CommonRoad: Documentation of the XML Format

(Version 2017a)

Markus Koschi, Stefanie Manzinger, and Matthias Althoff

Technische Universität München, 85748 Garching, Germany

Abstract

This document presents the XML format in *CommonRoad* for specifying road traffic scenarios. The provided scenarios in *CommonRoad* are described in the scenario documentation. The *CommonRoad* XML files are composed of (1) a formal representation of the road network, (2) static and dynamic obstacles, and (3) the planning problem for the ego vehicle(s). So far, we have not discovered any limitations when building scenarios using the proposed format.

Contents

1 Introduction

Within *CommonRoad* [1], XML files are used to store the data for specific driving scenarios. In the following, we present the basic structure of the XML files for building scenarios consisting of a road network, static and dynamic obstacles, initial states, and goal regions. The road network is composed from lanelets [2], which are interconnected road segments, geometrically represented by a left and right boundary.

2 Overview

The *CommonRoad* XML files are composed of (1) a formal representation of the road network (see Section 4.4), (2) static and dynamic obstacles (see Section 4.2), and (3) the planning problem of the ego vehicle(s) (see Section 4.3) as shown in Fig. 1. In non-collaborative scenarios, only one ego vehicle exists, while in collaborating scenarios the motion planning has to be conducted for several ego vehicles. Within the XML file, we use the auxiliary elements *state*, *occupancy*, and further primitives (see Fig. 2 and Section 4.1). Please note that the numbers in square brackets denote the number of allowed elements (while `N` can be different for each element), the data in round brackets the attributes of an element, `ref to` a reference to one element, the data behind a double column the value of the element.

```
/
├── [1..N] lanelet (id)
│   ├── [1] leftBound
│   │   ├── [2..N] point
│   │   └── [0..1] lineMarking
│   ├── [1] rightBound
│   │   ├── [2..N] point
│   │   └── [0..1] lineMarking
│   ├── [0..N] predecessor (ref to lanelet)
│   ├── [0..N] successor (ref to lanelet)
│   ├── [0..1] adjacentLeft (ref to lanelet, drivingDir=same/opposite)
│   ├── [0..1] adjacentRight (ref to lanelet, drivingDir=same/opposite)
│   ├── [0..1] speedLimit
│   └── [0..N] trafficSign
├── [0..N] obstacle (id)
│   ├── [1] role: static
│   ├── [1] type: parkedVehicle/.../unknown
│   └── [1] shape
├── [0..N] obstacle (id)
│   ├── [1] role: dynamic
│   ├── [1] type: car/truck/.../unknown
│   ├── [1] shape
│   └── [1] trajectory
│       └── [1..N] state
│   OR
├── [0..N] obstacle (id)
│   ├── [1] role: dynamic
│   ├── [1] type: car/truck/.../unknown
│   └── [1] occupancySet
│       └── [1..N] occupancy
│   OR
├── [0..N] obstacle (id)
│   ├── [1] role: dynamic
│   ├── [1] type: car/truck/.../unknown
│   ├── [1] shape
│   └── [1] probabilityDistribution
└── [1..N] planningProblem (id)
    ├── [1] initialState
    └── [1] goalRegion
        └── [1..N] state
```

Figure 1: Structure of the XML files encoding each scenario

```
/
├── point OR center
│   ├── [1] x
│   └── [1] y
├── rectangle
│   ├── [1] length
│   ├── [1] width
│   ├── [0..1] orientation
│   └── [0..1] center
├── circle
│   ├── [1] radius
│   └── [0..1] center
├── polygon
│   └── [3..N] point
├── shape
│   └── [1..N] rectangle/circle/polygon
├── position
│   ├── [1] point
│   │   OR
│   ├── [1..N] rectangle/circle/polygon
│   │   OR
│   └── [1..N] lanelet (ref to lanelet)
├── orientation/time/velocity/acceleration
│   ├── [1] exact
│   │   OR
│   ├── [1] intervalStart
│   └── [1] intervalEnd
├── state
│   ├── [1] position
│   ├── [1] orientation
│   ├── [1] time
│   ├── [0..1] velocity
│   └── [0..1] acceleration
├── initialState
│   ├── [1] position
│   ├── [1] velocity
│   ├── [1] orientation
│   ├── [1] yawRate
│   ├── [1] slipAngle
│   └── [1] time
└── occupancy
    ├── [1] shape
    └── [1] time
```

Figure 2: Auxiliary elements of the XML file

For our benchmarks we use *lanelets* [2] as drivable road segments to represent the road network. A lanelet is defined by its *left* and *right boundary*, where each boundary is represented by an array of points (a polyline), as shown in Fig. 3. We have chosen lanelets since they are as expressible as other formats, such as e.g. OpenDRIVE[1], yet have a lightweight and extensible representation. By using lanelets, the road network is modeled as a *directed graph*, where each node has four types of outgoing edges: successor, predecessor, left, right (see Fig. 1; predecessor is not required but added for implementation reasons). Our road networks are constructed in a way that all laterally adjacent lanes have the same length, as shown in Fig. 3 and explained in more detail in Section 4.4.

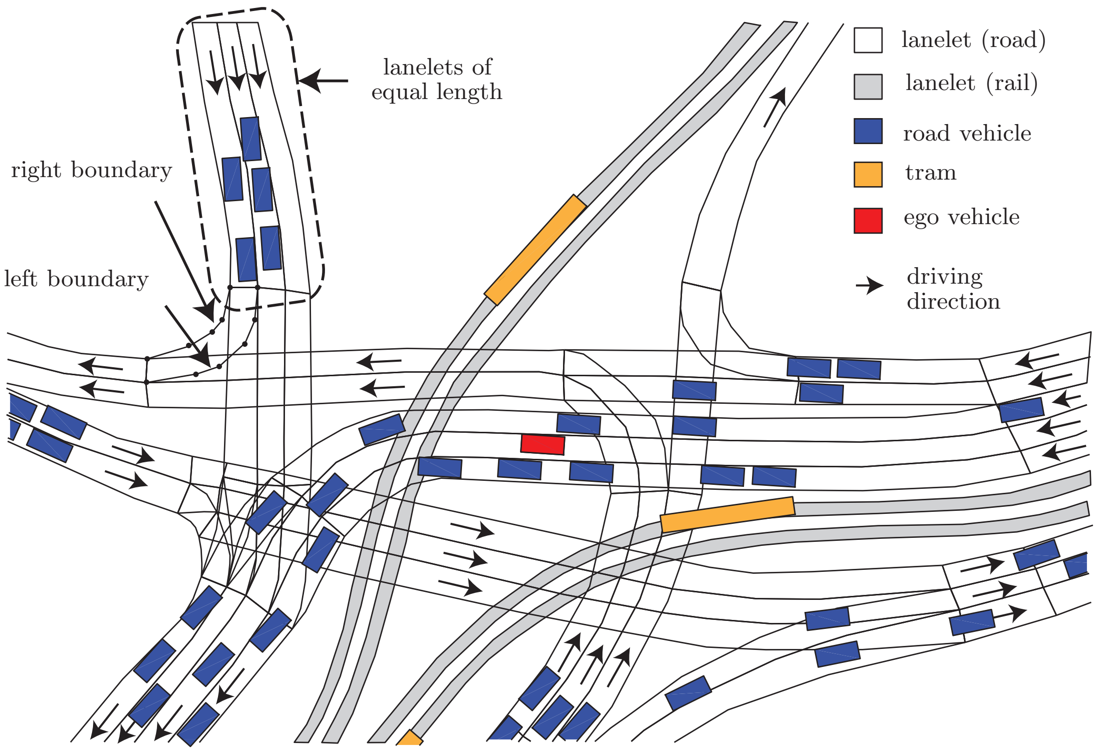

Figure 3: Lanelets of a complex intersection in the city center of Munich (scenario ID S=GER_Muc_1a). Besides roads, also tram rails are modeled by lanelets.

3 Connection to OpenStreetMap

As proposed in [2][2], our XML structure is inspired by the data structure of OpenSteetMap[3] (OSM), which uses the elements *nodes*, *ways* and *relations*.

Nodes A Node is a single point in space consisting of an ID and the coordinates in WGS84 reference system[4], which is the used as a global coordinate system for the benchmark scenarios.

```
<node id='-74' lat='48.08765700619' lon='11.09185012843' />
```

[1]opendrive.org
[2]github.com/phbender/liblanelet
[3]openstreetmap.org
[4]wiki.openstreetmap.org/wiki/Node

Ways Ways are polyline connections of up to 2000 nodes. The nodes are stored as an ordered list and referred to by their IDs[5].

```
<way id='-144'>
   <nd ref='-74' />
   <nd ref='-54' />
</way>
```

Relations Relations can be used to specify logical or geometrical relationships between nodes or ways. The elements are stored as members in an ordered list and have a type associated with them. Also the role of the referenced element role has to be specified[6]. Thus, lanelets can be modeled using relations.

```
<relation id='-3301'>
   <member type='way' ref='-144' role='right' />
   <member type='way' ref='-195' role='left' />
   <tag k='type' v='lanelet' />
</relation>
```

With the XML format of OpenStreetMap, it easy to create new road networks using *JavaOpenStreetMap* (JOSM)[7] and also to edit existing road networks. However, the OSM format has several shortcomings when representing traffic scenarios, e.g. non-Cartesian coordinate frame and inconsistent use of attributes/references instead of elements. Thus, we have developed the *CommonRoad* XML format, which uses a different XML notation and incorporates only a small subset of the OSM attributes, but offers all elements required to specify a traffic scenario.

Note that we use the XML format of OSM[8] only to extract their road data with JOSM to create the road network for a traffic scenario. The lanelets are saved in an intermediate XML file (in OSM format), which is also available online for each *CommonRoad* scenario based on real roads. When adding the obstacles and the planning problem, we convert the OSM data into the *CommonRoad* XML format, which is the final XML file of each scenario. In the following, we present the details of the *CommonRoad* XML format.

4 Specification of the XML Format

A *CommonRoad* XML file consists of an XML header (XML version and the encoding type) and the *CommonRoad* element. The latter has the attributes `commonRoadVersion` (version of the XML specification), `benchmarkID` (ID of the scenario), `date` (date when scenario was written), and `timeStepSize` (global step size of the time-discrete scenario in seconds).

The *CommonRoad* element contains one or more elements of type *lanelet*, *obstacle*, and *planningProblem*. Each has a unique[9] ID making it possible to reference it. It also contains several auxiliary elements as shown in Fig. 1 and Fig. 2. As mentioned earlier, the numbers in square brackets denote whether an element is compulsory for the *CommonRoad* XML format and how many elements of the same type are allowed. If an element is not required compulsorily, we comment it as *optional* in the following XML snippets.

[5]wiki.openstreetmap.org/wiki/Way
[6]wiki.openstreetmap.org/wiki/Relation
[7]josm.openstreetmap.de
[8]wiki.openstreetmap.org/wiki/JOSM_file_format
[9]Unique within the whole XML document.

To check the compliance of an XML file, we additionally provide the XML Schema Definition (XSD) of our format on our website.

All variables are given in SI units and based on a common Cartesian coordinate frame. Angles are measured counter-clockwise around the positive z-axis with the zero angle along the x-axis.

```
<?xml version="1.0" encoding="utf-8"?>
<commonRoad commonRoadVersion='2017a' benchmarkID='documentationExample'
date='2017-08-22' timeStepSize='0.01'>
    <lanelet id='10'>
        ...
    </lanelet>
     ...
    <obstacle id='50'>
        ...
    </obstacle>
    ...
    <planningProblem id='100'>
        ...
    </planningProblem>
    ...
</commonRoad>
```

The specification of each element type is presented in this section. In Sec. 4.1, we will introduce all auxiliary elements (see Fig. 2). Then, we will use the auxiliary elements to build the elements *lanelet* (Sec. 4.4), *obstacle* (Sec. 4.2), and *planningProblem* (Sec. 4.3).

4.1 Auxiliary Elements

First, we require geometric primitives.

Point A point is the simplest primitives and described by an x-coordinate and a y-coordinate:

```
<point>
        <x>0</x>
        <y>0</y>
</point>
```

Rectangle The element *rectangle* can be used to model rectangular obstacles, e.g. a vehicle. It is specified by the length (longitudinal direction) and the width (lateral direction), the orientation, and a center point (reference point of a rectangle is its geometric center).

```
<rectangle>
        <length>4.2</length>
        <width>1.9</width>
        <orientation>0.8</orientation>
        <center>
                <x>1.0</x>
                <y>1.0</y>
        </center>
</rectangle>
```

If the orientation and the x and y coordinates of the center are zero, both elements can be omitted as shown below.

```
<rectangle>
        <length>4.2</length>
        <width>1.9</width>
</rectangle>
```

Circle The element *circle* can be used to model circular obstacles, for example a pedestrian or a vehicle by using three circles. A circle is defined by its radius and its center (reference point of a circle is its geometric center). Analogously to the rectangle, the center can be omitted if its x and y values are zero.

```
<circle>
        <radius>1.3</radius>
        <center>
                <x>-1.0</x>
                <y>-1.0</y>
        </center>
</circle>
```

Polygon The element *polygon* can be used to model any other two-dimensional obstacle. A polygon is defined by an ordered list of points, in which the first one is its reference point. We adhere to the convention that the polygon points are ordered clockwise.

```
<polygon>
        <point>
                <x>-1</x>
                <y>0</y>
        </point>
        <point>
                <x>0</x>
                <y>1</y>
        </point>
        <point>
                <x>1</x>
                <y>-1</y>
        </point>
</polygon>
```

Shape Elements of type *shape* specify the dimension of an object and can contain one or more elements of the geometric primitives (i.e. rectangle, circle, or polygon). Please note that we separate the representation of the dimension and position/orientation of an object into the elements shape and position/orientation (described subsequently), respectively. Thus, the shape elements should usually use the origin as center point and an orientation of zero, unless a certain offset is desired.

```
<shape>
        <circle>
                ...
        </circle>
        <circle>
                ...
        </circle>
        ...
</shape>
```

Position The position of an object is specified by the element *position* which contains either a point, rectangle, circle, polygon, or lanelet. (Note that if an area is specified as position, it does not enclose the geometric shape of an object, but only models the interval of possible positions, e.g. the uncertainty of the position measurement.)

```
<position>
        <point>
                ...
        </point>
        <!-- or -->
        <rectangle>
                ...
        </rectangle>
        ...
        <!-- or -->
        <lanelet ref='12'/>
        ...
<position>
```

Orientation/Time/Velocity/Acceleration Generally, elements can have either an exact value or an interval of values, e.g. to specify the goal region or to include uncertainties.

The position above is given exact by using a point or given as an interval by using any other geometric primitive. To describe dynamic objects, we further require values for the orientation, time, velocity, and acceleration. For each, one can specify the value `exact` or as an interval by specifying the `intervalStart` and `intervalEnd`. For example, a *time* element is defined as

```
<time>
        <exact>0.0</exact>
        <!-- or -->
        <intervalStart>0.0</intervalStart>
        <intervalEnd>1.0</intervalEnd>
</time>
```

The elements *orientation*, *velocity*, and *acceleration* are defined analogously.

States The time-discrete state of an object is specified by the element *state* with the following state variables, which are introduced above: *position*, *orientation*, and *time*. The variables *velocity* (signed speed) and *acceleration* are optional, and both are defined in the body frame of the the object.

We use states to describe the goal region of planning problems (see Sec. 4.3) and to describe the movement of dynamic obstacles (see Sec. 4.2). For obstacles, we assume that the velocity is aligned with their current orientation (i.e. their slip angle $\beta = 0$), while for the goal region of ego vehicle(s), β is not restricted so far. Note that we optionally include acceleration as a state variable for obstacles to provide additional information, e.g. for motion prediction, even though acceleration is often used as input for vehicle models.

```
<state>
        <position>
                ...
        </position>
        <orientation>
                <exact>0.04</exact>
                <!-- or -->
                <intervalStart>0.0</intervalStart>
                <intervalEnd>0.08</intervalEnd>
        </orientation>
        <time>
                <exact>0.0</exact>
                <!-- or -->
                <intervalStart>0.0</intervalStart>
                <intervalEnd>1.0</intervalEnd>
        </time>
        <!-- optional -->
        <velocity>
                <exact>15.5</exact>
                <!-- or -->
                <intervalStart>15.0</intervalStart>
                <intervalEnd>16.0</intervalEnd>
        </velocity>
        <!-- optional -->
        <acceleration>
                <exact>0.0</exact>
                <!-- or -->
                <intervalStart>0.0</intervalStart>
                <intervalEnd>-0.2</intervalEnd>
        </acceleration>
</state>
```

Initial states We use the element *initial state* to describe the initial state of an ego vehicle. In contrast to the general element *state*, the variables *yaw rate* and *slip angle* are also included. The element *initial state* of each planning problem (see Sec. 4.3) allows the initialization of each vehicle model, as described in more detail in our *vehicle model documentation*.

```
<initialState>
        <position>
                ...
        </position>
        <velocity>
                ...
        </velocity>
        <orientation>
                ...
        </orientation>
        <yawRate>
                ...
```

```
       </yawRate>
       <slipAngle>
               ...
       </slipAngle>
       <time>
               ...
       </time>
</initialState>
```

Occupancies If the dimension of an object is time-dependent and one is only interested in the region which is occupied by the object (and the other state variables are not relevant), one can use the element *occupancy*. Occupancies are used to represent the unknown behavior of dynamic obstacles bounded by sets.

```
<occupancy>
       <shape>
               ...
       </shape>
       <time>
               ...
       </time>
</occupancy>
```

Probability Distribution One can also use probability distributions to describe unknown stochastic behavior. Since many different probability distributions are used, we provide a placeholder for probability distributions.

```
<probabilityDistribution>
       ...
</probabilityDistribution>
```

4.2 Obstacles

The element *obstacle* is used to represent different kinds of traffic participants within the scenario. An obstacle is either static or dynamic, which is specified by the element *role*. Each role allows different types of an obstacle as listed in Table 1.

Table 1: Types of obstacles.

Role	**Type**
Static	parkedVehicle, constructionZone, unknown
Dynamic	car, truck, bus, bicycle, pedestrian, priorityVehicle, unknown

Static Obstacles A static obstacle is specified by the elements *role*, *type*, and *shape*.

```
<obstacle id='57'>
       <role>static</role>
       <type>parkedVehicle</type>
```

```
        <shape>
                ...
        </shape>
</obstacle>
```

In addition to static obstacles, traffic scenarios can contain dynamic obstacles. Please note that only elements of either of the following three behavior models (with known behavior, with unknown behavior, or with unknown stochastic behavior) may be present. We do not use these different behavior models together within one traffic scenario as indicated in Fig. 1.

Dynamic Obstacles with Known Behavior A dynamic obstacle with known behavior contains the same elements as a static obstacle and additionally a trajectory of states. The trajectory allows us to represent the states of a dynamic traffic participant along a path. The trajectory is e.g. obtained from a dataset or measurements, which can be exact or with uncertainties.

```
<obstacle id='58'>
        <role>dynamic</role>
        <type>car</type>
        <shape>
                ...
        </shape>
        <trajectory>
                <state>
                        ...
                </state>
                ...
        </trajectory>
</obstacle>
```

The shape of the obstacle is not included in the trajectory, since the shape is usually not varying over time. However, time varying shapes could be included with a new element *timeVaryingShape* instead of the standard *shape* element, but we advise to use dynamic obstacles with unknown behavior (i.e. with occupancy elements) instead.

```
<timeVaryingShape>
        <shapeWithTime>
                <shape>
                        ...
                </shape>
                <time>
                        ...
                </time>
        </shapeWithTime>
        ...
</timeVaryingShape>
```

Dynamic Obstacles with Unknown Behavior For motion planning, we often do not know the trajectory of dynamic obstacles and have to predict their future occupancy. Dynamic obstacles with a unknown behavior represent the occupied area over time by bounded sets.

```
<obstacle id='59'>
        <role>dynamic</role>
        <type>car</type>
        <occupancySet>
                <occupancy>
                        <shape>
                                ...
                        </shape>
                        <time>
                                ...
                        </time>
                </occupancy>
                ...
        </occupancySet>
</obstacle>
```

Dynamic Obstacles with Unknown Stochastic Behavior One can describe unknown stochastic behavior by probability distributions of states.

```
<obstacle id='60'>
        <role>dynamic</role>
        <type>car</type>
        <shape>
                ...
        </shape>
        <probabilityDistribution>
                ...
        </probabilityDistribution>
</obstacle>
```

Optional Tags for Obstacles Elements of type *tag* can be optionally added to an obstacle element, e.g. properties which might be required for some computations. They are given as a key-value pair.

```
    <tag k='' v=''/>
```

4.3 Planning problem

The aim of the *CommonRoad* benchmarks is to compare trajectory planners. Thus, the element *planningProblem* is used to specify the initial state and the goal region, i.e. one or more goal state(s), for the motion planning problem. The shape of the ego vehicle is not included in the XML scenario description, since this property depends on which vehicle parameter set is chosen (see the *vehicle model documentation* on our website).

```
<planningProblem id='100'>
        <initialState>
                ...
        </initialState>
        <goalRegion>
                <state>
                ...
            </state>
            ...
        </goalRegion>
</planningProblem>
```

4.4 Lanelets

Elements of type *lanelet* are used to model the drivable road segments (cf. Section 3). The left and right bound define the driving direction and are represented by polylines, i.e. a sorted list of points. Optionally, line markings (solid, dashed, ...) can be included to model the traffic conditions more precisely.

In order to represent the graph of the road network, the elements *predecessor*, *successor*, *adjacentLeft*, and *adjacentRight* are used, which are omitted if they are empty. Since these elements only contain objects which are already present in the XML file, we refrain from copying their data but introduce references to the neighboring lanelets by an attribute referring to their unique ID. The elements *predecessor* and *successor* can be present multiple times to represent multiple longitudinally adjacent lanelets, e.g. for a road fork. In contrast, a lanelet can have at the most one *adjacentLeft* and one *adjacentRight* neighbor and thus at the most one element of this type. The additional attribute *drivingDir* specifies the driving direction of the neighboring lanelet as `same` or `opposite`.

Optionally, the speed limit of the lanelet can be included. Further traffic signs will be added to the *CommonRoad* XML format in a later release to represent traffic conditions more precisely.

```
<lanelet id='10'>
        <leftBound>
                <point>
                        ...
                </point>
                ...
                <!-- Optional -->
                <lineMarking>solid</lineMarking>
        </leftBound>
        <rightBound>
                <point>
                        ...
                </point>
                ...
                <!-- Optional -->
                <lineMarking>dashed</lineMarking>
        </rightBound>
        <predecessor ref='13'/>
        <successor ref='17'/>
        <adjacentLeft ref='14' drivingDir='opposite'/>
        <adjacentRight ref='15' drivingDir='same'/>
        <!-- Optional -->
        <speedLimit>16.67</speedLimit>
</lanelet>
```

Geometrical requirements of lanelets All *CommonRoad* scenarios meet the following requirements, which assure that lanelets form a road without holes or incorrect overlaps.

- The two polylines forming the right and left bound of a lanelet must consist of the same amount of nodes. In addition, the imaginary straight line connection between two corresponding nodes, one in the left and one in the right bound, should be perpendicular to the center line of the lanelet.
- In case of a two-lane or multi-lane road, a polyline can be shared by two lanelets, i.e. the same points are used to mark the right respectively left boundary of the corresponding lanelets.
- For longitudinal adjacent lanelets, the connection nodes of two consecutive lanelets have to be identical, i.e. the end nodes of the predecessor are identical to the start nodes of the successor.
- To ensures continuous lanes, the bounds of merging and forking lanelets start/end at the corresponding left or right bound of another lanelet, as shown in Fig. 4.

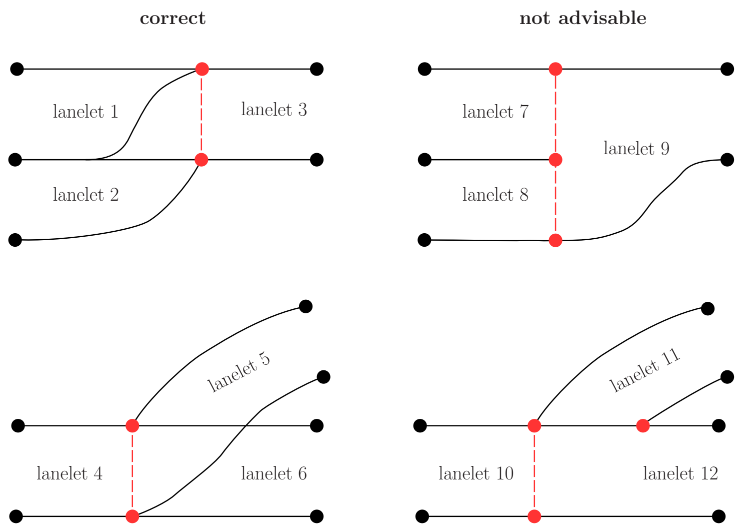

Figure 4: Spatial division of merging and forking lanelets.

- Roads are divided in so called *Lane Sections*[10]. As shown in Fig. 5, each lane section has the same number of lateral adjacent lanelets and all lanelets start and end at the border of a lane section. Thus, all laterally adjacent lanes have the same *length*, which allows us to set the lateral adjacencies correctly (e.g. in Fig. 5, lanelet 1 and 2 are lateral adjacent to each other; as well as lanelet 4, 5, and 6; and lanelet 7 and 8).

5 Conclusions

The *CommonRoad* XML format is a platform-independent format for specifying road traffic scenarios for motion planning. Complex traffic situations can be encoded by specifying the road network, static and dynamic obstacles, and the planning problem. Details on models for the ego vehicle dynamics can be found in the *vehicle model documentation*. Examples of traffic situations that are specified by this format can be found on the *CommonRoad* website[11]. Please contact us if you have any comments.

[10]opendrive.org/
[11]commonroad.in.tum.de

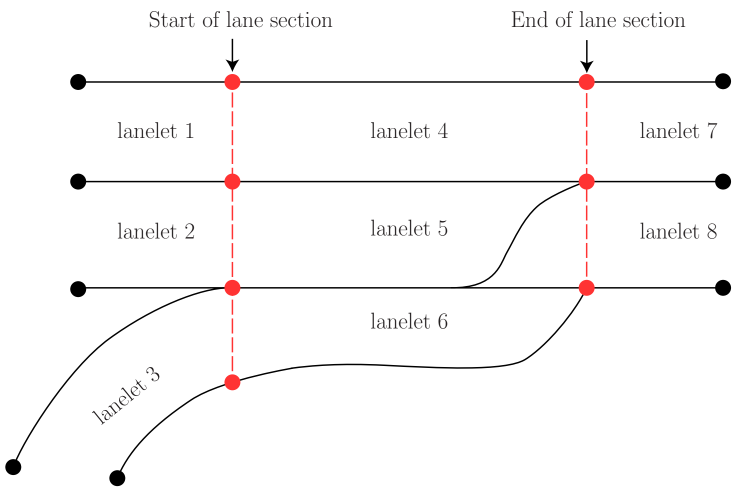

Figure 5: Definition of lane sections.

Acknowledgment

The author gratefully acknowledge financial support by the BMW Group within the Car@TUM project and by the Free State of Bavaria.

References

[1] M. Althoff, M. Koschi, and S. Manzinger. CommonRoad: Composable benchmarks for motion planning on roads. In *Proc. of the IEEE Intelligent Vehicles Symposium*, pages 719–726, 2017.

[2] P. Bender, J. Ziegler, and C. Stiller. Lanelets: Efficient map representation for autonomous driving. In *Proc. of the IEEE Intelligent Vehicles Symposium*, pages 420–425, 2014.