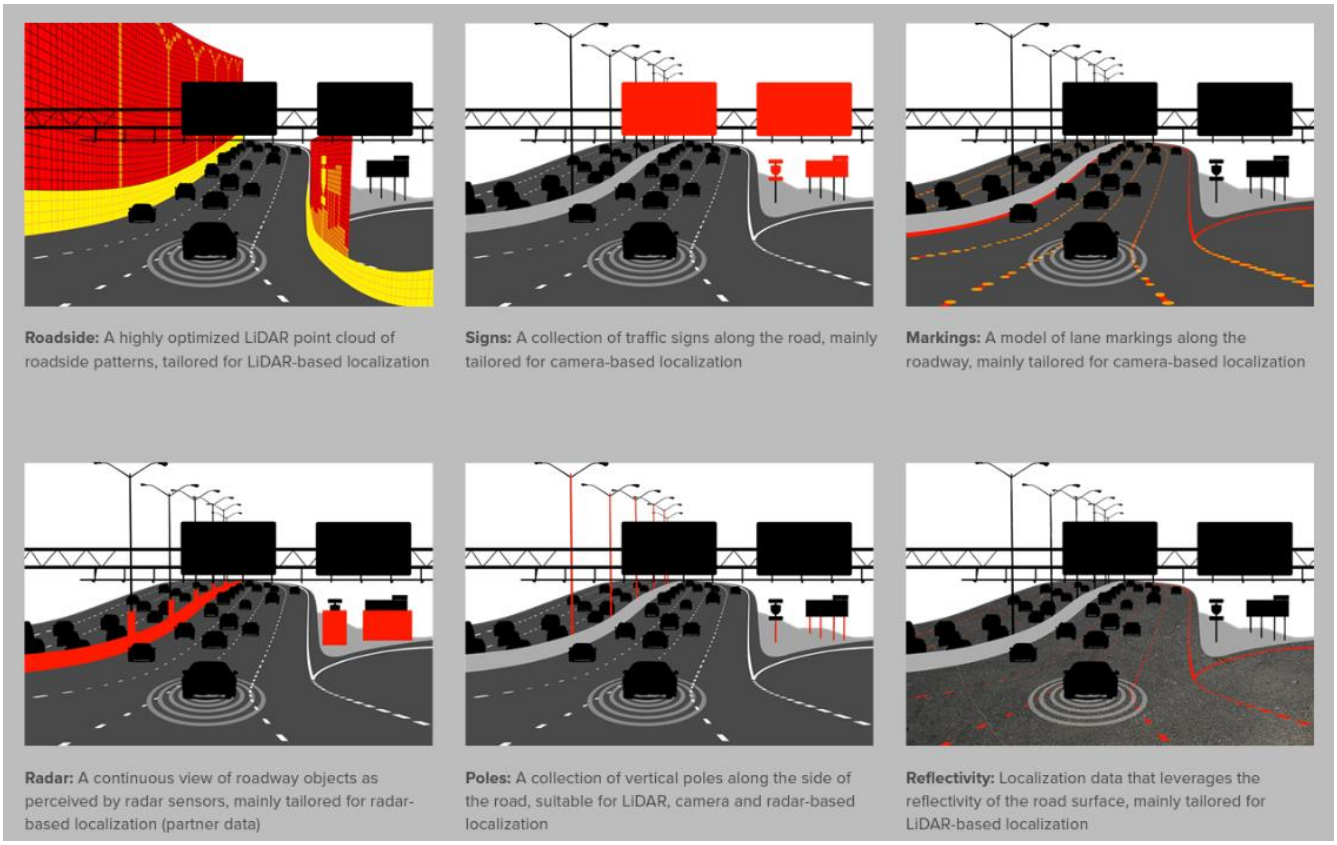


Figure 4. Localisation layers of TomTom RoadDNA[4].

HERE HD Live Map[5] is a highly-detailed, cloud-based map for AD. It is updated in near-real time with sensor data from connected vehicles applying so-called self-healing map technology and continuous evaluation of map quality index for data. The Road Model (Figure 5) contains an ADAS map based on the HERE navigation map with extensive global coverage, including road topology, ADAS attributes, and embedded semantic data. The HD Lane Model provides precise, lane-level topology and geometry, modelled with 3D positions. The HD Localization layer contains accurate road furniture, such as signs and poles.

The base map is the foundation of the HERE HD Live Map. Data for the base map is captured using HERE True Vehicles, equipped with LiDAR. Using crowdsourced vehicle sensor data, HERE also collects drive paths, lane markings, and more information. These sources are combined with others - like satellite imagery – to maintain a fresh HERE HD Live Map, which is then made available through HERE Marketplace and is published in the HERE Native format (Protobuf) and as NDS 2.5.4 compatible.

Figure 5. HERE Live HD Map layers[5].

Even though several commercial providers have defined their HD map formats, there are arguments that these formats are not research-friendly enough due to their confidentiality. Apart from commercial HD maps, **open map solutions** evolved from different industries, such as vehicle navigation systems and driving simulators, or have been proposed by academic researchers in open-source projects. Some of them became *de facto* standards; among them are ASAM OpenDRIVE, Autoware Vector Map, and Lanelet2.

ASAM OpenDrive[8] was initially developed by a driving simulator company called VIRES Simulationstechnologie GmbH and became a standard of the Association for Standardization of Automation and Measuring Systems (ASAM)[9] in 2018. The primary purpose of ASAM OpenDrive is to provide a road network description that can be fed into simulations to develop and validate ADAS and AD features [8]. The standard describes a road network with eXtensible Markup Language (XML) syntax using the file extension XODR. Thanks to this organisation, the map can be extended from each *node* with user-defined data. This functionality allows for a high degree of specialisation for particular applications (usually simulations) and keeps the interoperability of the data exchange between different applications. OpenDrive file stores the geometry of roads, lanes, and objects, such as traffic signs and lights. The road network is modelled along the *reference line*, which is the basic primitive. Roads, lanes, incl. their elevation profiles are all attached to the reference line (Figure 6). Road objects, such as traffic lights, can be placed using either the reference line or spatial coordinate reference system in which the road network is placed. Various coordinate frames are possible from global (latitude/longitude) to local projected coordinate systems. In addition, the format allows for the construction of hierarchical junctions and controllers of the junctions.

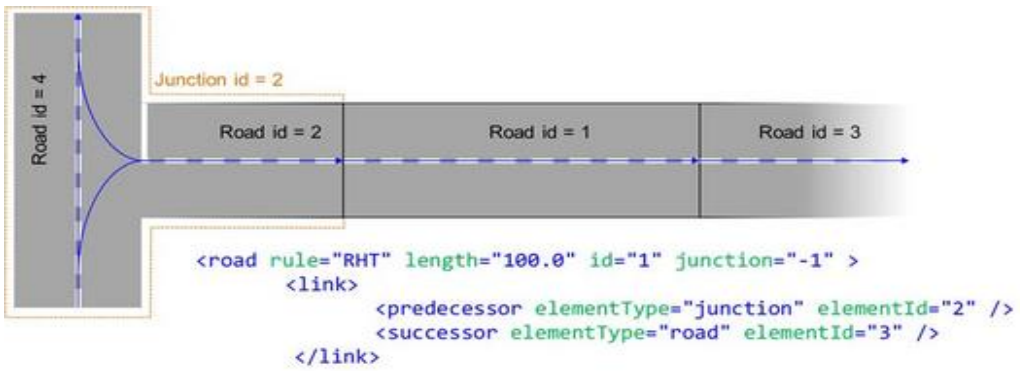


Figure 6. Individual interconnected road segments of ASAM OpenDrive[9] HD map.

[8] https://www.opendrive.org
[9] https://www.asam.net/standards/detail/opendrive/

Although the specification is open, no freely available libraries exist for interpreting and processing the data [8]. The related ASAM projects are OpenCRG[10], developed to describe road surfaces and OpenScenario[11] for representing the dynamic content of driving and traffic simulators.

Autoware Vector Map format was designed for Autoware.ai[12], one of the world's leading open-source software projects for AD by Autoware Foundation (AWF). The software is designed for scalability across a broad range of AD applications and has a modular architecture. The format is a simplified version of the proprietary Aisan Technology ADAS map from which it is originating.

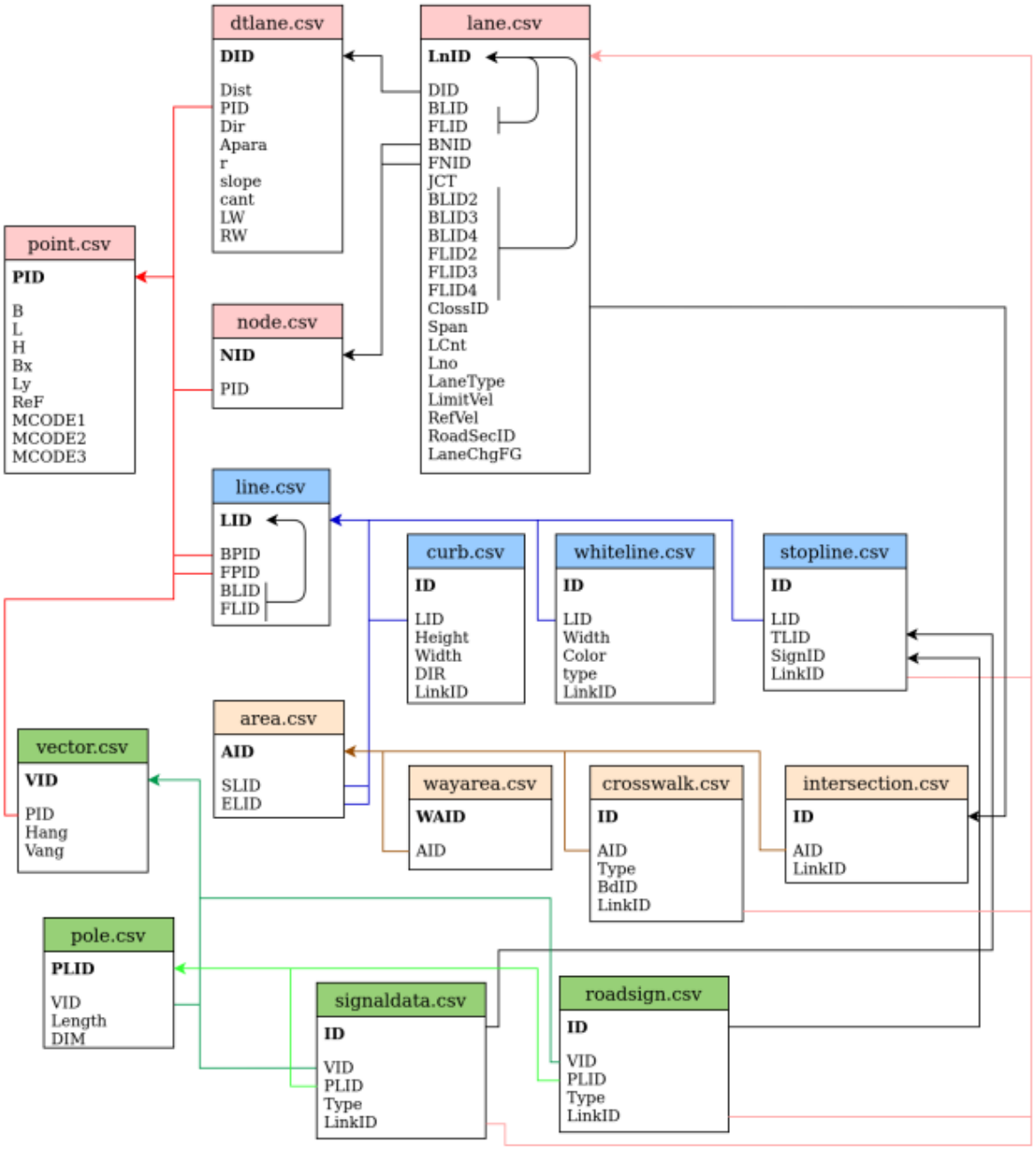


Figure 7. Files' structure and relations between the features used in Autoware Vector Map format in ADL lab, after [3].

[10] https://www.asam.net/standards/detail/opencrg
[11] https://www.asam.net/standards/detail/openscenario/
[12] https://www.autoware.ai/

Unfortunately, the documentation on Autoware Vector Map is scarce. However, some clues can be found in the tutorials and documentation of dedicated creation and editing tools. According to [11], it is formatted as a set of comma-separated value files, up to thirty-two categories. The categories are: point, vector, line, area, pole, box, dtlane, node, lane, way area, road edge, gutter, curb, white line, stop line, zebra zone, crosswalk, road mark, road pole, road sign, signal, street light, utility pole, guardrail, sidewalk, drive-on-portion, crossroad, side strip, curve mirror, wall, fence, and rail crossing. But the minimum working set comprises only four files, which participate in the path planning: point, node, lane, and dtlane.

Point is a basic spatial primitive which includes the PID and x, y, and h values in the local coordinate system. B and L in Figure 7 stay for *latitude* and *longitude*. The nodes file stores IDs from the points' file. Dtlane file lists the centre lines of the lane using point/node IDs with direction and heading values. The lane file defines lanes that consist of connected dtlanes primitives [10]; also, connectivity between lanes is stored here – BLID and FLID stand for before lane ID and following lane ID correspondently. Other geometrical primitives are *line* (based on points), *area* (based on lines), *vector*, and *pole* (both based on points). They are the basis for constructing road furniture, areas, and driving environment elements.

The AWF also developed the Vector Map Builder tool[13] to generate both Autoware Vector Map and Lanelet2 maps. However, since it is an online tool many users share, it sometimes has its overload. Also, the tool requires Point Cloud Data (PCD) file, and lanes must be drawn on the PCD manually point by point, which is time-consuming [11]. It is why [11] designed their own tool called Simple Vector Mapper Tool (SVMT). There are also other tools for Autoware maps, such as MapToolBox[14] and Assure Maps[15]. The comparison of mapping tools for Autoware can be found in [3].

Lanelet2[16] is a C++ library for handling map data in the context of automated driving. It is a generalisation of the LibLanelet approach of the Bertha Drive Project [5], [6] that addresses later shortcomings [8]. According to its authors, important concepts applied in Lanelet2 development are flexibility and extensibility, which can handle the upcoming challenges of future maps and applications. Furthermore, Lanelet2 library and API perform several AD tasks for traffic actors such as ego vehicle, other cars, pedestrians, and cyclists. The data format is based on Liblanelet, an extension of the XML-based OpenStreetMap[17] (OSM) data format, for which several editors like Java OpenStreetMap Editor[18] (JOSM) are available.

The map consists of three layers [8] (Figure 8):

- *physical layer* for real observable elements;
- *relational layer*, in which the elements of the physical layer are connected to lanes, areas, and traffic rules;
- *topological layer*, where the elements of the relational layer are combined into a network of potentially passable regions, depending on the road user and situation.

There are five primitives distinguishable in the Lanelet2 format (Figure 8). *Points* and *line strings* (*nodes* and *ways* in OSM terms) belong to the physical layer. Line strings consist of points' sequences. Further, *lanelets*, *areas*, and *regulatory elements* belong to the relational layer. All elements have a unique ID and set of attributes or so-called *tags* in the form of key-value pairs. The format can be extended by adding custom attributes to the fixed ones. *Points* are only primitives that can be described by latitude/longitude and by X, Y local projected coordinates. Since height information is essential in AD, a height coordinate is also required, for example, to determine the road height profile. Each *point* represents a single vertical structure such as a pole or participates in line string definition. Line strings describe either observable road elements such as road markings, curbs, etc. or can be imaginable or *virtual*, indicating the border of lane or area. *Lanelets* define an atomic section of the map in which directed motion occurs, such as standard lanes or pedestrian crossings. Atomic means traffic rules and topological relationships do not change within a lanelet. A lanelet is defined by exactly one line string as the left and exactly one as the right border. In addition, the centre line or trajectory can be stored. Lanelet may reference several regulatory elements that express traffic rules applying to this lanelet. Lanelets can also overlap

[13] https://tools.tier4.jp/feature/vector_map_builder_ll2/
[14] https://github.com/autocore-ai/MapToolbox
[15] https://GitHub.com/hatem-darweesh/assuremappingtools
[16] https://GitHub.com/fzi-forschungszentrum-informatik/Lanelet2
[17] https://www.openstreetmap.org/
[18] https://josm.openstreetmap.de/

or intersect. Topological relations between lanelets are guarded by shared endpoints of the left and right border. *Areas* are parts on the map such as parking, squares, green spaces or buildings, where undirected or no movement is possible. Finally, *regulatory elements* define traffic rules, such as speed limits, priority, or traffic lights, which are applied to one or more lanelets.

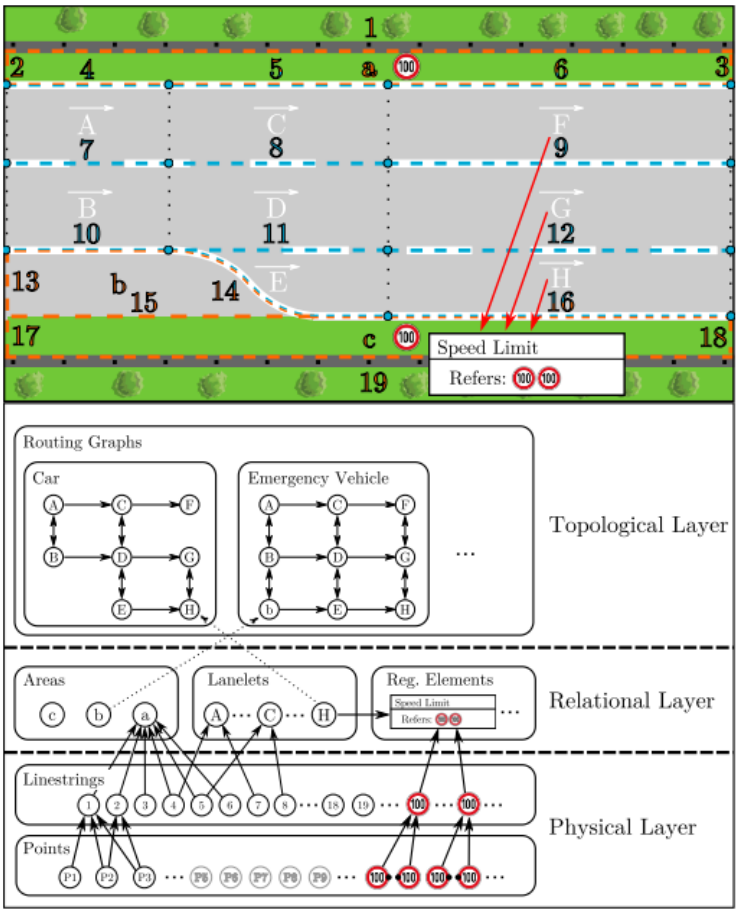


Figure 8. Map example for a highway road and the resulting map structure. Lanelets have capital letters, areas lowercase letters, and linestrings have numbers. Source [8].

The advantage of Lanelet2 is well defined and maintained API with several modules:

- *Core* module contains the basic primitives and layers described above as well as, geometry functionalities, including calculation of centre lines, distances, and overlaps;
- *Traffic Rules* module interprets the rules contained in the map depending on the type of road, and it also determines whether lane changes are possible;
- *Physical* module allows direct access to the elements of the physical layer;
- *Routing* model produces routing graphs with the help of traffic rules, including possible lane changes, or to predict possible routes and points of conflict for other road users;
- *Matching* model assigns lanelets to road users or to determine possible positions on the map based on specific observations of the sensors.

There are also models handling input and output, re-projection of the coordinate system, and map validation.

Interestingly, Autoware.ai starting from version 1.13 and the most recent Autoware.auto[19] also use the Lanelet2 maps and have its own extension[20,21] for the general Lanelet2 format.

[19] https://gitlab.com/autowarefoundation/autoware.auto/AutowareAuto/
[20] https://github.com/autowarefoundation/autoware_common/tree/main/tmp/lanelet2_extension
[21] https://autowarefoundation.gitlab.io/autoware.auto/AutowareAuto/lanelet2-map-for-autoware-auto.html

2.4 Mapping methods

Making HD maps requires accuracy, completeness, verifiability, and extensibility. However, because of the complexity of HD mapping, it is currently expensive and difficult to implement, particularly in the urban environment [2]. In addition, the mapping process involves a high proportion of manual labour for digitising and structural editing, making it challenging to maintain frequently changing road conditions [12]. Moreover, outdated or erroneous maps risk the safety and performance of stationary driving environment modelling [13].

A common approach is missing when it comes to the definition of scope, the requirements and the configuration of workflow [7], [14]. Rehrl et al. [7] propose and evaluate a 4-step workflow for creating an HD map which includes sub-flows for (1) definition of the scope based on use cases, (2) mapping and semi-automatic extraction of objects from a LiDAR point cloud, (3) HD map composition with OpenDRIVE and Lanelet2 as target formats and (4) testing and iterative refinement for the intended use cases. In step 1, the HD map requirements are specified, and a catalogue of mapping features is produced according to the type of AV, level of automation, and target map format.

Tsushima et al. [14] proposed a 5-step workflow: 1) 3D point cloud survey, 2) 3D plotting and semi-automatic feature extraction, 3) editing of feature attributes and building of road network topology, 4) target format conversion and 5) quality assurance (QA). Some researchers concentrate on the data survey and feature extraction steps dividing feature extraction into several sub-flows [15] by feature type in question or applied algorithms. In some workflows, the ground data collection with car sensors is substituted with aerial or/and satellite orthophoto imagery and Aerial Lidar Systems (ALS) point cloud data produced by national authorities or mapping companies [3]. The tasks of features' vectorisation, attribution, topology building and composition of complex map elements from geometric primitives are less covered topics in the literature than automatic feature extraction algorithms. Some authors view vectorisation and attribution as dependent on target map format, while others create a kind of "universal" dataset, which can be used for several map formats [5]. Figure 9 represents the generalisation of the mapping workflow, which accommodates steps found in different sources.

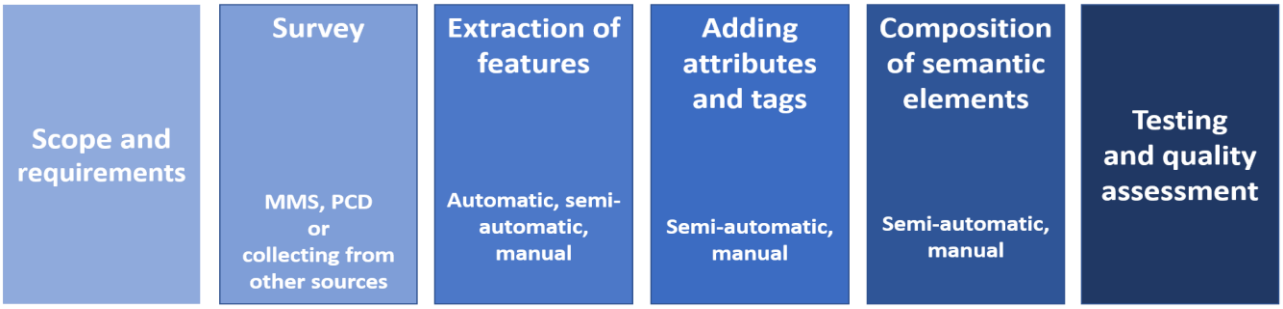


Figure 9. General mapping workflow

While some authors claim a high level of automation in HD map creation, it is still time-consuming and prone to errors due to the level of human intervention needed and shortcomings of extraction algorithms. The workflows of [7] were evaluated by applying them on a 2-kilometres-long rural test track during the Digibus Austria project for EZ10 shuttle AV. Estimated human efforts for feature extraction and map composition are presented in Table 2 and Table 3 correspondently.

Table 2. Estimated efforts in person-hours for extracting objects from a 3D point cloud for a 2-kilometres-long rural test track. After [7].

Object category	**Extraction notes**	**Effort [h]**
Road middle axis, Lane boundaries	Semi-automated where road markings exist, manual where road markings are absent	12
Road markings	Semi-automated or manual depending on the wear	6
Lamp posts	Automated extraction (position and height), check for false positives, some manual placements at the edge of forest	2
Curbs	Automated in stretched sections, manual rework at intersections	24
Verge	Due to the smooth transition between road and verge often difficult to extract, substantial manual intervention	12
Traffic signs	Position and height are detected automatically, and content is derived manually from imagery	4
Total effort		60

Table 3. Estimated effort in person-hours for the six steps of composing the HD map for the 2-kilometres-long rural test track. The target map format is Lanelet2. After [7].

Step	Description	Effort [h]
1	Set Lanelet2-specific linestring type tags	6
2	Split lanes to sections	4
3	Create virtual lines to create a topologically correct graph	12
4	Compose lanelets from line boundaries	8
5	Create areas	4
6	Create regulatory elements	4
Total effort		38

The mapping procedure is specific to each map format and map layer. The heaviest part is the creation of geometric and semantic layers, the latter may include the connectivity of the road/lane network [7]. The geometric layer consists of ***dense 3D point cloud data (PCD)*** representing the driving environment. Points are basics for other co-called ***vector geometry*** primitive's construction such as lines, areas, vectors, and nodes. A semantic map composes geometry primitives into lanes with their left and right bounds, pedestrian crossings, traffic regulatory elements, etc. Also, semantic maps add metadata on road connectivity. Both layers are produced in one process when the semantic layer is usually created by overlaying on top of a PCD map or satellite/airborne orthophoto imagery.

Data for creating maps can be captured in many ways, from satellite to aerial to terrestrial sources. These sources may capture a wide variety of data, from high-resolution RGB imagery to 3D point clouds to thermal imagery [9]. For AD, the focus is on the acquisition of the PCD map which is the capture of sensor data from vehicle-mounted sensors also known as *Mobile Mapping Systems (MMS)*. MMS is typically equipped with GNSS-IMU coupled highly accurate positioning system, calibrated video cameras, and LiDAR sensors. Data is collected by a single car or fleet of capture vehicles that can produce many local maps which are then merged into a single global map [9]. The systems differ in data management strategies such as physical storage and data transfer or cloud-based transmission of data. Autoware captures and stores data using ROSBAG files. Nowadays the need for data frequent updates – so-called map self-healing - is pushing towards cloud-based systems and harvesting data from private vehicles.

SLAM – simultaneous localisation and mapping – is a base technology for 3D PCD creation from LiDAR [3]. Two processes undergo simultaneously – the localisation of the vehicle in geographic space and the recording of sensored points. It is a research field by itself and is out of the scope of this thesis. Briefly, the steps are:

- Local positioning tracking by matching sensor data in sequential frames
- Accumulation of sensor data from a series of key frames – poses along the vehicle path
- Integrate data into a consistent global map;
- Check for loops in the travel path: loop closing and error correction via pose graph optimisation.

Other sources of data that come from conventional or "traditional" mapping are aerial or satellite imagery collected by both the private and public sectors. The advantage of those sources is that they can provide information in the areas which are hidden from MMS sensors. Aerial orthophotos are characterised by high resolution and positional accuracy (10 cm), they do not suffer from GNSS occlusions, and their accuracy is usually verified by employing well-established methods using ground control points by data providers.

Creating the semantic map is done either manually by drawing geometry primitives on the point cloud or digital image by using dedicated HD map editors of more general geographic information system (GIS) software [3], [12], [14]. GIS software provides general tools for managing spatial data, handling coordinate reference systems (SRS), geoprocessing of imagery or vector geometry as well as for conversions between two of them. In parallel to the drawing, the semantic labels and attributes/tags are added [3], [14]. The connections between primitives are added via attributes during editing by the operator [3] or can be found automatically via geoprocessing and establishing GIS-based topology rules [14]. Examples of free dedicated editors are Java OpenStreetMap Editor (JOSM) and Autoware VectorMapBuilder. GIS software used and mentioned in the literature are ArcGIS[22] and Quantum GIS[23] (QGIS). In the case of GIS processing, data is stored in shapefiles [3], [7], [9], geopackage files [3], in the PostGIS database [4], [7], in ASCII files [14] or in OSM format [6], [7].

There are also examples of automatic or semi-automatic semantic feature extraction in the end-to-end domain using LiDAR and camera data. Still, automatic extraction needs to be post-processed by a human. Choi and Kim [12] used MMS data and proposed a deep learning model to extract areas of road lane markings from point clouds and then automatically generate them. They used the ArcGIS tool to expand bits of lane markings into one line. Shipitko et al [16] proposed an algorithm based on an occupancy grid mapping method with an inverse sensor model for mapping road linear features. At the end of mapping, the values of the cells were binarised according to the threshold, and vectorised linear features were constructed. No driveable maps were produced. Tsushima et al [14] described a machine learning method to detect traffic lights and signs. Roads and markings are extracted from aerial imagery using fully convolutional neural networks followed by remote sensing processing techniques such as filters, image segmentation, and pattern recognition [17]. Aerial imagery is also used in combination with MMS data via different matching scenarios [18].

[22] https://www.esri.com/en-us/arcgis/products/index
[23] https://www.qgis.org/en/site/

3 Methodology

3.1 Data sources and data collection

3.1.1 Aerial imagery

The **Estonian Land Board** is a national provider of a broad range of geospatial data for topographic maps, land management registries such as cadastre, nature protection registries etc. It produces and updates on a regular basis nationwide layer of aerial orthophotos. *Orthophoto*[24] is processed aerial imagery from which distortions caused by terrain relief, camera tilt relative to the ground at the moment of exposure removed and camera-centred projection changed to map projection. Digital orthophoto has a certain pixel size or resolution of 20-40 cm for nationwide coverage and 10-16 cm in urban areas. Therefore, it makes the product an ideal source of information for HD maps. Images are acquired preferably in spring to minimise the effect of trees' crowns and shadows. One-fourth of the country's territory is updated yearly for national coverage. The new flights for cities are done according to the number of changes, but not fewer than three years[25]. The orthophoto production cycle takes up to several months, and imagery acquired in spring is usually made available via the Land Board portal Web map service[26] (WMS) in late autumn. An example of an orthophoto around the Delta building of Tartu University is in Figure 8. This layer is an open dataset freely available via the Estonian Land Board web service.

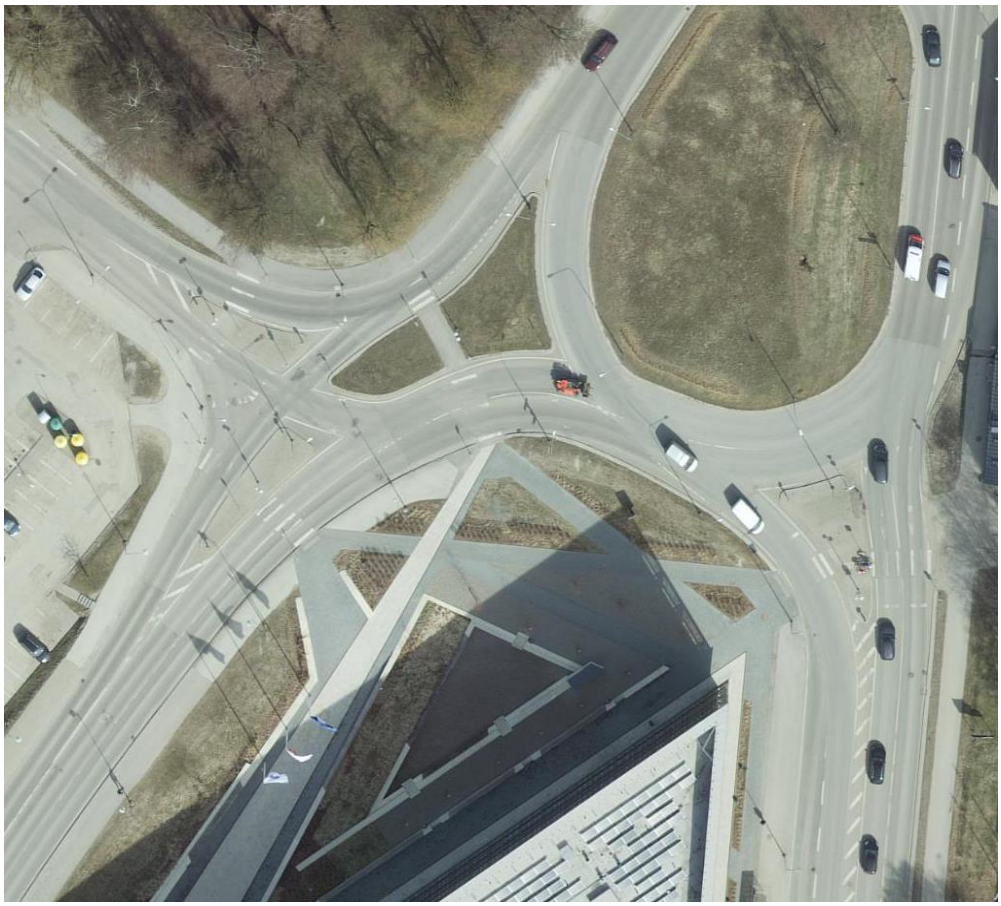


Figure 8. Orthophoto north to the Delta building, Tartu University. Estonian Land Board 2021.

[24] https://geoportaal.maaamet.ee/eng/Spatial-Data/Orthophotos-p309.html
[25] https://geoportaal.maaamet.ee/eng/Spatial-Data/Orthophotos/Orthophoto-metadata-by-year-p350.html
[26] https://geoportaal.maaamet.ee/eng/Services/Public-WMSWFS-p346.html

3.1.2 Elevation data

Estonian Land Board performs Aerial Laser Scanning (ALS) at the same time as aerial imagery capturing. As a result, digital elevation models (DEM) and 3D ALS point clouds are produced. The most detailed DEM is a regular grid that has a 1 m spatial resolution with elevation data precision of millimetres. These data can be used for automatic height extraction for HD vector map elements (Figure 9). Georeferenced ALS point clouds and orthophotos could be matched with MMS point clouds to perform automatic georeferencing [19]. This layer is an open dataset and freely available.

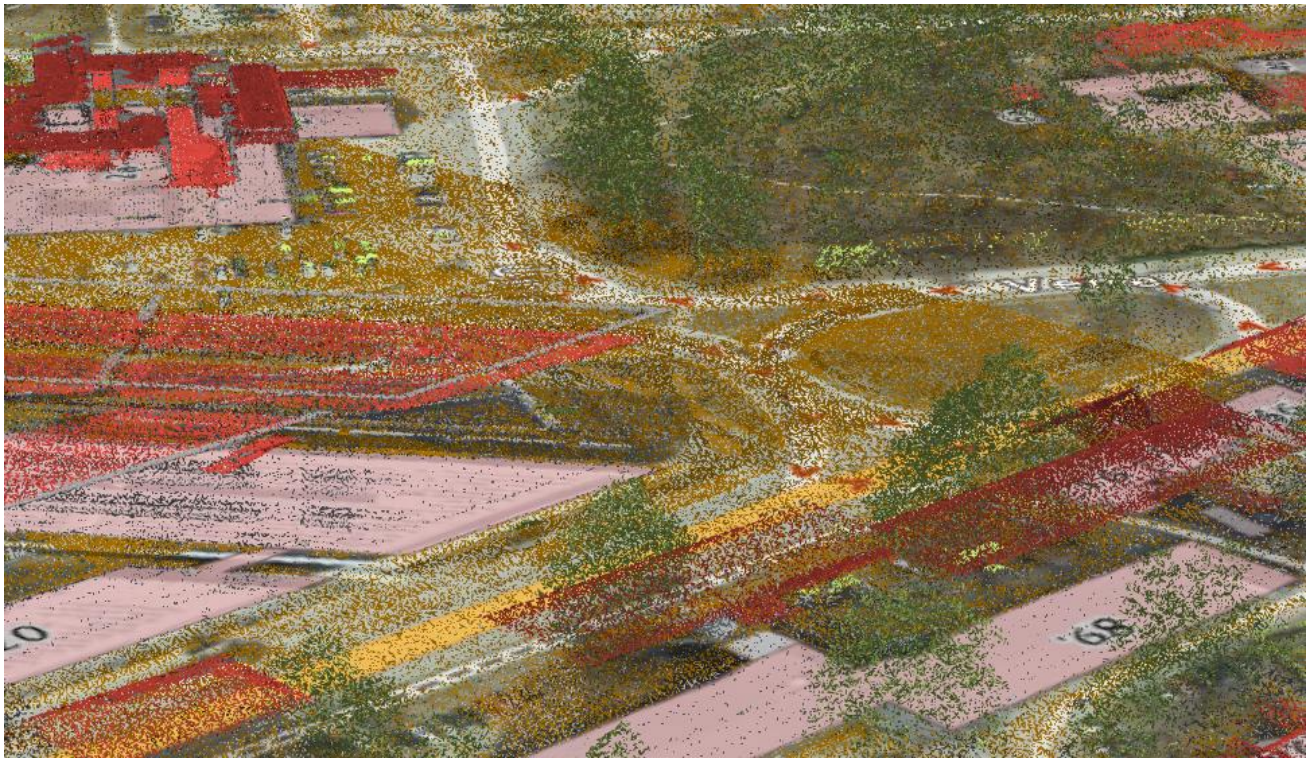


Figure 9. ALS point cloud data from Estonian Land Board 2020.

3.1.3 AD vehicle trajectories

Vehicle trajectories recorded using the vehicle-mounted GMSS device can be used for automating the generation of lane centrelines. They were surveyed during the ADL data collection and test drives for previous mapping campaigns (next section) and used for on-the-screen digitising reference.

3.1.4 Data mapping campaigns in the ADL

Up to now, data for HD maps was manually extracted from aerial orthophoto imagery by digitising on screen using the geospatial software QGIS. Data was kept in two ways: (1) in a set of shapefiles[27] (further SHP data) with attributes necessary for Autoware Vector Map and (2) in geopackage[28], bearing in mind the needs of Lanelet2 format (further GPKG data). Both HD map formats require a number of similar elements with coincident geometry but also need some specific features. Table 4 presents several datasets for different test sites created previously in the ADL. After the previous data was inserted, the Tiksoja test site data was added to the database already using QGIS as a client of PostGIS.

[27] https://www.esri.com/content/dam/esrisites/sitecore-archive/Files/Pdfs/library/whitepapers/pdfs/shapefile.pdf

[28] https://www.geopackage.org/

Table 4. The ADL map digitalisation campaigns prior to 2022.

No	Site	Format
1	Tartu_demo_route	SHP & GPKG
2	Auvetec_tartu_ut	SHP
3	Cleveron_track	SHP
4	Tallinn_bolt_route	SHP

Feature extraction workflow was proposed in [12] and is not a part of this thesis. The workflow was built using QGIS, an open-source GIS software to handle spatial vector information. The recent orthophotos served by Estonian Land Board via WMS service were used as a background map in the QGIS environment without the need for downloading. As data storage, the sets of shapefiles or geopackage files were used. First, a simplified description of the real-world features that need to be mapped was produced (reality model). Second, the standard digitising rules were described to make the work of different operators comparable (data model). Finally, the presentation model - the cartographic visualisation of the features – was created to display the data as a map (Figure 10). The presentation model is essential for humans to make it more interpretable and tackle digitising errors visually. One of the negative aspects was the lack of support for large 3D point clouds and 3D map editing [12]. On the other hand, from testing different HD map editors, it was evident that extracting features in 3D is slower than in 2D. Finally, the third dimension (elevation) was automatically added to vector features from DEMs during the conversion step.

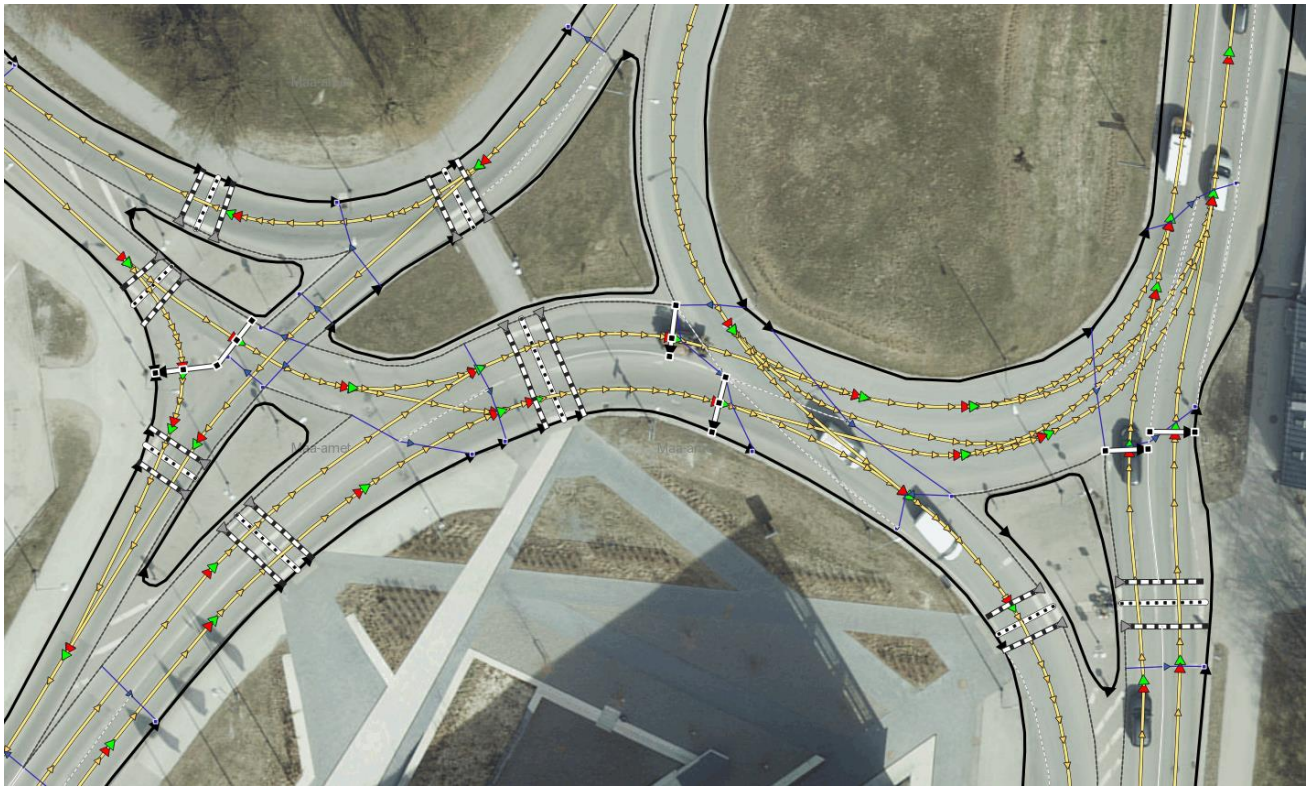


Figure 10. HD map features extracted from orthophoto imagery during digitalisation.

3.2 PostGIS Database

The current thesis investigates the possibility of storing data underlining the HD maps in **PostGIS**[29], the spatial database extender for PostgreSQL[30] object-relational database. The PostGIS is not only allowing for storing and

[29] https://postgis.net/
[30] https://www.postgresql.org/

re-using of data but also for geoprocessing of it. Geoprocessing may include the generation of lane boundaries where elements are not evident from the imagery as well as for the structural editing – finding spatial relationships between the lane elements and geometry primitives. These capabilities are promising significant reduction of human manual work in HD map preparation.

PostGIS adds support for spatial objects allowing location queries and spatial joins to be run in Structured Query Language[31] (SQL). It stores and manipulates spatial objects like any other object in the database, and spatial geometry data types are like any other data type in a relational database. First, to be spatial, in addition to spatial data types, the database should have constructors to generate and handle geometry based on points, lines, and areas. It includes generating new spatial features from existing ones, making complex geometry types from basic types, and accessing geometry primitives from which created geometry features. Next, it should include spatial predicates and functions, posed in SQL, for querying spatial properties and relationships. Finally, multi-dimensional spatial indexing is necessary for the efficient processing of spatial operations. Database clients can access spatial objects using general proprietary or open geospatial software such as QGIS. All this is possible due to the application of the OGC Simple Feature Access standard[32] in PostGIS development, which database clients can also understand.

Since the GIS applications were first developed, they were based on spatial "flat" files such as shapefile[33], which do not allow concurrent usage. The standard database solution of PostgreSQL is used to perform concurrent access without corruption and slowdowns. Complex questions and exciting analyses, including spatial and non-spatial data, can be expressed in one SQL statement on the database.

Basic geometry primitives (data types) of PostGIS are *points, linestrings and polygons* that correspond to general points, lines and areas. Points are characterised by X, Y, and H coordinates expressed in the local coordinate reference system (CRS) or geographic coordinates. Linestrings compose of points and polygons from one exterior and zero or many interior linestrings. Three primitives are the basis for the construction of more complicated geometries. Every geometry column is constrained with the value indicating the CRS of geometry stored in the column. Moreover, PostGIS tables are not restricted to having only one geometry column. So, a table can keep, for example, the linestring representation of the real-world object in one column plus the particular geometries of start and end points corresponding to the same line and found by spatial constructors of PostGIS – functionality which is not possible in "flat" geospatial formats.

PostGIS is powerful because it not only stores geometry but also has the ability to find *relationships between geometries*, described by the OGC Simple Feature Access standard. Questions like "Which is the closest lane marking that does not intersect the vehicle trajectory?" or "To which lane end node the stop line can be associated?" can only be answered by comparing geometries representing the lanes, markings and regulatory elements.

The main idea of creating **the ADL primary map database** is to combine and standardise all map creation efforts undertaken so far. The data collected under the hood of the basic map should be useful for several target formats and serve different areas for testing. Both envisaged target formats, the Autoware Vector Map and Lanelet2, require a number of similar elements with coincident geometry but also need some specific features. The challenge of this exercise was to provide a specification for a single spatial database, which can store all features necessary for both formats and help avoid double digitalisation. Currently, the database structure contains only features used for HD maps generation so far, but the original specifications of Autoware Vector Map and Lanelet2 include more features. The rationality of incorporating more features into the database should be addressed in the future case-by-case.

After careful cross-checks of SHP and GPKG datasets, it was ensured that all necessary elements are preserved or can be created by the converter from the remaining ones. All attributes, which are not changing in the data and eventually assigned by the converter, were removed from the database. Tables' and attributes' name convention was developed. Imported data is stored in the database **hdmap_v0** in the **public** schema[34].

[31] https://www.w3schools.com/sql/
[32] https://www.ogc.org/standards/sfa
[33] https://www.esri.com/content/dam/esrisites/sitecore-archive/Files/Pdfs/library/whitepapers/pdfs/shapefile.pdf
[34] https://docs.google.com/document/d/1wSGYhxzAvgnkRXN3foGNtDh7k87FCirhbF4ztEcDUeY/edit?usp=sharing

3.3 Semi-automatic generation of HD map elements and converting to Lanelet2 format

Created during some research projects, existing open HD map editors that allow extraction of the geometry primitives while simultaneously combining them into primitives of the HD map are slow for mass data production [12]. Therefore, this thesis uses a semi-automatic approach based on (1) the advantage of using open GIS software optimised for mass digitalisation in 2D and (2) the functionality of PostGIS.

First, lane trajectories are produced with vehicle sensors' help, and the linear elements visible on the imagery are extracted during on-the-screen digitalisation. Still, digitalisation is a time-consuming process. Therefore, to minimise human efforts, the rest of the elements needed to construct a proper lane, so-called virtual boundaries, are produced through geoprocessing using PostGIS functionality. Later, we again minimise manual work by finding the relationships between the lane elements and lane versus regulatory elements with the help of spatial operations in PostGIS. Finally, a map corresponding to the Lanelet2 format is generated.

For a better understanding of the further explanation of the algorithm, some notes on the naming conventions of basic geometry types are necessary. In PostGIS, three basic geometry primitives are *point*, linestring and *polygon*, while in Lanelet2, they are *node*, *way* and *relation* (Table 5). PostGIS linestring can be decomposed into points, where the start and endpoints are colloquially addressed as nodes, sometimes dangling nodes, which may create confusion with the general name of points in the Lanelet2. The rest of the points constituting linestring in PostGIS are called vertices. In both formats, an area is a composition of one exterior line and, if needed, one or many interior line primitives. In the case of Lanelet2, this aggregation is realised through an XML element called *relation*. It should be pointed out that the relations mechanism is also widely used in Lanelet2 for creating lanes, semantic elements, and their relationships to the lane. The *lanelet* is an XML relation element which consists of references to the three ways representing left, right and, optionally, centreline. A regulatory element, for example, is a relation referring to a stop line and traffic lights. Furthermore, the relation representing a lane can include a reference pointing to regulatory element relation. To be workable, both formats require unique identifiers for stored elements. In PostGIS tables, the term "feature ID" is used and abbreviated as 'fid', while Lanelet2 uses 'id'.

In geoprocessing, to establish relationships between three linestrings/ways defining a lane, a linestring primitive which passes the endpoints of all three linestrings/ways was introduced. This primitive is also called *'relation'* in the processing flow (Figure 20).

Table 5. Geometry primitives in PostGIS and Lanelet2.

	PostGIS	Lanelet2/OSM
Point geometry	Point	Node
Line Geometry	Linestring	Way - sequence of nodes
Area geometry	Polygon	Relation of closed ways

First, SQL scripts were made to test concepts of geoprocessing and their order – create buffers, linestring offsets, and produce snapping, cleaning, and aggregation of geometric objects. Then parts of the processing which needed to be repeated several times along the workflow were transformed into database stored procedures. Finally, different steps were combined under the hood of python script, which calls SQL scripts and database procedures. The general workflow consists of two main blocks: (1) generation of spatial features and finding of spatial relationships (Figure 11) and (2) extraction of geometry and tags information from spatial features of PostGIS into Lanelet2 XML elements and an HD map generation (Figure 12). Further subsections describe each part of the workflow in detail.

It is not necessary to perform these two parts together each time when HD map should be generated. Processed spatial features for one or several sites can be stored in the database, and an HD map can be generated on demand. The particular solution for storing and updating generated elements strategy is not a part of this thesis.

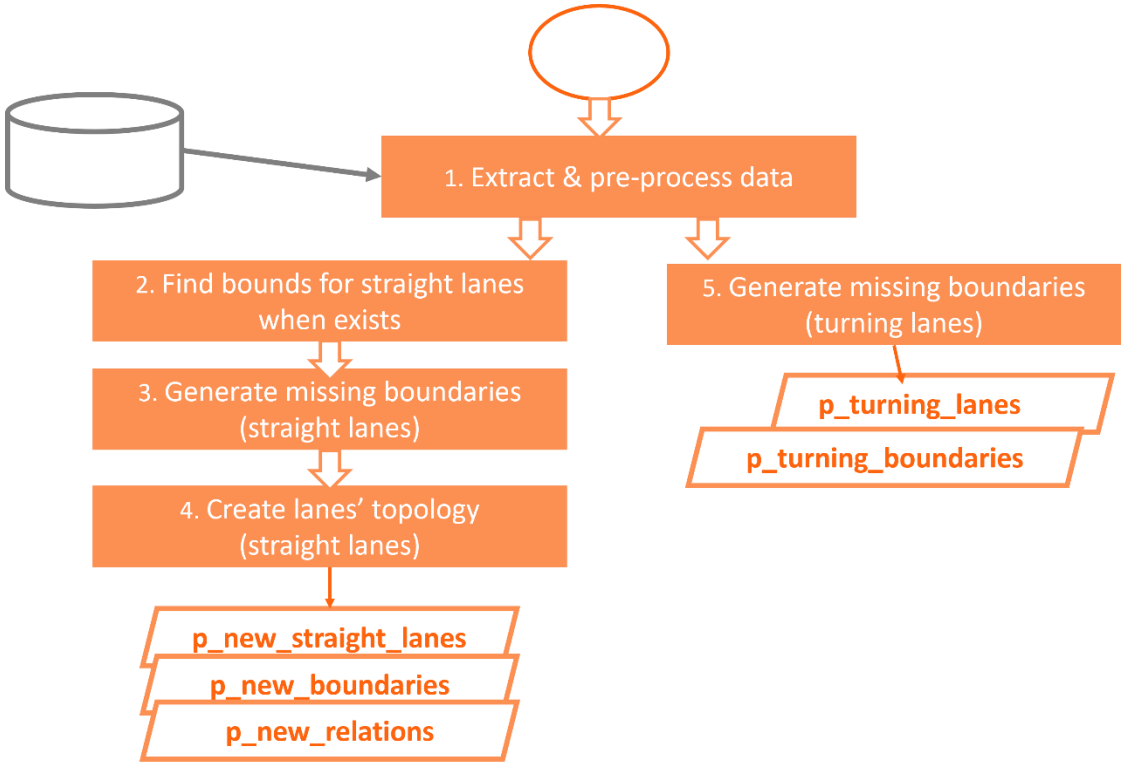


Figure 11. Generation of PostGIS layers for Lanelet2 map.

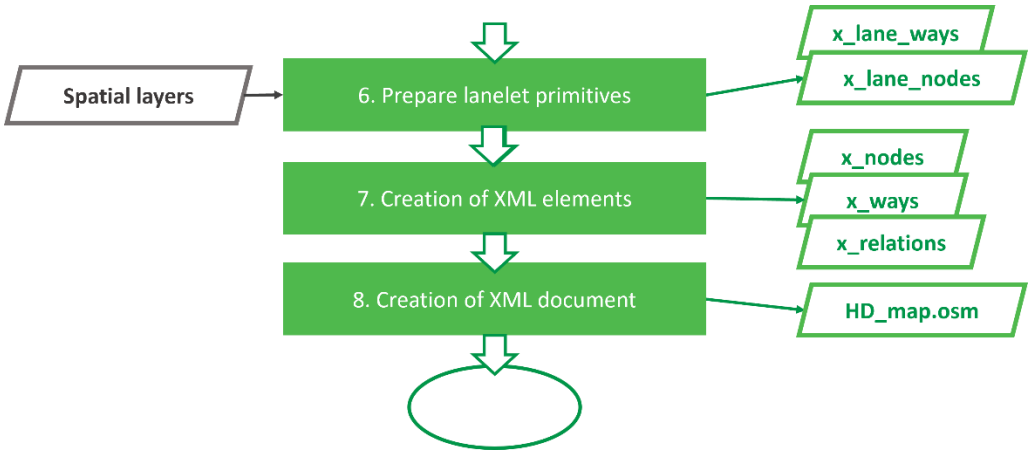


Figure 12. Decomposition of PostGIS spatial objects into Lanelet2 primitives and HD map generation.

The algorithm was developed and tested for the test area located in Tartu, next to the University of Life Sciences campus (Figure 13). The test area comprises a 500 metres long road section with two intersections of classical X-shape. The total length of lanes is 1.78 km.

Figure 13. The algorithm development area. Oproximetly 500 m road of Tallinna mnt. street next to University of Life Sciences in Tartu

4 Results

4.1 Database

The structure of the ADL map database is presented in Figure 14. Table *lanes* contains vehicle trajectories or lanes' centrelines digitised from imagery. Tables *lane_boundaries* and *road_markings* contain observable linear features of the driving environment. In some places, the hypothetical virtual bounds of the lane borders were inserted where they do not exist in the real world. Table *relations* contains the imaginable perpendicular linestring dividing lanes into segments with the same semantics – same speed limit, the same type of left and right bounds, which allow or prohibit lane changes or turns etc. These tables define the area on the road where the directed movement of vehicles is permitted, and direction is set by the lane linestring direction. The particular kind of lanes – *crosswalks* - are also stored in the *lanes* table. Movement in both directions is allowed for crosswalks. The area of crosswalks is generated later in target format converters by buffer function or by creating polygon topology with elements of *road_markings* and *lane_boundaries* tables.

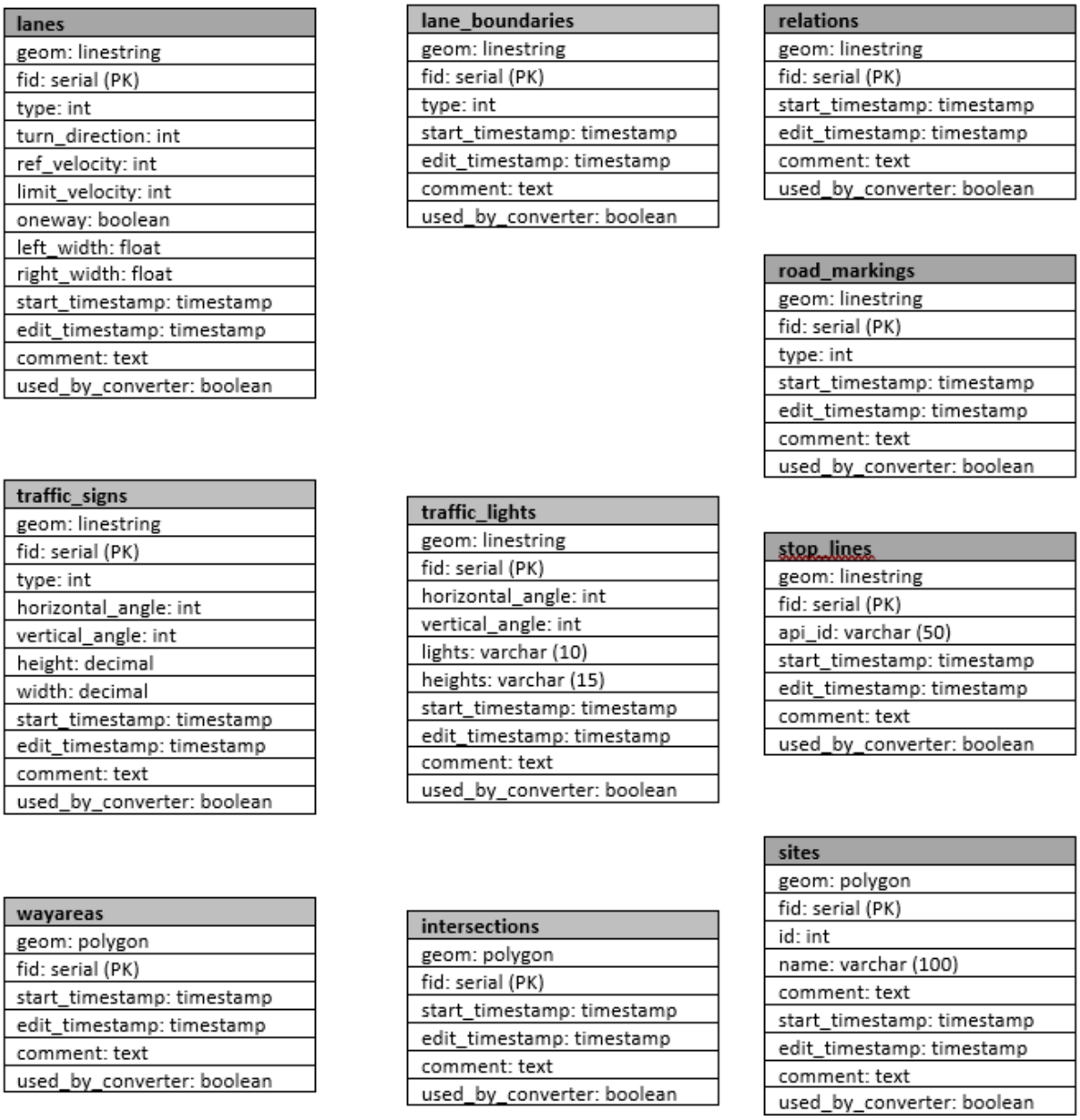


Figure 14. The structure of the hdmap_v0 database schema.

The next block of features – *tables traffic_sign, traffic_light and stop_lines* – are so-called regulatory elements in HD maps. They are associated with the end node of the lane centreline. Please notice that the geometry type of the traffic lights and signs is a directed linestring (vector), which allows us to associate it with the lane centreline to which they are effective.

Tables *intersections* and *wayareas* hold corresponding areas. Table *sites* stores sites of interests' bounding boxes or buffers, for example, Tartu_demo_route. Converters use site boundaries to extract corresponding features from the database.

All features have additional attributes to facilitate digitalisation work or handle inclusion/exclusion from converting. The attributes are *start_timestamp, edit_timestamp, comment, used_by_converter*. For example, some lanes are digitised but are under road construction.

4.2 Algorithm development for generation of HD map missing elements

4.2.1 Data extraction and pre-processing.

It selects the spatial extent for the map and clips the data out of the digitised map layers. The table *lanes*, which contains lane centrelines, is split into two: one for driving lanes and one for crosswalks. Because potential lane boundaries are stored in two different tables in the PostGIS (see section Result for database structure), the script combines them into one data layer, changing values for attribute *type* to avoid values' collision. Also, the merge of linestrings by attributes is performed to eliminate artificial splits of linestrings, for example, when too long elements are digitised as a sequence of several linestrings like several kilometres long road boundary in the countryside. The details of the algorithm are presented in Annex A.1.

4.2.2 Find borders for straight lanes when exists

Lanes in Lanelet2 have tags corresponding to the turning direction: 'straight', 'left' or 'right'. Corresponding attributes are added to lane centrelines during digitisation. But road intersections naturally have a complex interlacement of all possible straight and turning trajectories. The algorithm addresses the task of lane bounds generation step by step. First, straight lanes' left and right bounds are produced on the regular road and in intersection areas. Then the boundaries of turning lanes, which overlap but do not split the areas of the straight lanes, are produced. The step of processing straight lanes consists of two parts:

- Finding straight lane segments where both bounds exist
- Generating bounds for lane segments where one or both bounds are missing

The step of finding segments with boundaries consists of two essential procedures, repeated several times during the missing element generation and lane topology creation steps (section 4.2.4). The first procedure's concept is splitting lanes according to the candidate boundaries into lanes with uniform driving environment conditions – meaning that the type of boundary on the left and right sides does not change. For this purpose, the lane centreline is divided into approximately one meter (up to 1.2 meters) length (Figure 15 A and B) segments and over 3-metre perpendicular transects are created from the centre of the lane segment to the left and to the right. Then, the first (counting from lane centreline) intersection with potential boundaries is found, and type and fid attributes of the left and right boundaries are assigned to the little lane segment. These attributes are the basis for merging consecutive lanes with uniform driving environment conditions into one lane segment. Finally, the perpendicular transects are created from the start and end points of the newly constructed lanes, and linestrings of boundaries are split to make candidates for lane left and right bounds (Figure 15 C and D). Code steps are presented in Annex A.2

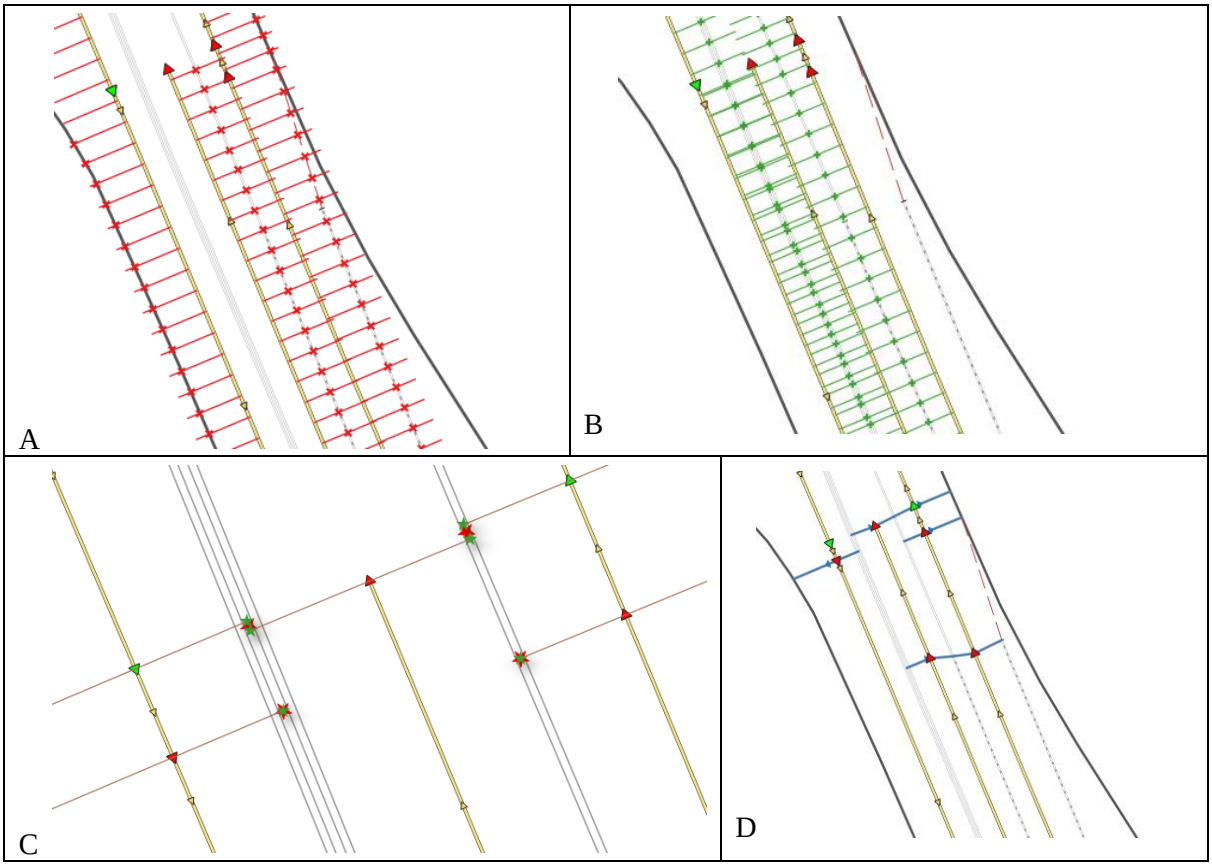


Figure 15. Illustrations for algorithm steps: A – right transects and their points of intersection with lane digitised boundaries; B – left transects and their points of intersection with lane digitised boundaries; C – snapping of the lane end node transects towards boundaries nodes; D – the creation of relation elements.

4.2.3 Generation of bounds for lane segments where one or both bounds are missing

At this step's beginning, lanes with missing boundaries are filtered into the temporary layer. First, the virtual boundaries are generated between the lanes using buffer approach (Figure 16.A and 16.B). Then generated lanes are put together with existing boundaries, and the procedure of step 2 for lane segmentation is repeated. Finally, lanes still without bounds on the left or right are filtered, and boundaries are created by creating offset linestrings from centreline (Figure 16.C and 16.D). Since the quality of the results of this step is vital for the rest of the procedures, the visual check on the screen is highly recommended. The details of the algorithm are presented in Annex A.3.

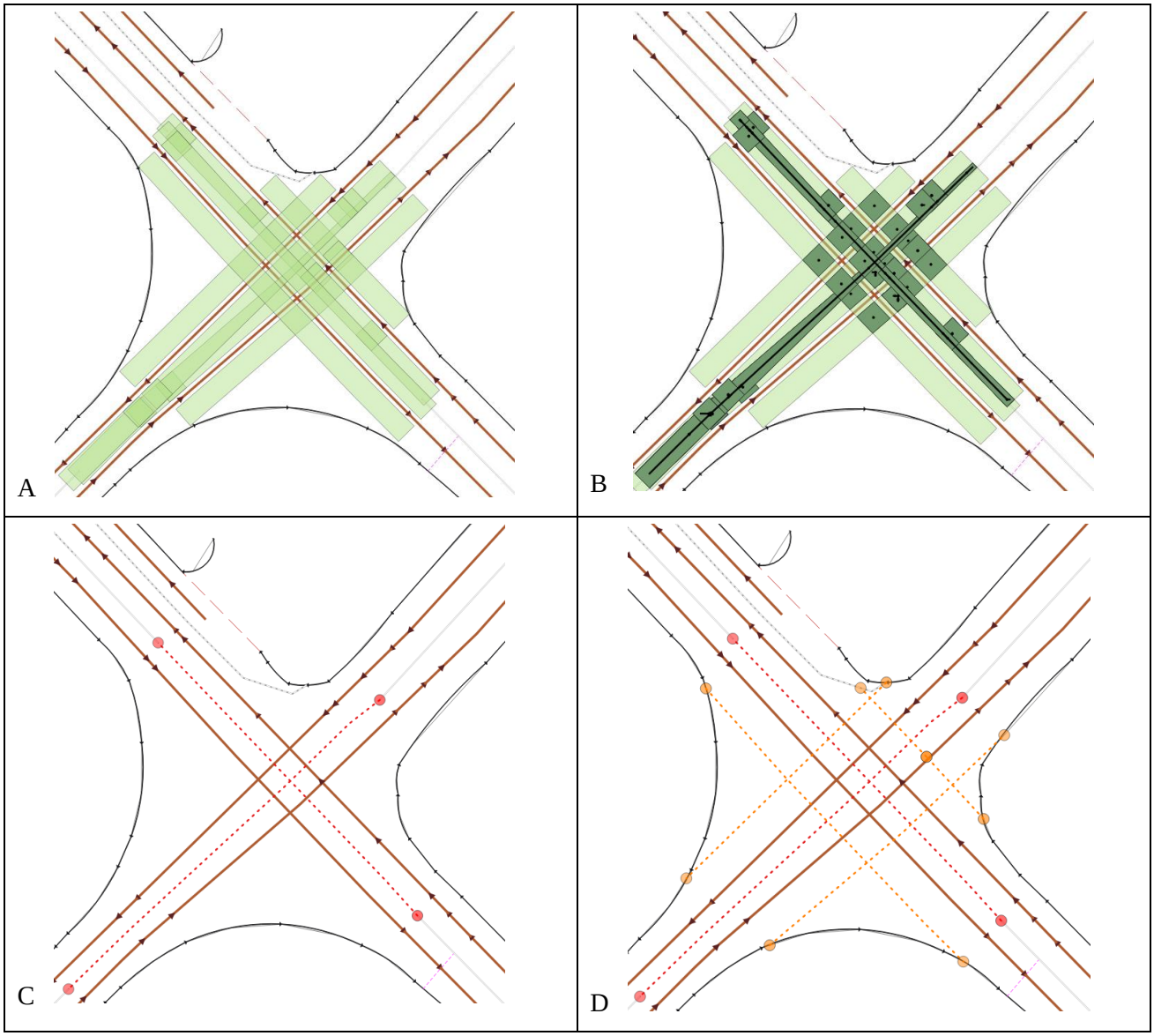


Figure 16. Generating bounds for lane segments where bounds are missing: A – generating buffers; B – intersecting buffers and creating centrelines inside of intersecting polygons; C – generated and snapped bounds between lanes, little circles represent search radius for snapping; D – generated by offset and snapped bounds (legend: dark brown – lane centrelines, light green – buffers, dark green – the intersection of buffers, black – polygon centreline, dashed red – generated bound, first step; dashed orange – the rest of generated bounds by offset)

4.2.4 Creation of lanelets topology.

According to the Lanelet2 format, a lanelet is defined by exactly one line string as the left and exactly one as the right boundary. In addition, the centre line or trajectory can be stored. The type of the bound, for example, dashed markings or curb, defines the kind of possible manoeuvre to make. Furthermore, the manoeuvre from one lane to another is only possible if the lanelets share the same bounding line.

Therefore, if the boundary on one side is changing - the curb is over, and the dashed marking line is starting – one lane should stop and another start. It causes the boundary splitting on the other side of the same lane, even if it is a continuous road boundary or marking. Finally, since boundaries are shared, breaking the boundary from one side by one lane calls splitting the adjacent lane. And so on…

Here the procedures written for step 2 for finding lane segments versus boundaries with homogeneous conditions are used repeatedly. The illustration of the process is presented in Figure 17.

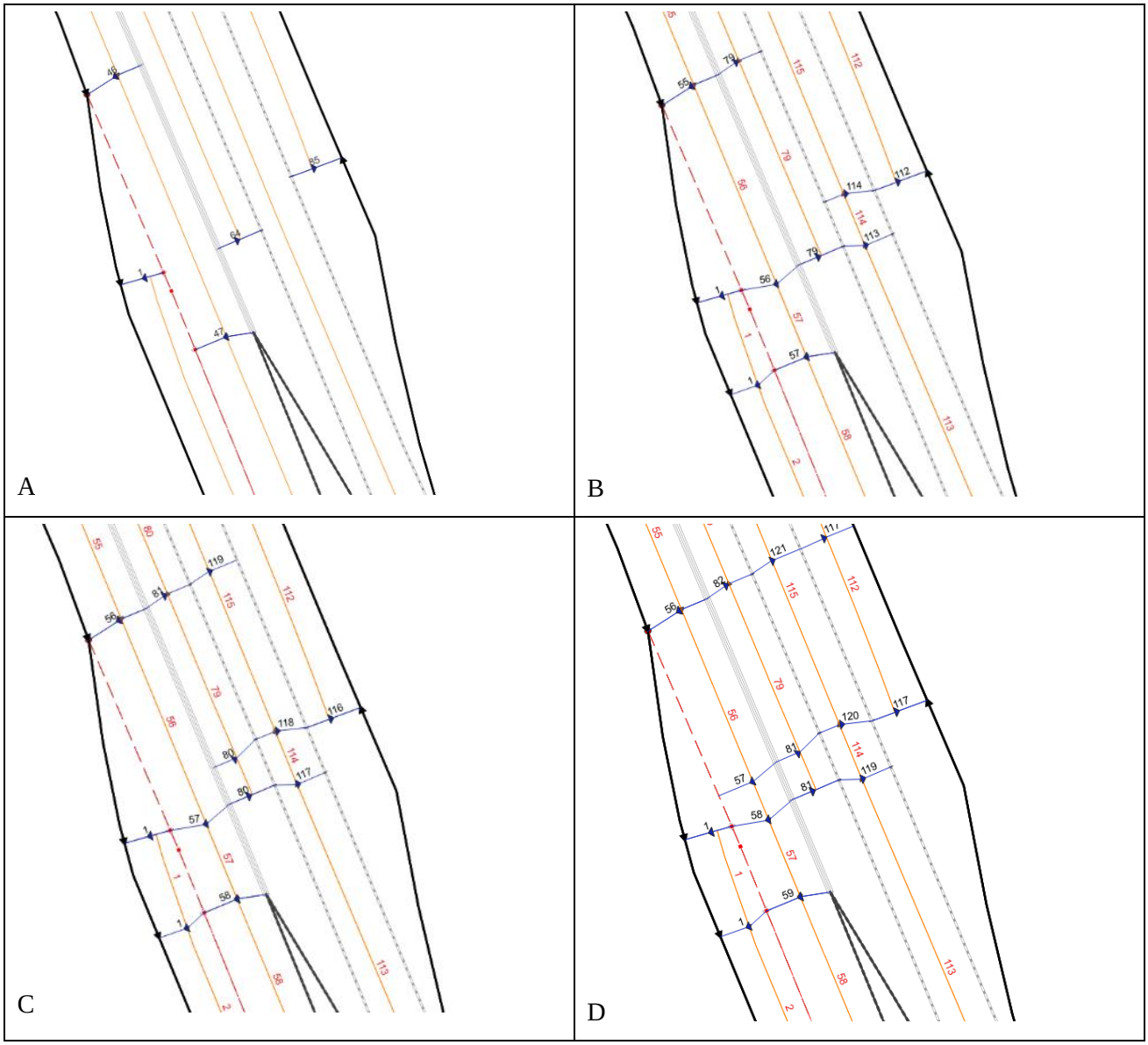


Figure 17. Four steps of splitting lane centrelines and boundaries into lanelets with the same driving conditions. The number of iterations depends on the number of adjacent lanes.

4.2.5 Generate missing boundaries for turning lanes.

Step 5 from Figure 11 uses turning lanes attribute right_width and left_width to create bounds as offset linestrings from the centrelines. The code uses the *relation* primitives to smooth the transition to the previous and following straight lanes: end and start nodes of the offset linestrings are substituted by nodes of *relation* linestrings (Figure 18). The code details are presented in Annex A.4.

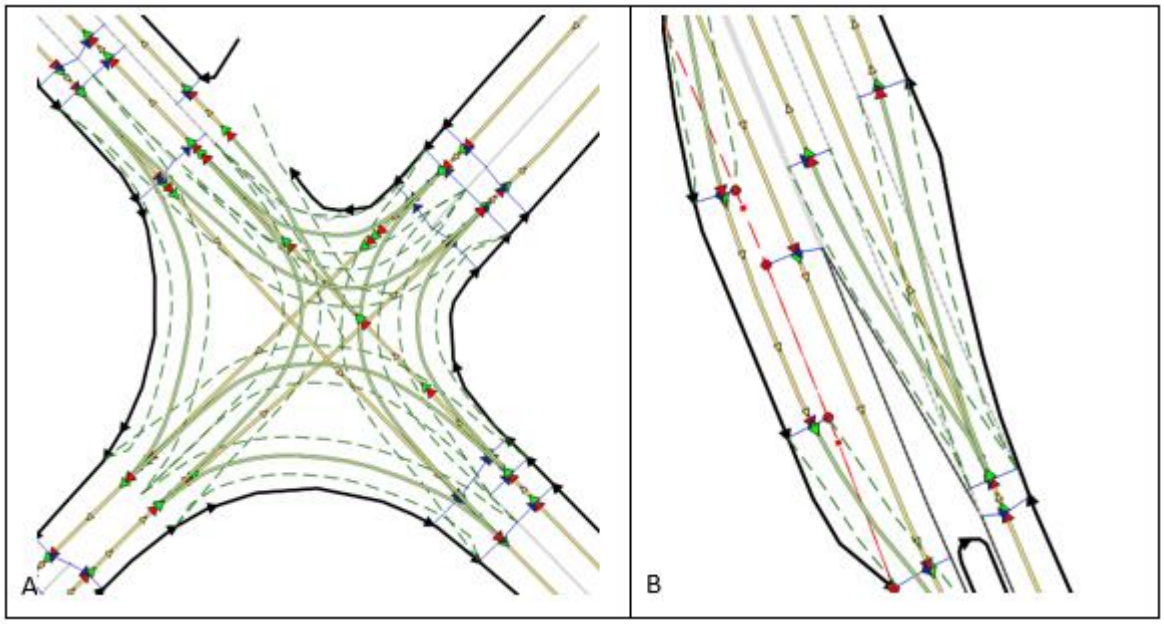


Figure 18. Creation of turning lanes boundaries' geometry. Green lines are turning lanes; green dashed lines are their bounds. A – intersection area; B – bus stops' pockets.

4.3 Converter for Lanelet2 HD map

As was mentioned above, to create the Lanelet2 HD map, the PostGIS geometries should be decomposed to produce lanelet primitives - *nodes*, *ways* and *relations*. Linestrings should be decomposed into *nodes* with geographic and, optional, projected coordinates. *Ways* should contain references to the *nodes* in sequential order and attribute tags. Further, the *relation* elements should be composed with references to the *way* lines and attribute tags on the relation level. The connectivity of two subsequent lanelets in Lanelet2 requires that the *ways* comprising these lanes should share the same *nodes*. Therefore, connected PostGIS linestrings should be decomposed when nodes are not overlapping at the same location as it takes place in PostGIS geometry, so the ID-s of the *nodes* should be unique. The same ID should be referring in all *ways* where the node participates. In the same fashion, the lanelet *relations* share *ways*, ensuring that manoeuvre can be performed from one lane to another if the separating *way* is of the proper type (dashed, dashed-solid markings, for example).

4.3.1 Preparing the lanelet primitives

This step prepares tables for XML elements. First, the table containing lanes, centerline and boundaries is created from two separate tables, and the unique fids are added. Second, the table of nodes, shared by two or more *ways*, is made, and distinct node geometries receive their unique IDs. Third, by intersection with linestrings, the references to the unique nodes are added to the linestrings table. Similarly, the ways of linestrings of crosswalks, stop lines, traffic lights and signs are generated. Finally, nodes representing poles, traffic light bulbs and traffic signs are added to the x_***_ways tables. The converter code steps are presented in Annex B.1

4.3.2 Creation of XML elements

This part of the algorithm makes use of the XML datatype in PostgreSQL. The combination of functions such as xmlelement(), xmlattribute(), xmlconcat() and xmlagg() can produce XML elements of different depths and store them in a database table.

Linestrings are decomposed into arrays of vertices, and each one becomes a *node* in the node table, except the first and the last. In the linestring table, the arrays of node references are produced for each linestring, where references to the unique shared nodes substitute the first and the last nodes. Code details are presented in Annex B.2

4.3.3 Creation of XML document

This step uses xml.etree.ElementTree library[35] to create the XML root element in the python script and psycopg2 library to access the PostgreSQL tables with the XML elements.

Next, if adding elevation to the coordinates is specified as a parameter, the height is obtained using 2D coordinates from the DEM of the Estonian National Land board. The python libraries rasterio and geopandas are used for this procedure. Finally, all three arrays containing XML elements for nodes, ways and relations are added to the XML root.

[35] https://docs.python.org/3/library/xml.etree.elementtree.html#module-xml.etree.ElementTree

5 Testing

After establishing the workflow on the site presented in Figure 13, two more testing sites (Figure 19) were selected to estimate the time for the algorithm and manual post-processing. Also, the goal was to examine how the proposed solution performs in addressing driving environments of various complexity: oneway roads, roundabouts, and intersections with complex configurations.

Figure 19. Location of the testing sites in Tartu. Green – Lai-Jakobi-Kroonuaia, orange – Narva-Roosi-Puistee.

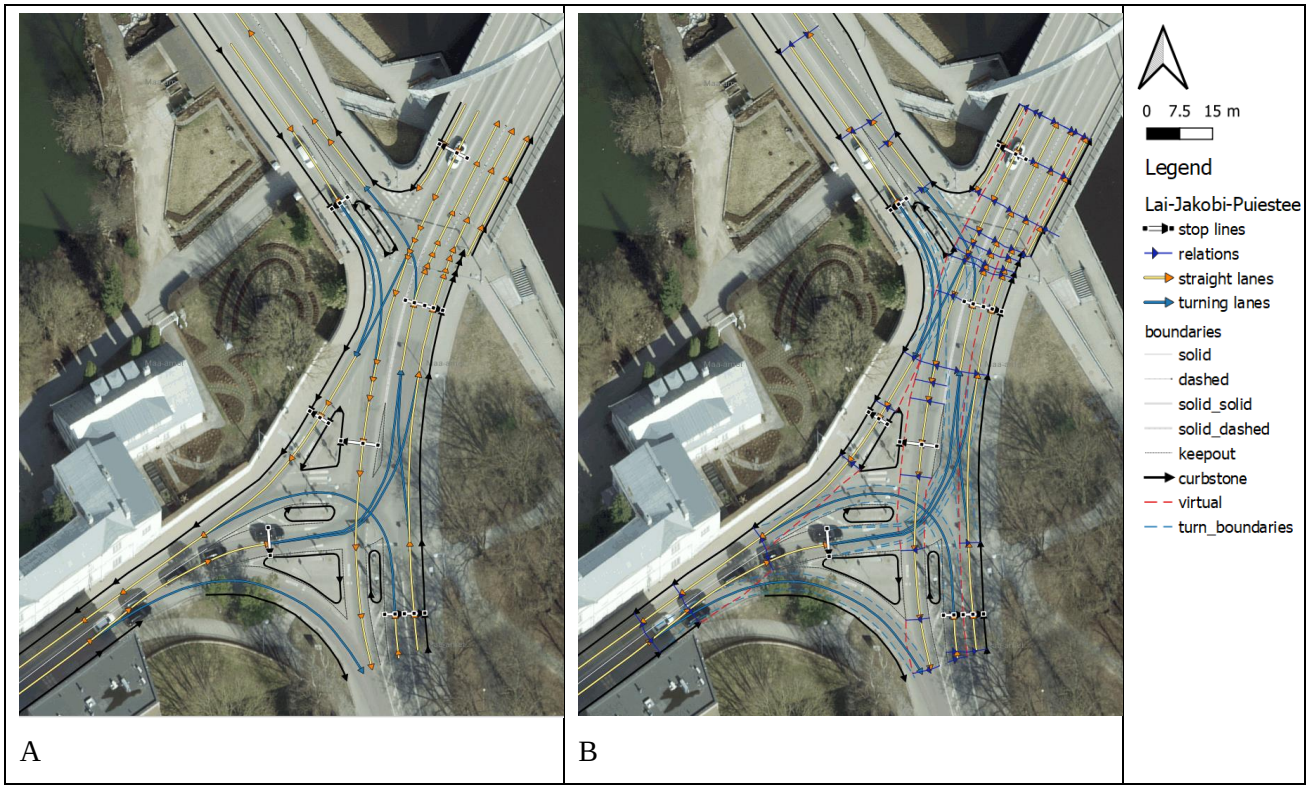
The **Lai_Jakobi_Kroonuaia** selected site has a proximal length of 2.25 kilometres of streets, 3.746 km of lanes and 22 traffic lights (green in Figure 19). It includes Lai str. – Vabaduse ave. - Emajõe str. intersection with complex configuration (Figure 20). Table 6 estimates the time spent for HD creation with the proposed solution and the time needed for digitising all elements. Table 6 composed though to make estimates comparable with estimates of [7] presented as tables 2 and 3 of this thesis. Table 6 consists of two parts: (1) feature extraction and (2) efforts for composing the lanelets. For a 2-kilometres-long rural test track, [7] estimated 60 hours for automated extraction of features and 38 hours for lanelets post-processing. The paper mainly concentrated on the workflow's first part and specified fewer details of post-processing.

In the proposed solution, a minimal set of features was digitised on the screen; therefore, the assessments are the same for the semi-automated and fully manual processes. Still, our results for feature extraction for approximately the same road length are better than the results of [7]. And they are considerably better for automated post-processing. Moreover, considering total time, our solution ends up with a 45% reduction of time compared to manual digitalisation for the entire workflow.

Table 6. Time spent for HD map and comparison with fully manual solution, Lai-Jakobi-Kroonuaia test site.

	Quantity	Proposed solution, hh:mm:ss	Manual digitalization, hh:mm:ss
Features extraction via digitising *			
Driving trajectories digitising	3.746 km	04:00:00	04:00:00
Lane boundaries (incl. curbstones, verges, fences) and road markings	6.522 km	06:45:00	06:45:00
Traffic lights and signs	22	05:30:00	05:30:00
Time spent on digitising		**16:15:00**	**16:15:00**
Composing the lanelets			
Create virtual lines to create correct lanelets + with editing	3.132 km	00:00:19 01:15:00	03:15:00
Split lanes to sections and create relation elements + with editing	169	00:00:39 02:30:00	11:15:00
Compose lanelets from line boundaries / regulatory elements + with editing	152 / 22	00:00:21 02:30:00	10:10:00
Time spent on lanelets composing		**6:16:19**	**24:40:00**
Total efforts		**22:31:19**	**40:55:00**
Efforts per 1 km of lanes		**06:00:44**	**10:55:22**

* Including setting Lanelet2-specific type tags as attributes



A B

Figure 20. Lai str. – Vabaduse ave. - Emajõe str. Intersection. A – after digitising, B – after features semi-automate generation

The **Narva-Roosi-Puiestee** site has a proximal length of 3.49 kilometres of streets, 9,92 kilometres of lanes and 21 traffic lights (orange in Figure 19). It includes the roundabout next to the Delta building.

It includes the roundabout next to the Delta building (Figure 21). Table 7 estimates the time spent for HD creation with the proposed solution and the time needed for digitising all elements. It turns out that with the automation of the most tedious exercise of lane splitting and establishing the relations, the time required for composing lanelets may drop by 65%.

Table 7. Time spent for HD map and comparison with fully manual solution, Narva-Roosi-Puiestee test site.

	Quantity	Proposed solution, hh:mm:ss	Manual digitalisation, hh:mm:ss
Features extraction via digitising *			
Driving trajectories digitising	9.922 km	10:30:00	10:30:00
Lane boundaries (incl. curbstones, verges, fences) and road markings	9.669 km	10:00:00	10:00:00
Traffic lights and si	21	05:15:00	05:15:00
Time spent on digitising		**25:45:00**	**25:45:00**
Composing the lanelets			
Create virtual lines to create correct lanelets + with editing	7.700 km	00:00:44 01:45:00	08:00:00
Split lanes to sections and create relation elements, pieces + with editing	524	00:04:54 04:00:00	35:00:00
Compose lanelets from line boundaries and regulatory elements + with editing	495 21	00:00:31 04:00:00	33:00:00
Time spent on lanelets composing		**09:51:09**	**76:00:00**
Total efforts		**35:36:09**	**101:45:00**
Efforts per 1 km of lanes		**03:35:17**	**10:15:18**

Concerning the amount of human intervention, it should be mentioned that performance is very good with standard road structures where plain street stretches meet in T-shape or X-shape intersections. Little or none of the editing of results is needed. Also, the algorithm is performing well in areas of complex street intersections, but more human assistance is required. The statistics obtained for comparison of the proposed solution with a fully manual process of map elements digitisation demonstrated a considerable reduction in time and human efforts in HD map creation.

6 Conclusions

HD maps are essential in AD research by providing data about the static driving environment for vehicle localisation, route planning, perception and manoeuvre decision-making. Hence, the quest to propose the most efficient mapping pipeline has become a research topic by itself. The general motivation behind this thesis was to present the database structure and the effective workflow for the semi-automatic generation of HD map elements. More specifically, the task was to find and apply the PostGIS functionality to establish a suitable workflow for the ADL that would minimize the amount of manual work as much as possible (in the context of Lanelet2 map creation).

Firstly, the ADL's previous mapping efforts were overviewed, and the database design for storing spatial data required in various HD map formats was proposed. Next, the open-source PostGIS solution was chosen for the database engine, considering its capabilities in storing and processing spatial data efficiently. Further on, datasets generated during previous mapping campaigns were transformed into the new data structure and inserted into the database capable of serving the needs of Autoware Vector Map and Lanelet2 map converters.

Secondly, existing data was used to develop algorithms for the semi-automatic generation of so-called missing elements – spatial features needed to construct proper lanelets, fundamental primitives in Lanelet2 format. These features include left and right boundaries of the lanelet when they can not be extracted via digitising or it is too time-consuming to insert them manually. Thus, the lanelet centrelines were split into little segments, and each segment was tested on the existence of left and right boundaries at certain proximity. The type of road boundary or marking was assigned to the segment. Then, the segments were merged back corresponding to the attributes of found boundaries, and centrelines with missing boundaries on the left and/or right side were separated for further processing. The buffering and linestring offsetting functionalities of PostGIS were applied to create virtual lane boundaries; the algorithm analysed some potential errors and produced their list, and the step was followed by an on-the-screen check.

Thirdly, the Lanelet2 requirements for the shared bounds and automated finding of semantic relationships between primitives comprising a lanelet were addressed. The same method for splitting centrelines into segments and splitting boundaries according to the new set of centrelines was applied repeatedly until the numbers of centrelines and shared boundaries were stabilised. The key step here is producing spatial elements, which we call 'relations' that establish the relationships between a centreline and its bounds in the database.

Fourthly, the data converter for HD maps in Lanelet2 format was proposed. It transforms spatial primitives of PostGIS, which are points, linestrings and polygons, into primitives of OSM and Lanelet2, which are nodes, ways and relations. Due to PostgreSQL's capability to store the XML datatype, elements for each node, way and relation were created in the database. Further on, the database was accessed from the python script, where XML root was made, and the following elements were loaded from the database into the root to create a proper Lanelet2 file.

The workflow was developed using a test site with a road stretch of 500 meters and two classical X-shape intersections. Further, it was tested on two more sites: Lai-Jakobi-Kroonuaia and Narva-Roosi-Puiestee. It helped to assess the algorithm's robustness in various driving environments and improve its performance. It turned out that the algorithm performs very well with standard road structures where plain street stretches meet in T-shape or X-shape intersections. In this driving environment, the algorithm can produce good results based only on vehicle trajectories and digitising minimum road markings and crucial road edges. Also, the algorithm is capable of performing well in areas of complex street intersections such as Lai str. – Vabaduse ave. - Emajõe str. or ring road next to the Delta building, but some human help is needed. Still, due to very diverse road situations in city centres, it is virtually impossible to correctly set all thresholds and search distances to tackle all possible combinations of road elements. Therefore, a thorough check of intermediate results is necessary, which can be facilitated by the automated creation of potential error lists.

The statistic obtained for comparison of the proposed solution with a fully manual process of map elements digitisation demonstrated a considerable reduction in time and human efforts in HD map creation. Due to the PostGIS functionality, the data geoprocessing was impressively fast. Depending on road structure complexity, the amount of effort needed per one kilometre of lanes can drop by 45% to 65%.

Acknowledgement

This work was funded by a cooperation project between the University of Tartu and Bolt "Applied Research on Development of Autonomous Driving Lab for Level 4 Autonomy".

I would like to express my thanks to my supervisors and ADL employers for giving me the most valuable support possible.

I'm very grateful to the supervisors also for the fruitful discussions throughout the writing the thesis.

References

[1] R. Liu, J. Wang, and B. Zhang, "High Definition Map for Automated Driving: Overview and Analysis," *J. Navig.*, vol. 73, no. 2, pp. 324–341, Mar. 2020, doi: 10.1017/S0373463319000638.

[2] B. Ebrahimi Soorchaei, M. Razzaghpour, R. Valiente, A. Raftari, and Y. P. Fallah, "High-Definition Map Representation Techniques for Automated Vehicles," *Electron. 2022, Vol. 11, Page 3374*, vol. 11, no. 20, p. 3374, Oct. 2022, doi: 10.3390/ELECTRONICS11203374.

[3] E. Sepp, "Creation of High Definition Map for Autonomous Driving," *Univ. Tartu, Master Thesis*, p. 84, 2021.

[4] A. Tampuu, T. Matiisen, M. Semikin, D. Fishman, and N. Muhammad, "A Survey of End-to-End Driving: Architectures and Training Methods," *IEEE Trans. Neural Networks Learn. Syst.*, vol. 33, no. 4, pp. 1364–1384, Apr. 2022, doi: 10.1109/TNNLS.2020.3043505.

[5] J. Ziegler *et al.*, "Making Bertha Drive—An Autonomous Journey on a Historic Route," *IEEE Intell. Transp. Syst. Mag.*, vol. 6, no. 2, pp. 8–20, 2014, doi: 10.1109/MITS.2014.2306552.

[6] P. Bender, J. Ziegler, and C. Stiller, "Lanelets: Efficient map representation for autonomous driving," in *2014 IEEE Intelligent Vehicles Symposium Proceedings*, Jun. 2014, pp. 420–425. doi: 10.1109/IVS.2014.6856487.

[7] K. Rehrl, S. Groechenig, T. Piribauer, R. Spielhofer, and P. Weissensteiner, "Towards a standardised workflow for creating high-definition maps for highly automated shuttles," *J. Locat. Based Serv.*, vol. 16, no. 2, pp. 119–151, Apr. 2022, doi: 10.1080/17489725.2022.2059587.

[8] F. Poggenhans *et al.*, "Lanelet2: A high-definition map framework for the future of automated driving," *IEEE Conf. Intell. Transp. Syst. Proceedings, ITSC*, vol. 2018-Novem, pp. 1672–1679, Dec. 2018, doi: 10.1109/ITSC.2018.8569929.

[9] K. Wong, Y. Gu, and S. Kamijo, "Mapping for Autonomous Driving: Opportunities and Challenges," *IEEE Intell. Transp. Syst. Mag.*, vol. 13, no. 1, pp. 91–106, 2021, doi: 10.1109/MITS.2020.3014152.

[10] Y. Kang and A. Magdy, "HiDaM: A Unified Data Model for High-definition (HD) Map Data," in *2020 IEEE 36th International Conference on Data Engineering Workshops (ICDEW)*, Apr. 2020, pp. 26–32. doi: 10.1109/ICDEW49219.2020.00-11.

[11] W. N. Tun, S. Kim, J.-W. Lee, and H. Darweesh, "Open-Source Tool of Vector Map for Path Planning in Autoware Autonomous Driving Software," in *2019 IEEE International Conference on Big Data and Smart Computing (BigComp)*, Feb. 2019, pp. 1–3. doi: 10.1109/BIGCOMP.2019.8679340.

[12] I.-H. Choi and E.-M. Kim, "Automatic Construction of Road Lane Markings Using Mobile Mapping System Data," *Sensors Mater.*, vol. 34, no. 7, p. 2625, Jul. 2022, doi: 10.18494/SAM3872.

[13] J. Fricke, C. Plachetka, and B. Rech, "Towards Knowledge-based Road Modeling for Automated Vehicles: Analysis and Concept for Incorporating Prior Knowledge," in *2021 IEEE Intelligent Vehicles Symposium Workshops (IV Workshops)*, Jul. 2021, pp. 49–56. doi: 10.1109/IVWorkshops54471.2021.9669220.

[14] F. Tsushima, N. Kishimoto, Y. Okada, and W. Che, "CREATION OF HIGH DEFINITION MAP FOR AUTONOMOUS DRIVING," *Int. Arch. Photogramm. Remote Sens. Spat. Inf. Sci.*, vol. XLIII-B4-2, pp. 415–420, Aug. 2020, doi: 10.5194/isprs-archives-XLIII-B4-2020-415-2020.

[15] W. Xia, "High-definition map creation and update for autonomous driving," *KTH R. Inst. Technol. Master Thesis*, p. 76, 2021.

[16] O. Shipitko, A. Kabakov, A. Grigoryev, and K. Smirnov, "Road Markings and Road Edges Mapping With Inverse Visual Detector Model," in *2022 IEEE 95th Vehicular Technology Conference: (VTC2022-Spring)*, Jun. 2022, pp. 1–6. doi: 10.1109/VTC2022-Spring54318.2022.9860752.

[17] P. Fischer, S. M. Azimi, R. Roschlaub, and T. Krauß, “Towards HD Maps from Aerial Imagery: Robust Lane Marking Segmentation Using Country-Scale Imagery,” *ISPRS Int. J. Geo-Information*, vol. 7, no. 12, p. 458, Nov. 2018, doi: 10.3390/ijgi7120458.

[18] P. Jende, F. Nex, M. Gerke, and G. Vosselman, "A fully automatic approach to register mobile mapping and airborne imagery to support the correction of platform trajectories in GNSS-denied urban areas," *ISPRS J. Photogramm. Remote Sens.*, vol. 141, pp. 86–99, Jul. 2018, doi: 10.1016/j.isprsjprs.2018.04.017.

Annexes

Annex A Algorithmfor generation of HD map missing elements

A.1 Data extraction and pre-processing.

The algorithm is as follows (the numeration of steps is according to Figure 11):

1.1 Select the site polygon and create an area of interest (AOI) table

- Select the site polygon from *public.sites* into *clip_table*

1.2 Clip, filter lanes and crosswalks, merge lanes by attributes and split by stop lines:

- Clip data by selected extent from the public schema (*lanes* and *stop_lines*, see database section in the Results) into working schema;
- Merge lanes with the same attributes, in case of too long elements were digitised as a sequence of several linestrings, for example, several kilometres long road in the countryside**;**
- Lanes shall not be merged if they intersect with the stop line! Therefore, split lanes from the previous step by stop lines;

1.3 Clip and combine into one table all lane boundaries

- clip *lane_boundaries* and *road_markings* tables by AOI
- change the *type* attribute codes to avoid values collision between elements coming from different original tables;
- put together in one table;
- merge linestrings with the same attributes, in case too long elements were digitised as a sequence of several linestrings, then split them by stop lines.

A.2 Find borders for straight lanes when exists

The algorithm is as follows (the numeration of steps is according to Figure 11):

2.1 Finding straight lane segments where both bounds exist Split lanes according to existing boundaries:

- densify lanes for each linestring in the lanes table by creating vertices over a 1.2 m distance;
- create nearly 1.2 m segments using vertices;
- filter only 'straight' lanes;
- create perpendiculars (transects) to segments from the segment's middle point to the left and to the right from the lane segment;
- find the closest to the lane segment intersection (if any) between transect and digitised boundaries;
- assign attributes *l_type*, *r_type* and *l_fid*, *r_fid* of the boundary to the lane segment;
- aggregate lane segments by same *type* and *fid* attributes on both sides into the new lane with uniform driving environment condition;

2.3 Splitting boundaries according to new lanes, procedure proc_new_boundaries_rel, output p_new_boundaries and new_relations.

- create perpendiculars (transects) to segments from the start and end point of merged lanes to the left and to the right;
- find the closest to the lane intersection between transects and boundaries;
- if there the start or end node of the boundary closer than 0.6 m to the transect (half of the length of elementar segments which is 1.2 m) snap the transect towards the boundary start or end node (Figure 17.C);
- split boundaries' linestrings by transects;
- create relation linestrings (Figure 17.D);

A.3 Generating bounds for lane segments where one or both bounds are missing.

3.1. Find lane bounds between lanes:

- filter lanes without boundaries (*l_type* or r_type is null) and create offset from the lane centreline using right_width and left_width attributes;
- create buffers around offset lines (0.7 m);
- intersect buffer polygons and find centrelines of buffers' intersections (Figure 19.A and 19.B);
- select only intersections of Polygons type, filter by area (power(buffer_size * 2, 2));
- find centrelines of polygons;
- clean centerlines (snap with tolerance 0.7 m);
- merge centerlines before insertion and snap them towards digitised boundaries linestrings;
- create *p_generate1* table, which is the first intermediate result of missing bounds' generation (Figure 19.C);

NB! ST_ApproximateMedialAxis function for polygons' centrelines is a part of PostGIS extension posgis_sfcgal

3.2 Find the rest of the lanes without bounds:

- put generated lines together with *p_new_boundaries* from the previous step → *n_bnd*
- recheck lanes against *n_bnd* boundaries, and find lanes which still don't have boundaries on the left or the right;
- if they are, put them into *_candidates* table;

3.3 Generate new boundaries with offset:

- on the left side, if *l_type* is null, offset with distances from *left_width;*
- on the right side, if *r_type* is null, offset with a fixed distance of 3.0 m;
- snap the resulting lines towards the previous set of boundaries;
- create *p_generate2*, which is the second intermediate result of missing bounds' generation;
- put together p_generate1 and p_generate2 into one table (Figure 19.D),
- Put together digitised and generated lane bounds together for further processing;
- After the last step, the list of potential errors is created for visual check and editing (small dangling linestings, line islands, improper snapping);

A.4 Generate missing boundaries for turning lanes.

5.1 Find start and end nodes

- For each turning lane starting point, find the *relation* with which it intersects, find the geometry of the *relation* start and end points, and add to the turning lane table;
- For each turning lane end point, find the *relation* with which it intersects, find the geometry of the *relation* start and end points, and add to the turning lane table;

5.2 Create left and right bounds for turning lanes by offset using left_width and right width attribute

5.3 Update original offset linestings with end and start points of relations

5.4 Create errors list: check if generated boundaries intersect with digitised hard boundaries such as curbs, road border, guard_rail, fence or wall. Eliminate / correct errors on the screen. Also the geometry of some boundaries can be corrected to be smoother.

Annex B Converter for Lanelet2 HD map

B.1 Prepare lanelet primitives

The algorithm is as follows (the numeration of steps is according to Figure 12):

6.1 Create an *x_lane_ways* table**:**

- put together in one geometry table all lanes elements: trajectories which are presented by centreline and left/right bounds;
- add linestrings from road borders and markings which are not bounds for any lanes;
- add turning lanes' centerlines;
- add turning lanes' left/right bounds;
- create a Look-up-table (LUT) for lanes which contain lanelet id, centerline fid and fids of the bounds.

6.2 Create an *x_unique_nodes* table for nodes which are shared between two or more lines:

- make a table of ways of start and end points; find unique geometries;
- return references to unique node IDs to x_lane_ways as attributes.

6.3 Prerare regulatory elements and crosswalks:

- using geospatial joins to put together Look-up-table for regulatory elements (*x_regulatory*) vs participating elements FIDs and associate them with x_lane_ways.ID;
- create linestrings to x_stopline_ways (nodes are already in the shared nodes table);
- create *x_traffic_light_ways, x_bulb_ways, traffic_sign_ways* for ways; nodes geometry is added to the same tables
- find bounds of crosswalk & bike_crossings *ways* using buffers for searching in road_markings table linestrings *type ='zebra_marking' OR 'bike_marking'*.

B.2 Creation of XML elements

The algorithm is as follows (the numeration is according to Figure 12):

- 7.1 Create a table for XML nodes x_point_seq_xml
- insert dumped vertecies from x_lane_ways linestrings with coordinates, lane ID and sequence number;
- add dumped linestring vertexes from x_crosswalk_ways with coordinates, lane ID and sequence number;
- substitute way start and end point with ID and geometry from x_unique_nodes
- produce geom_4326 column with geographic coordinates transformed from projected coordinates
- produce node XML element and the reference to the node, for example <nd ref=' 10000042'>

7.2 Create Lanelet2 ways XML elements table → *x_ways*

- produce way XML elements for lanelet centerline and bounds from the sequences of nodes and add tags.
- add stop lines way xml elements
- add ways from *x_traffic_light_ways, x_bulb_ways, traffic_sign_ways*

7.3 Create XML relations → *x_relations*

- relation XML elements using lanes LUT table, add tags
- relation XML elements using regulatory elements LUT table, add tags

7.4 Add nodes from *x_traffic_light_ways*, *x_bulb_ways*, *traffic_sign_ways* tables

License

Non-exclusive licence to reproduce thesis and make thesis public

I, Valentina Sagris,

(author's name)

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to reproduce, for the purpose of preservation, including for adding to the DSpace digital archives until the expiry of the term of copyright,

Semi-automatic generation of Lanelet2 maps for autonomous driving,

(title of thesis)

supervised by Edgar Sepp and Tambet Matiisen.

(supervisors' name)

2. I grant the University of Tartu a permit to make the work specified in p. 1 available to the public via the web environment of the University of Tartu, including via the DSpace digital archives, under the Creative Commons licence CC BY NC ND 3.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright.

3. I am aware of the fact that the author retains the rights specified in p. 1 and 2.

4. I certify that granting the non-exclusive licence does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Valentina Sagris

09/03/2023