CommonRoad: Documentation of the Format

(Version 2023a)

Sebastian Maierhofer, Markus Koschi, Anna-Katharina Rettinger, Stefanie Manzinger, and Matthias Althoff

Technical University of Munich, 85748 Garching, Germany

Abstract

This document presents the *CommonRoad* format for specifying road traffic scenarios. The *CommonRoad* format is composed of (1) meta information about the scenario, (2) a formal representation of the road network, (3) static and dynamic obstacles, and (4) 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

1.1 Overview about CommonRoad

Within *CommonRoad* [1][1], Protocol Buffers[2] and XML files (currently only up to version 2020a) are used to store the data for specific driving scenarios. In this documentation, we present the definition of CommonRoad scenarios, which are composed of (1) meta information describing the scenario (see Section 2.1), (2) a formal representation of the road network (see Section 2.3-2.6), (3) static and dynamic obstacles (see Section 2.7), and (4) the planning problem of the ego vehicle(s) (see Section 2.8). In non-collaborative scenarios, only one planning problem exists, while in collaborating scenarios several planning problems have to be solved.

A visualization of all scenarios is on our website[3], where you can also search for specific types of scenarios.

Since CommonRoad version 2023a, the default representation of a CommonRoad scenario consists of three Protobuf files instead of a single (Protobuf/XML) file. However, for better exchanging files it is possible to write and read everything in just one file.

1.2 Changes Compared to Version 2020a

For a quick reference, we summarize the major changes of version 2023a compared to version 2020a:

- New intersection definition
- Separation into three files: map, dynamic, scenario
- Additional elements for link to licenses and license text
- Area for modelling parking lots, bus stops, or any other "drivable area" which is difficult to model via lanelets (concept derived from lanelet2 format)
- Meta-information series for obstacles
- Boundary definition separated from lanelet

[1] commonroad.in.tum.de
[2] https://protobuf.dev/
[3] commonroad.in.tum.de/scenarios

2 Specification of the Format

We designed our format such that it can represent all features, is unambiguous, and is easy to use. We put additional emphasis to augment the road description by its implications, i.e., our format does not just describe a traffic situation, but already provides the meaning for motion planning. As a result, our scenarios can directly be used by a motion planner and do not require much additional computations in terms of pre-processing. Subsequently, we describe our specification.

All variables are given by decimal numbers based on SI units. We use a common Cartesian coordinate frame with x-, y-, and z-axis to be able to represent all road networks including bridges and tunnels. If a scenario is only defined in a two-dimensional plane (which is often the case), we use the convention that all z-coordinates are zero. Angles are measured counter-clockwise around the positive z-axis with the zero angle along the x-axis.

Add figure for orientation since this often leads to questions.

The overall structure of the CommonRoad format is shown by Fig. 1, Fig. 2, and Fig. 3. A *CommonRoad* scenario contains one or more elements of type *lanelet*, *staticObstacle*, *dynamicObstacle*, *trafficSign*, *trafficLight*, *intersection*, and *planningProblem*. Each has a unique[4] ID (of type positive integer) making it possible to reference it. Additionally, we can model meta information about map location and environment (e.g., weather conditions). The numbers in square brackets denote the number of allowed elements (while `N` can be different for each element). If an element is omitted (number of allowed elements is ≥ 0), the default value is none or its default value is specified in the description of this element below.

```
[1] commonroad_map
 ├── [1] map_meta_information
 ├── [1] location
 ├── [0..N] lanelets
 ├── [0..N] stop_lines
 ├── [0..N] boundaries
 ├── [0..N] areas
 ├── [0..N] traffic_sign
 ├── [0..N] traffic_lights
 └── [0..N] intersections
```

Figure 1: Structure encoding map information.

```
[1] commonroad_dynamic
 ├── [1] dynamic_meta_information
 ├── [1] environment
 ├── [0..N] traffic_light_cycles
 ├── [0..N] traffic_sign_values
 ├── [0..N] static_obstacles
 ├── [0..N] dynamic_obstacles
 ├── [0..N] environment_obstacles
 └── [0..N] phantom_obstacles
```

Figure 2: Structure encoding dynamic scenario information.

[4] Unique within the whole scenario.

```
[1] commonRoa_scenario
 ├── [1] scenario_meta_information
 ├── [1] map_id
 ├── [1] dynamic_id
 ├── [0..N] planning_problems
 └── [0..N] coopearative_planning_problems
```

Figure 3: Structure encoding scenario information.

2.1 CommonRoad Map

2.2 CommonRoad Dynamic

2.3 CommonRoad Scenario

2.4 Meta Information Describing CommonRoad Scenarios

2.4.1 CommonRoad Root Element

The *CommonRoad* root element has the following attributes (its elements are shown in Fig. **??** and Fig. 2), where the timeStepSize is not present in files representing static information:

- `commonRoadVersion`: version of the specification,
- `benchmarkID`: benchmark ID of the scenario (see Sec. 2.1.2),
- `date`: date when scenario was generated,
- `author`: author(s) of the scenario in alphabetic order,
- `affiliation`: affiliation of the author(s), e.g., name and country of the university or company,
- `source`: if applicable, description of the data source of the scenario, e.g., name of dataset or map service,
- `sourceLink`: link(s) to source of the scenario,
- `license`: name or link to license of scenario,
- `timeStepSize`: global step size of the time-discrete scenario.

2.4.2 Benchmark ID

The benchmark ID of each scenarios consists of four elements:

COUNTRY_SCENE_CONFIG_PRED_Version.

The scenario ID has the prefix C- if the scenario has multiple planning problems, i.e. it is a cooperative planning problem (otherwise, it has no prefix).

COUNTRY is the capitalized three-letter country code defined by the ISO 3166-1 standard[5], e.g. Germany has DEU and United States has USA. If a scenario is based on an artificial road network, we use ZAM for Zamunda[6].

SCENE = MAP-{1-9}* specifies the road network. MAP is for rural scenarios a two/three letter city code (e.g. Muc) and for highways/major roads the road code (e.g. A9 or Lanker).

[5] `https://www.iso.org/obp/ui/#search/code/`

[6] en.wikipedia.org/?title=Zamunda

It is appended by an integer counting up. Note that if COUNTRY_SCENE is the same for two scenarios, all their lanelets are identical.

CONFIG = {1-9}* specifies the initial configuration of obstacles and the planning problem(s). Note that CONFIG is counting independently for non-cooperative scenarios (i.e. only one planning problem) and cooperative scenarios (i.e. multiple planning problems), since the prefix allows to distinguish between them. Thus, if PREFIX-COUNTRY_SCENE_CONFIG is the same for two scenarios, the road network, initial configuration of obstacles, and the planning problem(s) are equal, and only the prediction of the obstacles differs.

PRED = {S,T,P}-{1-9}* specifies the future behavior of the obstacles, i.e. their prediction, where S = set-based occupancies, T = single trajectories, P = probability distributions, appended by an integer to distinguish predictions on the same initial configuration but with different prediction parameters. If no prediction is used (i.e. the scenario has no dynamic obstacles), we omit the element PRED in the benchmark ID.

Version = {0-9}*-{0-9}*-{0-9}* specifies the scenario version. The first number represents the major revision and the second number the minor revision of the CommonRoad scenario version ID (see 1). Scenarios with the same major revision number are compatible. The last number is optional and represents minor updates in the scenario itself, e.g., if a small bug in the lanelet vertices is fixed. Note that before CommonRoad version 3.0, we had a different naming convention of the scenario version. We changed the CommonRoad version ID from year-based to number-based. Tab. 1 maps the year-based version representation to a number-based for versions.

Table 1: Year-base with corresponding number-based version.

year-based	**number-based**	**scenario version ID**
2017a	0.1	0-1
2017b	0.2	0-2
2018a	0.3	0-3
2018b	1.0	1-0
2020a	2.0	2-0
-	3.0	3-0

Examples: Possible examples of a benchmark ID are: C-USA_US101-1_123_T-1_3-0, DEU_FFB-2_42_S-4_3-0-2, DEU_Hhr-1_1_0-2.

2.4.3 Location of Scenarios

The location element consists of (1) a GeoName-ID[7], (2) a GPS latitude coordinate, (3) a GPS longitude coordinate, (4) an optional geometrical transformation introduced in the following subsection, and an optional environment information. If the GeoName-ID and the GPS coordinates are unknown, e.g., in artificial road networks, the GeoName-ID is set to -999 and the values of the coordinates are set to 999.

[7] geonames.org

GeoTransformation Element To specify the geometrical transformation which were performed while creating the lanelets, one can add a `geoTransformation` element. This may contain two children:

1. The optional `geoReference` element contains a *proj-strings*[8] describing a coordinate transformation from geodetic coordinates to the projected (Cartesian) coordinates. This projection can then be used to transform the Cartesian coordinates back to geodetic coordinates used by OSM (cf. Sec. **??**).
2. The `additionalTransformation` element (see Fig. 4) describes geometrical operations which were performed (after the geoReference) to transform the Cartesian coordinates. The execution order of the transformations is according to the order of the following list:
   - `xTranslation` Translating x-coordinates of all points by this value.
   - `yTranslation` Translating y-coordinates of all points by this value.
   - `zRotation` Rotating all points by this value around the origin, with respect to a right-handed coordinate system.
   - `scaling` Multiplying all x- and y-coordinates by this value.

```
/
└── [0..1] additionalTransformation
    ├── [1] xTranslation
    ├── [1] yTranslation
    ├── [1] zRotation
    └── [1] scaling
```

Figure 4: Element *additionalTransformation*

Environment The optional element *environment* contains information about the time (in hours, minutes and seconds) at which the scenario starts, the time of day, the current weather, and the underground.

```
/
├── [0..1] time
├── [0..1] timeOfDay: night/day
├── [0..1] weather: sunny/light_rain/heavy_rain/fog/snow/hail
└── [0..1] underground: wet/clean/dirty/damaged/snow/ice
```

Figure 5: Element *environment*

2.4.4 Tags for Scenarios

To allow users to select scenarios meeting their needs, the list of scenarios on our website can be filtered by the tags given in the element `tags`. Additionally, the filtering based on the number of static obstacles, dynamic obstacles, obstacle types, type of future behavior of obstacles, number of ego vehicles, number of goal states, and time horizon of the scenario is possible.

2.5 Auxiliary Elements

We use the following auxiliary geometry elements:

[8] proj.org

```
/
├── point OR center
│   ├── [1] x
│   ├── [1] y
│   └── [0..1] z
├── 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
```

Figure 6: Auxiliary elements used for defining scenarios.

Point A point is the simplest primitives and described by an x-, y-, and z-coordinate. If the z-coordinate is zero (for all two-dimensional scenarios), we omit the z-element.

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). If the orientation and the coordinates of the center are zero, both elements can be omitted.

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 all its coordinates are zero.

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.

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.

Positions The position of an object is specified by the element *position* which contains either a point, rectangle, circle, polygon, or lanelet (unless for a planning problem as specified later), as shown in Fig. 7.

Note that if the position of an object is given as an area (i.e. not a single point), the area does not enclose the geometric shape of the object, but only models the interval of possible positions, e.g. the uncertainty of the position measurement.

```
/
└─ [1] position
    ├─ [1] point
    │  OR
    ├─ [1..N] rectangle/circle/polygon
    │  OR
    └─ [1..N] lanelet (ref to lanelet)
```

Figure 7: Element *position*.

Numeric Values Elements describing the state of an object, e.g. orientation or velocity, can have either an exact value or an interval of values, e.g. to specify the goal state or to include uncertainties. For example, an *orientation* element can be defined using `exact` or `intervalStart` and `intervalEnd`:

```xml
<orientation>
        <exact>0.0</exact>
        <!-- or -->
        <intervalStart>-1.5708</intervalStart>
        <intervalEnd>1.5708</intervalEnd>
</orientation>
```

Time All time elements are not given as numeric values, but as integers (i.e. non-negative whole numbers). Thus, the time element can specify the time stamp of an time-discrete object. Since the initial time is always 0 and the constant time step size is given in the CommonRoad root element, the time in seconds can be directly calculated.

2.6 Lanelets

For our benchmarks we use *lanelets* [2] as drivable road segments to represent the road network. Fig. **??** shows the specification of a *lanelet* element. It is defined by its *left* and *right boundary*, where each boundary is represented by an array of points (a polyline), as shown in Fig. 8. Optionally, line markings (solid, dashed, broad solid, broad dashed, no line marking, unknown, where unknown is the default line marking) can be included to model the boundary more precisely. We have chosen lanelets since they are as expressible as other formats, such as e.g. OpenDRIVE[9], yet have a lightweight and extensible representation. Our converter from OpenDRIVE to Lanelets is available on our website.

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 (see Fig. **??**). Since these elements only contain objects which are already existing, 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 used 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`. The driving direction of a lanelet is implicitly defined by its left and right bound.

Additionally, the element *stopLine* can be used to model a stop line or give way line (see Fig. 9). This element is defined by two points, representing the line. If no points are given for a stop line,

[9] opendrive.org

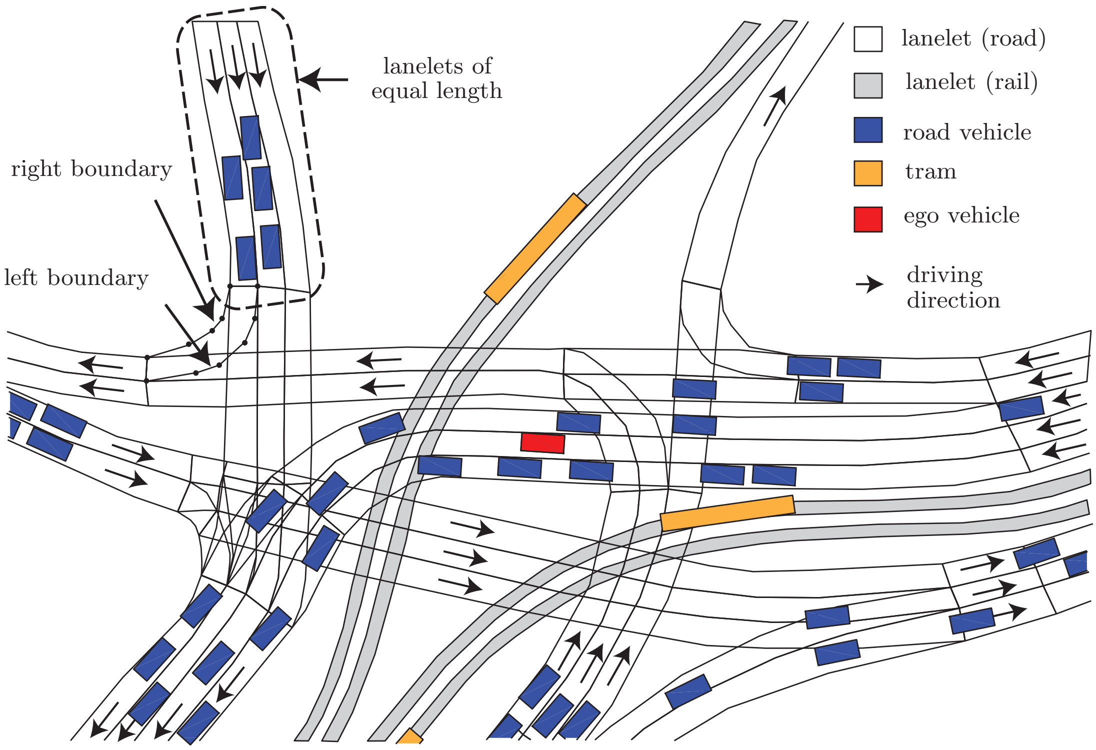

Figure 8: Lanelets of a complex intersection in the city center of Munich. Besides roads, also tram rails are modeled by lanelets.

we assume that its points correspond to the end points of the lanelet it belongs to. Optionally, traffic lights or traffic signs associated with this stop line can be specified to model the traffic conditions more precisely. Similarly to the neighboring lanelets, traffic signs and traffic lights are already existing. Thus, we use an attribute referring to their unique ID. By defining the line marking (solid or dashed), a stop line or give way line can be represented.

```
/
└─ stopLine
   ├─ [0..2] point
   ├─ [1] lineMarking
   ├─ [0..N] trafficSignRef (ref to trafficSign)
   └─ [0..N] trafficLightRef (ref to trafficLight)
```

Figure 9: Element *stopLine*

For precise modelling of traffic conditions, additional properties of a lanelet are required. The element *laneletType* can be used multiple times to clearly define the type of a lanelet. In order to specify which types of traffic participants are allowed to use a lanelet and in which direction, the elements *userOneWay* and *userBidirectional* can be included. The supported types and users of lanelets are listed in Tab. 2.

Optionally, traffic signs or a traffic light valid for a lanelet can be included by referring to the unique ID of the respective element.

2.6.1 Geometrical Requirements of Lanelets

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

Table 2: Types and users of lanelets.

LaneletType	urban, country, highway, driveWay, mainCarriageWay, accessRamp, exitRamp, shoulder, busLane, busStop, bicycleLane, sidewalk, crosswalk, interstate, intersection, unknown
User	vehicle (car, truck, bus, motorcycle, priorityVehicle, taxi), car, truck, bus, priorityVehicle, motorcycle, bicycle, pedestrian, train, taxi

- 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. 10.

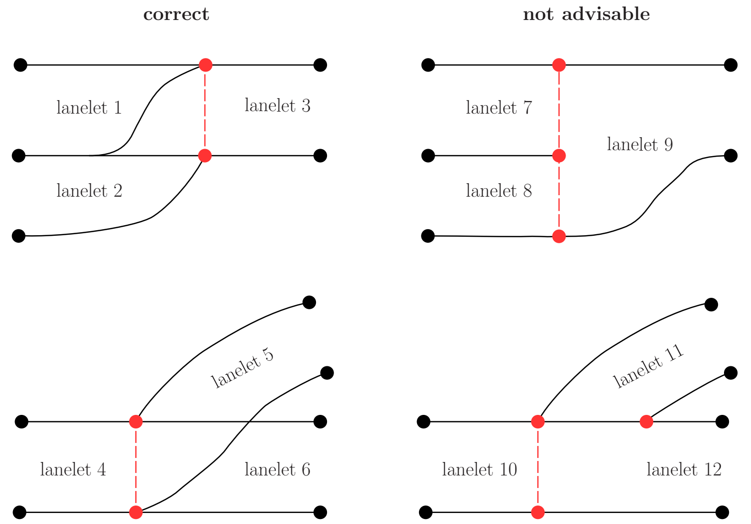

Figure 10: Spatial division of merging and forking lanelets.

- Roads are divided in so called *Lane Sections*[10]. As shown in Fig. 11, 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. 11, lanelet 1 and 2 are lateral adjacent to each other; as well as lanelet 4, 5, and 6; and lanelet 7 and 8).

[10] opendrive.org/

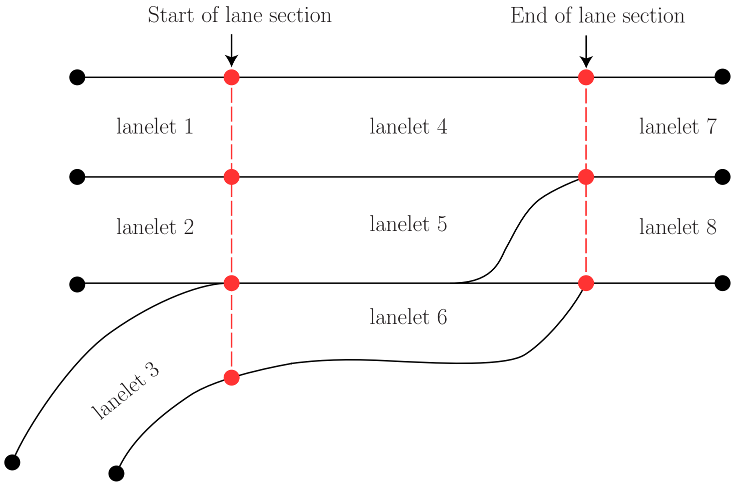

Figure 11: Definition of lane sections.

2.7 Traffic Signs

The element *trafficSign* is used to represent different traffic signs within the scenario (see Fig. 12). Traffic signs are always valid either from the beginning of a lanelet or the end of a lanelet. For example, a priority sign at an intersection is valid from the end of the incoming lanelet. Therefore, it is not necessary to place the sign on the lanelets which go straight, left, or right. On the other hand, a speed limit sign is valid from the beginning of a lanelet so that the speed limit has to be considered for the entire lanelet.
A *trafficSign* element consists of one or multiple *trafficSignElements*. This way, a combination of traffic signs can be represented. Additionally, the position of the traffic sign can be included and it can be specified whether the traffic sign is existent in the real world by the element *virtual*, e.g., in the case one wants to add a speed limit which is in the real world outside of the captured road network. If *virtual* is not specified or it is set to false, we assume that the traffic sign exists also in the real world at this position. A *trafficSignElement* is defined by the *trafficSignId* specified in national traffic laws and can be further described by one or multiple *additionalValues*. For example, the *trafficSignElement* for a speed limit of 16.66 m/s in Germany would be defined as:

```xml
<trafficSignElement>
<trafficSignId>274</trafficSignId>
<additionalValue>16.66</additionalValue>
</trafficSignElement>
```

The currently by CommonRoad supported national traffic signs are listed in Tab. 4. In Tab. 3, exemplary traffic signs from Germany with their description are shown.

```
trafficSign (id)
├── [1..N] trafficSignElement
│   ├── [1] trafficSignID
│   └── [0..N] additionalValue
├── [0..1] position
│   └── [1] point
└── [0..1] virtual
```

Figure 12: Element *trafficSign*.

Table 3: Exemplary German traffic sign IDs supported by CommonRoad with the corresponding symbol.

Symbol [11]	**Description**	Traffic Sign ID
	Right before left rule	102
	Yield	205
	Stop	206
	Ban on motorcycles and multi-lane vehicles	260
	No U-turn	272
	Speed limit	274
	Required speed	275
	No overtaking (except of non-motorized traffic participants, trains, and motorcycles without sidecar)	276
	Right of way	301
	Priority road	306
	Town sign	310
	Green arrow sign	720

2.8 Traffic Lights

The element *trafficLight* is used to represent traffic lights within the scenario (see Fig. 13). Each phase/color of a *trafficLight* and its duration is defined by the element *cycleElement*. Similarly to the *time* elements, the duration is not given as numeric value, but as integer. The color value inactive indicates that currently no phase is activated, e.g. a green right arrow traffic light can be activated iteratively. The order of the different phases is determined by the order of the *cycleElements* in the element *cycle*. By specifying the element *timeOffset*, the cycle is shifted by this value.

In order to define for which driving direction the traffic light is valid, i.e., the direction arrow(s) in a traffic light, the direction can be included. If the direction is not specified, the traffic light is valid for all directions. Additionally, the element *active* can be used to determine whether a traffic light is active or not. Optionally, the position of the traffic sign can be included. Traffic lights are always valid starting from the end of a lanelet.

Table 4: Overview of all supported traffic signs. Valid for complete lanelet expresses that the traffic sign is always valid from the start of a lanelet. All traffic signs which are not mentioned in this list are valid starting starting from the end of a lanelet.

Country	**Traffic Sign ID**	**Valid for complete lanelet**
Germany	101, 102, 108, 114, 123, 138, 142-10, 201, 205, 206, 208, 209-10, 209-20, 220-10, 220-20, 222-10, 222-20 ,237, 239, 242.1, 242.2, 244.2, 244.2, 245, 250, 251, 253, 254, 255, 257-54, 259, 260, 261, 262, 264, 265, 266, 26, 272, 274, 274.1, 274.2, 275, 276, 277, 278, 281, 282, 301, 306, 308, 310, 325.1, 325.2, 327, 330.1, 330.2, 331.1, 331.2, 333-21, 333-22, 350, 357, 625-10, 625-11, 625-12, 625-13, 625-20, 625-21, 625-22, 625-23, 626-10, 626-20, 626-30, 626-31, 720, 1000-10, 1000-11, 1000-20, 1000-21, 1000-30, 1000-31, 1001-30, 1001-31, 1002-10, 1002-12, 1002-13, 1002-20, 1002-22, 1002-23, 1002-11, 1002-14, 1002-21, 1002-24, 1004-30, 1004-31, 1020-30, 1022-10, 1024-10, 1026-36, 1026-37, 1026-38, 1040-30, 1053-33	101, 102, 108, 114, 123, 138, 142-10, 201, 208, 209-10, 209-20, 220-10, 220-20, 222-10, 222-20, 237, 239, 242.1, 244.1, 245, 250, 251, 253, 254, 255, 257-54, 259, 260, 261, 262, 264, 265, 266, 267, 274, 274.1, 275, 276, 277, 308, 310, 325.1, 327, 330.1, 331.1
USA	R2-1, R3-4	R2-1

```
trafficLight (id)
├── [1] cycle
│   ├── [1..N] cycleElement
│   │   ├── [1] duration
│   │   └── [1] color: red/redYellow/green/yellow/inactive
│   └── [0..1] timeOffset
├── [0..1] position
│   └── [1] point
├── [0..1] direction: right/straight/left/leftStraight/straightRight/leftRight/all
└── [0..1] active: true/false
```

Figure 13: Element *trafficLight*.

2.9 Intersections

The element *intersection* is used to represent an intersection within the road network (see Fig. 15). An *intersection* element is defined by at least one *incoming*, which consist of *incomingLanelets*, *outgoingsRight*, *outgoingsStraight*, *outgoingsLeft*, the element *isLeftOf*, and the element *crossing*.

The elements *outgoingsRight*/*Straight*/*Left* are used to infer for which lanelets the traffic light is valid for and to facilitate the calculation of priorities at intersections. The element *isLeftOf* is used to infer the right-before-left-rule. The element *crossing* models lanelets which cross other lanelets, e.g., crosswalks. Since all elements of a *incoming* are already existing, we use an attribute referring to their unique ID. An example for an *intersection* with all elements except crossings is illustrated in Fig. 15.

[11] commons.wikimedia.org

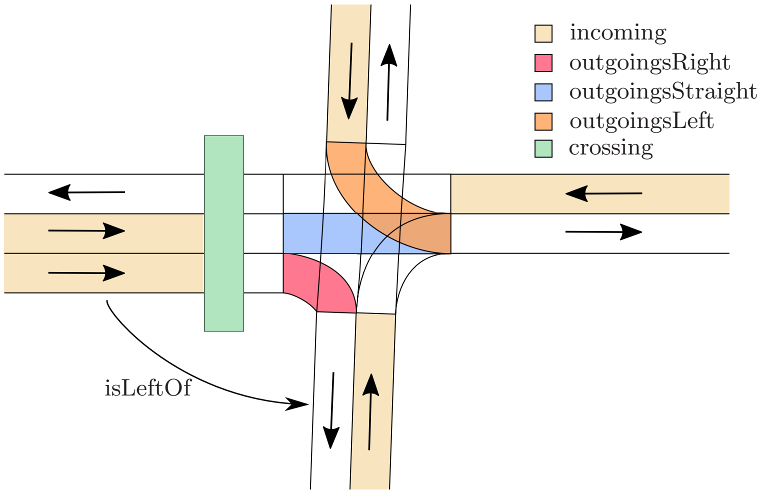

Figure 14: Example intersection. For better visibility, only one example lanelet is highlighted for each outgoingsRight/Straight/Left element.

```
intersection (id)
 └── [1..N] incoming (id)
      ├── [1..N] incomingLanelet (ref to lanelet)
      ├── [0..N] outgoingsRight (ref  to all lanelets that complete the right turn within
      │    this intersection)
      ├── [0..N] outgoingsStraight (ref to all lanelets that are going straight)
      ├── [0..N] outgoingsLeft (ref to all lanelets that are turning left)
      └── [0..1] isLeftOf (ref to incoming)
 └── [0..N] crossing
      └── [1..N] crossingLanelet (ref to lanelet (usually of type crosswalk))
```

Figure 15: Element *intersection*.

2.10 Obstacles

The elements *staticObstacle* and *dynamicObstacle* are used to represent different kinds of traffic participants within the scenario. Additionally, the element *environmentObstacle* is used to represent objects outside of the road network, e.g. buildings. Each obstacle element can have a *type* as listed in Table 5.

Table 5: Types of obstacles.

Role	Type
Static	parkedVehicle, constructionZone, roadBoundary, phantom, unknown
Dynamic	car, truck, bus, motorcycle, bicycle, pedestrian, priorityVehicle, train, phantom, unknown
Environment	building, pillar, median_strip, unknown

The dimensions of an obstacle is specified by the element *shape* (cf. Sec. 2.2), and its initial configuration by the element *initialState*. Additionally, the element *initialSignalState* can be included, to account for properties which are not related to the dynamic of an obstacle, e.g., whether the indicators are turned on or off.

Initial state of obstacles The configuration of an obstacle at the initial time ($t = 0$) is specified by the element *initialState* with the following state variables: *position*, *orientation*, *time*, *velocity* (scalar), *acceleration* (scalar), *yawRate*, and *slipAngle*, as shown in Fig. 16.

```
/
└─ initialState
   ├─ [1] position
   ├─ [1] orientation
   ├─ [1] time
   │  └─ [1] exact
   │     └─ [1] 0
   ├─ [0..1] velocity
   ├─ [0..1] acceleration
   ├─ [0..1] yawRate
   └─ [0..1] slipAngle
```

Figure 16: Element *initialState* of an obstacle, where each state variable (except time) can be exact or an interval.

Initial signal state of obstacles The state of various signals of an obstacle at the initial time ($t = 0$) is specified by the element *initialSignalState* with the following state variables: *time*, *horn*, *indicatorLeft*, *indicatorRight*, *brakingLights*, *hazardWarningLights*, and *flashingBlueLights*, as shown in Fig. 17.

```
/
└─ initialSignalState
   ├─ [1] time
   │  └─ [1] exact
   │     └─ [1] 0
   ├─ [0..1] horn
   ├─ [0..1] indicatorLeft
   ├─ [0..1] indicatorRight
   ├─ [0..1] brakingLights
   ├─ [0..1] hazardWarningLights
   └─ [0..1] flashingBlueLights
```

Figure 17: Element *initialSignalState* of an obstacle.

2.10.1 Static Obstacles

A static obstacle has no further information, as shown in Fig. **??**.

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

2.10.2 Dynamic Obstacles with Known Behavior

A dynamic obstacle with known behavior contains a trajectory of states and a series of signals (cf. Fig. **??**). The trajectory allows us to represent the states of a dynamic traffic participant along a path for $t > 0$. The trajectory is obtained from a dataset (whose measurements can be exact or with uncertainties), from a prediction (which generates a single trajectory for each obstacle), or created hand-crafted. The signal series is obtained from a simulator or created hand-crafted.

States The time-discrete states of a trajectory are specified by the element *state* with the following state variables: *position*, *orientation*, and *time*, *velocity* (scalar), *acceleration* (scalar), *yawRate*, and *slipAngle*, as shown in Fig. 18.

```
/
└── state
     ├── [1] time
     ├── [1] position
     ├── [1] orientation
     ├── [0..1] velocity
     ├── [0..1] acceleration
     ├── [0..1] yawRate
     └── [0..1] slipAngle
```

Figure 18: Element *state* of a trajectory, where each state variable can be exact or an interval.

Signal States The time-discrete states of a signal series are specified by the element *signalState* with the following state variables: *time*, *horn*, *indicatorLeft*, *indicatorRight*, *brakingLights*, *hazardWarningLights*, and *flashingBlueLights*, as shown in Fig. 19.

```
/
└── signalState
     ├── [1] time
     ├── [0..1] horn
     ├── [0..1] indicatorLeft
     ├── [0..1] indicatorRight
     ├── [0..1] brakingLights
     ├── [0..1] hazardWarningLights
     └── [0..1] flashingBlueLights
```

Figure 19: Element *signalState* of a signal series.

2.10.3 Dynamic Obstacles with Unknown Behavior

For motion planning, we often do not know the exact future behavior of dynamic obstacles, but we instead represent their future behavior by bounded sets. Thus, dynamic obstacles with a unknown behavior are specified by an *occupancy set*, which represents the occupied area over time by bounded sets. As shown in Fig. **??**, an *occupancy set* contains a list of *occupancy* elements.

Occupancies The *occupancy* element consists of a shape (occupied area) and a time, as shown in Fig. 20.

2.10.4 Dynamic Obstacles with Unknown Stochastic Behavior

One can describe unknown stochastic behavior by probability distributions of states. Since many different probability distributions are used, we only provide a placeholder for probability distributions.

```
/
└── occupancy
    ├── [1] shape
    └── [1] time
        ├── [1] exact
        │   OR
        ├── [1] intervalStart
        └── [1] intervalEnd
```

Figure 20: Element *occupancy* of an occupancy set.

2.10.5 Phantom Obstalces

The element *phantomObstacle* is used to specify potential occluded obstacles. Therefore, they have no trajectory, but an occupancy set.

2.10.6 Environment Obstalces

The element *environmentObstacle* is used to specify the outline of environmental or infrastructure objects to properly compute occlusions. A *environmentObstacle* is specified by its type, e.g. building, and its shape.

2.11 Planning Problem

The element *planningProblem* is used to specify the initial state and one or more goal state(s) for the motion planning problem. Note that the shape of the ego vehicle is not included in the scenario description, since this property depends on which vehicle parameter set is chosen (see the *vehicle model documentation* on our website).

Initial States We use the element *initial state* to describe the initial state of the planning problem. In contrast to the general element *state*, all state variables are mandatory and must be given exact, as shown in Fig. 21. The element *initial state* of each planning problem allows the initialization of each vehicle model, as described in more detail in our *vehicle model documentation*.

Goal States A planning problem may contain several elements *goal state* (cf. Fig. **??**). In contrast to the general element *state*, all state variables except time are optional and all variables can only be given as an interval, as specified in Fig. 22.

3 Conclusions

The *CommonRoad* 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[12]. Please contact us if you have any comments.

[12] commonroad.in.tum.de

```
/
└── initialState
    ├── [1] position
    │   └── [1] point
    ├── [1] velocity
    │   └── [1] exact
    ├── [1] orientation
    │   └── [1] exact
    ├── [1] yawRate
    │   └── [1] exact
    ├── [1] slipAngle
    │   └── [1] exact
    ├── [1] time
    │   └── [1] exact
    │       └── [1] 0.0
    └── [0..1] acceleration
        └── [1] exact
```

Figure 21: Element *initial state* of a planning problem

```
/
└── goalState
    ├── [1] time
    │   ├── [1] intervalStart
    │   └── [1] intervalEnd
    ├── [0..1] position
    │   ├── [1..N] rectangle/circle/polygon
    │   │   OR
    │   └── [1..N] lanelet (ref to lanelet)
    ├── [0..1] orientation
    │   ├── [1] intervalStart
    │   └── [1] intervalEnd
    └── [0..1] velocity
        ├── [1] intervalStart
        └── [1] intervalEnd
```

Figure 22: Element *goal state* of a planning problem

4 Conclusions

The *CommonRoad* 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[13]. Please contact us if you have any comments.

[13]commonroad.in.tum.de

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.