Simplifying Sim-to-Real Transfer in Autonomous Driving: Coupling Autoware with the CommonRoad Motion Planning Framework

Gerald Würsching, Tobias Mascetta, Yuanfei Lin, and Matthias Althoff

***Abstract*— Validating motion planning algorithms for autonomous vehicles on a real system is essential to improve their safety in the real world. Open-source initiatives, such as Autoware, provide a deployable software stack for real vehicles. However, such driving stacks have a high entry barrier, so that integrating new algorithms is tedious. Especially new research results are thus mostly evaluated only in simulation, e.g., within the CommonRoad benchmark suite. To address this problem, we present `CR2AW`, a publicly available interface between the CommonRoad framework and Autoware. `CR2AW` significantly simplifies the sim-to-real transfer of motion planning research, by allowing users to easily integrate their CommonRoad planning modules into Autoware. Our experiments both in simulation and on our research vehicle showcase the usefulness of `CR2AW`.**

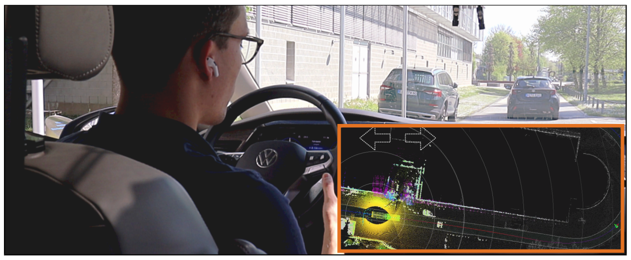
I. Introduction

To ensure robustness and safety of motion planning algorithms for automated vehicles, both extensive testing in simulation and validation on real systems are required. Real-world evaluations allow developers to analyze the performance of algorithms in a closed-loop setting embedded in the full software stack and thus identify problems that can not be captured in simulation. However, conducting real-world experiments is especially challenging for research groups, as the setup of a fully operational driving stack is time-consuming. Hence, to date, most car manufacturers validate their software on real systems, while the number of real-world experiments conducted within research remains scarce. Recent advances in open-source driving stacks have made real vehicle experiments more accessible to the research community [1], nevertheless, the transfer of research algorithms from simulation to full driving stacks remains cumbersome. Aiming to ease the sim-to-real transfer in motion planning research, we introduce the first publicly available interface between the CommonRoad benchmark suite [2] and Autoware [3], two widely used open-source frameworks for simulation and real-world evaluation, respectively (Fig. 1).

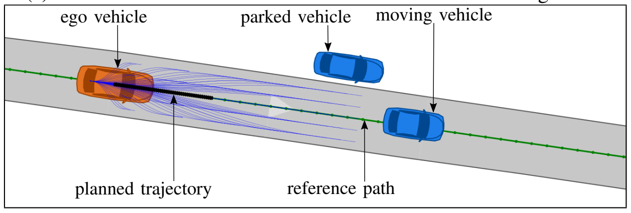
A. Related Work

1) CommonRoad Framework: CommonRoad facilitates the benchmarking of motion planning algorithms by offering a wide range of open-source toolboxes, easily configurable vehicle models, and cost functions. These toolboxes are capable of, e.g., checking the drivability of planned motions [4], conducting reachability analysis [5], [6], and measuring the criticality of traffic scenarios [7]. CommonRoad provides a vast collection of traffic scenarios, both interactive and non-interactive, partially generated from real datasets. Furthermore, CommonRoad offers converters for various scenario and map formats [8]–[10] and interfaces between different platforms [11], [12]. In particular, safety-critical scenarios can be generated automatically using real-world maps and sophisticated traffic simulators [9], [13]. The diversity of scenarios accelerates the development of learning-based frameworks for motion planning, as demonstrated in [14]–[16]. With its comprehensive and versatile framework, CommonRoad has gained significant popularity in the research community to evaluate and showcase motion planners for automated vehicles [17]–[28]. However, none of these works have so far been integrated with other software and hardware components for validation in a real-world environment.

2) Open-Source Driving Stacks: Despite the rich history of research on autonomous vehicles [29]–[33], only a few studies have made their source code publicly accessible or available under an open license. Driving stacks are realized either as modular systems or through end-to-end learning approaches [1]. However, the latter often lack interpretability and safety guarantees. To date, the most established open-source, modular driving stacks for real-world vehicles are Autoware and Apollo[1]. Both Autoware and Apollo provide software development kits that facilitate the validation of al-



(a) Real vehicle view and RVIZ visualization of the driving stack.



(b) Corresponding scenario in CommonRoad with planner output.

Fig. 1: Real-world scenario with two surrounding vehicles: We use `CR2AW` to integrate a motion planner using the CommonRoad format into the Autoware driving stack and run it on a real test vehicle.

All authors are with the School of Computation, Information and Technology at the Technical University of Munich, 85748 Garching, Germany. `{gerald.wuersching, tobias.mascetta, yuanfei.lin, althoff}@tum.de`

[1] `https://github.com/ApolloAuto/apollo`

gorithms using real demonstrators [34]–[37]. As Autoware is a) becoming increasingly popular, b) offers a more modular framework and a larger ecosystem than Apollo [38], and c) has more users compared to newer open-source platforms such as Pylot [39] and AVstack [40], we choose Autoware as the middleware for this work.

B. Summarizing Assessment

Both CommonRoad and Autoware facilitate researchers and practitioners with means of evaluating motion planning algorithms. CommonRoad enables testing algorithms on a vast array of benchmark scenarios and ensures reproducibility of research results. Due to its low entry barrier, the platform allows users to implement and test motion planners in a rapid prototyping fashion. However, influences arising from other components, e.g., perception or control, are not considered. Autoware, in contrast, provides a full driving stack, which is directly deployable on a real vehicle. Yet, the entire software stack is highly complex, and thus, not directly suited for developing algorithms for research. Integrating algorithms directly into the existing stack is time-consuming and requires a certain level of proficiency with Autoware.

C. Contributions

The complementary capabilities of CommonRoad and Autoware show the gap between rapid prototyping and real-world experiments in motion planning research. We address precisely this issue: we present `CR2AW`, the first open-source interface between the CommonRoad framework and Autoware. In particular, `CR2AW`

- significantly simplifies the integration of planning algorithms running in the CommonRoad environment into a full driving stack;
- is designed in a modular fashion with algorithm-agnostic interfaces to effortlessly exchange and compare different planning algorithms;
- integrates seamlessly a wide range of publicly available CommonRoad tools into Autoware, thus making novel features such as a reachability analyzer [6] or a criticality evaluator [7] available in Autoware;
- allows users to automatically generate motion planning benchmarks in the CommonRoad format from data collected in real test drives.

The remainder of this paper is structured as follows: We introduce necessary preliminaries in Sec. II and provide an overview of our interface including its implementation details in Sec. III. In Sec. IV, we showcase our interface both in simulation and on a real test vehicle. Finally, we draw conclusions in Sec. V.

II. Preliminaries

A. CommonRoad Scenario Format

A CommonRoad scenario consists of an environment model and one or several planning problems (see Fig. 2).

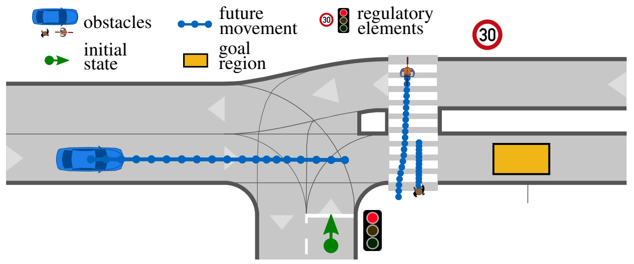


Fig. 2: Illustration of a CommonRoad scenario where the predicted movements of dynamic obstacles are represented by trajectories.

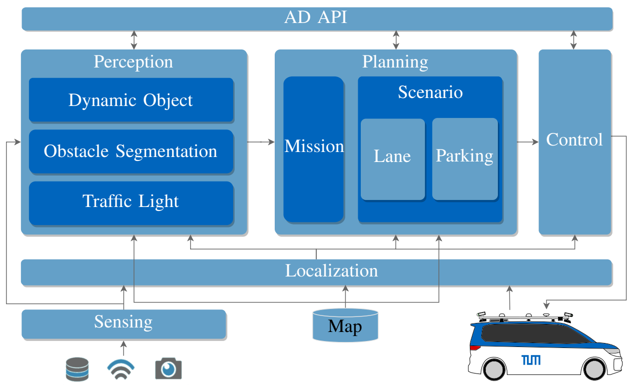


Fig. 3: Architecture of the components and modules in Autoware[2].

1) *Environment Model:* CommonRoad provides a detailed representation of the driving environment as a temporal sequence with a fixed step size Δt. This includes obstacles, road networks, and regulatory elements. Obstacles are characterized by their role, type, shape, and current state. For dynamic obstacles, a prediction of their future movement is specified, which can be provided as a trajectory; alternatives such as sets and probability density functions are also supported. The road network is constructed using a set of lanelets [41], which are modeled with left and right boundary polylines and a driving direction. A lanelet can reference regulatory elements, such as stop lines, traffic signs, and traffic lights, which are defined by their signal state, direction, and switching cycle. Notably, Autoware employs the Lanelet2 map format [42], which extends and generalizes the lanelets as described in [41]. Given that CommonRoad emphasizes motion planning while maintaining full compatibility with Lanelet2, we use the CommonRoad map format in this work.

2) *Motion Planning:* Motion planning problems for the *ego vehicle* consist of an initial state and a goal region. The goal region contains a set of desired states, which typically constrain the ego vehicle to reach certain positions, orientations, and velocities within a specified time.

B. Autoware.Universe

In this work, we adapt the Autoware.Universe framework illustrated in Fig. 3, which is built on ROS2 [43]. This framework provides all essential functionalities for autonomous driving, including perception, planning, and control, all

[2]Source: `https://github.com/autowarefoundation/autoware`

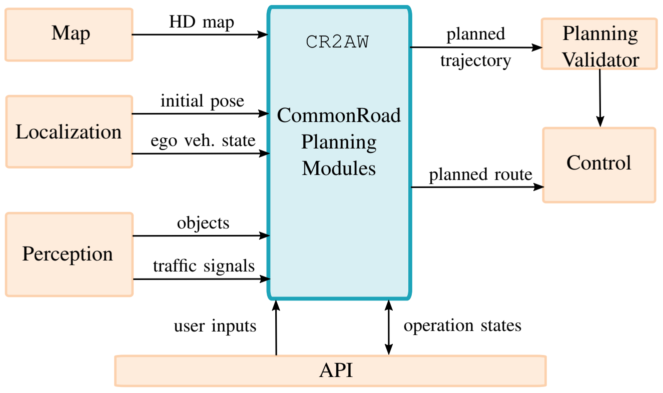


Fig. 4: Overview of the inputs and outputs of the `CR2AW` interface.

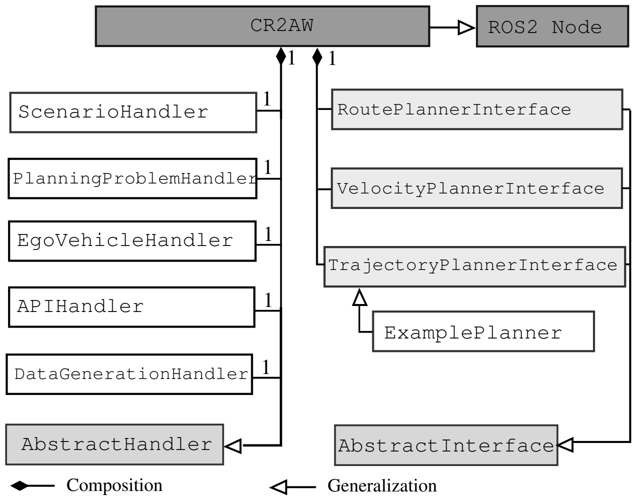


Fig. 5: Architecture of the `CR2AW` interface shown as a UML class diagram.

structured modularly. Moreover, Autoware.Universe includes an API, enabling external operation of the vehicle outside the autonomous driving system. Communication between these modules is facilitated through, the subscriber-publisher and request-response messaging patterns [43, Sec. III.C], utilizing predefined topics and services for efficient data exchange and coordination.

III. CommonRoad-Autoware Interface

We first describe the design goals of our CommonRoad-Autoware (`CR2AW`) interface in Sec. III-A. Next, we provide an overview of the integration of `CR2AW` into Autoware in Sec. III-B and highlight implementation details in Sec III-C.

A. Design Goals

In order to simplify the transfer of algorithms to a full driving stack, we base our implementation on the following design goals:

D1. **Ease of use:** To lower the entry barrier for real-world experiments, our interface manages all communication with the software stack, such that users are not required to re-implement their code.

D2. **Modularity:** To retain the modular architecture of Autoware, `CR2AW` integrates directly into the existing modules and does not define new interfaces.

D3. **Extensibility:** To facilitate benchmarking and comparisons of planners, `CR2AW` allows users to easily exchange and compose planning algorithms.

B. Overview

In Fig. 4 we illustrate how our interface is integrated into the full driving stack of Autoware. The relevant Autoware modules with their inputs and outputs to `CR2AW` are shown. Following design goal D2, we utilize the existing interfaces as defined by Autoware for all inputs and outputs in `CR2AW`.

1) Inputs: The map module provides a high-definition (HD) map in the Lanelet2 format. To obtain the initial state of the ego vehicle and update its current state for re-planning, we subscribe to the corresponding inputs from the localization module. The information about the dynamically changing environment is obtained from the perception module. We subscribe to both static and dynamic objects with their predictions as well as the signal state of surrounding traffic lights. Additionally, we subscribe to inputs originating from the API (cf. Sec. II-B), e.g., user inputs (such as setting a goal pose) and the operation state of the overall software.

2) Outputs: Each output of `CR2AW` is a computed route through the road network and the trajectory of the ego vehicle. Each trajectory is published periodically of a fixed frequency and first checked by the planning validation module of Autoware before being forwarded to the control module. Moreover, we return required information to the API module, e.g., the operation state of our planning module.

3) Planning Modules: The motion planning modules are encapsulated in `CR2AW` in which the planning problem is represented in the CommonRoad format (cf. Sec. II-A.2). An established approach for autonomous vehicles is to structure the planning task hierarchically into three levels, which differ in terms of how the planning problem is abstracted [44]: (global) route planning, (high-level) behavior planning and (low-level) trajectory planning. In `CR2AW` we follow a similar structure and provide abstract interfaces for planners at all three levels (cf. Sec. III-C).

C. Implementation Details

We now present the architecture of `CR2AW` (see Fig. 5) and highlight implementation details of its core modules. The class `CR2AW` serves as the main class of our interface and is implemented as a Python ROS2 node such that it can communicate with other nodes via subscribers and publishers. We choose Python to offer convenient prototyping capabilities to users (see D1), yet, our interface is also directly compatible with C++ code.

The core modules within `CR2AW` are grouped into two types of classes with distinct roles each, following the *separation of concerns* principle [45]. *Handler* classes are responsible for processing the input stream from other Autoware modules and converting the required information to CommonRoad. *Interface* classes solely operate within the CommonRoad environment. With this structure, we decouple the CommonRoad part from Autoware, which realizes

Algorithm 1 Overview of core functionality of `CR2AW`

```
Input: initial pose, goal pose, ego vehicle state, user input, objects $\mathbf{O}$, traffic signals $\mathbf{T}$, mapping $m_\mathrm{O}$, mapping $m_\mathrm{T}$   ▷ cf. Fig. 4
Output: planned trajectory, planned route, operation states (AW topics)   ▷ cf. Fig. 4
 1: CommonRoad scenario $S \leftarrow$ LOADCONVERTEDMAP(HD map)   ▷ cf. [9, Sec. III-C]
 2: if APIHANDLER.GET_LOCALIZATION_STATE() == INITIALIZED then
 3:   planning problem $P \leftarrow$ PLANNINGPROBLEMHANDLER.GENERATE(initial pose, goal pose)   ▷ cf. Tab. I (a)
 4:   ego vehicle $E \leftarrow$ EGOVEHICLEHANDLER.INITIALIZE(initial pose)   ▷ cf. Tab. I (a)
 5:   route, reference path $\leftarrow$ ROUTEPLANNERINTERFACE.PLAN($S$, $P$)
 6:   ROUTEPLANNERINTERFACE.PUBLISH(route)
 7:   APIHANDLER.SET_ROUTING_STATE(SET)
 8:   reference trajectory $\leftarrow$ VELOCITYPLANNERINTERFACE.PLAN(reference path, user_input.max_velocity)
 9:   data generator $D \leftarrow$ DATAGENERATIONHANDLER.START_RECORDING()
10:   goal_reached $\leftarrow$ False
11:   while not goal_reached do
12:     $S \leftarrow$ UPDATESCENARIO($S, \mathbf{O}, \mathbf{T}, m_\mathrm{O}, m_\mathrm{T}$)   ▷ cf. Alg. 2
13:     $E \leftarrow$ UPDATEEGOVEHICLE($E$, ego vehicle state)   ▷ cf. Tab. I (a)
14:     $\mathbf{traj}_\mathrm{planned} \leftarrow$ TRAJECTORYPLANNERINTERFACE.PLAN($S$, $E$, reference trajectory)
15:     TRAJECTORYPLANNERINTERFACE.PUBLISH($\mathbf{traj}_\mathrm{planned}$)   ▷ cf. Tab. I (c)
16:     goal_reached $\leftarrow$ PLANNINGPROBLEMHANDLER.CHECK_GOAL($E$)
17:   end while
18:   APIHANDLER.SET_ROUTING_STATE(ARRIVED)
19:   DATAGENERATIONHANDLER.STOP_RECORDING_AND_SAVE_DATA()
20: end if
```

our design goal D1. For both types of classes we define abstract classes which implement all required methods and attributes, such that new modules can easily be added via inheritance, conforming to design goal D3 (e.g., see class `ExamplePlanner` in Fig. 5). `CR2AW` consists of the following main modules:

- `ScenarioHandler`: processes map and perception inputs, creates and updates the CommonRoad scenario;
- `PlanningProblemHandler`: creates the CommonRoad planning problem;
- `EgoVehicleHandler`: processes localization information, updates the ego vehicle state for re-planning;
- `APIHandler`: manages communication with the API, receives user inputs and operation states;
- `DataGenerationHandler`: automatically generates benchmark scenarios and stores planning results;
- `RoutePlannerInterface`: general interface for a high-level planner to compute a route and reference path through the lanelet network;
- `VelocityPlannerInterface`: general interface to compute a velocity profile for the reference path;
- `TrajectoryPlannerInterface`: general interface for a low-level trajectory planner, which computes a feasible trajectory for the controller to execute.

The planner interface classes correspond to the hierarchical structure mentioned in Sec. III-B.3, such that users can integrate planning algorithms of different abstraction levels with `CR2AW`. Next, we describe the core working procedure of `CR2AW`, which is also summarized in Alg. 1.

1) Initializing and creating the planning problem: We initially load the converted map in the CommonRoad format, which consists of the lanelet network and regulatory elements (line 1). To automatically convert the original Lanelet2 map to CommonRoad and verify correctness of the converted

Algorithm 2 UPDATESCENARIO

```
Input: previous scenario $S$, list objects $\mathbf{O}$, list traffic signals $\mathbf{T}$,
    mapping $m_\mathrm{O}$, mapping $m_\mathrm{T}$
Output: updated scenario $S$
 1: time step $\Delta t \leftarrow S$.time_step
 2: for each object $o$ in $\mathbf{O}$ do
 3:   $\mathbf{traj}_o \leftarrow$ RESAMPLEPREDICTION($o, \Delta t$)   ▷ see [12]
 4:   Autoware-ID $\mathrm{id_{aw}} \leftarrow o$.object_id
 5:   if $\mathrm{id_{aw}}$ in $m_\mathrm{O}$ then
 6:     Commonroad-ID $\mathrm{id_{cr}} \leftarrow m_\mathrm{O}[\mathrm{id_{aw}}]$
 7:     $S$.UPDATEOBSTACLE($o$, $\mathbf{traj}_o$, $\mathrm{id_{cr}}$)
 8:   else
 9:     CommonRoad-ID $\mathrm{id_{cr}} \leftarrow$ GENERATEID($S$)
10:     $m_\mathrm{O}$.ADD($\mathrm{id_{aw}}$, $\mathrm{id_{cr}}$)
11:     $S$.ADDOBSTACLE($o$, $\mathbf{traj}_o$, $\mathrm{id_{cr}}$)
12:   end if
13: end for
14: $S, m_\mathrm{O} \leftarrow$ REMOVEOBSTACLES()
15: for each traffic signal $t$ in $\mathbf{T}$ do
16:   Autoware-ID $\mathrm{id_{aw}} \leftarrow t$.traffic_signal_id
17:   CommonRoad-ID $\mathrm{id_{cr}} \leftarrow m_\mathrm{T}[\mathrm{id_{aw}}]$
18:   $S$.UPDATETRAFFICLIGHT($\mathrm{id_{cr}}$, $t$.color, $t$.shape)
19: end for
20: return $S$
```

map, we use [9, Sec. III-C] and [46]. Our interface receives the initial pose from the localization module and a desired goal pose from the user input, which are used to construct the CommonRoad planning problem (lines 2-3). Then we initialize the state of the ego vehicle (line 4) using the received initial state. To convert the message types from Autoware to CommonRoad types, we use Tab. I (a).

2) Generating a route: Next, we generate a route and a reference path for the created planning problem in the lanelet network using a high-level planner in the `RoutePlannerInterface` (see line 5). If a valid route is found, we publish the route (line 6) and the `APIHandler` informs the other modules by setting the routing state accordingly (see line 7). Afterwards, a velocity profile is computed for the reference path by the `VelocityPlannerInterface`, considering a velocity limit which can be set by the user via the API (see line 8).

3) Running the planning loop: Starting in line 11 of Alg. 1, we run the planning loop, which is called periodically with a fixed planning frequency using a ROS timer. At the beginning of each loop (lines 12-13), we update the CommonRoad scenario and the ego vehicle with the inputs from the perception and localization module (cf. Fig. 4).

The procedure for updating the CommonRoad scenario is described in Alg. 2, which includes updating the state of objects and traffic lights. We iterate over the list of incoming objects $\mathbf{O}$ (line 2): First, we resample their predicted trajectory to match the time step of the CommonRoad scenario (line 3), similar to [12, Sec. 3.3]. To uniquely assign an incoming object to an existing CommonRoad object, we store a mapping m_{O} of the Autoware-ID to the CommonRoad-ID for each object. If the object already exists in the scenario, we update its state, shape and predicted trajectory using the type conversions in Tab. I (b)-(c) (see lines 5-7). If the object ID is new, we create a new obstacle in the CommonRoad scenario using the same type of conversion and store the ID in our mapping m_{O} (lines 9-11). Finally, we remove all obstacles from the scenario and from m_{O} which are not present in the tracked objects $\mathbf{O}$. Similarly, we keep track of a mapping m_{T} of the corresponding traffic light IDs in Autoware and CommonRoad. This mapping is generated during the initial conversion of the HD map (cf. line 1, Alg. 1). We update the state (consisting of the color and shape) of each traffic light t in the input list $\mathbf{T}$ (see line 16-19) using the type conversions in Tab. I (d).

After updating the scenario, we run the trajectory planner (see line 14, Alg. 1). The output trajectory $\mathbf{traj}_{\mathrm{planned}}$ is converted to the corresponding Autoware type by converting each planned state in the state list using Tab. I (c). Finally, we check if the goal is reached (see line 16). If yes, we exit the planning loop and set the routing state to `ARRIVED` in line 18. We note that it is also possible to have multiple goals along a global route. In this case, if the ego vehicle has reached the first goal (line 16), we use the next goal in the list and resume the planning loop in line 11.

4) Generating benchmark data: The `DataGeneration Handler` is a separate entity that subscribes to the same topics shown in Fig. 4. Thus, it handles the collection of data to the CommonRoad format in parallel to the other modules described in Fig. 5. `CR2AW` triggers the start of the data recording once the planning problem is computed (see line 8) and stops the recording once the goal is reached (see line 17). `CR2AW` supports the automatic generation of CommonRoad benchmark scenarios, including the road network, the planning problem, the reference trajectory, and dynamic obstacles. Additionally, we store data for evaluation purposes, such as the planned and measured states of the ego vehicle over time.

TABLE I: Type conversions between Autoware and CommonRoad.

Autoware	CommonRoad
(a) Ego Vehicle State	
Odometry.	State.
pose.pose.position	position
pose.pose.orientation	orientation
twist.twist.linear.x	velocity
twist.twist.angular.z	yaw_rate
AccellwithCovariance.	State.
accel.accel.linear.x	acceleration
(b) Objects	
PredictedObject.	DynamicObstacle.
classification	obstacle_type
kinematics.	initial_state.
initial_pose.pose.position	position
initial_pose.pose.orientation	orientation
initial_twist.twist.linear.x	velocity
shape.	obstacle_shape.
dimensions.x	length
dimensions.y	width
predicted_paths	prediction.trajectory
(c) State	
TrajectoryPoint.	State.
pose.position	position
pose.orientation	orientation
longitudinal_velocity_mps	velocity
acceleration_mps2	acceleration
heading_rate_rps	yaw_rate
front_wheel_angle_rad	steering_angle
(d) Traffic Light States	
color	TrafficLightState.
`RED, AMBER, GREEN`	`RED, YELLOW, GREEN`
shape	TrafficLightDirection.
`LEFT_ARROW, RIGHT_ARROW`	`LEFT, RIGHT`
`UP_ARROW`	`STRAIGHT`
`UP_LEFT_ARROW`	`LEFT_STRAIGHT`
`UP_RIGHT_ARROW`	`STRAIGHT_RIGHT`

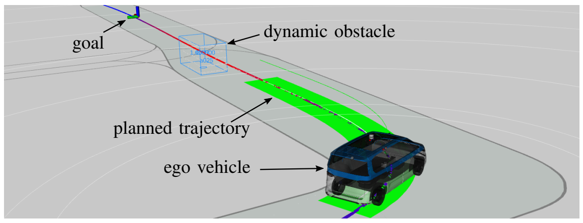
IV. Experiments

We tested the `CR2AW` interface in simulation and on our research vehicle EDGAR [37]. In both cases, we used a map from the Garching Campus of the Technical University of Munich. Within our interface, we utilized the following planning modules: As the route planner we used the CommonRoad Route Planner[3] and as the velocity planner we used the approach in [47]; As a trajectory planner we used the CommonRoad Reactive Planner[4], which implements the sampling-based approach of [48]. The planning algorithms are written in Python with some parts (e.g., collision checks) written in C++ for performance.

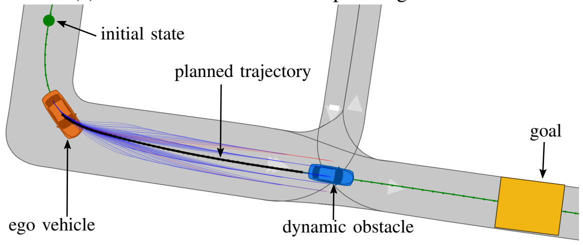
A. Experiment with the Autoware Planning Simulation

We tested our interface within the Autoware Planning Simulation. Fig. 6 provides an overview over the evaluation. We manually set up the initial pose and a desired goal pose as well as one dynamic obstacle in the Autoware planning simulation (Fig. 6a). The corresponding scenario in CommonRoad is shown in Fig. 6b, where the planning problem, the set of

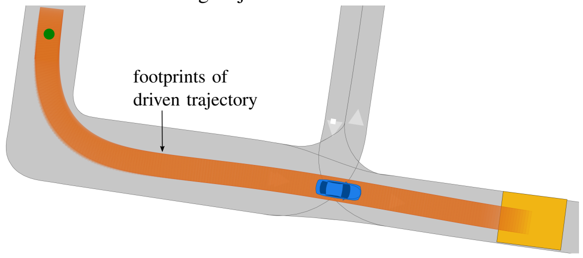
[3]`https://commonroad.in.tum.de/tools/route-planner`
[4]`https://commonroad.in.tum.de/tools/commonroad-reactive-planner`



(a) Scenario in the Autoware planning simulation.

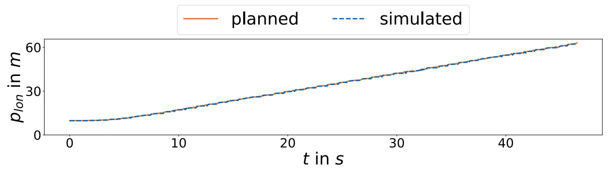


(b) Corresponding scenario in CommonRoad. Feasible trajectories are shown in blue and colliding trajectories are shown in red.

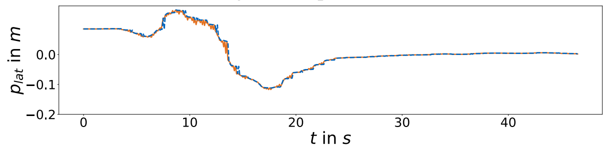


(c) Generated benchmark scenario with the footprints of the previously driven solution trajectory. Please note that the dynamic obstacle is shown at the initial state.

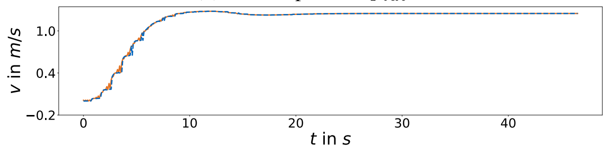
Fig. 6: Evaluation of `CR2AW` in the Autoware planning simulation with one dynamic obstacle.



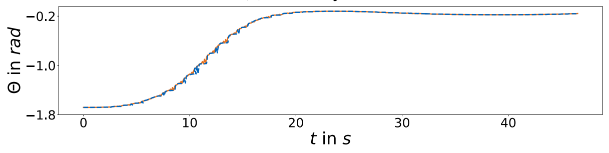
(a) Longitudinal position p_{lon}.



(b) Lateral position p_{lat}.

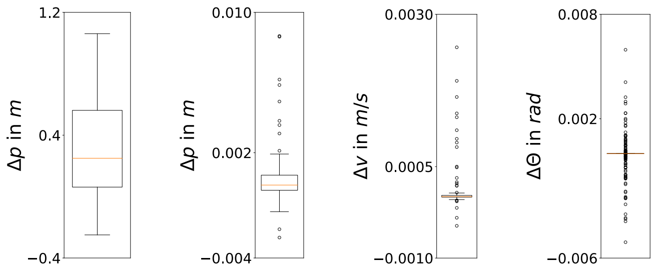


(c) Velocity v.



(d) Orientation θ.

Fig. 7: Planned and simulated states for the scenario in Fig. 6.



(a) Lon. position (b) Lat. position (c) Velocity (d) Orientation

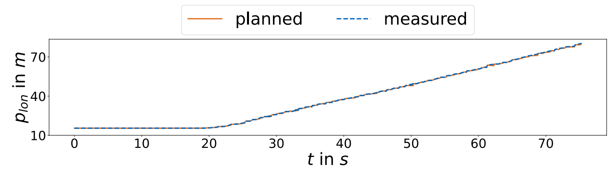
Fig. 8: Box plots of errors between planned and simulated states in Fig. 7.

sampled trajectories as well as the planned trajectory are visualized. Fig. 6c shows the benchmark scenario generated by the `DataGenerationHandler` (cf. Sec. III-C.4). In addition, we stored the driven trajectory of the ego vehicle, which can be used for benchmarking or improving the planner.

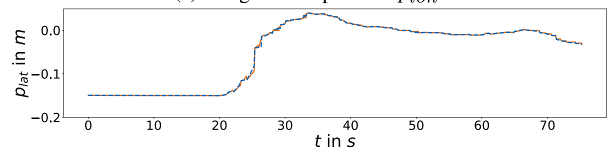
Moreover, the `DataGenerationHandler` stores quantitative data about the planned trajectory and simulated states executed by the controller, which we analyze in Fig. 7. The data includes the longitudinal position p_{lon}, the lateral position p_{lat}, the longitudinal velocity v and the orientation Θ. The planning simulation added a Gaussian measurement noise to the simulated states with standard deviations $\sigma_{\text{pos}} = 0.01$ for the position, $\sigma_\theta = 0.001$ for the orientation and $\sigma_{\text{steer}} = 0.001$ for the steering angle. Since the simulated measurement noise was relatively low, we observed that the controller follows the reference states of the planned trajectory well with only slight deviations. This is also evident when analyzing the error distributions in Fig. 8. Please note that we used the default implementation and parameterization of the controller in Autoware.Universe.

B. Experiment with EDGAR

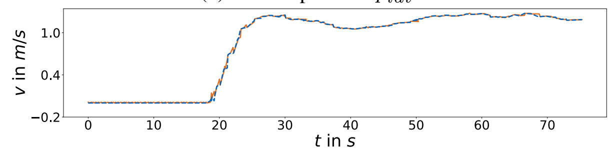
We integrated `CR2AW` in our research vehicle EDGAR and tested it in the urban scenario shown in Fig. 1, where the ego vehicle was driving on a straight road behind a slowly moving vehicle; another parked vehicle was alongside the road. Fig. 1b shows that we could reliably use the perception input from Autoware to continuously update the CommonRoad scenario (cf. Sec. III-C), which our planning algorithm can use to compute the desired trajectory. Again, we analyze the planned and measured state of the maneuver in Fig. 9 and the corresponding error distributions in Fig. 10. We notice that the measured state followed the planned trajectory well. The longitudinal position error tends to be negative, i.e., the measured position was slightly behind the planned position. We attribute this to the low re-planning frequency of our planner, which we set to 2 Hz for our experiment. Moreover, the lateral position error and the orientation error were very small.



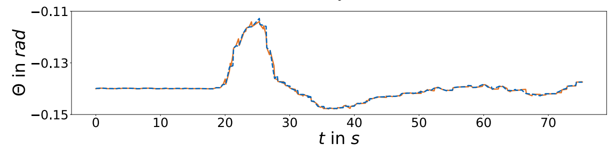
(a) Longitudinal position p_{lon}.



(b) Lateral position p_{lat}.

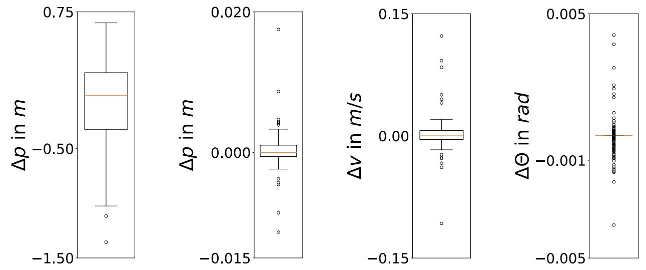


(c) Velocity v.



(d) Orientation θ.

Fig. 9: Planned and measured states for the scenario in Fig. 1.



(a) Lon. position (b) Lat. position (c) Velocity (d) Orientation

Fig. 10: Box plot of errors between planned and measured states in Fig. 9. Please note that the first 20 seconds (stand-still) are not included in the box plots.

V. Conclusions

We present CR2AW, a publicly available interface between CommonRoad and Autoware – two widely used open-source frameworks for evaluating motion planners in simulation and real-world settings. CR2AW lowers the entry barrier for real vehicle tests by significantly simplifying the integration of planning algorithms into a full driving stack. Thus, researchers and developers can first evaluate their planners on a multitude of CommonRoad benchmark scenarios before directly testing them on a real vehicle with Autoware using CR2AW. In addition, benchmark scenarios can be generated from test drives, which allows users to improve their algorithms afterwards for certain scenarios. We have demonstrated the capabilities of CR2AW both in simulation and with our research vehicle EDGAR.

Acknowledgments

This work was funded by the German Federal Ministry of Education and Research (BMBF) within the *Munich Cluster for the Future of Mobility in Metropolitan Regions (MCube)* under grant 03ZU1105AA, the German Federal Ministry for Digital and Transport (BMDV) within the project *Cooperative Autonomous Driving with Safety Guarantees* (KoSi) and the European Union, under the Horizon Europe program, grant 101076165 (i4Driving). Moreover, the authors thank Gemb Kaljavesi, Florian Pfab, and Rainer Trauth for assisting with the test drives and all students who assisted with the implementation.

References

[1] E. Yurtsever, J. Lambert, A. Carballo, and K. Takeda, "A survey of autonomous driving: Common practices and emerging technologies," *IEEE access*, vol. 8, pp. 58 443–58 469, 2020.

[2] M. Althoff, M. Koschi, and S. Manzinger, "CommonRoad: Composable benchmarks for motion planning on roads," in *Proc. of the IEEE Intell. Veh. Symp.*, 2017, pp. 719–726.

[3] S. Kato, S. Tokunaga, Y. Maruyama, S. Maeda, M. Hirabayashi, Y. Kitsukawa, A. Monrroy, T. Ando, Y. Fujii, and T. Azumi, "Autoware on board: Enabling autonomous vehicles with embedded systems," in *Proc. of the ACM/IEEE Int. Conf. on Cyber-Physical Systems*, 2018, pp. 287–296.

[4] C. Pek, V. Rusinov, S. Manzinger, M. C. Üste, and M. Althoff, "CommonRoad drivability checker: Simplifying the development and validation of motion planning algorithms," in *Proc. the IEEE Intell. Veh. Symp.*, 2020, pp. 1013–1020.

[5] M. Koschi and M. Althoff, "SPOT: A tool for set-based prediction of traffic participants," in *Proc. the IEEE Intell. Veh. Symp.*, 2017, pp. 1686–1693.

[6] E. I. Liu, G. Würsching, M. Klischat, and M. Althoff, "CommonRoad-Reach: A toolbox for reachability analysis of automated vehicles," in *Proc. the IEEE Int. Conf. on Intell. Transp. Syst.*, 2022, pp. 2313–2320.

[7] Y. Lin and M. Althoff, "CommonRoad-CriMe: A toolbox for criticality measures of autonomous vehicles," in *Proc. the IEEE Intell. Veh. Symp.*, 2023, pp. 1–8.

[8] M. Althoff, S. Urban, and M. Koschi, "Automatic conversion of road networks from OpenDRIVE to lanelets," in *Proc. the IEEE Int. Conf. on Service Operations and Logistics, and Info.*, 2018, pp. 157–162.

[9] S. Maierhofer, M. Klischat, and M. Althoff, "Commonroad scenario designer: An open-source toolbox for map conversion and scenario creation for autonomous vehicles," in *Proc. the IEEE Int. Conf. on Intell. Transp. Syst.*, 2021, pp. 3176–3182.

[10] Y. Lin, M. Ratzel, and M. Althoff, "Automatic traffic scenario conversion from OpenSCENARIO to CommonRoad," in *Proc. the IEEE Int. Conf. on Intell. Transp. Syst.*, 2023.

[11] M. Klischat, O. Dragoi, M. Eissa, and M. Althoff, "Coupling SUMO with a motion planning framework for automated vehicles," in *SUMO User Conf.*, 2019, pp. 1–9.

[12] X. Wang, A.-K. Rettinger, M. T. B. Waez, and M. Althoff, "Coupling Apollo with the CommonRoad motion planning framework," in *FISITA World Congress*, 2020.

[13] M. Klischat, E. I. Liu, F. Holtke, and M. Althoff, "Scenario factory: Creating safety-critical traffic scenarios for automated vehicles," in *Proc. the IEEE Int. Conf. on Intell. Transp. Syst.*, 2020, pp. 1–7.

[14] X. Wang, H. Krasowski, and M. Althoff, "Commonroad-RL: A configurable reinforcement learning environment for motion planning of autonomous vehicles," in *Proc. the IEEE Int. Conf. on Intell. Transp. Syst.*, 2021, pp. 466–472.

[15] S. Khaitan and J. M. Dolan, "State dropout-based curriculum reinforcement learning for self-driving at unsignalized intersections," in *Proc. the IEEE Int. Conf. on Intell. Robots and Sys.*, 2022, pp. 12 219–12 224.

[16] E. Meyer, M. Brenner, B. Zhang, M. Schickert, B. Musani, and M. Althoff, "Geometric deep learning for autonomous driving: Unlocking the power of graph neural networks with CommonRoad-Geometric," in *Proc. the IEEE Intell. Veh. Symp.*, 2023, pp. 1–8.

[17] T. Nyberg, C. Pek, L. Dal Col, C. Norén, and J. Tumova, "Risk-aware motion planning for autonomous vehicles with safety specifications," in *Proc. the IEEE Intell. Veh. Symp.*, 2021, pp. 1016–1023.

[18] A. Zanardi, G. Zardini, S. Srinivasan, S. Bolognani, A. Censi, F. Dörfler, and E. Frazzoli, "Posetal games: Efficiency, existence, and refinement of equilibria in games with prioritized metrics," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 1292–1299, 2021.

[19] J. Li, X. Xie, Q. Lin, J. He, and J. M. Dolan, "Motion planning by search in derivative space and convex optimization with enlarged solution space," in *Proc. the IEEE Int. Conf. on Intell. Robots and Sys.*, 2022, pp. 13 500–13 507.

[20] M. Geisslinger, F. Poszler, and M. Lienkamp, "An ethical trajectory planning algorithm for autonomous vehicles," *Nature Machine Intell.*, vol. 5, no. 2, pp. 137–144, 2023.

[21] S. Deolasee, Q. Lin, J. Li, and J. M. Dolan, "Spatio-temporal motion planning for autonomous vehicles with trapezoidal prism corridors and Bézier curves," in *Proc. of the American Control Conf.*, 2023, pp. 3207–3214.

[22] R. Trauth, M. Kaufeld, M. Geisslinger, and J. Betz, "Learning and adapting behavior of autonomous vehicles through inverse reinforcement learning," in *Proc. the IEEE Intell. Veh. Symp.*, 2023, pp. 1–8.

[23] N. Kochdumper and S. Bak, "Real-time capable decision making for autonomous driving using reachable sets," *arXiv preprint arXiv:2309.12289*, 2023.

[24] R. Trauth, K. Moller, and J. Betz, "Toward safer autonomous vehicles: Occlusion-aware trajectory planning to minimize risky behavior," *IEEE Open Journal of Intell. Transp. Syst.*, vol. 4, pp. 929–942, 2023.

[25] M. Geisslinger, R. Trauth, G. Kaljavesi, and M. Lienkamp, "Maximum acceptable risk as criterion for decision-making in autonomous vehicle trajectory planning," *IEEE Open Journal of Intell. Transp. Syst.*, vol. 4, pp. 570–579, 2023.

[26] S. Sun, J. Chen, J. Sun, C. Yuan, Y. Li, T. Zhang, and M. H. Ang, "FISS+: Efficient and focused trajectory generation and refinement using fast iterative search and sampling strategy," in *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 2023, pp. 10 527–10 534.

[27] G. Würsching and M. Althoff, "Sampling-Based Optimal Trajectory Generation for Autonomous Vehicles Using Reachable Sets," in *Proc. of the IEEE Int. Conf. on Intell. Transp. Syst.*, 2021, pp. 828–835.

[28] R. Kensbock, M. Nezami, and G. Schildbach, "Scenario-based decision-making, planning and control for interaction-aware autonomous driving on highways," in *Proc. the IEEE Intell. Veh. Symp.*, 2023, pp. 1–6.

[29] C. Urmson, J. Anhalt, D. Bagnell, C. Baker, R. Bittner, M. Clark, J. Dolan, D. Duggins, T. Galatali, C. Geyer, *et al.*, "Autonomous driving in urban environments: Boss and the urban challenge," *Journal of field Robotics*, vol. 25, no. 8, pp. 425–466, 2008.

[30] M. Montemerlo, J. Becker, S. Bhat, H. Dahlkamp, D. Dolgov, S. Ettinger, D. Haehnel, T. Hilden, G. Hoffmann, B. Huhnke, *et al.*, "Junior: The Stanford entry in the urban challenge," *Journal of Field Robotics*, vol. 25, no. 9, pp. 569–597, 2008.

[31] J. Ziegler, P. Bender, M. Schreiber, H. Lategahn, T. Strauss, C. Stiller, T. Dang, U. Franke, N. Appenrodt, C. G. Keller, *et al.*, "Making Bertha drive—an autonomous journey on a historic route," *IEEE Intell. Transp. Syst. magazine*, vol. 6, no. 2, pp. 8–20, 2014.

[32] Ö. Ş. Taş, F. Kuhnt, J. M. Zöllner, and C. Stiller, "Functional system architectures towards fully automated driving," in *Proc. the IEEE Intell. Veh. Symp.*, 2016, pp. 304–309.

[33] K. Burnett, J. Qian, X. Du, L. Liu, D. J. Yoon, T. Shen, S. Sun, S. Samavi, M. J. Sorocky, M. Bianchi, *et al.*, "Zeus: A system description of the two-time winner of the collegiate SAE autodrive competition," *Journal of Field Robotics*, vol. 38, no. 1, pp. 139–166, 2021.

[34] T. Kessler, J. Bernhard, M. Buechel, K. Esterle, P. Hart, D. Malovetz, M. T. Le, F. Diehl, T. Brunner, and A. Knoll, "Bridging the gap between open source software and vehicle hardware for autonomous driving," in *Proc. the IEEE Intell. Veh. Symp.*, 2019, pp. 1612–1619.

[35] M. Tsukada, T. Oi, A. Ito, M. Hirata, and H. Esaki, "AutoC2X: Open-source software to realize V2X cooperative perception among autonomous vehicles," in *Proc. of the IEEE Vehicular Technology Conf.*, 2020, pp. 1–6.

[36] Z. Zang, R. Tumu, J. Betz, H. Zheng, and R. Mangharam, "Winning the 3rd Japan Automotive AI Challenge-autonomous racing with the Autoware.Auto open source software stack," in *Proc. the IEEE Intell. Veh. Symp.*, 2022, pp. 1757–1764.

[37] P. Karle, T. Betz, M. Bosk, F. Fent, N. Gehrke, M. Geisslinger, L. Gressenbuch, P. Hafemann, S. Huber, M. Hübner, *et al.*, "EDGAR: An autonomous driving research platform–from feature development to real-world application," *arXiv preprint arXiv:2309.15492*, 2023.

[38] V. M. Raju, V. Gupta, and S. Lomate, "Performance of open autonomous vehicle platforms: Autoware and Apollo," in *Proc. of the IEEE Int. Conf. for Convergence in Technology*, 2019, pp. 1–5.

[39] I. Gog, S. Kalra, P. Schafhalter, M. A. Wright, J. E. Gonzalez, and I. Stoica, "Pylot: A modular platform for exploring latency-accuracy tradeoffs in autonomous vehicles," in *Proc. of the IEEE Int. Conf. on Robotics and Automation*, 2021, pp. 8806–8813.

[40] R. S. Hallyburton, S. Zhang, and M. Pajic, "AVstack: An open-source, reconfigurable platform for autonomous vehicle development," in *Proc. of the ACM/IEEE Int. Conf. on Cyber-Physical Systems*, 2023, pp. 209–220.

[41] P. Bender, J. Ziegler, and C. Stiller, "Lanelets: Efficient map representation for autonomous driving," in *Proc. of the IEEE Intell. Veh. Symp.*, 2014, pp. 420–425.

[42] F. Poggenhans, J.-H. Pauls, J. Janosovits, S. Orf, M. Naumann, F. Kuhnt, and M. Mayr, "Lanelet2: A high-definition map framework for the future of automated driving," in *Proc. of the IEEE Int. Conf. on Intell. Transp. Syst.*, 2018, pp. 1672–1679.

[43] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, and W. Woodall, "Robot operating system 2: Design, architecture, and uses in the wild," *Science Robotics*, vol. 7, no. 66, 2022.

[44] B. Paden, M. Čáp, S. Z. Yong, D. Yershov, and E. Frazzoli, "A survey of motion planning and control techniques for self-driving urban vehicles," *IEEE Transactions on intelligent vehicles*, vol. 1, no. 1, pp. 33–55, 2016.

[45] R. C. Martin, *Clean Architecture: A Craftsman's Guide to Software Structure and Design*. Prentice Hall, 2017.

[46] S. Maierhofer, Y. Ballnath, and M. Althoff, "Map verification and repairing using formalized map specifications," in *Proc. of the IEEE Int. Conf. on Intell. Transp. Syst.*, 2023.

[47] Y. Shimizu, T. Horibe, F. Watanabe, and S. Kato, "Jerk constrained velocity planning for an autonomous vehicle: Linear programming approach," in *Proc. of the IEEE Int. Conf. on Robotics and Automation*, 2022, pp. 5814–5820.

[48] M. Werling, J. Ziegler, S. Kammel, and S. Thrun, "Optimal trajectory generation for dynamic street scenarios in a Frenét frame," in *Proc. of the IEEE Int. Conf. on Robotics and Automation*, 2010, pp. 987–993.