David OBERLERCHNER, BSc

Road Data-Modelling for Autonomous Driving

MASTER'S THESIS

to achieve the university degree of

Master of Science

Master's degree program: Geospatial Technologies

submitted to

Graz University of Technology

Supervisor

Ass.Prof.Dipl.-Ing. (FH) Dr.techn. Johannes SCHOLZ

Institute of Geodesy

in cooperation with

AVL List GmbH

Graz, September 2019

AFFIDAVIT

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used. The text document uploaded to TUGRAZonline is identical with the present master thesis.

Date	Signature

Abstract

Autonomous driving is one of the most challenging research topics for the automobile industry. Autonomous vehicles are equipped with several sensor techniques. Cameras, RADAR as well as LIDAR support vehicles in autonomous driving. Although these sensors are able to detect lanes, road objects and objects near the road, they are not capable of interpreting specific actions. Therefore, a knowledge representation of the data is necessary in order to describe the driving environment. Ontologies are frameworks for knowledge representations of the real world, which consists mainly of properties and relations between classes and data. Consequently, an ontology has been constructed, which describes the road in the driving environment. The ontology represents knowledge derived from open source data, like OpenStreetMap, which is the most ambitious collection of spatial data. Further simulation tasks for autonomous driving presume an adequate road network. Hence, OpenDRIVE is a common specification for describing road network data and a standard for different driving simulators.

Keywords

OSM road data, curvature, curve, ontology, road data model, OpenDRIVE

Contents

List of Figures

Abbreviations

AI	Artificial Intelligence
API	Application Programming Interface
CityGML	City Geographic Markup Language
DL	Description Logics
DLR	German Aerospace Center
DTD	Document Type Definition
FME	Feature Manipulation Engine
GI	Geographic Information
GIS	Geographic Information System
GTFS	General Transit Speed Specification
HTML	Hypertext Markup Language
ID	Identifier
IRI	Internationalized Resource Identifier
OSM	OpenStreetMap
OWL	Web Ontology Language
RDF	Resource Description Format
SUMO	Simulation of Urban Mobility
UML	Unified Modeling Language
UNA	Unique Name Assumption
URI	Uniform Resource Identifier
VTD	Virtual Test Drive

W3C	World Wide Web Consortium
XML	Extensible Markup Language
2D	2-dimensional

1 Introduction

1.1 Motivation and Problem Definition

The availability of knowledge-based road data for autonomous driving is a promising and an essential task. When using road data as a map base the identification of each attribute from the roads can be retrieved. However, the meaning of the attributes is not interpretable for machine-readable systems and the representation of knowledge is necessary. Some data formats like OpenDRIVE are limited in their knowledge representation on a few attributes and not all the attributes can be represented sufficiently. This implies that road data hold a lot of basic attributes, like speed, lane number, road names etc. However, there is no identification of the curvature and therefore the detection of a curvy segment. Nevertheless, there are limited road models available for such a task.

Therefore, the usage of an ontology is preferred to solve this problem. A road in the first instance is described by its geometry, mostly in form of a polyline, and such a road often contains additional describing attributes. Among those attributes, which exactly define the characteristics of a road segment, the meaning of the horizontal curvature of road segments plays a major role in contrast to the vertical curvature. The information about the horizontal curvature is necessary to determine the frequency of curves within a road. This task can be easily solved by generating an ontology for spatial road data.

In order to create an adequate ontology for road data from specific test areas all over the world suitable data with a high geographical coverage must be used. The availability of high- quality spatial data is often limited and cost-intensive. The usage of free and open source data may be an effective alternative. The best known data are OpenStreetMap (OSM), which cover a huge part of the world on a high level of detail. Therefore, they are ideal candidates to implement a road data model.

However, the knowledge representation of the road data is not the only interesting task, also the further usage in a simulation network might be significant. There are a few data formats that can be used within simulation tools. One of the most common data formats for simulation purposes is represented by OpenDRIVE. This data format encompasses some parameters that can be retrieved from OSM as data source. The transformation into OpenDRIVE is a challenging endeavor, as there are hardly any tools available to automate this complex transformation. Due to the widespread usage of OpenDRIVE for simulation applications the transformation of road data into OpenDRIVE is of grand importance anyway.

1.2 Goal and Research Question

This thesis is divided into three main parts that relate to the calculation of the curvature, to the generation of a road data model and to the transformation into the OpenDRIVE format. For the implementation of a road data model the usage of open source data is favored. Therefore, a huge range of open source platforms offers suitable spatial data. OSM is maybe the fastest and most efficient way to gain open source road data all over the world. In addition, this model is enriched with many attributes to derive descriptive road data.

- Are OSM data applicable to knowledge representation and for transformation into the OpenDRIVE format, and - if so - which attributes from the OSM data might be useful to enrich road data models?

The horizontal curvature of a road is an additional feature for describing a road segment. Unfortunately, it is not described within the OSM data. This task can be accomplished by creating an adequate ontology.

- Which approach might be the most effective to calculate the horizontal curvature of a road?
- How can we describe that a road segment is part of a curve by using an ontology?

The creation of an ontology is an essential element to enable autonomous driving. Consequently, the usage of road data assumes an output which is usable for simulation purposes.

- How can we transform digital road data into the OpenDRIVE format for simulation purposes?

1.3 Related Work

Geographic Knowledge Extraction and Semantic Similarity in OpenStreetMap

This paper is about the creation of crowd-sourced geographic datasets. OSM is the leading project in creating an open-content world map through user contributions. The semantics of OSM consists of geographic classes and descriptive properties. Due to the simple and open semantic structure, the OSM approach often results in ambiguous data. Semantic similarity of the OSM classes can reduce this semantic gap (Ballatore et al. 2013).

Curviness as a Parameter for Route Determination

In this paper the parameter for the curviness of a road is discussed. Therefore, special groups of traffic participants have specific requirements as to the curviness of roads. Truck drivers may want to avoid sharp curves whereas motorbike drivers try to avoid long and straight roads. The paper represents three different approaches for modelling the results. These are the determination of an optimal route, the specific requirements for different traffic participants and finally the modelling of the costs. The costs are represented as the curvature of a road segment, which is the inverse of the radius. Due to the calculation of the curvature optimal routes may be calculated for different purposes of the traffic participants (Navratil 2012).

Identifying Curviness of Overpass Mountain Roads from Remote Sensing Data

This thesis introduces the modelling of curvy roads. The identification can be accomplished by the use of continuous objects in a digital form or by discrete objects, like extracted roads from remote sensing data. To estimate the shape of the road the concept of road boundary detection is used which will output a polygon for the identification of curvy road segments. Therefore, two approaches can be applied to identify the curvature. They are both based on the angle and the deviation of a point at a straight line by using a fix-sized moving window. The first approach measures all the curvature values of a road, and the second approach calculates only one curvature value based on the mid, the start and the end point. This method for identifying curves can be used for data with different quality and spatial resolution, and therefore, this method can be used for different applications (Alian 2007).

Ontology-based Retrieval of Geographic Information

This paper presents an approach to ontology-based Geographic Information (GI) retrieval where semantic heterogeneity occurs. The problems of semantic heterogeneity are caused by synonyms and homonyms during free-text searches in catalogues. Those catalogues provide information descriptions, but the GI retrieval is still inadequate. Therefore, attributes within geographic datasets are often difficult to interpret. The concept of an ontology can overcome this problem by allowing the requester the make use of a well-known vocabulary of a specific domain (Lutz and Klien 2006).

A Semantic Similarity Measure for Formal Ontologies

This thesis introduces a model for comparing the semantics of at least two data sources. The aim of the Semantic Web is to provide a more intelligent web by combining information from heterogeneous and different systems. The usage of the same integration service can help overcome this heterogeneity. This service provides integration on a syntactic and semantic level. Through a semantic similarity measure the semantics of at least two data sources can be compared and embedded into the integration services (Hall 2006).

Road Data as Prior Knowledge for Highly Automated Driving

In this paper the priority of road data for highly automated driving is described. Therefore, vehicles have to recognize and to record road data and all associated road data attributes in real time. To facilitate highly automated processes the software in the vehicle should be able to identify any road infrastructure on the planned route. Consequently, highly developed maps are necessary for the navigation system, where the localization of the vehicle can be accomplished faster and more accurately. The knowledge can be directly retrieved from separate layers within the highly developed maps, which encompass the kinematic parameters and the digital geometric data. The road data can be processed in the OpenDRIVE format for setting speeds, which can be determined from the road geometry (Kühn et al. 2017).

Core Ontologies for Safe Autonomous Driving

This paper is about the usage of ontologies for representing the knowledge of maps, driving paths and driving environments. This structured machine-readable representation should improve the safety of autonomous vehicles. Therefore, some ontologies have been developed for Advance Driver Assistance Systems. Those ontologies are divided in three different types and can be used for the construction of knowledge base for autonomous vehicles. A map ontology is used for receiving the driving environments by describing the road network, like e.g. roads, intersections, lanes and traffic light information. A control ontology manages the driving actions and the paths of autonomous vehicles. A car ontology consists of different types of vehicles and their attributes, like e.g. sensors and engines (Zhao et al. 2015).

1.4 Scope of the Work

This thesis is organized as follows. The first chapter gives an introduction and an overview concerning the problem definitions, the goal of the work and the research question. It also includes a summary of related work. Those parts have already been mentioned before.

From the second chapter onwards to the fourth chapter we will represent the data, the methodology, the implementation and the results of the thesis (Figure 1). In the second chapter the study area and the data are described. Driven by the requirements for data enrichment and manipulations many different applications and software packages had to combined to achieve the intended results. Those applications and software packages include:

- applications for downloading and implementing the data,
- software packages to create road data ontologies, and
- applications for generating and validating OpenDRIVE files.

The next chapter describes the methodologies used to create this thesis. In section 3.1 we will discuss options for data retrieval from OSM – which forms the base of this research. Chapter 3.2 is mainly about the calculation of the horizontal and vertical curvature and its implementation within road data. Chapter 3.3 exemplifies the creation of an ontology. In chapter 3.4 will focus on the generation of simulation networks and its subsequent transformation into the OpenDRIVE files.

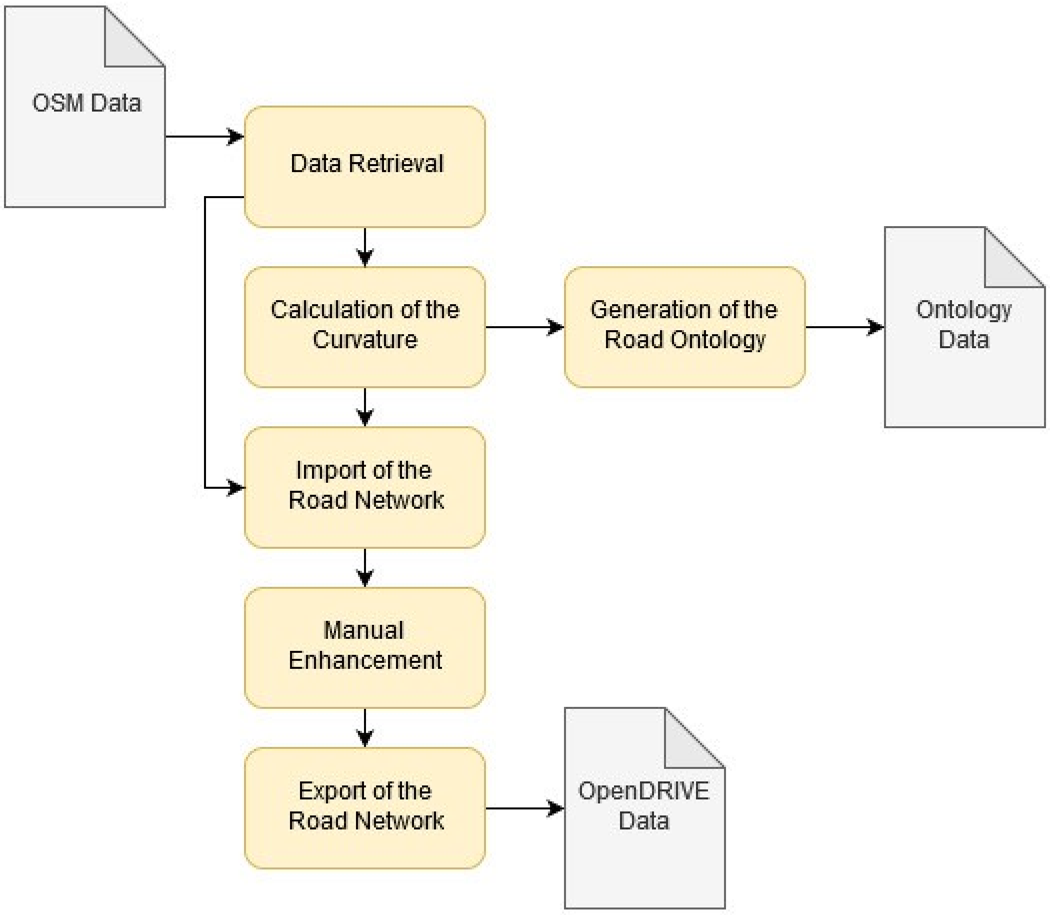

OSM Data
Data Retrieval
Calculation of the Curvature
Generation of the Road Ontology
Ontology Data
Import of the Road Network
Manual Enhancement
Export of the Road Network
OpenDRIVE Data

Figure 1: Workflow of the methodology

Thus, the part of the methodology consists of the three main parts on which we will focus within the thesis in chapter 3.2, 3.3 and 3.4. The end of each chapter contains the implementation details. The fourth chapter contains research results as well as a critical reflection about selected steps within the workflow. The final chapter presents conclusions derived from this research and provides suggestions for further investigative work.

2 Study Area, Data and Used Software

2.1 Study Area

The main focus is on road areas with freeways where autonomous driving can be applied quite easily. Therefore, the area around the freeway junction Peggau-Deutschfeistritz, a local area in Styria, Austria, in the north of Graz, is used as the first test area. This area of the road has a lot of straight sections due to the freeway as well as a lot of curved sections to enter and exit the freeway. This study covers the area starting at the freeway junction Peggau-Deutschfeistritz in the south, continuing onto the freeway service area Deutschfeistritz in the west and closing at the driveway Peggau Mitte in the east and in the north (Figure 2).

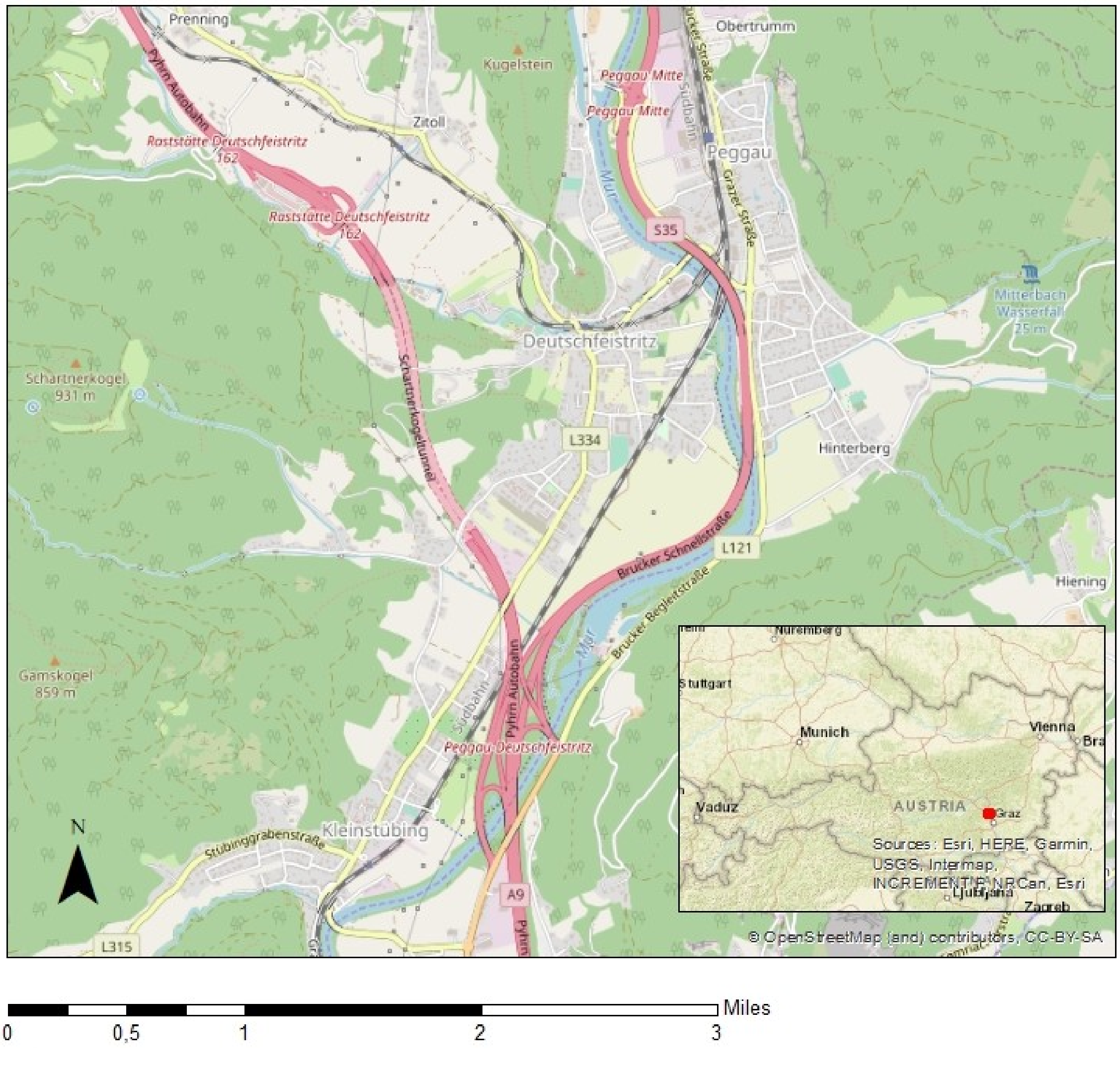


Figure 2:Junction Peggau-Deutschfeistritz, Austria

Besides the usage of freeways main roads are also of high relevance as they can serve as possible links to freeways. However, both of them show special attributes. Therefore, two sites in Bavaria, Germany between Munich in the south and Ingolstadt in the north are chosen. The first site is the area around the freeway junction Langenbruck near Ingolstadt, which is a part of the southern section of the freeway A9. It consists of some main roads in addition to the freeway.

The second site also composes of an area including a freeway and some main roads as mentioned above. This test area is located at the junction Allershausen on the A9 near Munich, some miles down south of the previous site around Langenbruck (Figure 3).

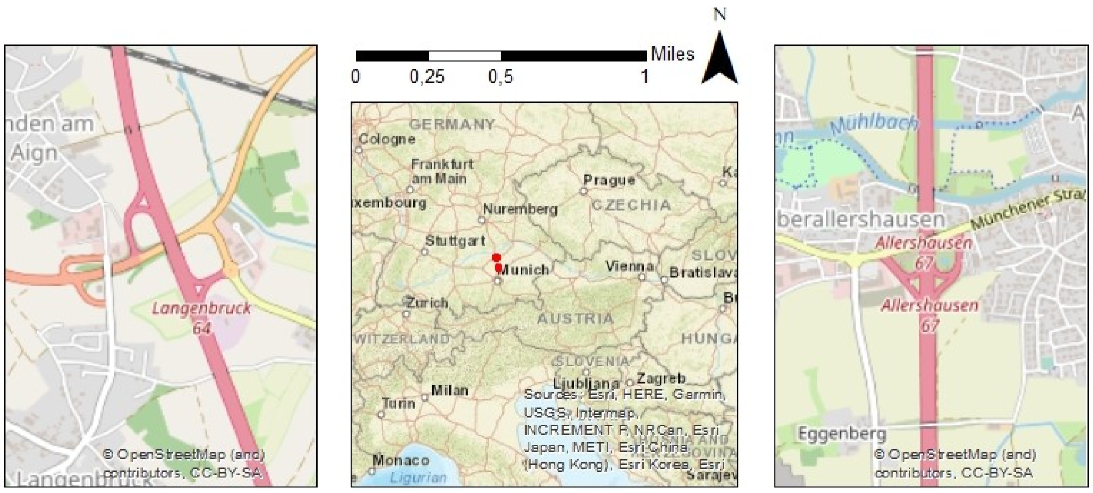

Langenbruck
64
© OpenStreetMap (and) contributors, CC-BY-SA
Miles
0
0,25
0,5
1
N
GERMANY
Cologne
Frankfurt am Main
Prague
Nuremberg
CZECHIA
Stuttgart
Munich
Vienna
Bratislava
Zurich
AUSTRIA
SLOVENIA
Ljubljana
Zagreb
Milan
Turin
Monaco
Sources: Esri, HERE, Garmin, USGS, Intermap, INCREMENT P, NRCan, Esri Japan, METI, Esri China (Hong Kong), Esri Korea, Esri
Mühlbach
Münchener Straße
Allershausen
67
Eggenberg
© OpenStreetMap (and) contributors, CC-BY-SA

Figure 3: Junction Langenbruck (left) and Junction Allershausen (right)

2.2 Data

Free and open source data are the most cost-effective way when it comes to generating larger road models. Their worldwide availability is another essential benefit. In order to guarantee their world-wide application it is absolutely necessary to ensure a high geographical coverage. Hence, the most ambitious and also most successful collaborative data project for this purpose is OSM road data. Those vector data are available in most countries worldwide. Moreover, they account for a high amount of accessible data. Therefore for all the three study areas the data are used from OSM (Ballatore et al. 2013, pp. 1–2).

The OSM concept follows the principle of Volunteered Geographic Information, to which a large group of only registered users make a contribution. By restricting the editing of the data to registered users the OSM project enables a detailed inspection of the source of information regarding copyright conflicts. The project's aim is free usage and editing of a set of map data. In Europe, accurate digital geographic information is very expensive, hence the availability of free geographical information is the key motivation of OSM. The data are edited by a considerable number of volunteers, which are using the technical infrastructure. The technical infrastructure of OSM is created and continuously improved by a small amount of approximately 40 volunteers. That work consists of server maintenance, software implementation and the creation of the cartographical outputs. The availability of OSM data for further usage across different applications, software platforms and hardware devices is being developed by another growing group of contributors (Haklay and Weber 2008, pp. 12–14). The data format of OSM can often be translated into a table format that can easily be embedded into relational databases. However, this can turn out to be difficult for the extraction of large data amounts. The amount of the road infrastructure in OSM data increases very fast, and there is a lot of additional data available (Richter et al. 2016, p. 24).

One important aspect of the coherence and the quality of the OSM vector data is its semantic structure, which is stored in a text file based on the Extensible Markup Language (XML) format. An OSM dataset consists of objects and associated properties, which are constructed in a key-value pair structure. Those properties are called "tags" in the OSM world and represent the semantic content of an object. The OSM tags are described on the OSM Wiki website[1]. The OSM keys specify groups of geographic entities (e.g. *highway*) or encode properties with unrestricted values (e.g. *name*) (Ballatore et al. 2013, pp. 6–7). The geographical entities of the vector datasets are constructed as different types. The simplest data form are nodes (points) which include the geographical longitude and latitude coordinates and the associated information. Another type of features are ways (lines and polygons) which are defined by referencing

[1] https://wiki.openstreetmap.org/wiki/Taginfo (checked on 06/15/2019)

to a list of ordered nodes. Although polygons are not defined as explicit area features, they are rather associated with a closed way, i.e. the first node of the way is also the last node. More over those vector datasets can be represented as relations (groups and objects) (Haklay and Weber 2008, p. 15).

The extent of a key with its set of defined values ranges from small (e.g. *junction*) to very large (e.g. *amenity*), the latter containing more than 150 values. By the way, semantic difficulties can also occur by defining similar tags with different keys (e.g. *landuse=garages* and *amenity=parking*). That semantic gap can cause some discrepancies among the mapping users. An association between similar entities can be found with the help of a semantic similarity measure, which is currently not possible through the structure of OSM tags, due to its parent-sibling concept (Ballatore et al. 2013, pp. 6–7).

2.3 Used Software

2.3.1 QGIS

QGIS[2] is a free and open source Geographic Information System (GIS) application for viewing, editing and analyzing geospatial data. QGIS supports raster and vector data, whereby vector data are stored as point, polyline or polygon features. The software allows the use of many geospatial formats and data from external sources (QGIS Development Team 2019). In this work the main use with this software is to handle geographic open source data. It is highly compatible with OSM data for importing, visualizing and converting into other data formats like Shapefiles and also many others.

2.3.2 GIS/Python

The interpreted programming language Python is a good choice for performing geographic data analysis and map automation due to its connection with ArcGIS and the ArcPy site package. This package includes a huge range of modules for general purposes

[2] https://www.qgis.org/de/site/ (checked on 07/15/2019)

and it is appropriate for interactive work and for scripting. Therefore, Python scripting with additional support of ArcPy packages is used for the calculation of the curvature.

2.3.3 Protégé

Protégé[3] is a free and open source editor for modelling ontologies. The software was originally developed by the Stanford Center for Biomedical Informatics Research at the Stanford University School of Medicine. The software environment of Protégé is implemented in Java, which guarantees its platform independency. Protégé is used to create, manage and edit knowledge databases of specific fields of knowledge and to query the knowledge with SPARQL queries. An ontology can be validated through deductive reasoners. Those classifiers reason if models are consistent and infer new information based on the analysis of an ontology. The Protégé framework includes various plugins for different application requirements. Moreover, it allows for individual adjustments. The Protégé editor is based on Web Ontology Language (OWL) constructs, which will be used for generating an ontology for the road data and its related parameters. The editor offers two basic types for knowledge representation:

- Protégé-frames
- Protégé-OWL plugin

Protégé-frames is the originally used frame-based approach for knowledge representation. The information about a specific knowledge in a domain will be represented in a hierarchical structure with the components of classes, slots and instances. Protégé-OWL is based on the Semantic Web standard OWL. In addition, logical mechanisms can be used, which are able to infer implicit knowledge from one or more ontologies with different sources (Stanford University/Protégé community 2019). The components in an OWL ontology are quite similar to a frame based ontology, but they differ in their terminology. Therefore, they are described as classes, properties and individuals (Horridge 2011, p. 10).

[3] https://protege.stanford.edu/ (checked on 07/15/2019)

For the generation of the ontology the Protégé version 5.5.0[4] will be used in this thesis. This version is the latest stable release that involves various improvements at the user interface and in bug fixing. Furthermore, some useful embedded tools are implemented, like e.g. the Cellfie plugin for importing data from Excel spreadsheets and the Pellet reasoner among others (Stanford University/Protégé community 2019). Since the Protégé version 4 a guiding document for creating OWL ontologies has been available. That document focuses on the generation of an OWL-DL-based ontology and on the integrating of a Description Logics (DL) reasoner for checking the ontology's consistency (Horridge 2011, p. 7).

2.3.4 SUMO Netconvert

Simulation of Urban Mobility (SUMO)[5] is an open source road traffic simulation package introduced by the German Aerospace Center (DLR) for handling large road networks. It includes a network import from different source formats, a framework for automatic driving simulations for imported networks and the ability to export various data formats (Behrisch et al. 2011, p. 1). The software supports the import of several data formats, like OSM and Shapefiles and also provides the export of the open format specification OpenDRIVE 1.4 for further traffic simulations. Despite its limited functions in exporting to the OpenDRIVE format it is the only sufficient tool that supports this data format for that purpose.

2.3.5 OpenDRIVE Viewer

For visualizing data in the OpenDRIVE format the OpenDRIVE Viewer can be made use of. This software is suited for Linux systems and allows a quick and efficient visualization of the data. Additionally, some options are available for switching on and off features for the presentation on the display. The latest version we use is the OpenDRIVE ODR Viewer 1.9.1.

[4] https://github.com/protegeproject/protege-distribution/releases/tag/v5.5.0 (checked on 07/11/2019)
[5] https://sumo.dlr.de/wiki/Simulation_of_Urban_MObility_-_Wiki (checked on 07/15/2019)

2.3.6 OpenDRIVE Validator

The OpenDRIVE Validator[6] is a free tool for the validation of OpenDRIVE files. The tool is available as a Java application for Linux systems within the usability of VDT. It contains several possibilities for rule adaptation through a configuration file according to the desired quality criteria of validation. The validator was invented for the OpenDRIVE version 1.4 by VIRES Simulationstechnik GmbH within a corporation with Audi Electronics Venture[7].

[6] https://redmine.vires.com/projects/vtd/wiki/Wiki (checked on 07/11/2019)
[7] https://www.vires.com/OpenDRIVE/ODRMeeting20151015_VIRES.pdf (checked on 07/11/2019)

3 Methodology

3.1 Data Retrieval

The retrieval of geographic information can be accomplished by using OSM data. An integrated export function enables downloading OSM information in various vector and raster formats for further usage. This function allows a quick export of generated maps, images, document files and also raw data for a selected bounding area. There are also developed ordinary sets of map tiles for specific usage and user tasks available (Haklay and Weber 2008, p. 14). However, this is a pre-built solution and contains a lot of information which is irrelevant for the task in this thesis.

With the help of web application Overpass Application Programming Interface (API) specific data can be selected effectively from a certain geographical area. Those extracted data are usually filtered by a region and maybe also by a subject (Jokar Arsanjani et al. 2015, p. 101). The querying of the data is built for a specific bounding box, where the desired data mostly in form of nodes and ways, but also as relations can be retrieved. This is a very sufficient solution for small areas, but it turns out to be extremely time-consuming, if not to say almost impossible for very large areas (Jokar Arsanjani et al. 2015, p. 110). A further usage of the retrieved data has already been described in chapter 2.2.

3.2 Curvature

The term curvature includes two types in form of the horizontal and the vertical curvature. For an easier understanding of the horizontal curvature, the parameter will just be called curvature, whereas the parameter for the vertical curvature is then referred to as incline.

3.2.1 Curvature Definition

A road in a virtual space consists of at least two vertices and polylines between each of the two vertices. Therefore, a curve is made as a segment of a circle and its osculating radius, and there is also a connection to the straight parts of a road (Andrášik et al. 2013, p. 75). Therefore, the circumference of a circle is given as:

$$2 \times \pi \times radius$$

The curvature parameter is derived from the slope of the circle at a specific point on the circumference. Hence, the curvature on the edge of the circle is defined as:

$$\frac{1}{radius}$$

This imparts the knowledge that the smaller the circle, the higher the curvature and vice versa (Öberg 2012, p. 22).

A curve might also be described by the angles between two vector polylines and the resulting change of direction. The information we get for the curve or the curvature of a road segment relates to polylines, which consist of at least one line. The entire road segment is assumed to be a polygon of which the curvature parameter will be derived. For identifying the curvature in a 2-dimensional (2D) space the arc length or the length of the polyline and the curvature are the two essential factors (Alian 2007, p. 16).

3.2.2 Curvature Calculation

An easy way to determine the curvature is the computation of the quotient q between the Euclidean distance $l_{Euclidean}$ of the start and the end point of a road and the regular length of the road l_{true} between the same points:

$$q = \frac{l_{Euclidean}}{l_{true}}$$

The result would show that a straight road without any sense of curvature would have a length quotient with a value of one. Whereas a road with loops or a road with infinite length would have a value of zero, which might only occur with specific form of e.g. roundabouts represented by a closed polyline. In general, roundabouts have a relative small size and a small geometrical distance. Therefore, the determined quotient of zero should have less or even no impact on the curvature parameter. All other types of roads will have a quotient between one and zero. The disadvantage of this approach is that it does not describe the shape of the road. This means that roads with different shapes may have the same length quotient (Navratil 2012, pp. 357–358).

In order to determine the parameter of the curvature of a road we apply two simple approaches of an average curvature. The first approach uses a moving window size always containing three points for the curvature estimation: a start, a middle and an end point as vertices. Those vertices are connected with one polyline between the start and the middle point and between the middle and the end point. For the second approach we use all the points of a polyline. This signifies that a polyline with only two points occurs with one point at the start and one at the end with a polyline between the two. Without the existence of a middle point, this results in a straight line with a zero curvature value (Alian 2007, p. 16).

The following process will be applied for the determination of the curvature of a road segment by assuming of the consistence of three points within a polygon in a not intense generalized road network (Figure 4):

- For each three points p_{n-1}, p_n, p_{n+1} in a polyline, where n is the middle point, we determine the radius of an osculating circle through these three points.
- Assuming that each line segment is a part of a polygon of three points, the length of a polyline segment will be determined.
- The radius of a circle is being inverted to get the curvature of a polyline segment.
- The window shifts by dropping the first point in the polyline and adds the next point in the polyline. This process repeats until no point in the polyline is left.

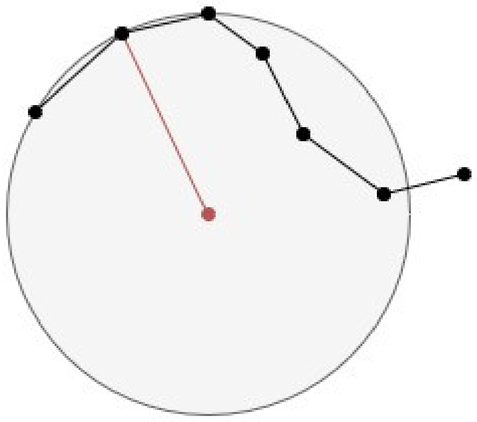


Figure 4: Radius of a circle (red) of three vertices (black point) within a polyline (black line)

The result of the process is a list of curvatures to their corresponding length (Navratil 2012, p. 358). The ideal form of the available data of polylines, would consist of points as vertices of a polyline with equal intervals to each other and with only small measurement errors (up to 0.1 m) (Andrášik et al. 2013, p. 75). In reality, the line segments are of different length, but also similar curvature values. Therefore, a weighted average has to be applied to the network:

$$c = \frac{\sum_i c_i l_i}{\sum_i l_i}$$

The curvature is represented as a function where c_i is the curvature and l_i is the length of an arc or a part of the polyline. To get a balance between road segments with different curvature values but with similar length, the average curvature has to be multiplied with the length of the corresponding road segment (Navratil 2012, p. 358).

3.2.3 Process of Curvature Calculation

The process for calculating the average curvature is further described in the following Unified Modeling Language (UML) diagram. Therefore, we refer to the enhanced approach of the osculating radius and the average curvature by Navratil (Figure 5).

For the calculation of the curvature we need two data files as input. The first file is the raw data in form of a Shapefile and the second file contains the same Shapefile but with its polylines split at its vertices. In the first loop we enter a polyline in the Shapefile containing the raw data. Within this loop we step through each vertex in the polyline feature to get the longitude *x* and the latitude *y* of each vertex. By checking the amount of vertices in the current polyline we decide on the calculation type.

If the polyline contains exactly two vertices, i.e. as start and an end point and no other points occur we assign an average curvature with the value *0* to the output of a straight line (Alian 2007, p. 16). The output will then be assigned to each polyline segment in the split Shapefile and represent its curvature. After that process we go back to the beginning, where we enter the polyline and start the loop with the next chosen polyline from anew.

If the polyline consists of at least three vertices we can apply the intended approach of the osculating radius. Therefore, the center point of the circle through the first three points will be computed. Then we calculate the radius of this circle and invert the radius like mentioned above at the beginning of this chapter (Öberg 2012, p. 22). Furthermore, it is necessary to determine the length of each polyline segment. Therefore, the length between the first vertex to the second one and from the second vertex to the third and last one will be computed.

This process needs to undergo several checks in order to calculate the correct number of vertices as each polyline contains a different one. If there are exactly three vertices within the polyline, the average curvature has to be calculated at the first and the last segment of the polyline. Therefore, the average curvature will be weighted by means of the length at the particular polyline segment. Again, after the processing and the assigning of values of the average curvature to the split output file we go back to choose the next polyline for calculation.

The last case appears when a polyline contains of more than three vertices. There we have the same process of radius calculation and length determination. We also compute the average curvature for the first and the last line of the polyline by considering the length of the specific polyline segment. In addition, we calculate all the polyline segments within the middle of the entire polyline based on the weighted aspect. Finally, it is necessary to assign the average curvature to the output data and to go back again in order to repeat the processing until no polylines are left in the input data.

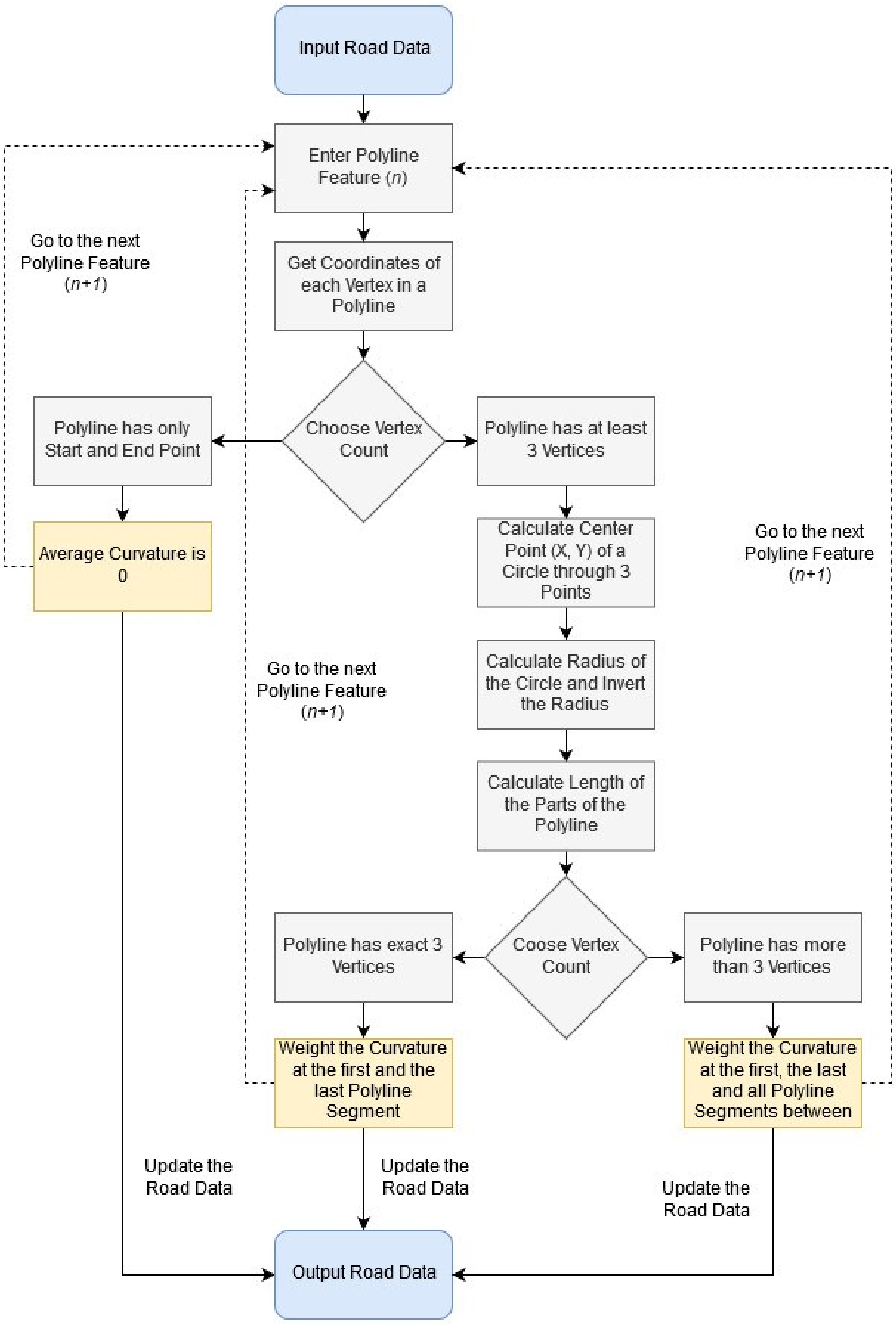

Input Road Data
Enter Polyline Feature (n)
Get Coordinates of each Vertex in a Polyline
Choose Vertex Count
Polyline has only Start and End Point
Average Curvature is 0
Polyline has at least 3 Vertices
Calculate Center Point (X, Y) of a Circle through 3 Points
Calculate Radius of the Circle and Invert the Radius
Calculate Length of the Parts of the Polyline
Coose Vertex Count
Polyline has exact 3 Vertices
Polyline has more than 3 Vertices
Weight the Curvature at the first and the last Polyline Segment
Weight the Curvature at the first, the last and all Polyline Segments between
Go to the next Polyline Feature (n+1)
Go to the next Polyline Feature (n+1)
Go to the next Polyline Feature (n+1)
Update the Road Data
Update the Road Data
Update the Road Data
Output Road Data

Figure 5: UML diagram of the curvature calculation process

3.2.4 Incline Calculation

The incline of a slope is well-known as the tangent of the angle between the surface and the horizontal level of the reference surface. For the calculation a DEM is needed which can be retrieved for different levels in different ways. For a better representation of the incline we will need a DEM with a high geographic resolution. For this purpose DEM's[8] with a geographic resolution of 10x10 meters made from airborne laser scanning are available in Austria.

The incline of the road data will be done for every single road segment (Figure 6). Therefore, we need the length of the road segments and we have to identify the absolute height of the start and the end point of each road segment out of the DEM. After that the calculation of the incline can be accomplished.

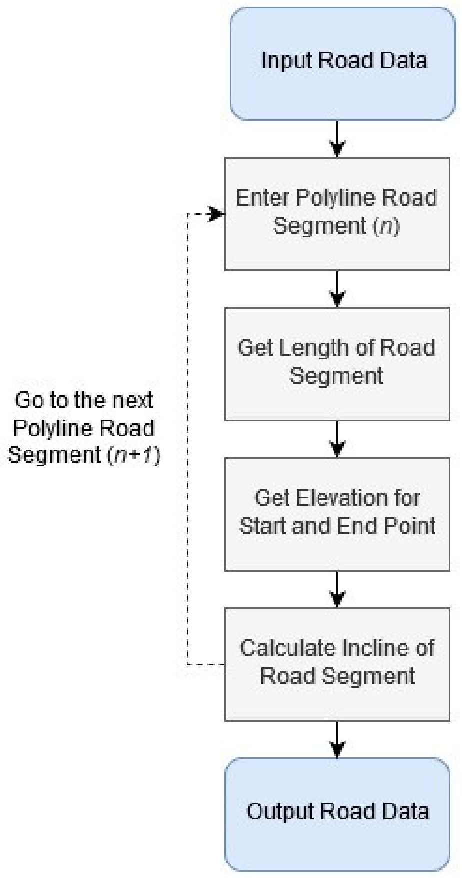

Input Road Data
Enter Polyline Road Segment (n)
Get Length of Road Segment
Get Elevation for Start and End Point
Calculate Incline of Road Segment
Output Road Data
Go to the next Polyline Road Segment (n+1)

Figure 6:UML diagram of the incline calculation process

[8] https://www.data.gv.at/katalog/dataset/d88a1246-9684-480b-a480-ff63286b35b7 (checked on 07/15/2019)

3.3 Ontology

3.3.1 Ontology Definition

An ontology is a structural framework for knowledge representation about a domain of the real world or a part of it. Ontologies consist of concepts and relationships between them. Concepts are also known as classes and relationships that can be called properties (Zhao et al. 2015, p. 1). This is a rather simple interpretation of many interpretations of the term ontology. Originally, the term comes from philosophy, where things in the real world should be classified, while Artificial Intelligence (AI) operates with models of the world for knowledge sharing and reuse of ontologies (Linková et al. 2005, p. 3). Therefore, the term ontology is taken for describing the computational representation of the world in a program for AI.

As mentioned above, a lot of definitions have been used to describe an ontology. The most accurate one indicates that "an ontology is a formal, explicit specification of a shared conceptualization". Formal means that an ontology is machine readable. Explicit specifications refers to the used concepts, attributes, functions and explicit defined axioms. Shared means that an ontology contains consensual knowledge. The term conceptualization describes an abstract model concluding various phenomena of the real world (Studer et al. 1998, p. 25).

The aim of an ontology as a shared concept model is to provide structural information to merge already existing knowledge for a much easier information search. The information in form of spatial data may be defined and semantically specified. Moreover, it can also be machine-readable (Wang et al. 2007, p. 205). For representing information of the real word it is necessary to consider the difficulty of different key aspects of mapping from different knowledge bases. For this purpose an ontology is the most common way to describe real objects und their relationships. In information processing it has the task to formalize real objects and their relationships for communicating between the human experience and the technical information representation. The challenges are on the right classification within a context. Therefore, real objects of the world may be distinguished into concrete and abstract objects, where concrete objects are e.g. persons and abstract objects are e.g. organizations (Pfeiffer 2010, 12-13).

3.3.2 Ontologies in Knowledge Engineering

Ontologies can be categorized into their intended purpose of knowledge representation. In a knowledge engineering environment ontologies enable the construction of a domain model in a specific range of knowledge (Pfeiffer 2010, p. 13). Therefore, the domain is modeled by terms and relations between them. Various types of ontologies exist for different demands in the process of creating a domain model. However, they all have the same basis of determining explicit static knowledge about a domain. As different domains can also be considerably diverse it may be useful to distinguish between different types of ontologies:

- Application ontologies describe the modelling of a certain domain with all the necessary knowledge. This type of ontology is mostly used for a specific task.
- Domain ontologies are applied to a particular type of domain with all the valid and specific knowledge.
- Generic ontologies (task ontologies) are used across various domains and consist of definitions for describing general activities.
- Representation ontologies (top ontologies) are not applied to any certain domain. They operate independently and represent knowledge of general nature. Therefore, representational entities are provided in a Frame Ontology, which represents knowledge in a frame-based or in an object-oriented way (Studer et al. 1998, p. 27).

Among the differentiation of the diverse ontologies the main function for knowledge representation remains the same. Hence, the concrete description of the real world data should support the retrieving information on the user’s side (Pfeiffer 2010, p. 13).

3.3.3 Ontology Languages

3.3.3.1 XML

XML had been introduced as an extensible language for defining data structures, which is not provided by the former Hypertext Markup Language (HTML) format due to its fixed annotation scheme. The advantage of the XML format is the exchange of data in a structured and syntactical way over the World Wide Web. For that purpose XML schemata have been established as a definition language for the restriction of structures of documents and datatypes.

An XML schema is presented as a single XML document which defines the valid structure of the given data which follows the guidelines for the documents provided from the World Wide Web Consortium (W3C) committee. That specification offers a formal grammar and further grammatical restrictions on the structure of a document. This restriction in such a well-formed document is called Document Type Definition (DTD). The components used in an XML schema definition have the type "element" and associated attributes which define the restrictions. The element itself contains the information as a list of further element definitions, which are implemented in a nested way inside the defined element:

```
<element name="value" type="value" ...>
    <element name="value" type="value" ...>
    ...
</element>
```

A further advantage of the XML data format is that it provides additional features to define data structures. The following additional features are helpful for encoding complex data structures:

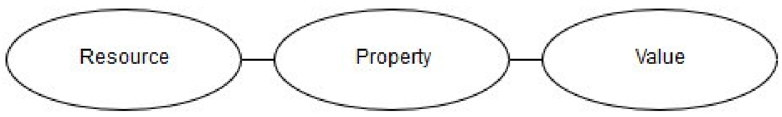
- Possibility of a combination of different XML schemata
- Restrictions on attributes
- Sophisticated structures
- Support of basic data types

Therefore, specific data models of applications can be mapped for sharing its information on an XML schema. For data exchange on an XML document its information can be encoded in the form of an XML document and can be made available on the World Wide Web by using it within a combination of an XML schema document. Such an XML document only provides a definition of the structured data and does not provide information about the content and the possible usage of the information. Hence, we cannot retrieve any meta-information from this data definition (Stuckenschmidt 2003, pp. 10-12).

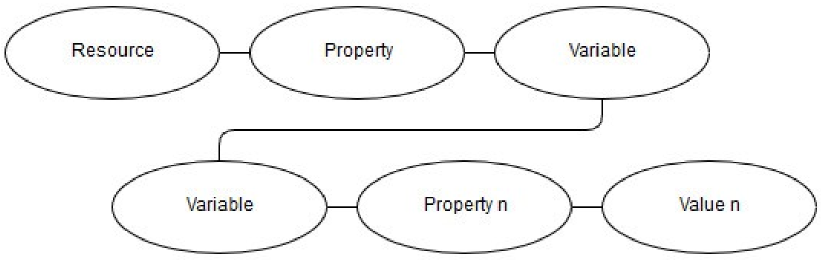
3.3.3.2 Resource Description Format

Resource Description Format (RDF) was introduced as a framework for describing meta-information and to define its meaning. The RDF standard is an additional approach based on the XML syntax for describing the previously mentioned meta-information and their content in form of simple semantics.

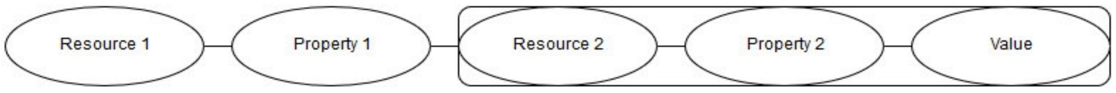
The RDF model is built in form of triples in a schema, which contains the information about an XML element. In this context the XML element can be expressed as a resource with properties and values. A property is defined as a relation which connects a resource to a specific value of this property. A value is expressed as a simple data type or as a resource:



In addition, a value can also be represented as a variable which is described as a resource that is further described by one or more new triples which are linked to a variable with asserted properties of the resource:



It is also possible to use a triple as a value for the property in a resource. This feature is called reification mechanism which allows us to make statements about specific facts. The representation will occur in a nested way:



Furthermore, multiple values can be associated to a single property. Therefore, the RDF model contains collections, which are defined as three built-in data types for providing aggregation mechanisms in a certain way:

- Lists
- Ordered lists
- Sets of alternatives.

Name conflicts of different RDF models may occur when sharing those models over the World Wide Web as a reference to different web sites. To avoid this problem name-spaces are used from the RDF which are provided by XML. The definition of those name-spaces refers to an Uniform Resource Identifier (URI) that provides the names and a connection to the source Identifier (ID). By defining the origin of a certain name this source ID is then used to annotate a name in a RDF specification:

Source ID:name

The RDF statements, which express the meaning of information, are implemented in a standard syntax for representing those as meta-information (Stuckenschmidt 2003, pp. 12–14).

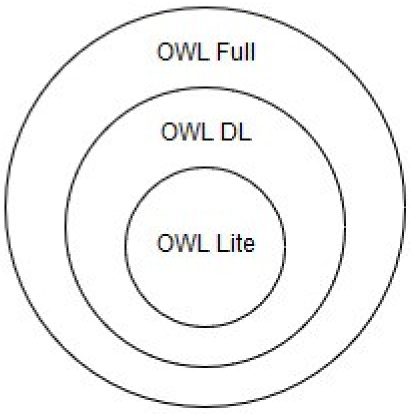
3.3.3.3 OWL

OWL is a very expressive language for exact description of semantic knowledge in the Semantic Web that is supported by many tools, like e.g. Protégé (Linková et al. 2005, p. 5). It was originally developed by the W3C as a standard ontology language and is based on a logical model for defining and describing simple as well as complex concepts (Horridge 2011, p. 10). The origin of this OWL language comes from three areas:

- Description Logics (DL)
- Frame systems
- Resource Description Format (RDF)

DL is a concept language for logical systems that derives their formal semantics of constructors and their language features from their semantics. Frame Systems describe the view of a frame-based system into a knowledge base for simplifying the understanding and handling of OWL ontologies. This frame-based view helps unskilled user working with DL (Hall 2006, pp. 10–11). RDF has already been mentioned above. However, it is not very expressive in the description of knowledge (Linková et al. 2005, p. 5). Since OWL is based on RDF, those two formats are compatible with each other and, therefore, a RDF graph is also a valid OWL ontology (Hall 2006, p. 11).

With the help of OWL it is possible to define and describe concepts in the same way like other ontology languages. However, the recent OWL development provides more functionalities over others (Horridge 2011, p. 10). Therefore, OWL is available in three expressive variants of languages:



OWL Lite is the simplest form comprising simple constraints and a basic classification hierarchy for the user. Therefore, it is built on a low formal complexity. OWL DL represents an extension of OWL Lite and concludes all OWL language constructions. By using this sublanguage a maximum range of expressions is assured. However, the usage of the entire language package goes along with some specific restrictions. OWL Full is an extension of OWL DL, which offers the maximum expressiveness and the entire syntactic possibilities of the RDF format. With that language the meaning in an ontology can be extended in addition to the predefined RDF and OWL vocabularies. Unfortunately, there is no computational guarantee anymore. Therefore, a complete reasoning for every feature is not assured from any reasoning software packages (McGuinness and van Harmelen 2004, p. 5).

3.3.4 Ontology Schema

The coverage of the OWL language features depends on the type of sublanguage. Nevertheless, we will discuss some basic components that occur in the schemata of all OWL sublanguages. OWL Lite and OWL DL have more limitations by using these components with respect to OWL Full:

- Classes
- Properties
- Individuals

- Domain
- Range

A class is defined by a group of individuals with the same properties. Classes can be organized in a hierarchical way, which is called taxonomy. By setting a class as a subclass of another class the class hierarchies may be created. The basis in an ontology is always represented by the built-in class named “Thing”. There can also be a built-in class named “Nothing” after inferring the ontology, which does not contain any individuals and forms a subclass of all classes in an ontology.

Properties are used to describe relationships from individuals to data values or relationships between two or more individuals. The first case is made in form of datatype properties and the second case might be solved by applying object properties. Hierarchies with properties can be created by setting a property as a subproperty of one or more properties (McGuinness and van Harmelen 2004, pp. 8–9). Furthermore, an object property can hold some characteristics like *functional*, *inverse functional*, *transitive, symmetric, asymmetric, reflexive* and *irreflexive*. In contrast to object properties, datatype properties only hold the characteristic type *functional* (Horridge 2011, p. 11).

An individual is used as an instance of one or more classes, where the properties are relating those individuals to each other (McGuinness and van Harmelen 2004, p. 9). Those individuals may model abstract or concrete objects in a domain of an ontology (Zhao et al. 2015, p. 3). As OWL is not applying the Unique Name Assumption (UNA), we have to explicitly describe the similarity and the difference of individuals in relation to each other (Horridge 2011, p. 10).

The domain of a property might be used for the assignment of this property to a class. It limits the individuals to which the property relates. From that the reasoner can infer the relations between the individuals. If an individual is related to another individual through a property and a class is described in the property’s domain, the related individual will be related to the class.

The range of a property assigns a value as a limitation for the associated individuals. If an individual is related to another individual through a property and a class is described in the property's range, the related individual will be related to the class. From that the reasoner can infer the relations between the occurring individuals once again (McGuinness and van Harmelen 2004, 9).

3.3.5 Spatial Ontology

The class hierarchy for the geometry in a spatial ontology is assumed for objects in a 2D coordinate system with longitude and latitude. Therefore, those objects can be categorized into eight types of geometry, whereby the first two geometry categories are relevant to the ontology in this thesis. A detailed description can be found below:

- Point
- Single line (Polyline)
- Connected line (Not a ring)
- Connected line (Ring)
- Polygon
- Multipoint
- Multicurve
- Multipolygon

A point *p* is defined by its longitude *x* and its latitude *y*. The spatial relationship between one or more points is represented by the given coordinates. Two or more points can be equal if the longitude and the latitude are exactly the same. Otherwise, they are just not equal.

A single line is also called a polyline, which is described by two points each at the start p_1 and the end p_2 of the polyline. For the spatial relationship of a polyline two relevant operations for road data are possible. If two polylines have exactly the same start and end point coordinates, they are equal to each other. Two polylines can meet at one point, either a start or an end point.

Another type of spatial relationship is the connection between points and the start or the end point or any other vertex of a polyline. This occurs when the coordinates of a point are equal to the coordinates of one vertex of a polyline (Jitkajornwanich et al. 2011, pp. 2–3).

3.3.6 Reasoning

The complexity of reasoning depends on the restriction of first order logic, which defines the structure of knowledge by using a very detailed definition for the concepts and their relations in the real world (Hall 2006, p. 6). The reasoner itself checks the consistency of the definitions and statements in a logical model and also recognizes the compatibility of concepts and definitions (Horridge 2011, p. 10). The implicit knowledge of the axioms in the concepts can be inferred to make them explicit. In the case of DL four mainly used inferences exist:

- Satisfiability
- Subsumption
- Equivalence
- Disjointness

The first two categories Satisfiability and Subsumption are the most used inferences in practice. Satisfiability operates if a knowledge base will be updated by adding, changing and removing some concepts, as we have to know if the updated concepts are valid and if other concepts will become invalid through the update. Subsumption creates the hierarchy of the concepts by considering their universality. The other two inferences Equivalence and Disjointness may also be applied to Subsumption. If the concepts in a DL language have full negation we can reduce all the inferences to Unsatisfiability. In the case of full negation in DL languages the process of reasoning can be executed by a tableau-based algorithm (Hall 2006, pp. 8–9).

Some popular reasoners that support Subsumption are e.g. Pellet, Racer and FaCT. A major advantage of these DL's is their decidability within the inference processing (Lutz and Klien 2006, p. 112). In the Semantic Web reasoning should be a fast and complete process. Some reasoners like FaCT (FaCT++) and Racer (RacerPro) are very popular, but they are not applicable to some necessary performances in the Semantic Web, e.g. for XML schema datatypes.

The Pellet reasoner complies with this task in the process of OWL Lite and OWL DL. In that process Pellet might perform a full testing of the consistency and of the syntax by reasoning the entire concepts in an ontology (Bauer et al. 2008, pp. 81–82). Whereas OWL DL can be interpreted and evaluated completely, OWL Full is not testable at all (Pfeiffer 2010, p. 34). The required time for a simple reasoning process of the Pellet reasoner compared to the FaCT++ and RacerPro reasoners is negligibly higher in practice, but for the reasoning of complex ontologies it is much faster. Furthermore, the performance of the Pellet reasoner compared to the two other reasoners is much better (Bauer et al. 2008, p. 84).

3.3.7 Semantic Translation

Interoperability between different data sources can be achieved through integration of data from a system into another system. To achieve the interoperability between different data sources some applicable tools are required, as the integration process may not be supported by all the user systems. Possible problems may occur through the heterogeneity of data, which can be divided into three categories:

- Syntax refers to the heterogeneity of data formats.
- Structure means that a database table contains homonyms, synonyms of different attributes.
- Semantics refers to the meaning of terms in a certain application of context (Visser et al. 2002a, pp. 104–105).

Throughout this master thesis the focus will be on the semantic integration of the retrieved data, especially on the already discussed definition of curves within the road data through the calculated curvature value.

3.3.7.1 Syntax

Due to information sharing and a grown usage of computer networks an increase of the importance of standard language takes place which has been advanced by the W3C committee. The focus is on the three languages XML, RDF and OWL (Stuckenschmidt 2003, p. 10). These languages have already been mentioned in this thesis in chapter 3.3.3. The syntactic integration of data deals with the source of the information at a syntactic level. This means that the information source will be restructured by using wrappers, which will be discussed shortly in chapter 3.3.7.3. Thereby, the contents of the internal data structure can be transformed to a uniform and structural data model (Visser et al. 2002b, p. 2).

3.3.7.2 Structure

The integration of heterogeneous database schemata can be solved by transformation queries which have to be manually coded. Because of this intricate solution the use of flexible mapping relations between the different data models may be much more sophisticated. This can be achieved by using middle-ware components which define certain mapping rules between the different information structures. Therefore, some approaches for the integration of structural information with basic technologies are available.

A schema consists of a set of specific relations where every single relation as a part of the schema is also called a column. To access information of a schema we use queries by applying the mentioned relations and those associated element tuples in a clause. The result of such a query can be shown in a view as a set of tuples.

For integrating heterogeneous schemata normally the use of a global schema linked to the single heterogeneous schemata is sufficient. An XML document provides a solution for information sharing in a global schema due to its semi-structured information. Unfortunately, the semantics of the information is not included in the XML schema specification. The information structures of a global schema can be mapped with additional information about the returned tuples in the results. Therefore, that additional information can be encoded to return contextual parameters.

When encoding individual information from different information sources it will be necessary to handle different measures and scales. Therefore, we have to transform the individual information which are returned by a query. This problem can be solved by applying context transformation rules for the purpose of database integration. However, the context of a certain information is often not directly included in such a database system. Therefore, data dictionaries define the data type, the valid ranges and possible restrictions, the latter defining the basis for a transformation and explaining the relation to other data types. By applying transformation rules it is impossible to determine the form of transformation, because there is no definition of the source and the goal context of a well-defined data model. We can only use the relation of the different context conceptions without an associated schema (Stuckenschmidt 2003, pp. 14–16).

3.3.7.3 Semantics

To achieve semantic interoperability we have to apply semantic translation techniques between heterogeneous information systems. When using different information systems the problem lies in their different interpretation of information. Therefore, sematic conflicts occur in the case of the heterogeneity. The different systems have to understand the meaning of the interchanged information between them. The following two well-known types of disagreement exist:

- Homonyms occur by using the same word but with different meaning in the applied information systems.
- Synonyms result by using different words but with the same meaning in the applied information systems.

To solve the semantic conflicts for the types mentioned above specific converter and mediator systems have been established which use one-to-one structural mapping solutions. For conflicts which cannot be solved with the one-to-one mapping technique we have to take the difference of the information elements and their relation among each other into account. Therefore, some approaches can be applied to access the semantics of information (Stuckenschmidt 2003, pp. 16–17).

Semantics from Structure

In order to gain semantics of information we can use its structure. A common approach is the Entity-Relationship Approach by using conceptual models for stored information. This approach mainly focuses on handling the structural information in complex domains. This connection to the stored information is very useful for information sharing by accessing and validating the information. By using so called wrappers which are derived from a conceptual model structural information can be accessed. The implementation of wrappers to access and extract less structured information can be solved by applying machine-learning techniques. Those are represented as a set of extraction rules which extract the information and put it into a new generated structure for further processing.

As extraction rules are only defined for structural information, we have to build a logical model to integrate semantic information from different data sources. There are two approaches for integrating the semantics of the information:

- Structure Resemblance
- Structure Enrichment

In the Structure Resemblance Approach we use a logical model which represents a one-to-one copy of the database and its conceptual structure. That model might be encoded into a language with the possibility for automatic reasoning. The integration of the data will be realized on the copy of the model which is associated with the source data. Both the so called SIMS mediator (Arens et al. 1993) and the TSIMMIS system (Garcia-Molina et al. 1995) include this approach.

The Structure Enrichment Approach describes a logical model that contains the structure of the original information and additional information about the concept definitions. Some systems that use this approach are called DWQ (Calvanese et al. 1998), KRAFT (Preece et al. 1999), OBSERVER (Kashyap and Sheth 1998) and PICSEL (Calvanese et al. 1998).

Those two approaches assume the existence of semantics in the structure of the information. In reality, there is mostly a lack of conceptual data models. Therefore, those approaches are often not applicable (Stuckenschmidt 2003, pp. 17–18).

Semantics from Text

Another method to gain semantic information is to derive the semantics from the structure of information within the text. This approach is often used on the World Wide Web, where a lot of free-text resources are available. By using indexed terms with a relation to their contents the relevant information can be retrieved from those free-text documents as a result of natural language processing. An increasing precision could be achieved by using compound expressions and meaningful statements, which are very similar to the already mentioned RDF format.

The aim of using a natural language is to index descriptions from a text in a document and to analyze those documents. The problem of this approach is that repeatedly occurring terms may be implied differently. That different meaning of the same term can result in different degrees of relevance. In order to improve data retrieval, the function of a term can be made explicit in the text of a document. Thereby, the same term in a natural language can have a different meaning in the same text.

A common approach is the analyzation of the entire context of a specific term. The results are different interpretations that are based on the existence of other words in the context which effect the meaning of the term. The decision for an interpretation often depends on the general natural language vocabulary. An explicit description of the relations between the terms must be created for documents with a specific vocabulary. That can be accomplished by using a domain specific vocabulary or a semantic network.

To retrieve textual information with terms from everyday language it may be useful to apply text-understanding techniques. The main problem is the lack of specific semantic information causing a lot of limitations. The use of contextual language can solve the issue of ambiguity in the processing of natural languages. However, it is very difficult to deviate the meaning from artificial terms, especially when they are used as a specific term in the information source. Therefore, we may use potentially available background information of the data (Stuckenschmidt 2003, pp. 18–19).

Explicit Semantics

In order to extract more complex indexing terms specific models for the semantics of information from a text in a document as well as specific vocabulary from the scientific community and the technical field can be used. The semantics of information can be used in one of the models outlined below:

- Information Extraction
- Processible Semantics
- Ontologies

Information Extraction relies on the accessibility to resources of information. Therefore, we have to induce wrapping technology with information extraction techniques to get the access to the required information.

Processible Semantics deals with the development of formal annotation languages, like the already mentioned formats XML and RDF. Therefrom we can get the structure of information and the meta-information from the information source by applying the syntactic and structural approaches (Stuckenschmidt 2003, 16–20).

Ontologies represent the semantic information that is based on a formal and explicit specification of the shared conceptualization (Studer et al. 1998, p. 25) in a domain. This process is equate with an enrichment of the information sources with additional semantic information (Stuckenschmidt 2003, pp. 19–20)

3.3.8 Ontologies for Road Data

Within the task of ontology-based processes for autonomous driving some developments are already available. Those ontologies aim at creating scenarios for vehicles in different driving situations (Bagschik et al. 2018, p. 1). Furthermore, the interaction and the relationship between the road objects, which may be identified by sensor techniques, and the subject vehicle can be described by contextual information (Armand et al. 2014, p. 1).

The challenges for simulations with road data and transportation networks can also be performed by applying the usage of ontologies. Due to the amount and the diversity of digital spatial vector data from road networks ontologies play an important role in that area. Through their own formal semantics, the integration of data can be supported by the creation of standards. The use for better expressive definitions for concepts and their properties goes beyond UML diagrams and thus improves the effort for standardization. Ontologies provide a common and explicit language which is useful for the translation of the data between different applications. The common language also supports accessing databases as sources of information. However, it requires either a special technique for retrieving the results directly from the database or the possibility to translate the data directly into instances in the ontology which may be achieved by querying. Furthermore, such an ontology can also be useful as a tool for analysis purposes like simple queries for certain problems and for exploring and understanding the domain.

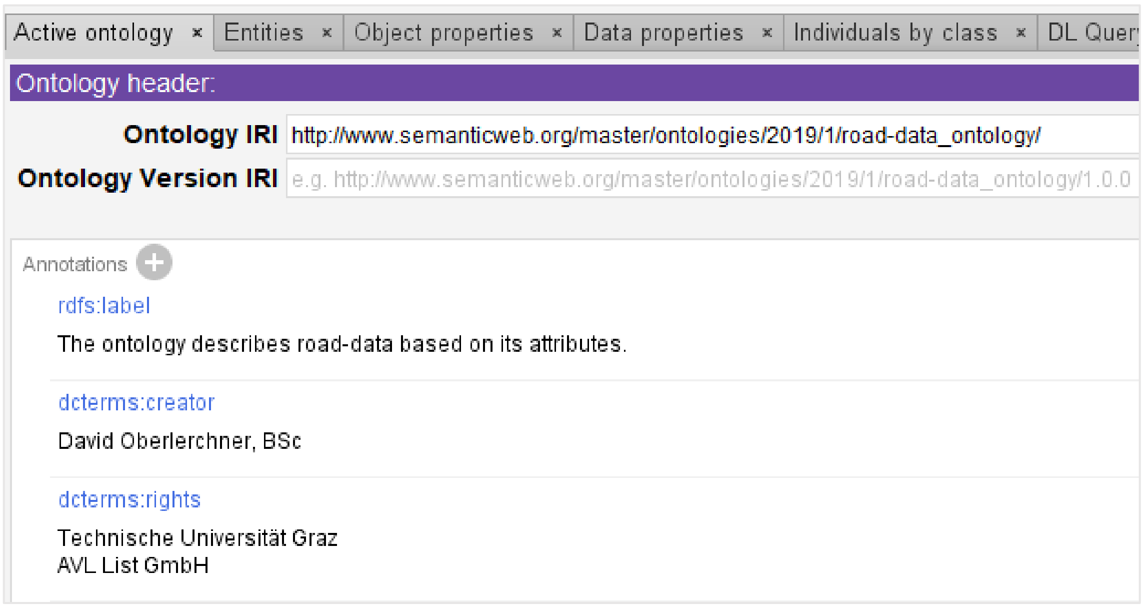
Ontologies are suited for some different languages as already mentioned before. Nowadays, OWL is the best choice for comparing ontologies and it is also the most used language over the Semantic Web. For practical applications the use of other ontology languages is not really relevant. For instance, transportation ontologies mostly use the OWL format. Some popular projects in the transportation domain are the General Transit Speed Specification (GTFS) and the City Geographic Markup Language (CityGML) formats, which are based on the XML format with a specific vocabulary for the transportation domain in a particular schema classification (Katsumi and Fox 2018, pp. 56–57). Therefore, the modeling of an adequate descriptive ontology for specific road data in the field of autonomous driving is an essential task.

3.3.9 Modelling of the Ontology

3.3.9.1 Basics

In order to generate a new Protégé ontology project we have to define an Internationalized Resource Identifier (IRI) for the ontology. This can be done for the header of the ontology in the "Active ontology" tab (Figure 7). An IRI is an internationalized form of an URI with an extended sequence of characters that describes the definite identification of resources (Duerst and Suignard 2005, p. 6). Therefore, every single ontology holds a unique IRI which can be changed continuously. All classes, properties and individuals within that ontology will belong to the defined IRI. In the case of merging ontologies we can distinguish the ontologies by their respective IRI identifications. Furthermore, an ontology may be described by some annotations like comments or labels etc. There is also the possibility to extend some terms to the annotation. We can therefore add some meta-information to describe the resource from the Dublin Core Metadata Initiative[9] by using the term *dcterms*.

[9] http://www.dublincore.org/specifications/dublin-core/dcmi-terms/ (checked on 07/12/2019)



Ontology header:

Ontology IRI http://www.semanticweb.org/master/ontologies/2019/1/road-data_ontology/

Ontology Version IRI e.g. http://www.semanticweb.org/master/ontologies/2019/1/road-data_ontology/1.0.0

Annotations

rdfs:label
The ontology describes road-data based on its attributes.

dcterms:creator
David Oberlerchner, BSc

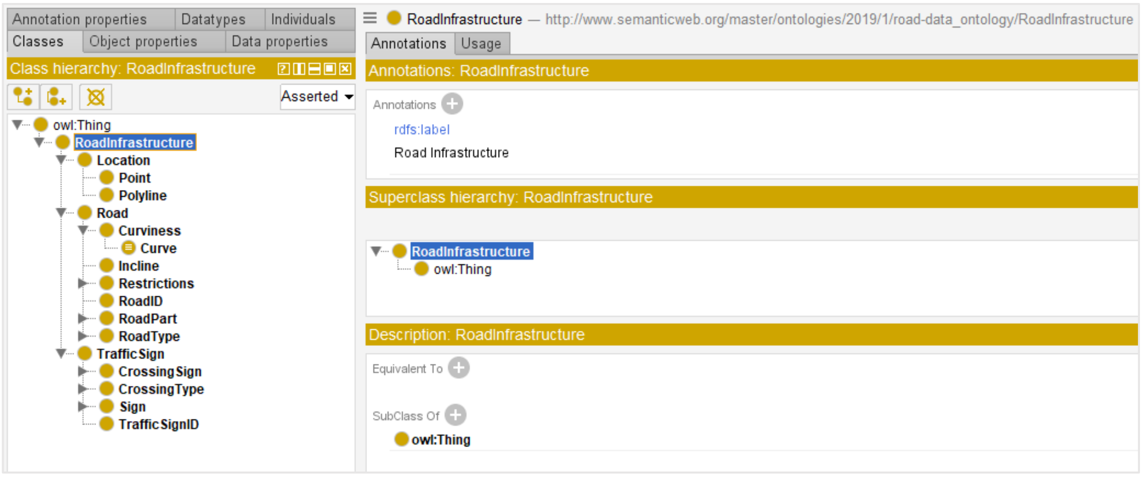
dcterms:rights
Technische Universität Graz
AVL List GmbH

Figure 7: Ontology header tab

3.3.9.2 Classes

When the setting of the general definition of the ontology is done, we can create and define the entities in the ontology, like classes, object properties, data properties and individuals. In the course of the work we are always able to adjust the structure of the dataset.

First we start with the creation of classes by creating a first subclass. Depending on how the structural configuration should be we will create the appropriate subclasses and also sibling classes. In contrast to subclasses, sibling classes share the same hierarchical level in the ontology. The basis class in an ontology is the class *owl:Thing*, which means that all underneath created classes are subclasses of *owl:Thing*: The main class is called *RoadInfrastructure,* which gather all the underlying classes. Those are the classes *Location*, *Road* and *TrafficSign*, where every class contains a different amount of subclasses. These three classes are defined as a collective term for specific classes. Altogether, there are 30 subclasses at the last level of the ontology hierarchy. Whilst the classes *Road* and *TrafficSign* are interpreted to be disjoint to each other, they are presented both within the class *Location*, even though in different subclasses. A road is presented in form of a polyline and a traffic sign must be in form of a point. This means that the two classes *Polyline* and *Point* are also disjoint to each other. The disjointedness of classes means that they are not allowed to own the same individuals. Consequently, a road cannot be a traffic sign and vice versa (Figure 8):


Annotation properties
Datatypes
Individuals
Classes
Object properties
Data properties
Class hierarchy: RoadInfrastructure
Asserted
Annotations
Usage
Annotations: RoadInfrastructure
Superclass hierarchy: RoadInfrastructure
Description: RoadInfrastructure
Equivalent To
SubClass Of

RoadInfrastructure — http://www.semanticweb.org/master/ontologies/2019/1/road-data_ontology/RoadInfrastructure

- owl:Thing
  - RoadInfrastructure
    - Location
      - Point
      - Polyline
    - Road
      - Curviness
        - Curve
      - Incline
      - Restrictions
      - RoadID
      - RoadPart
      - RoadType
    - TrafficSign
      - CrossingSign
      - CrossingType
      - Sign
      - TrafficSignID

Annotations: rdfs:label Road Infrastructure

Superclass hierarchy: RoadInfrastructure — owl:Thing

SubClass Of: owl:Thing

Figure 8: OWL classes tab

3.3.9.3 Properties

Object Properties

Object properties connect two or more individuals or classes. For that purpose we can further make use of specific property characteristics which allow us to enrich the meaning of the properties (Figure 9). The object properties may be described in different ways. However, they must refer to a specific domain with a particular range. Thereby, individuals will be connected from the domain to individuals from the range. In the following process of reasoning these connections will be used as axioms. It is also possible to assign multiple classes to the range of the object properties (Horridge 2011, p. 35).

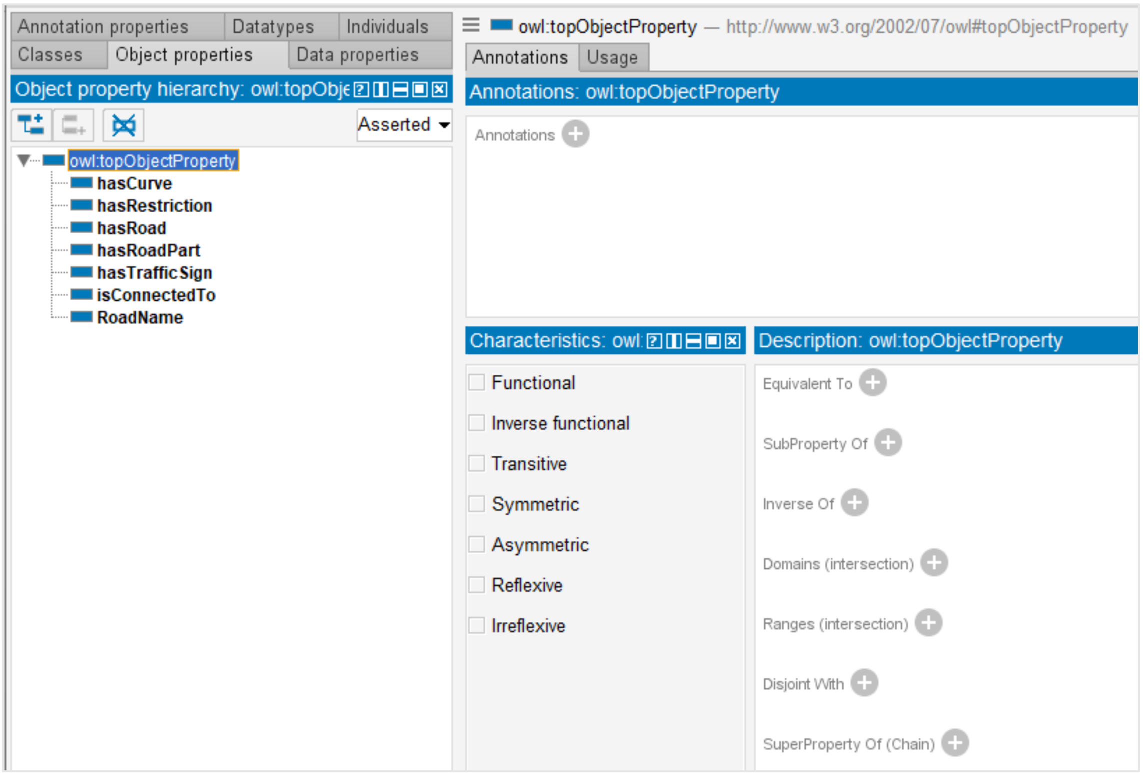
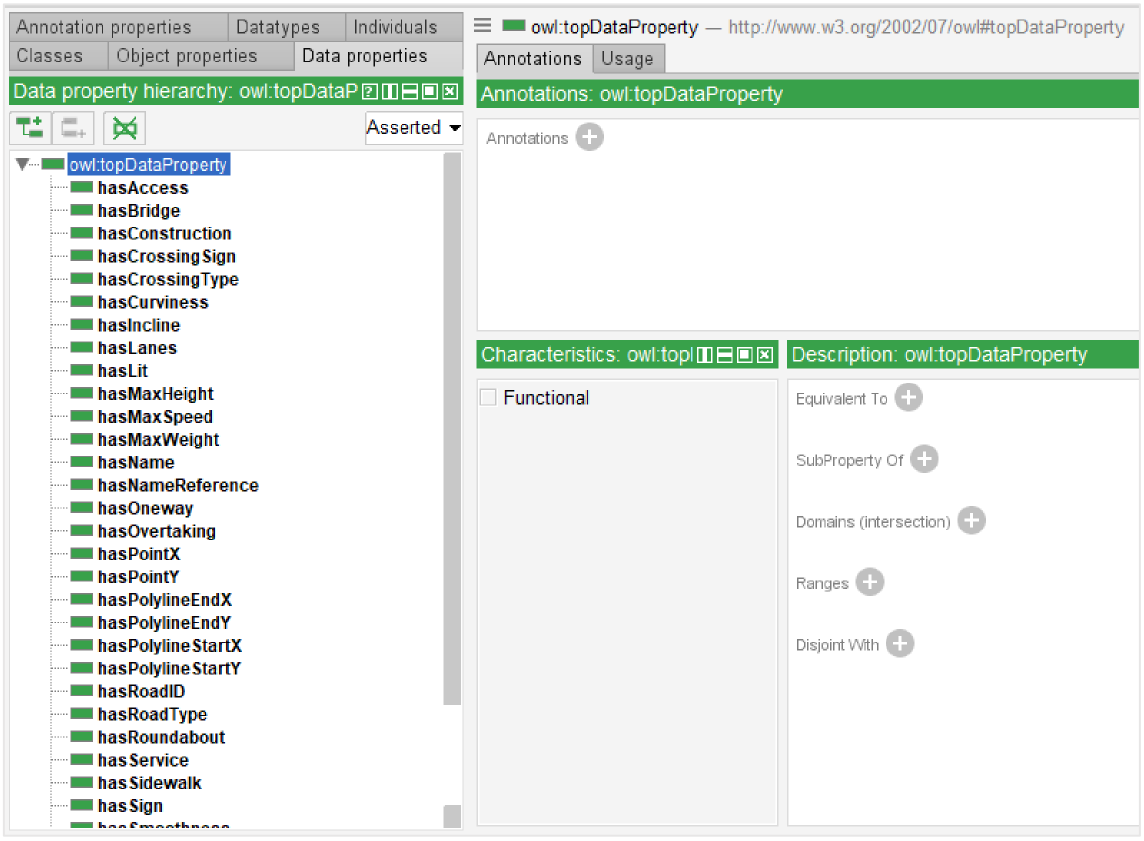


Figure 9: Object properties tab

Data Properties

Data properties are connecting classes and individuals with a specific datatype value of an XML schema or an RDF literal. The property characteristics of the object properties are mostly not available. The only characteristic we can apply is *Functional* for assigning exactly one value. As we use the description for object properties, we can also apply the description for domain and ranges for data properties. Therefore, the domain of the data property is linked to the corresponding class. The range is defined by a specific datatype for the output of the attributes of the data. For this thesis we only make use of three different datatypes. These are string, integer and decimal which are assigned to at least 34 data properties. Those data properties characterize the attributes of the road data and also of the traffic sign data (Figure 10).

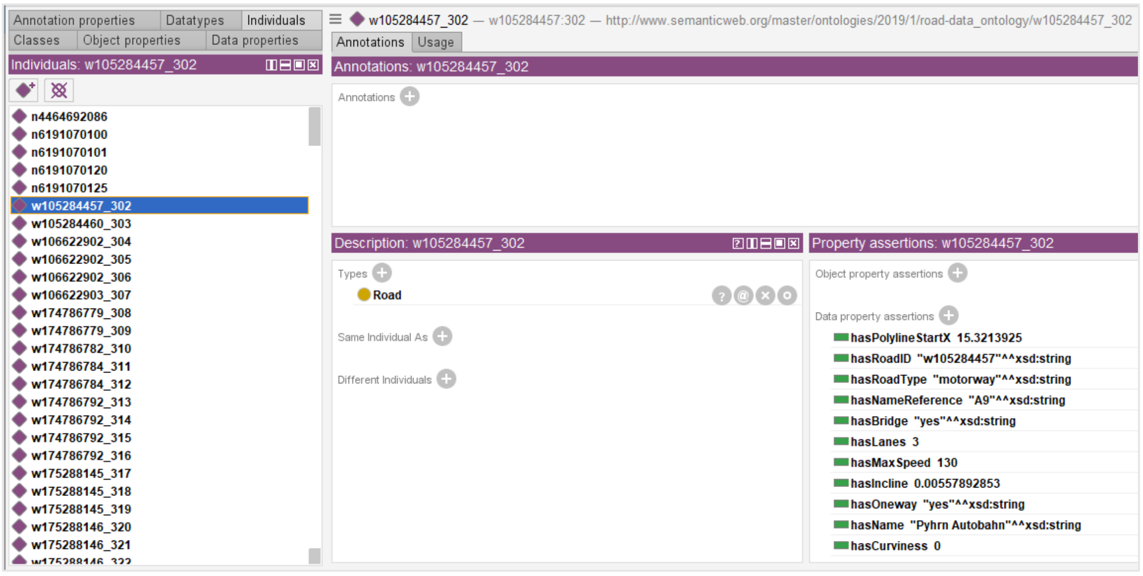

Annotation properties
Datatypes
Individuals
Classes
Object properties
Data properties
Data property hierarchy: owl:topDataP
Asserted
owl:topDataProperty — http://www.w3.org/2002/07/owl#topDataProperty
Annotations
Usage
Annotations: owl:topDataProperty
Annotations
Characteristics: owl:topI
Description: owl:topDataProperty
Functional
Equivalent To
SubProperty Of
Domains (intersection)
Ranges
Disjoint With

owl:topDataProperty
- hasAccess
- hasBridge
- hasConstruction
- hasCrossingSign
- hasCrossingType
- hasCurviness
- hasIncline
- hasLanes
- hasLit
- hasMaxHeight
- hasMaxSpeed
- hasMaxWeight
- hasName
- hasNameReference
- hasOneway
- hasOvertaking
- hasPointX
- hasPointY
- hasPolylineEndX
- hasPolylineEndY
- hasPolylineStartX
- hasPolylineStartY
- hasRoadID
- hasRoadType
- hasRoundabout
- hasService
- hasSidewalk
- hasSign

Figure 10: Datatype properties tab

3.3.9.4 Individuals

Within the individuals tab all the individuals, which represent real objects of the world, may be created. The input can be accomplished by entering the individuals one by one or by importing them as a group all at the same time. For this thesis we will import the individuals as axioms from an Excel workbook by using the embedded Protégé plugin Cellfie (Figure 11). Thus, every single individual is represented as a road segment in the data. We also have to assign the attributes which occur in the road dataset to the data properties.



w105284457_302 — w105284457:302 — http://www.semanticweb.org/master/ontologies/2019/1/road-data_ontology/w105284457_302

n4464692086
n6191070100
n6191070101
n6191070120
n6191070125
w105284457_302
w105284460_303
w106622902_304
w106622902_305
w106622902_306
w106622903_307
w174786779_308
w174786779_309
w174786782_310
w174786784_311
w174786784_312
w174786792_313
w174786792_314
w174786792_315
w174786792_316
w175288145_317
w175288145_318
w175288145_319
w175288146_320
w175288146_321
w175288146_322

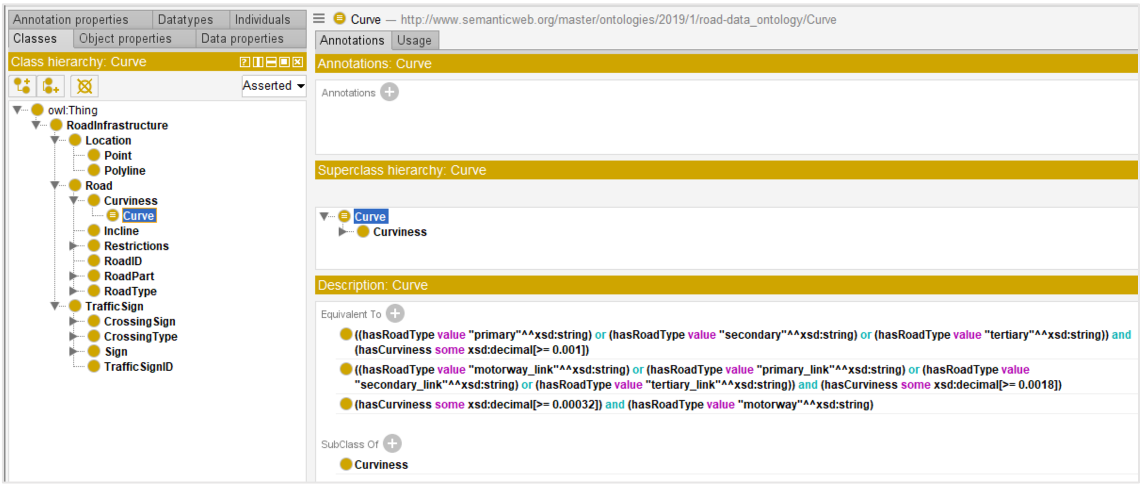
hasPolylineStartX 15.3213925
hasRoadID "w105284457"^^xsd:string
hasRoadType "motorway"^^xsd:string
hasNameReference "A9"^^xsd:string
hasBridge "yes"^^xsd:string
hasLanes 3
hasMaxSpeed 130
hasIncline 0.00557892853
hasOneway "yes"^^xsd:string
hasName "Pyhrn Autobahn"^^xsd:string
hasCurviness 0

Figure 11: Individuals tab

3.3.9.5 Restrictions

Within Protégé we can restrict the data in various ways. The most common type of restrictions is called "Existential Restriction". Thereby, individuals in a class are described, at least sharing one relationship with individuals from another class. The restriction may be defined by a specific property, whereas the relationship is represented through the term *some* between the two classes (Horridge 2011, p. 40). Another type of restriction is the "hasValue Restrictions". That restriction defines a group of individuals with at least one relationship to a specific individual. This specific individual is again described through a certain property (Horridge 2011, p. 91).

For the thesis we define "Essential Restrictions" for the determination of a curve. The "hasValue Restrictions" are used for the different road types. This is necessary as different road types result in different curve determinations. For example, a freeway has less sharp curves than a main road and, therefore, the threshold for curve determination is smaller. Consequently, we set the value for the data property *motorway* to *0.00032*. The road types for a *link* to the freeway and other main roads are set to a curvature value of *0.0018*. Furthermore, the types of the main roads *primary*, *secondary* and *tertiary* are set to *0.001* (Figure 12).



Equivalent To

((hasRoadType value "primary"^^xsd:string) or (hasRoadType value "secondary"^^xsd:string) or (hasRoadType value "tertiary"^^xsd:string)) and (hasCurviness some xsd:decimal[>= 0.001])

((hasRoadType value "motorway_link"^^xsd:string) or (hasRoadType value "primary_link"^^xsd:string) or (hasRoadType value "secondary_link"^^xsd:string) or (hasRoadType value "tertiary_link"^^xsd:string)) and (hasCurviness some xsd:decimal[>= 0.0018])

(hasCurviness some xsd:decimal[>= 0.00032]) and (hasRoadType value "motorway"^^xsd:string)

SubClass Of

Curviness

Figure 12: Restrictions

3.4 OpenDRIVE Simulation Data

3.4.1 OpenDRIVE Format

The OpenDRIVE[10] format is an open source specification and a de-facto standard within driving simulation tools for the description of road networks logics. It was established by a cooperation between the Daimler AG Driving Simulator and the VIRES Simulationstechnologie GmbH in 2005. The simulation format is based on the XML format and enables an exact description of roads with specific attributes from the real world. The advantage of the usage of the OpenDRIVE format is its exchangeability between different road network simulation tools for solutions like scenario simulations, traffic simulations and vehicle dynamics etc. Therefore, it is used by an increasing community in the driving simulation industry.

The format contains many attributes which supply the purpose of road network simulations. First the geometry of a road is given as a reference line with lane properties in terms of width and number and if available the elevation of the data. Additional attributes may be represented by traffic signals at or beside the road, the road type and its speed properties. Moreover, the road surface can be described by its material properties, and the infrastructure can be given as bridges and tunnels. Occasionally, road and roadside objects may be included for an extended road environment within the simulation process. Furthermore, an important feature of the format is the logical interconnection of junctions and of groups of junctions. However, the relationship between sequential lanes and road segments in a road simulation network is an essential feature (Menzel et al. 2018, p. 9).

Nowadays, data in the OpenDRIVE format is mostly generated manually, but in a very accurate way. As the available data are created on the basis of aerial images, this process turns out to be very time-consuming and cost-intensive. Furthermore, there is an insufficient amount of data available. This states the need of more data for a higher coverage, especially for autonomous driving simulations. The use of open source data

[10] http://www.opendrive.org/project.html (checked on 07/10/2019)

can fill this gap but may also lack information about some important descriptions of the road, because of its concept (Richter et al. 2016, p. 27).

3.4.2 Road Specifications

The road specifications have already been mentioned in chapter 3.4.1 as part of the OpenDRIVE format. Those categories are important for the data that are used within the generated road network and for the further generation of the OpenDRIVE files (Öberg 2012, p. 11).

3.4.2.1 Geometry

The basic geometries of a road are present in form of reference lines along a distance *s* at the center of a road. A reference line is composed of some elements of the basic types. Overall, five basic types are available (Figure 13):

- Straight line
- Spiral
- Curve or arc
- Cubic polynomial
- Parametric cubic curves (Dupuis, Marius e.a. VIRES GmbH 2015, p. 19)

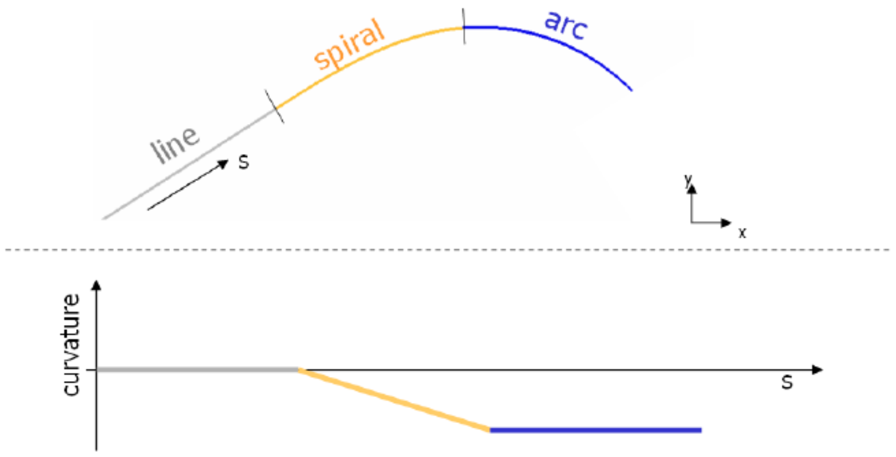


Figure 13: Reference line with the elements straight line, spiral and curve (from left to right) (Dupuis, Marius e.a. VIRES GmbH 2015, p. 19)

3.4.2.2 Lanes

Lanes are distinguished into right, center and left lanes, whereas there is always exactly one center lane in the middle of a road. However, there can be any number of the right and the left lanes which are located right or left of the center lane. Every lane has a successor and a predecessor from and to the next road segment. In addition, the lanes may be described by their lane type, by certain restrictions and by the width of a lane. In OpenDRIVE a polynomial will be used for describing the lane width. The lane width changes along a road segment. It depends on the width of the previous and the following lane especially at the beginning and the end of a road segment (Öberg 2012, p. 12). The center lane is always defined with the value *0* and may not explicitly contain a lane width. This lane is located along the reference line at a road segment (Figure 14) (Dupuis, Marius e.a. VIRES GmbH 2015, p. 20).

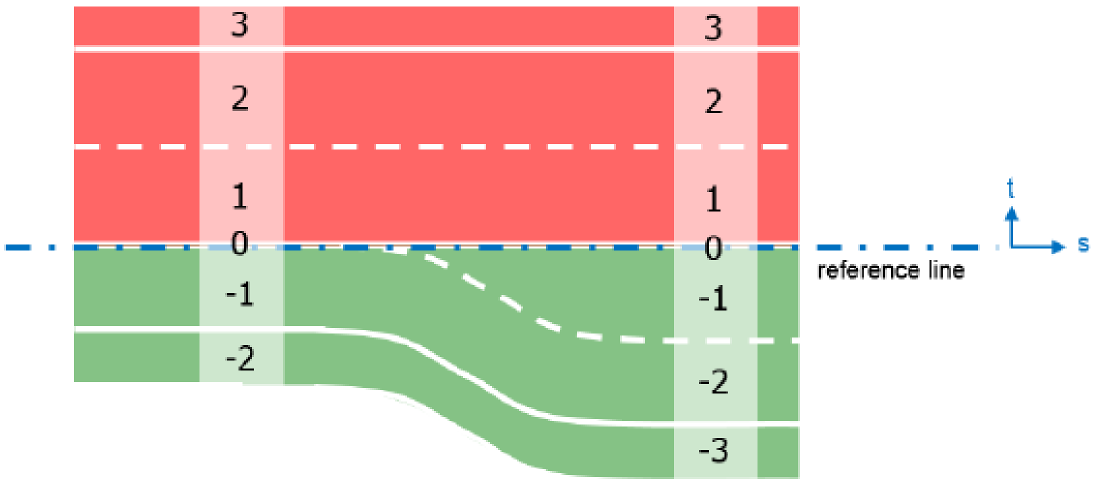
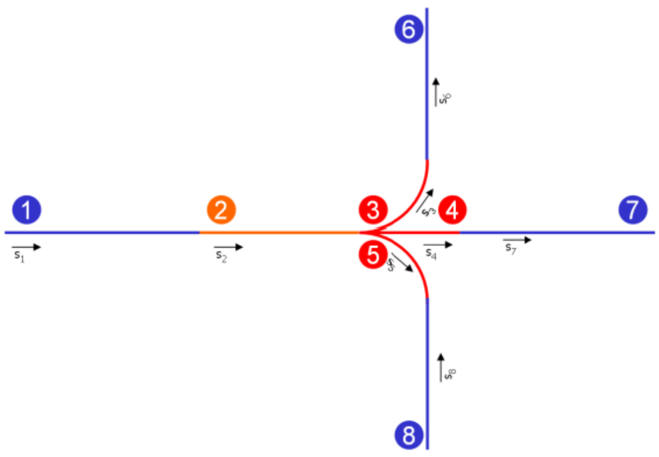


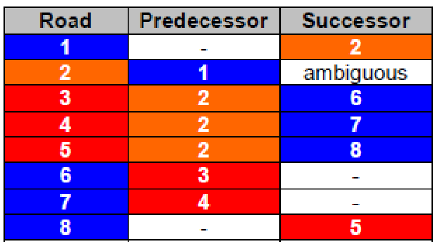
Figure 14: Lanes with number and width along a reference line (Dupuis, Marius e.a. VIRES GmbH 2015, p. 20)

3.4.2.3 Road Connection

For the navigation on a road network the connection between the sequential road segments must be known. That linkage can be accomplished by applying successors and predecessors at the IDs of the single road segments. For the standard case the connection with successors and predecessors is sufficient. If there is an ambiguous connection we have to use junctions instead.

The following example shows the connections, which are possible at a junction area. The road segments are each located along the reference line s_n at the road. The road segment number *2* is the “incoming road”, the numbers *3*, *4* and *5* are defined as “connecting roads” and the numbers *6*, *7* and *8* are called “outgoing roads” (Figure 15) (Dupuis, Marius e.a. VIRES GmbH 2015, p. 25).





Road	Predecessor	Successor
1	-	2
2	1	ambiguous
3	2	6
4	2	7
5	2	8
6	3	-
7	4	-
8	-	5

Figure 15: Road connection
(Dupuis, Marius e.a. VIRES GmbH 2015, p. 25)

A junction is a feature in the OpenDRIVE format where a road segment continues into one or more road segments. Therefore, we have a list of linkages with the name and the ID of the junction to other roads like mentioned above (Öberg 2012, p. 17). If a road belongs to a junction it will be treated as a path which connects the incoming road via the patch to the outgoing roads (Figure 16) (Dupuis, Marius e.a. VIRES GmbH 2015, p. 26).

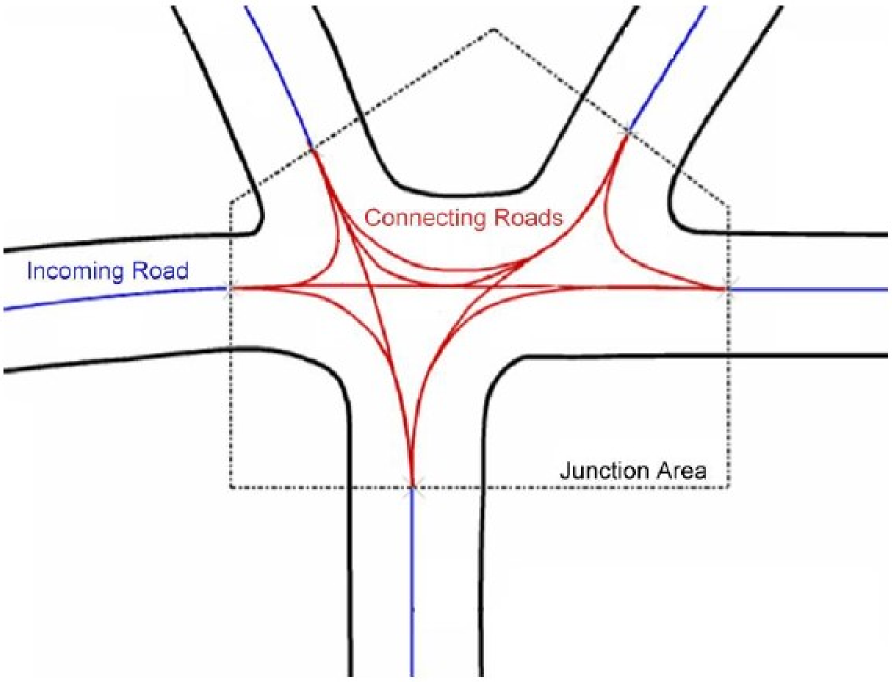


Figure 16: Junction with the path as a connection and the incoming and outgoing roads (Dupuis, Marius e.a. VIRES GmbH 2015, p. 26)

3.4.3 Tools for Road Network Generation

For the generation of road networks for simulation purpose just a few tools are available. Either the generation may be accomplished by using a conversion tool like e.g. Feature Manipulation Engine (FME)[11] or hale studio[12]. Those tools support the import of various data formats for spatial and non-spatial data. Among the plenty available formats for an export process the well-known XML format may also be generated. Unfortunately, a direct export of the OpenDRIVE format is not supported by those tools. In theory, it may be possible because of its XML based specifications (Coduro 2018, p. 76).

SUMO is a traffic simulation package, which supports the OpenDRIVE format as well as the import of a manageable amount of data formats from other traffic simulation tools like MATsim, Vissim and VISUM. Another option is the import of common spatial data

[11] https://www.safe.com/ (checked on 07/10/2019)
[12] https://www.wetransform.to/products/halestudio/ (checked on 07/10/2019)

formats like OSM, RoboCup and Shapefiles (Behrisch et al. 2011, p. 1). The SUMO package also allows to export data into various supported formats like the required OpenDRIVE format (Krajzewicz et al. 2012, p. 128).

3.4.4 Road Network Generation

A road network is described as a graph of a real world network with roads represented as edges and the intersections between them represented as nodes. The edges in the network are unidirectional connections between two nodes. The edges additionally contain the geometry, the allowed speed and a specific number of lanes with its changes in the number of lanes on the road segment. As mentioned above, the connection between lanes occurs across intersection, where the correct lane will be chosen to reach the subsequent lane.

The road network can be generated by using the SUMO application “netgenerate” for building a new network. Therefore some abstract types of road networks are available. Those can be a grid, a circular and a random network, where each of the algorithms have a specific set of allowed options to adjust the properties of a network.

By using already existing digital simulation road data we have to make use of the application “netconvert”. With this application the reading of the SUMO native XML based format is possible. This XML format is subdivided into five different parts. Two mandatory files include the description of edges and nodes. The other three optional files describe the edge types, the connections between the edges and the traffic lights. Furthermore, we can import road networks from other traffic simulation tools and also read other well-known and less common spatial data formats (Krajzewicz et al. 2012, pp. 128–129).

3.4.4.1 SUMO Import

For the import of spatial data into a road simulation network with the application "netconvert" we have two adequate options. Either we import the data in the original OSM format or we import the data as Shapefiles. For importing data as a SUMO network we can use a command line application which is an integrative part of the SUMO package.

To import the OSM data natively we must use the command *netconvert*. Access to the data can be accomplished by the basis option *--osm-files* and we save it to the SUMO internal road network with the option *-o*. This allows us a basic import of general data. However, in order to make sufficient use of the data we have to add some necessary options that define the import of the OSM data. Therefore, there are two OSM-specific options applicable for the import. The first option *--osm.oneway-spread-right* is necessary for the alignment of oneways. The second option *--osm.all-attributes* is used for importing all the attributes of an OSM dataset which may be relevant for the SUMO specific road network.

Moreover, we have to take into consideration that the specification of the velocity on a road in the SUMO road network is defined in meter per second. So it may be necessary to use the option *--speed-in-kmh*, as the original speed unit of an OSM dataset is also given in kilometers per hour. The OSM data are usually defined as sequential polylines with different changes of angles between them. This results in small gaps between polylines with sharp turns or a high curvature angle. With the option *--geometry.remove* the gap at those positions can be partly closed. Obviously, some gaps can still be left, because of a very high curvature angle which cannot be closed automatically. Therefore, we have to solve this problem manually with the application "NetEdit" after the entire import process. As to the process of importing OSM data there are still some options left. In the case of freeways a lot of acceleration lanes are often located with a higher angle connecting the main part of the freeway with the corresponding ramp. This problem may be solved by automatically identifying the acceleration lanes by using the option *--ramps.guess-acceleration-lanes* and allows a smooth transition from freeway links to the freeways. The last relevant option for the road representation is called *--junctions.join* for joining junctions that are close to each other, especially in an OSM- specific import process. Due to the polylineal structure it may be necessary to identify numerous polylines that converge at junction areas. For the driving process itself we have to define some options relating to turnarounds and lane connections. Therefore, we disallow turnarounds within the road network except for the endings of a road, where no connections are available. This can be accomplished with the option *--no-turnarounds.except-deadend*. With the option *--no-left-connections* the possibility for a lane change at junctions may be disallowed which can prohibit a forbidden U-turn at junction areas.

In addition, we will add two parameters that include the original OSM ID and the street names as a reference with the options *--output.original-names* and *--output.street-names*. With that we might accomplish the import of OSM data:

```
netconvert --osm-files RoadDataImport.osm -o RoadNetwork.net.xml
--osm.oneway-spread-right --osm.all-attributes --speed-in-kmh
--geometry.remove --roundabouts.guess --ramps.guess-acceleration-lanes
--no-turnarounds.except-deadend --no-left-connections --junctions.join
--output.original-names --output.street-names
```

The import of Shapefiles into the SUMO road network can be done similarly to the OSM import. Especially the general options concerning the import remain the same. However, there are some different Shapefile specific options for the import. First, the option*--shapefile-prefix* allows for the loading process of a Shapefile, whereas the option *-o* is used for saving the new road network. By importing a Shapefile we have to define the wanted attributes individually, unlike the OSM import process. Therefore, fixed options are given and can be applied for the street ID, the type of the street, the number of lanes and the allowed speed. For all other parameters it takes *--shapefile.add-params*. Supplementary parameters that do not exist in the original OSM data may be added. These parameters refer to the horizontal and the vertical curvature. During the import there can appear some problems on the edges which can be avoided with the option *--shapefile.use-defaults-on-failure*. Finally, we will add the option *--shapefile.guess-projection* for using the defined projection system of the data. Thereby the import of

Shapefile data is quite similar to the import of OSM data except of some specific import definitions:

```
netconvert -v --shapefile-prefix RoadDataImport -o RoadNetwork.net.xml
--shapefile.street-id segmentID --shapefile.type-id highway
--shapefile.laneNumber lanes --shapefile.speed maxspeed
--shapefile.add-params oneway;surface;name;ref;toll;bridge;tunnel;curviness;incline
--shapefile.use-defaults-on-failure --shapefile.guess-projection --speed-in-kmh
--geometry.remove --roundabouts.guess --ramps.guess
--no-turnarounds.except-deadend --no-left-connections --junctions.join
--output.original-names --output.street-names
```

After the data is successfully imported as a SUMO road network, we can view and optionally edit the gaps between the road segments within the SUMO tool “NetEdit”. The network consists of nodes and edges that result from the various street types. When editing the road network sufficiently all the gaps between the road segments should be eliminated by moving the affected nodes. We also have to check the connectivity between the lanes in a junction. If there are some connections between lanes which do not exist in reality, a manual change of these faulty connections is required (Figure 17).

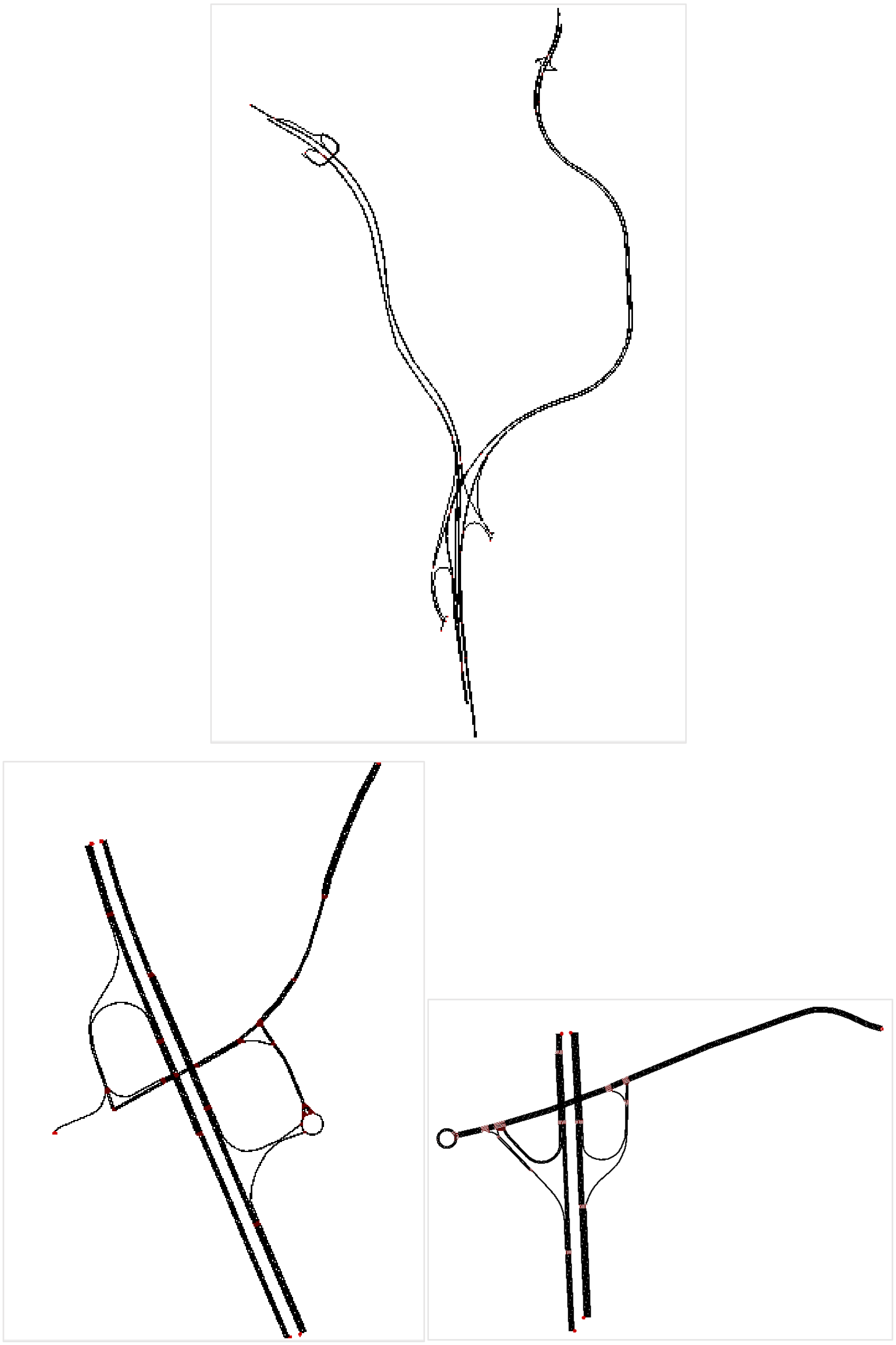


Figure 17: Road network of the area Peggau-Deutschfeistritz, Austria (up) Langenbruck (down left) and Allershausen, both Germany (down right)

3.4.4.2 OpenDRIVE Generation

Just like the import the export of the road network can be accomplished with the same option called *netconvert*. Therefore, we have to open the created road network with the option *–s* for the conversion process. For exporting the OpenDRIVE format the option *--opendrive-output* must be set. Additionally, some options for the export of the OpenDRIVE format are recommended. The option *--junctions.scurve-stretch* is needed for a smooth transition along of junction areas. By adjusting the value around *1.0* the stretching of the junction shape might be increased and reduced. Therefore, we set the value for the stretching of the junction to *1.5*. For the import process of the road data in this thesis the option *--output.original-names* is set. With that the original OSM ID has been assigned to the road network for every single road. The same option can be set for the export into OpenDRIVE for assigning the OSM ID to the exported format again:

```
netconvert -s RoadNetwork.net.xml --opendrive-output RoadNetworkExport.xodr
--junctions.scurve-stretch 1.5 --output.original-names
```

By opening the OpenDRIVE data in the OpenDRIVE Viewer the road networks can be visualized (Figure 18).

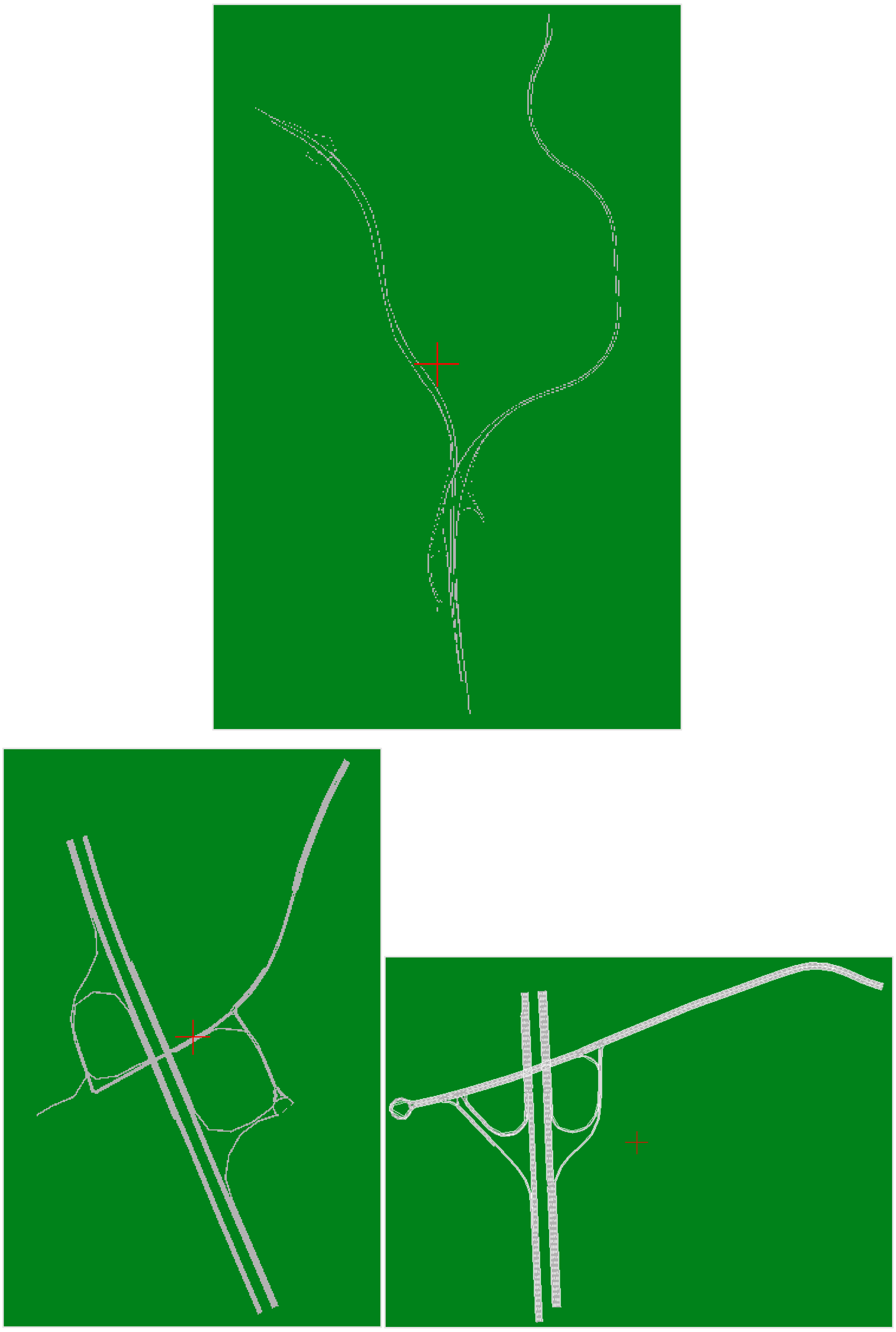


Figure 18: OpenDRIVE road data of the area Peggau-Deutschfeistritz, Austria (up), Langenbruck (down left) and Allershausen (down right)

4 Results and Discussion

4.1 Result of Curvature Calculation

The calculation of the curvature results in an appended column with the corresponding values to every single road segment (Figure 19). There we can see that the values are generally very low down to the value *0*. Road segments where the value *0* occurs represent straight elements without any curvature. Generally, those are polylines with only a start and an end point without any vertices within the polyline. For straight road segments with more than two vertices the curvature value is usually very low and, therefore, represented with a value around *10^{-4}* (green). For road segments within a curve the value is very low too, but with around *10^{-3}* significantly higher (red).

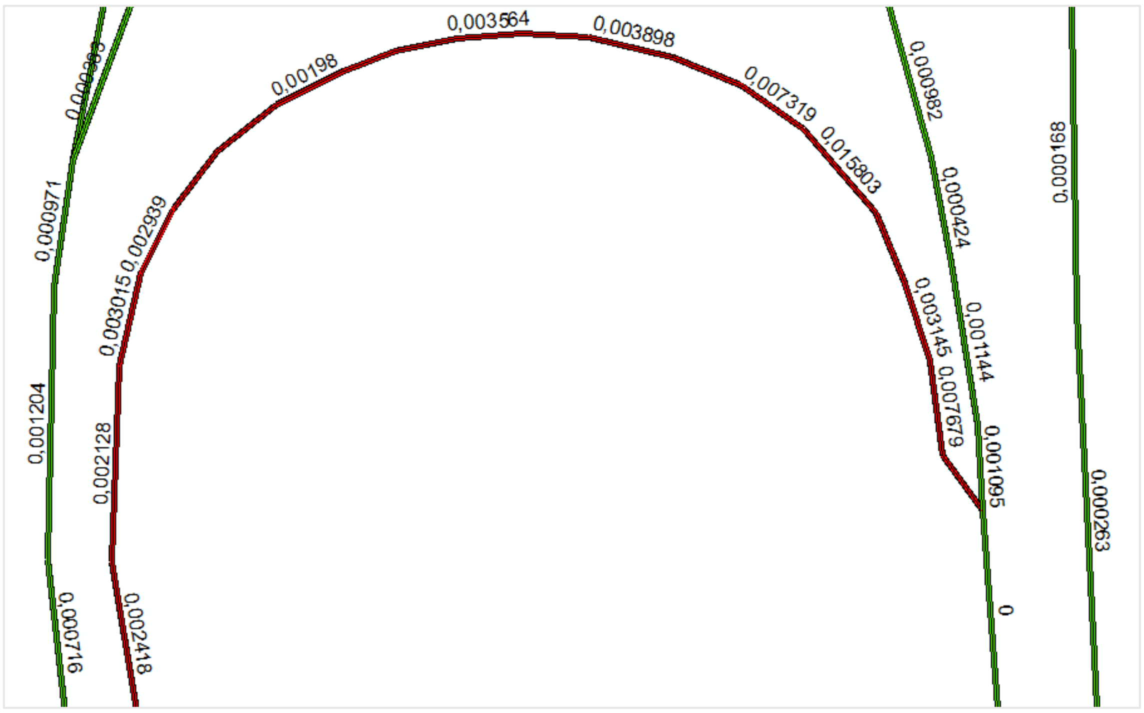


Figure 19: Curvature values of the road segments

For every road segment an incline value has been calculated. That value represents the slope in degrees of a road segment from the start to the end point (Figure 20). This value is usually low because of the fact that the road segments represent short parts of the entire road. Therefore, the value for the incline between the start and the end point is small.

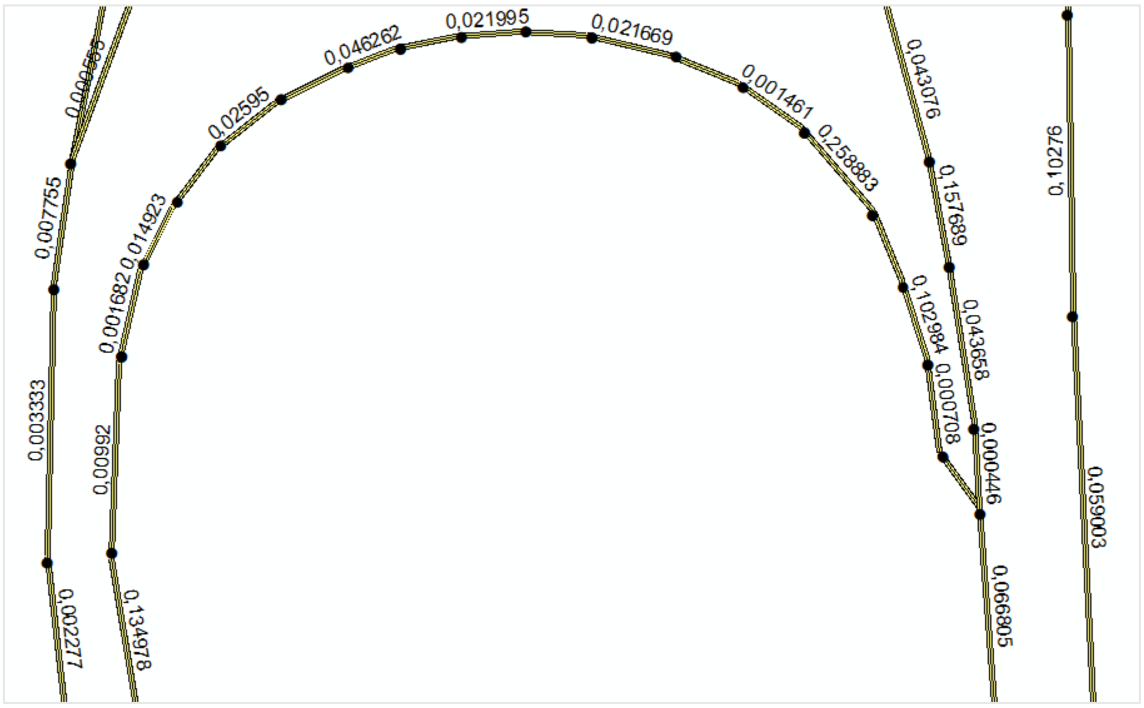
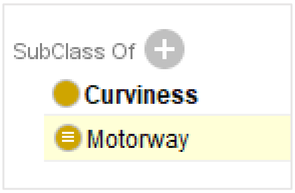


Figure 20: Incline values of the road segments

4.2 Road Ontology

The ontology for the road data represents some major attributes of the OSM dataset and also holds the description of the curvature of the road segments. By making use of the Pellet reasoner we are able to represent the knowledge within the ontology. Therefore, appropriate thresholds for identifying the curve have been set according to the curvature values and the type of the road segments.

After the reasoning process we can see that the class *Curve* is now also a subclass of the already existing class *Motorway*, as this class encompasses some common individuals (Figure 21).

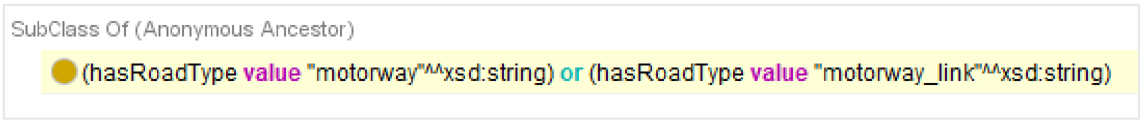

SubClass Of

Curviness

Motorway

Figure 21: Superclass of the class Curve

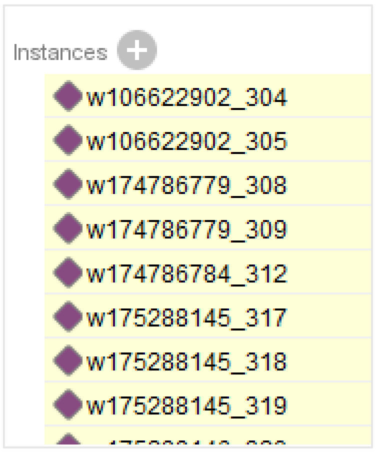
Furthermore, processing of the reasoner results in a new class which depends on the definition of the class expression. Therefore, the expression for the values *motorway* and *motorway_link* is represented as a new but anonymous superclass class within the ontology (Figure 22).


SubClass Of (Anonymous Ancestor)

(hasRoadType value "motorway"^^xsd:string) or (hasRoadType value "motorway_link"^^xsd:string)

Figure 22: Anonymous superclasses of the class Curve

All individuals that match the class expression are assigned to the class as instances. This means that all the added individuals within the class are represented as a part of a curve (Figure 23).


Instances

w106622902_304

w106622902_305

w174786779_308

w174786779_309

w174786784_312

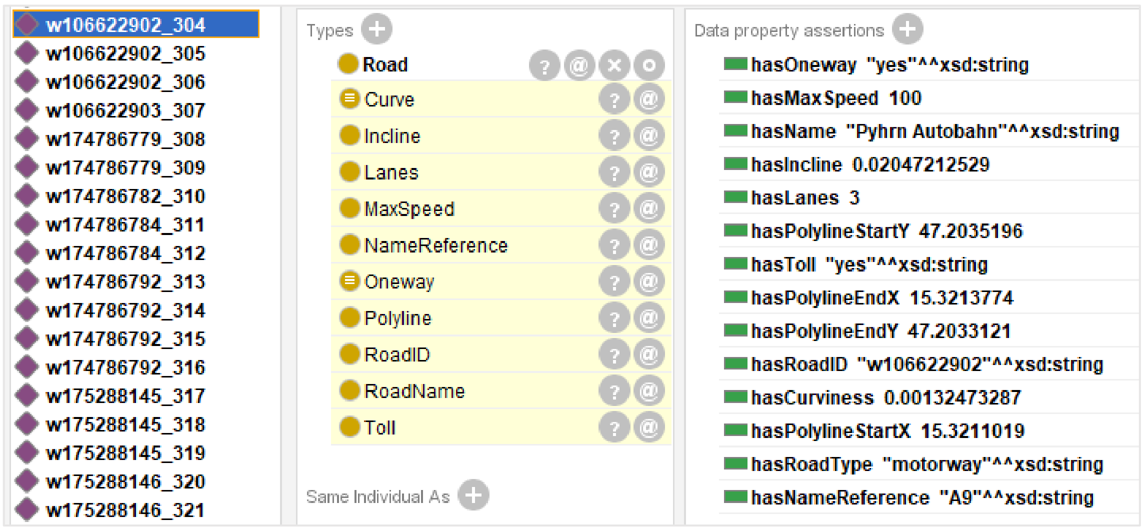
w175288145_317

w175288145_318

w175288145_319

Figure 23: Individuals as instances of the class

The individuals are represented under the “Individuals tab” with its descriptions and property assertions (Figure 24). The “Types” of the road segments as individuals are added to the description, i.e. some classes have been added to the main class *Road* or *TrafficSign* holding the assigned properties. Those properties of the individuals are shown as data property assertions.



w106622902_304
w106622902_305
w106622902_306
w106622903_307
w174786779_308
w174786779_309
w174786782_310
w174786784_311
w174786784_312
w174786792_313
w174786792_314
w174786792_315
w174786792_316
w175288145_317
w175288145_318
w175288145_319
w175288146_320
w175288146_321

Types
Road
Curve
Incline
Lanes
MaxSpeed
NameReference
Oneway
Polyline
RoadID
RoadName
Toll

Same Individual As

Data property assertions
hasOneway "yes"^^xsd:string
hasMaxSpeed 100
hasName "Pyhrn Autobahn"^^xsd:string
hasIncline 0.02047212529
hasLanes 3
hasPolylineStartY 47.2035196
hasToll "yes"^^xsd:string
hasPolylineEndX 15.3213774
hasPolylineEndY 47.2033121
hasRoadID "w106622902"^^xsd:string
hasCurviness 0.00132473287
hasPolylineStartX 15.3211019
hasRoadType "motorway"^^xsd:string
hasNameReference "A9"^^xsd:string

Figure 24: Individuals tab

4.3 OpenDRIVE Road Data

The OpenDRIVE data are available for the three test areas Peggau-Deutschfeistritz in Austria, Langenbruck and Allershausen in Germany. Data gained within these three areas have been exported with different road types. The area Peggau-Deutschfeistritz solely consists of freeways and connections to the freeways, whereas the other two areas Langenbruck and Allershausen also contain different main roads. This results in a different conversion procedure of the data.

By handling the road network from Shapefile data sources it is only possible to either use roads with unidirectional or bidirectional characteristics. However, due to the different handling and the available settings within the simulation toll SUMO this is not a sufficient solution. By contrast, when applying OSM data we can use all the road types for the OpenDRIVE conversion, which is allowed because of the key-value pair structure of the OSM data that can be recognized by the simulation tool SUMO. Therefore, the direct conversion from OSM data is a better solution because of its applicability than the conversion from Shapefiles (Figure 25).

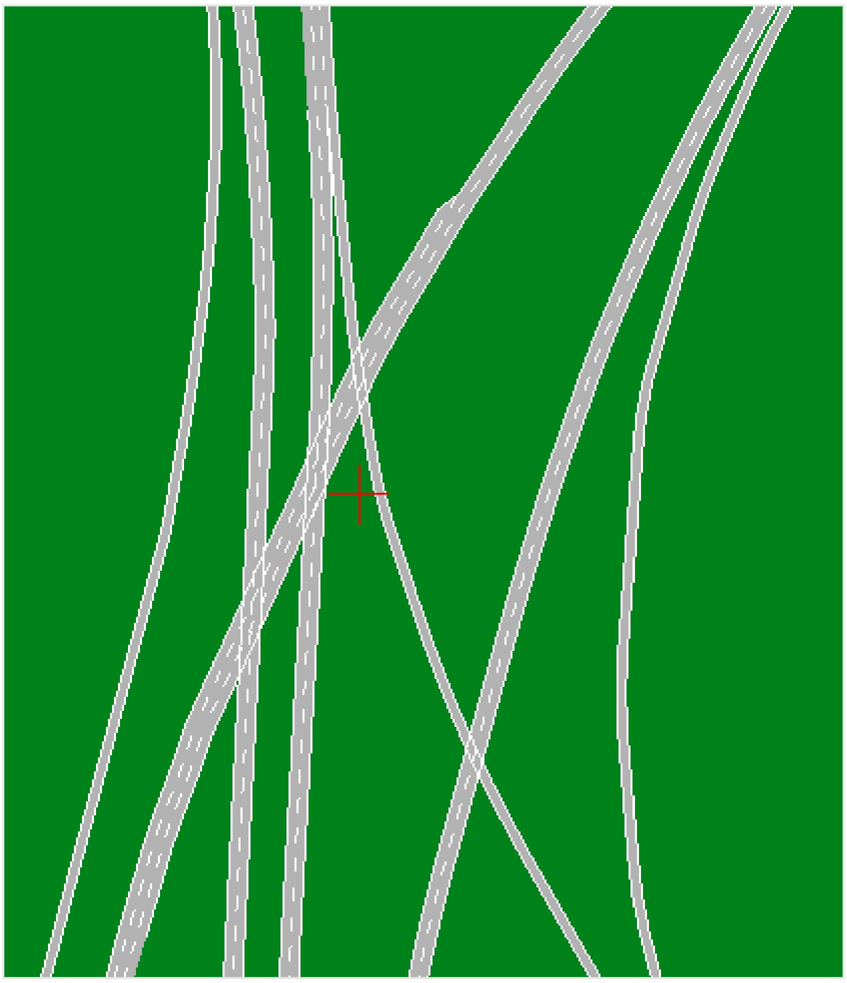


Figure 25: OpenDRIVE data the junction at Peggau-Deutschfeistritz, Austria

The challenges in the conversion to OpenDRIVE lie within the correct representation of the data for real world applications. Therefore, the road network may not contain any gaps and the transitions of junctions and lanes must be smooth. In principle, a transition of lanes is possible. However, correct visualizations of transitions of lanes are scarce (Figure 26). If the number of lanes increases, the transition at the new lane is visualized in a smooth way. The bug occurs when the number of lanes decreases which is visualized with an abrupt cut at the lane. Fortunately, this bug does not affect the simulation itself, as the road ends anyway. Therefore, the application lacks a correct visualization.

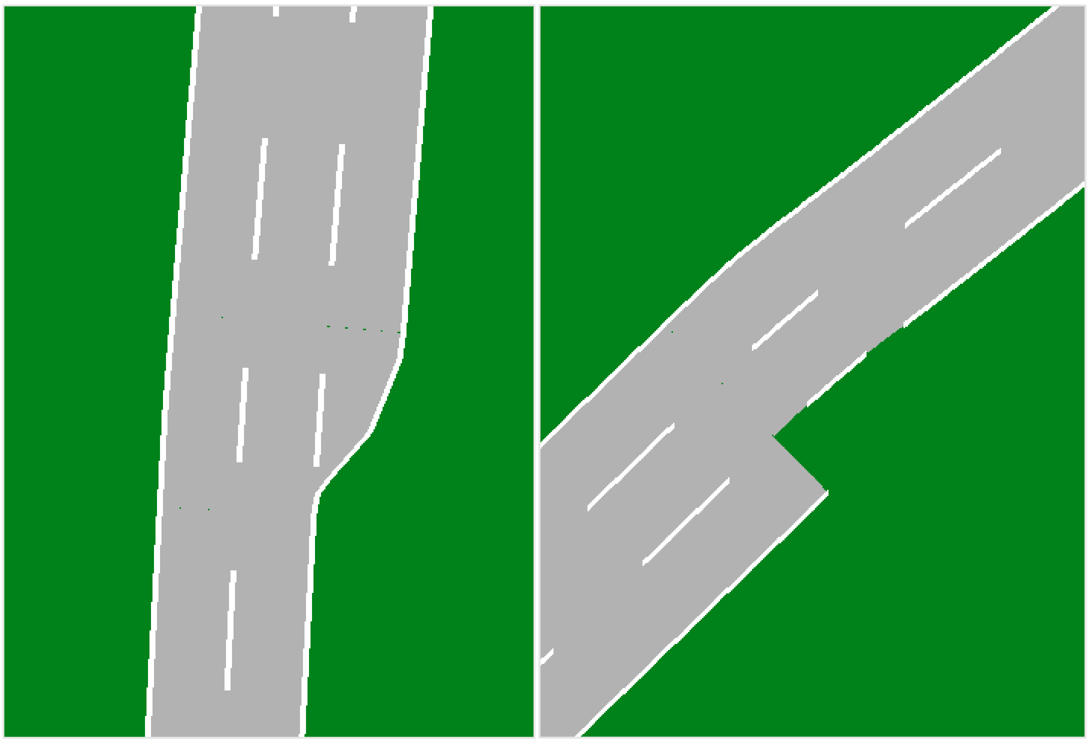


Figure 26: Increase (left) and decrease (right) of the lane number

As can be seen in Figure 27, the access and exit lanes are visualized as smoothly as possible. The representation depends on the angle of the linking lanes to the following lane, i.e. the less acute the angle the smoother the transition area. For example, if a lane has a less acute angle (as shown by the increase in the left graphic below), the transition of that lane into the main road is not completely smooth, which can be seen at the slight bending at the transition. If a lane has a more acute angle (as shown by the decrease in the right graphic below), the transition of the lane into the main road is represented in a very smooth way.

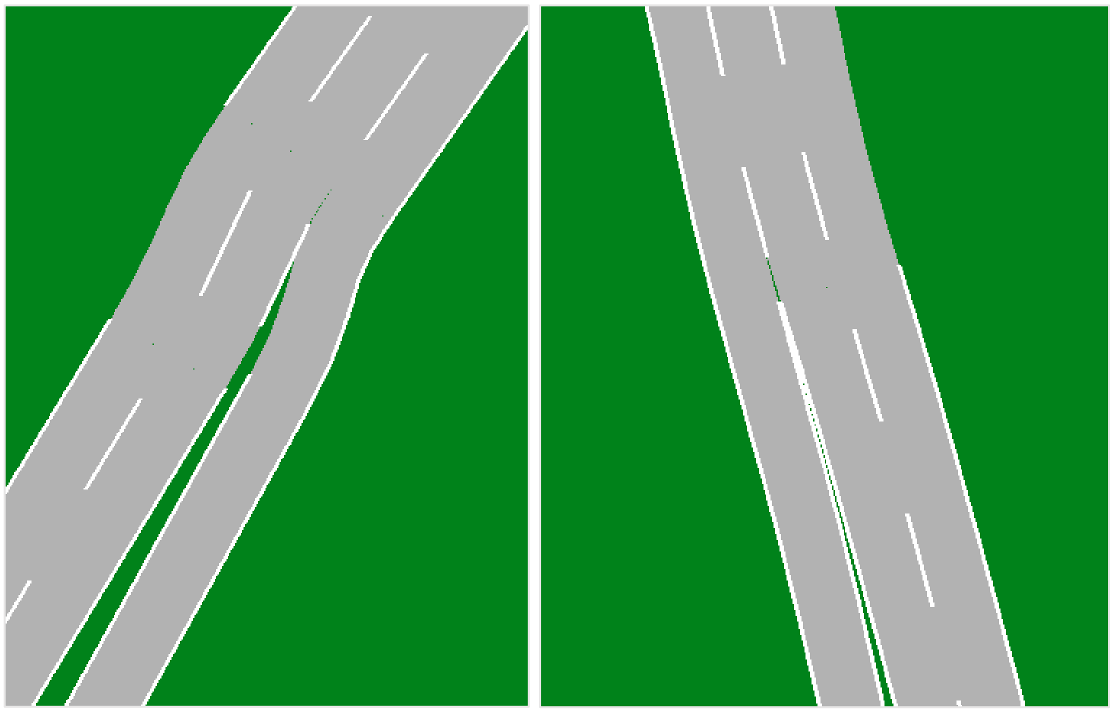


Figure 27: Access (left) and exit (right) lane

4.4 OpenDRIVE Validation

In order to correctly use OpenDRIVE data a validation of them must be performed. As a road is represented through a polyline, it is vital to check the connectivity and the curvature between sequential polylines. Therefore, different criteria will be assumed for the usability within the Virtual Test Drive (VTD). These criteria may be fixed within a configuration file. During the so called parsing process of the data more details can be outputted optionally. For determining the maximum deviation between two connected roads we can set a value for the tolerance of the connected reference lines in one direction. A tolerance value will be set for the maximum deviation between the natural length of the elements and their start position along its reference line and also between the total length of all original elements and the deviated total length of the road. Furthermore, the minimum length of a road element as well as the maximum curvature value of a road element must be described. At last, it is necessary to set a tolerance value for the maximum distance between the positions of two points to be considered as either identical or non-identical nodes. For each criterion default values are set and can be optimized for the validation process within the configuration file.

The validation of the OpenDRIVE data has been performed with the OpenDRIVE Validator by using the default criteria options within the configuration file. The results describe the type of observations on the data. The observations are classified into three outputs types called “INFO”, “WARNING” and “ERROR”. Whenever a conservation is finished, any database elements would be reported that fall within the observation categories. Every single reported element includes the line of the original OpenDRIVE data which can be retrieved. Due to the fact that the implementation has been conducted by SUMO, it is assumed that no errors should occur. The validation checks the consistency by observing road data. An error may occur if the geometry of at least one road within a road network is incomplete.

The validation of the road network for all three test areas has been completed successfully. After the validation process the output stated several warnings in the data concerning the inconsistence of angles at the end of a road. This result does not necessarily have to be a reason for concern as those affected nodes represent the end of a road anyway. Therefore, it is not possible to continue a potential simulation at those nodes. The remaining data do not show any warnings or errors.

5 Conclusion and Recommendation

5.1 Conclusion

This thesis focused on the calculation of the curvature as a parameter for the identification of a curve at a road segment within a road data model on the one hand. On the other hand, the generation of OpenDRIVE files for simulation purposes for autonomous driving was accomplished. Therefore, some steps had to be followed in order to reach the objectives of the research.

First we used open source data from OSM, because of their free availability and their high coverage. Furthermore, we converted OSM raw data into the well-known Shapefile format. On the basis of these Shapefile data the calculation of the curvature parameter was done with the help of Python scripting by using two specific approaches:

- By using a moving window the radius of an osculating circle through three points of a polyline was calculated. Additionally, the length of the polyline segment between those three points was determined. The curvature of a polyline segment is the inversion of the radius. By using the length of the corresponding arc the determination of an average curvature parameter was calculated. We started with the first three points of the polyline and shifted the moving window by dropping the first point in the polyline and adding the next one.
- In order to determine the curvature parameter in a polyline with only a start and an end point we calculated the curvature without the usage of a moving window. Therefore, a value of zero was assigned to the curvature parameter, as no curvature occurred.

The results are represented by curvature values corresponding to their polyline segments of a road. However, they do not specify if the road segment is a part of a curve. Therefore, the knowledge we need for the determination of curves was represented within an ontology by using the Protégé editor.

The ontology was built for road data with its original attributes and the calculated curvature parameter for the identification of curves. For determining the curve within the road an ontology was created containing all original attributes of OSM data as well as the calculated curvature parameter. Every single road segment represents an individual with specific road data attributes. Those data attributes are constructed in form of datatype properties which are included in the corresponding classes of the same name. In order to identify the curves a restriction for different types of roads was implemented by setting the parameter value according to their affiliation to freeways, main roads or accesses and exits. Road segments which fulfill those requirements were identified as a part of a curve. Therefore, those road segments are assigned to a class that holds all existing curves within the ontology. Consequently, the road segments are represented as a part of a curve. By using a reasoner we checked the classes, properties and individuals within the ontology and proved its consistency.

In order to generate OpenDRIVE files with open source data we searched for applications supporting the OpenDRIVE data format for simulation tasks. The SUMO application proved to be an adequate tool for the direct conversion of OSM and Shapefile data into the OpenDRIVE format. The special structure of OpenDRIVE data allows for a direct conversion of OSM and Shapefile data into the OpenDRIVE data format.

The conversion process was done by first importing the data as a road network containing all the relevant data attributes. Eventually, some adjustments to the road network must be done manually resulting from possible geometrical inconsistencies within data from open sources. The export of the road network was directly performed by converting it into the OpenDRIVE data format as it is a supported data format within the application. Specific attributes of the input data can be used within the OpenDRIVE data due to its structure. Therefore, we cannot represent all attributes from the open source data in the OpenDRIVE data format.

By validating the OpenDRIVE data we checked the consistency of the geometry for the connections of sequential road segments and for a continuous course of the road without the occurrence of any gaps between them.

5.2 Recommendation

In order to calculate the curvature of road segments we used the common approach following the determination of the radius of an osculating circle. However, we are convinced that there might be other approaches. In particular, when it comes to the usage of the angle of the curvature an equivalent approach could be applied.

The creation of an ontology for road data was done by using the well-known and well-developed ontology editor Protégé. Nevertheless, it should be noted that numerous other ontology editors exist that can be used for similar purposes. There are even editors that have been specifically designed for spatial data. However, none of them is as comprehensive as the Protégé editor. In fact, the availability of ontologies for road data remains limited. Therefore, it is a challenging task to advance the development of specific road data models and the refinement of existing models.

The demand for reliable OpenDRIVE data will especially increase in the field of autonomous driving. Since the creation of OpenDRIVE data is mostly done manually for small areas and is associated with high construction costs, we believe that the approach using open source data is certainly an adequate one. However, the creation of larger areas from digital road data generally lacks consistencies due to the conversion process. Therefore, manual editing is indispensable.

Currently, only a limited number of conversion tools that directly support the OpenDRIVE format can be found on the market. Another promising challenge regarding the conversion process is the application of specific conversion tools, which do not directly support OpenDRIVE.

6 Publication bibliography

Alian, Sahar (2007): Identifying Curviness of Overpass Mountain Roads from Remote Sensing Data. Master's dissertation. K. N. Toosi University of Technology, Tehran. International Institute for Geo-Information and Earth Observation (ITC).

Andrášik, Richard; Bíl, Michal; Janoška, Zbynek; Valentová, Veronika (2013): Identification of Curves and Straight Sections on Road Networks from Digital Vector Data. In *Transactions on Transport Science* 6 (2), pp. 73–80.

Arens, Y.; Chee, C.; Hsu, C.-N.; Knoblock, C. (1993): Retrieving and integrating data from multiple information sources. In *International Journal of Intelligent and Cooperative Information Systems* 2 (2), pp. 127–158.

Armand, Alexandre; Filliat, David; Ibañez-Guzman, Javier (Eds.) (2014): Ontology-Based Context Awareness for Driving Assistance Systems. 2014 IEEE Intelligent Vehicles Symposium Proceedings. Dearborn, MI, USA, June 8-11, 2014. IEEE.

Bagschik, Gerrit; Menzel, Till; Maurer, Markus (Eds.) (2018): Ontology based Scene Creation for theDevelopment of Automated Vehicles. 2018 IEEE Intelligent Vehicles Symposium (IV). Changshu, China, June 26-30, 2018. IEEE.

Ballatore, Andrea; Bertolotto, Michaela; Wilson, David C. (2013): Geographic Knowledge Extraction and Semantic Similarity in OpenStreetMap. In *Knowledge and Information Systems* 37 (1).

Bauer, Bernhard; Fischer, Wolf; Lautenbacher, Florian; Roser, Stephan (Eds.) (2008): Konzepte und Techniken für das Semantic Web. Universität Augsburg.

Behrisch, Michael; Bieker, Laura; Erdmann, Jakob; Krajzewicz, Daniel (Eds.) (2011): SUMO – Simulation of Urban MObility. An Overview. The Third International Conference on Advances (SIMUL). Barcelona, Spain, October 23-29, 2011. Institute of Transportation Systems.

Calvanese, D.; Giacomo, G. D.; Lenzerini, M.; Nardi, D.; Rosati, R. (1998): Description logic framework for information integration. In *Proceedings of the Internaltional Conference on Principles of Knowledge Representaiton and Reasoning*, pp. 2–13.

Coduro, Theresa (2018): Straßenraummodellierung mittels Mobile Mapping in OpenDRIVE und CityGML sowie Entwicklung geeigneter Visualisierungsmethoden. Master's dissertation. Technische Universität München, München. Ingenieurfakultät Bau Geo Umwelt.

Duerst, Martin; Suignard, Michel (2005): Internationalized Resource Identifiers (IRIs). Available online at https://tools.ietf.org/html/rfc3987#section-1.3, checked on 7/12/2019.

Dupuis, Marius e.a. VIRES GmbH (2015): OpenDRIVE® Format Specification, Rev. 1.4. H. VIRES Simulationstechnologie GmbH (VI2014.107).

Garcia-Molina, H.; Papakonstantinou, Y.; Quass, D.; Rajaraman, A.; Sagiv, Y.; Ullman, J.; Widom, J. (Eds.) (1995): The TSIMMIS approach to mediation: Data models and languages. Next Generation Information Technologies and Systems (NGITS-95). Naharia, Israel, June 27-29, 1995.

Haklay, Mordechai; Weber, Patrick (2008): OpenStreetMap: User-Generated Street Maps. In *IEEE Pervasive Computing* 7 (4), pp. 12–18.

Hall, Mark (2006): A Semantic Similarity Measure for Formal Ontologies. (With an application to ontologies of a geographic kind). Master's dissertation. Alpe-Adria Universität, Klagenfurt. Institut für Informatik-Systeme.

Horridge, Matthew (2011): A Practical Guide To Building OWL Ontologies Using Protégé 4 and CO-ODE Tools. Edition 1.3. With assistance of Holger Knublauch, Alan Rector, Robert Stevens, Chris Wroe, Simon Jupp, Georgina Moulton et al. The University Of Manchester. Manchester.

Jitkajornwanich, Kulsawasd; Elmasri, Ramez; Li, Chengkai; McEnery, John (Eds.) (2011): Formalization of 2-D spatial ontology and OWL/Protégé realization. Proceedings of the

International Workshop on Semantic Web Information Management. Athens, Greece, June 12-16,2011. Univ. of Texas at Arlington (9).

Jokar Arsanjani, Jamal; Mooney, Peter; Zipf, Alexander; Helbich, Marco (2015): OpenStreetMap in GIScience: Experiences, Research, Applications. With assistance of Roland Olbricht.

Kashyap, V.; Sheth, A. (1998): Semantic heterogeneity in global information systems. The role of metadata, context and ontologies. In *Cooperative Information Systems*, pp. 139–178.

Katsumi, Megan; Fox, Mark (2018): Ontologies for transportation research: A survey. In *Transportation Research Part C* 89, pp. 53–82.

Krajzewicz, Daniel; Erdmann, Jakob; Behrisch, Michael; Bieker, Laura (2012): Recent Development and Applications of SUMO – Simulation of Urban MObility. In *International Journal on Advances in Systems and Measurements* 5 (3 & 4), pp. 128–138.

Kühn, Wolfgang; Müller, Michael; Höppner, Tom (2017): Road Data as Prior Knowledge for Highly Automated Driving. In *Transportation Research Procedia* 27, pp. 222–229.

Linková, Zdeňka; Nedbal, Radim; Řimnáč, Martin (2005): Building Ontologies for GIS. Institute of Computer Science, Academy of Sciences of the Czech Republic (932).

Lutz, Michael; Klien, Eva (2006): Ontology-based Retrieval of Geographic Information. In *International Journal of Geographic Information Science* 20 (3), pp. 105–132.

McGuinness, Deborah L.; van Harmelen, Frank (2004): OWL Web Ontology Language. Overview. In *W3C Recommendation*.

Menzel, T.; Bagschik, G.; Isensee, L.; Schomburg, A.; Maurer, M. (2018): Detaillierung einer stichwortbasierten Szenariobeschreibung für die Durchführung in der Simulation am Beispiel von Szenarien auf deutschen Autobahnen.

Navratil, Gerhard (2012): Curviness as a Parameter for Route Determination. In *GI_Forum 2012: Geovisualization, Society and Learning*, pp. 355–364.

Öberg, Cim (2012): HORN - Hank and OpenDRIVE. Road Networks: An editor for creating HANK scenarios while working with OpenDRIVE. Bachelor's thesis. Linköpings Universitet, Linköping, Sweden. Department of Computer and Information Science.

Pfeiffer, Sabine (2010): Entwicklung einer Ontologie für die wissensbasierte Erschließung des ISDC-Repository und die Visualisierung kontextrelevanter semantischer Zusammenhänge. Master's dissertation. Hochschule Neubrandenburg, Neubrandenburg. Studiengang Geoinformatik.

Preece, A.; Hui, K.-Y.; Gray, W.; Marti, P.; Bench-Capon, T.; Jones, D.; Cui, Z. (Eds.) (1999): KRAFT architecture for knowledge fusion and transformation. 9th SGES International Conference on Knowledgebased Systems and Applied Artificial Intelligence (ES '99). Berlin, Germany: Springer.

QGIS Development Team (2019): QGIS Geographisches Informationssystem. Open Source Geospatial Foundation Projekt. Version 3.6.0. Available online at https://qgis.org/en/site/, checked on 6/7/2019.

Richter, Andreas; Friedl, Hartmut; Scholz, Michael (Eds.) (2016): Beyond OSM – Alternative DataSources and Approaches EnhancingGeneration of Road Networks forTraffic and Driving Simulations. Proceedings of the SUMO2016 - Traffic, Mobility, and Logistics. Berlin, Germany, May 23-25, 2016. DLR Institut für Verkehrssystemtechnik. 30 volumes: Deutsche Zentrum für Luft- und Raumfahrt.

Stanford University/Protégé community (2019): Protégé. Wiki. Version 5.5.0. Available online at https://protegewiki.stanford.edu/wiki/Main_Page, checked on 6/7/2019.

Stuckenschmidt, H. (2003): Ontology-Based Information Sharing in Weakly Structured Environments. Dissertation. Vrije Universiteit, Amsterdam. Dutch Research School for Information and Knowledge Systems (SIKS).

Studer, Rudi; Benjamins, Richard; Fensel, Dieter (1998): Knowledge Engineering: Principles and Methods. In *Data & Knowledge Engineering* 25 (1-2), pp. 161–197.

Visser, U.; Stuckenschmidt, H.; Schuster, G.; Vögele, T. (2002a): Ontologies for geographic information processing. In *Computers & Geosciences* 28, pp. 103–117.

Visser, Ubbo; Stuckenschmidt, Heiner; Schlieder, Christoph (2002b): Interoperability in GIS - Enabling Technologies. In *5th AGILE Conference on Geographic Information Science*, pp. 291–297.

Wang, Yandong; Gong, Jianya; Wu, Xiaohuang (2007): Geospatial semantic interoperability based on ontology. In *Geo-spatial Information Science* 10 (3), pp. 204–207.

Zhao, Lihua; Ichise, Ryutaro; Mita, Seiichi; Sasaki, Yutaka (Eds.) (2015): Core Ontologies for Safe Autonomous Driving. The 35th Semantic Web & Ontology Workshop. Toyota Technological Institute.

Appendices

A Overpass Queries for retrieving OSM data

```
/*
This query looks for ways with the given main key highway.
The region includes the test area around Peggau-Deutschfeistritz!
*/
[out:xml][timeout:300];
// gather results
(
  // query part for ways: “highway=*”
  way["highway"="motorway"](47.17314, 15.31567, 47.20999, 15.3467);
  way["highway"="motorway_link"](47.17314, 15.31567, 47.20999, 15.3467);
  way["construction"="motorway"](47.17314, 15.31567, 47.20999, 15.3467);
);
// print results
out body;
>;
out skel qt;


/*
This query looks for ways with the given main key highway.
The region includes the test area around Langenbruck!
*/
[out:xml][timeout:300];
// gather results
(
  // query part for ways: “highway=*”
  way["highway"="motorway"](48.64132, 11.51397, 48.64893, 11.52007);
  way["highway"="primary"](48.64132, 11.51397, 48.64893, 11.52007);
  way["highway"="motorway_link"](48.64132, 11.51397, 48.64893, 11.52007);
  way["highway"="primary_link"](48.64132, 11.51397, 48.64893, 11.52007);
  way["construction"="motorway"](48.64132, 11.51397, 48.64893, 11.52007);
  way["construction"="primary"](48.64132, 11.51397, 48.64893, 11.52007);
);
// print results
out body;
>;
out skel qt;
/*
This query looks for ways with the given main key highway.
The region includes the test area around Allershausen!
*/
```

```
[out:xml][timeout:300];
// gather results
(
  // query part for ways: “highway=*”
  way["highway"="motorway"](48.42615, 11.58563, 48.4303, 11.5965);
  way["highway"="primary"](48.42615, 11.58563, 48.4303, 11.5965);
  way["highway"="secondary"](48.42615, 11.58563, 48.4303, 11.5965);
  way["highway"="motorway_link"](48.42615, 11.58563, 48.4303, 11.5965);
  way["highway"="primary_link"](48.42615, 11.58563, 48.4303, 11.5965);
  way["highway"="secondary_link"](48.42615, 11.58563, 48.4303, 11.5965);
  way["construction"="motorway"](48.42615, 11.58563, 48.4303, 11.5965);
  way["construction"="primary"](48.42615, 11.58563, 48.4303, 11.5965);
  way["construction"="secondary"](48.42615, 11.58563, 48.4303, 11.5965);
);
// print results
out body;
>;
out skel qt;
```

B Python code for calculating the horizontal curvature of a road segment

```
# Curvature of OSM-road data
# Author: David Oberlerchner

# The programm takes calculations on a road feature, which consists polylines
# The road feature will be split at its vertices to apply a curvature value to each line
# The first loop searches in each row (polyine) of the feature for vertices
# Polylines with two vertices and more are separately treated
# Polyines with at least three vertices a calculated value based on the curvature is
assigned
# Polylines with two vertices a value of 0 is assigned, because they are straight
# The curvature value is based on the radius of curvature from three sequential points
# The curvature will be calculated for the vertices in each row
# The curvature from each polyine is written in a list
# The curvature from each polyline is then written in a common list for the roads
# The second loop updates the split feature with the values from the curvature list for
the roads

def curviness(inFeature, roadFeature):
   # Import system modules
   import arcpy
   from math import radians, acos, sin, cos, sqrt

   # Execute AddField for feature and create curvature list for the road data
```

```
    arcpy.AddField_management(roadFeature, "curviness", "DOUBLE")
    curvatureListRoads = []

    # WGS-84 ellipsoid semi-major axis
    R = 6371000.785

    # Enter for loop for each polyline feature
    print("Calculate the average curvature!")
    with arcpy.da.SearchCursor(inFeature,["OID@", "SHAPE@"]) as cursor:
        for row in cursor:
            print(row[0])
            vertexList = []
            curvatureList = []
            # Step through each vertex in the polyline feature
            for pnt in row[1].getPart(0):
                vertexList.append((pnt.X, pnt.Y))
            vertexCount = len(vertexList)
            i = 0
            averageCurvature = 0.0
            # Polyline has only two vertices
            if vertexCount == 2:
                averageCurvature = 0.0
                curvatureList.append(averageCurvature)
            # Check count of polyline vertices
            elif vertexCount >= 3:
                lengthList = []
                # Coordinate pairs to count until you run out of vertex triplicate
                while i + 3 <= vertexCount:
                    # Coordinates of the three vertices
                    x1 = radians(vertexList[i][0])
                    y1 = radians(vertexList[i][1])
                    x2 = radians(vertexList[i + 1][0])
                    y2 = radians(vertexList[i + 1][1])
                    x3 = radians(vertexList[i + 2][0])
                    y3 = radians(vertexList[i + 2][1])
                    # Center point (x, y) of a circle through three points
                    A = x1 * (y2 - y3) - y1 * (x2 - x3) + x2 * y3 - x3 * y2
                    B = (x1**2 + y1**2) * (y3 - y2) + (x2**2 + y2**2) * (y1 - y3) + (x3**2 +
y3**2) * (y2 - y1)
                    C = (x1**2 + y1**2) * (x2 - x3) + (x2**2 + y2**2) * (x3 - x1) + (x3**2 +
y3**2) * (x1 - x2)
                    # In case of zero matrix values a low value is assigned
                    if A == 0:
                        A = 0.0000001
                    elif B == 0:
```

```
                B = 0.0000001
            elif C == 0:
                C = 0.0000001
            # Center point of a circle
            xCenter = -(B / (2 * A))
            yCenter = -(C / (2 * A))
            # Radius of a circle
            radius = R * acos(sin(xCenter) * sin(x2) + cos(xCenter) * cos(x2) *
cos(yCenter - y2))
            # Length of the parts of the polyline
            length1 = R * acos(sin(x1) * sin(x2) + cos(x1) * cos(x2) * cos(y1 - y2))
            length2 = R * acos(sin(x2) * sin(x3) + cos(x2) * cos(x3) * cos(y2 - y3))
            length = length1 + length2
            # Invert the radius and calculate average curvature
            curvature = 1 / radius
            curvatureCount = len(curvatureList)
            # Check curvature list and append the average curvature
            # Polyline has more than three vertices
            if vertexCount >= 4:
                # First line gets first curvature value
                if curvatureCount == 0:
                    averageCurvature = curvature * length1 / length
                # Last line gets last curvature value
                elif curvatureCount == vertexCount - 3:
                    averageCurvature = (curvature + curvatureList[-1]) * length1 / (length +
lengthList[-1])
                    curvatureList.append(averageCurvature)
                    averageCurvature = curvature * length2 / length
                # Curvature average of lines with two curvature within a polyline
                else:
                    averageCurvature = (curvature + curvatureList[-1]) * length1 / (length +
lengthList[-1])
            # Polyline has exact three vertices
            elif vertexCount == 3:
                averageCurvature = curvature * length1 / length
                curvatureList.append(averageCurvature)
                averageCurvature = curvature * length2 / length
            lengthList.append(length)
            curvatureList.append(averageCurvature)
            i = i + 1
        # Extend the curvature list for the road data with the average curvature
        curvatureListRoads.extend(curvatureList)
    del row, cursor

    # Execute AddField for the coordinates and for the road segment ID
    arcpy.AddField_management(roadFeature, "startX", "DOUBLE")
    arcpy.AddField_management(roadFeature, "startY", "DOUBLE")
    arcpy.AddField_management(roadFeature, "endX", "DOUBLE")
    arcpy.AddField_management(roadFeature, "endY", "DOUBLE")
    arcpy.AddField_management(roadFeature, "segmentID", "TEXT")

    # Update the cursor with the average curvature
    print("Update the average curvature!")
    pointer = 0
    with arcpy.da.UpdateCursor(roadFeature,["OID@", "curviness", "SHAPE@", "startX",
"startY", "endX", "endY", "full_id", "segmentID"]) as cursor:
        for row in cursor:
            print(row[0])
            row[1] = curvatureListRoads[pointer]
            pointer += 1
            # Set start coordinates
            row[3] = row[2].firstPoint.X
            row[4] = row[2].firstPoint.Y
            # Set end coordinates
            row[5] = row[2].lastPoint.X
            row[6] = row[2].lastPoint.Y
            # Calculate the segment ID of the road
            row[8] = row[7] + "_" + str(pointer)
            cursor.updateRow(row)
    del row, cursor
```

C Python code for calculating the vertical curvature of a road segment

```
# Curvature of OSM-road data
# Author: David Oberlerchner

# The programm takes calculations on a road feature, which consists polylines
# A digital elevation model provides information about the surface
# The average slope for each polyline will be calculated

def incline(featureVertices, curvatureFeature):
    # Import system modules
    import arcpy
    from math import atan
    from arcpy.sa import *

    # List for DEM values at the points
    heightList = []
    print("Assign point values to a list!")
    with arcpy.da.UpdateCursor(featureVertices,["OID@", "Elev"]) as cursor:
        for row in cursor:
            heightList.append(row[1])
    del row, cursor

    # Execute AddField for feature
    arcpy.AddField_management(curvatureFeature, "incDeg", "DOUBLE")

    # Update the cursor with the average curvature
    print("Update the incline!")
    pointer = 0
    startPointer = 0
    endPointer = 1
    with arcpy.da.UpdateCursor(curvatureFeature,["OID@", "SHAPE@", "incDeg"]) as
cursor:
        for row in cursor:
            print(row[0])
            # Length of a polyline in meters
            length = row[1].getLength("GEODESIC", "METERS")
            # Incline in percent
            startHeight = heightList[startPointer]
            endHeight = heightList[endPointer]
            row[2] = atan((max(startHeight, endHeight) - min(startHeight, endHeight)) /
length)
            pointer += 1
            startPointer += 2
            endPointer += 2
            cursor.updateRow(row)
    del row, cursor
```

D Python code for updating the curvature

```
# Curvature of OSM-road data
# Author: David Oberlerchner

# The programm takes calculations on a road feature, which consists polylines
# The road feature will be split at its vertices to apply a curvature value to each line
# The first loop searches in each row (polyine) of the feature for vertices
# Polylines with two vertices and more are separately treated
# Polyines with at least three vertices a calculated value based on the curvature is
assigned
# Polylines with two vertices a value of 0 is assigned, because they are straight
# The curvature value is based on the radius of curvature from three sequential points
# The curvature will be calculated for the vertices in each row
# The curvature from each polyine is written in a list
# The curvature from each polyline is then written in a common list for the roads
# The second loop updates the split feature with the values from the curvature list for
the roads

# Import system modules
import arcpy, datetime
from arcpy import env
from arcpy.sa import *

# Calculating the time of process
print("Start of the calculations!")
start = datetime.datetime.now()

# Set environment settings
env.overwriteOutput = True
env.workspace = r"C:\Users\David\Documents\Masterarbeit\Daten"

# Set local variables
inFeature = r"osm_road-data\shape\peggaudeutschfeistritz_motorway.shp"
#inFeature = r"osm_road-data\shape\langenbruck_oneway.shp"
#inFeature = r"osm_road-data\shape\langenbruck_bidirectional.shp"
roadFeature = r"osm_road-
data\road_FINAL\road_peggaudeutschfeistritz_motorway.shp"
#roadFeature = r"osm_road-data\road_FINAL\road_langenbruck_oneway.shp"
#roadFeature = r"osm_road-data\road_FINAL\road_langenbruck_bidirectional.shp"
featureVertices = r"osm_road-
data\featureVertices\featureVertices_peggaudeutschfeistritz_motorway.shp"
#featureVertices = r"osm_road-
data\featureVertices\featureVertices_langenbruck_oneway.shp"
#featureVertices = r"osm_road-
data\featureVertices\featureVertices_langenbruck_bidirectional.shp"
inDEM = r"DEM\ALS_DGM_10M_UTM33N\ALS_DGM_10M_UTM33N.asc"

# Check out the ArcGIS Spatial Analyst extension license
arcpy.CheckOutExtension("Spatial")

# Run Splitline to get new lines, each of which is between two vertices
print("Split polyline at vertices")
arcpy.SplitLine_management(inFeature, roadFeature)

# Execute FeatureVerticesToPoints
print("Polyline feature vertices to points!")
arcpy.FeatureVerticesToPoints_management(roadFeature, featureVertices,
"BOTH_ENDS")
```

```
# Execute ExtractValuesToPoints
print("Extract DEM values to the points!")
ExtractMultiValuesToPoints(featureVertices, [[inDEM, "Elev"]], "BILINEAR")

# Function for curviness
import curviness_averageLength
curviness_averageLength.curviness(inFeature, roadFeature)

# Function for incline
import incline_degree
incline_degree.incline(featureVertices, roadFeature)

# End the time of calculation
print("End of the calculations in {0} seconds!".format(datetime.datetime.now() - start))
```