Mixed-Integer Motion Planning on German Roads within the Apollo Driving Stack

Tobias Kessler[1], Klemens Esterle[1], and Alois Knoll[2]

Abstract*—Traffic situations with interacting participants pose difficulties for today's autonomous vehicles to interpret situations and eventually achieve their own mission goal. Interactive planning approaches are promising solutions for solving such situations. However, most approaches are only assessed in simulation, as researchers lack the resources to operate an autonomous vehicle. Likewise, open-source stacks for autonomous driving, such as Apollo, provide competitive and resource-efficient state-of-the-art planning algorithms. However, promising planning concepts from research are usually not included within a reasonable time, possibly due to resource restrictions or technical limitations. Without evaluating these novel algorithms in reality, the benefits and shortcomings of proposed approaches cannot be thoroughly assessed. This work aims to contribute methodology and implementation to integrate a novel mixed-integer optimization-based planning algorithm in Apollo's planning component and assess its performance and real-time capability in theory and practice. It discusses the necessary modifications of Apollo for deployment on a different vehicle and presents three real-world driving experiments on a public road alongside a detailed experience report. The driving experiments show a smooth trajectory tracking performance operating robustly under varying perception data quality and the real-time capability of the closed-loop system.

I. Introduction

As long as autonomous vehicles share the road with human drivers, they need to make decisions interactively and cooperatively with them. Otherwise, dead-locks or dangerous situations may arise. Besides simulating these situations, testing and validating novel driving functions for motion or behavior planning requires closed-loop driving tests on vehicles equipped with a complete software stack. Research groups are usually unable to develop and operate such a state-of-the-art autonomous driving stack. Open-source stacks such as Apollo [1] or Autoware [2] can help research groups to conduct and validate their research on real demonstrators, not just in simulation environments. Applying such a stack to a new car is usually not straightforward: The vehicle actuation interfaces, network communication, and sensor configurations vary due to hardware availability, requirements, and local authorities. Our previous work [3] demonstrates how to apply the Apollo stack to a street-legal prototype vehicle in Germany.

As our goal is to validate the Mixed Integer Interactive Planning (MINIVAN) [4–6] algorithm for behavior and motion planning on the road, this work describes the process of integrating it in the stack. It also provides a thorough evaluation in multiple test scenarios on public roads in Germany, Fig. 1 illustrates one. By doing this, we display the applicability, capabilities, and limitations of MINIVAN in a full-size autonomous vehicle setup. Specifically, we contribute

- a thorough analysis of Apollo's architecture and planning module,
- a methodology on extending and working with Apollo,
- the demonstration of mixed-integer planning on a real road in a full-stack setup,
- and a set of open-source software modules [7] as an extension for Apollo to interface our Volkswagen Passat.

This paper is structured as follows: After reviewing existing open-source stacks, the planning approaches of Apollo, and interactive planning approaches in Sec. II, we introduce our planner MINIVAN in Sec. III. Starting from the system setup, Sec. IV introduces our adaptions made to Apollo. We elaborate on which modules we changed, replaced, or added. The integration of MINIVAN in Apollo is described in Sec. V. We discuss the pre- and postprocessing steps and all relevant software modules in detail. We then evaluate the complexity of MINIVAN with respect to a growing number of objects in Sec. VI in simulation to assess real-time capability. Sec. VII shows results from on-road evaluations analyzing timing and trajectory tracking aspects. We finish by summarizing our lessons learned in Sec. VIII, on the Apollo platform and the system setup and offer our conclusions in Sec. IX.

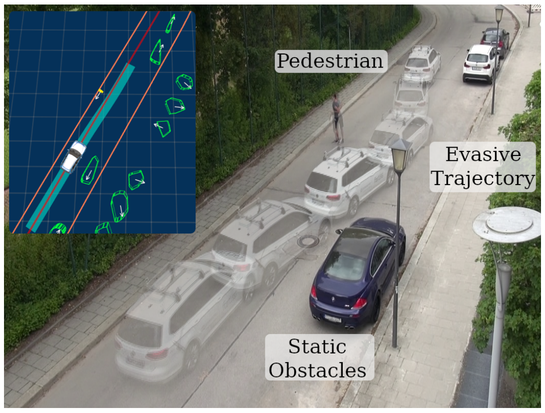


Fig. 1. Illustration of one benchmark scenario: An evasive maneuver around a pedestrian blocking parts of the lane.

II. Related Work

This section will discuss open-source autonomous driving stacks, especially Apollo, focusing on the planning component. We then briefly discuss interactive planning approaches.

[1] Tobias Kessler and Klemens Esterle are with fortiss GmbH, Research Institute of the Free State of Bavaria, Munich, Germany. Klemens Esterle left fortiss GmbH before the publication of this work.

[2] Alois Knoll is with Robotics, Artificial Intelligence and Real-time Systems, Technische Universität München, Munich, Germany.

A. Open-Source Autonomous Driving Stacks

Autoware.AI (formally known as Autoware [8]) is an open-source software stack for autonomous driving based on the Robot Operating System (ROS). It contains modules for localization, perception, prediction, and planning. The localization is based on a High Definition Map (HD-Map), Simultaneous Localization and Mapping (SLAM), and Global Navigation Satellite System (GNSS)/Inertial Measurement Unit (IMU) data. Perception fuses data from camera and lidar sensors. Planning uses probabilistic and rule-based techniques solving open space (e.g., parking) and lane following scenarios.

Autoware.Auto [9] is a recent reimplementation based on ROS 2 of Autoware.AI [2] with well-defined interfaces and an additional focus on functional safety. It is developed under the Autoware Foundation. Currently, the new stack supports valet parking but no comprehensive on-lane driving.

Baidu's Apollo [1] is an open-source autonomous driving stack with its customized middleware. The framework has been adapted or used in various works. Wang *et al.* [10] connected Apollo to a motion planning benchmark tool [11]. Fremont *et al.* [12] used Apollo to demonstrate the simulation to reality gap for a scenario-based testing approach.

Apollo comes with Cyber RT [13], a customized middleware for autonomous driving. Like popular robotic middlewares, Cyber RT is based on a publish-subscribe principle to interconnect different software components. The system is fully distributed and offers easy-to-use C++ and Python Application Programming Interfaces (APIs) to integrate new components. Similar to ROS 2, no central core component has to be started; each component is launched as a separate task in the userspace. Cyber RT resolves the components' data input/output dependencies and correctly links the components in a directed acyclic graph. The message format of Cyber RT is based on Google Protocol Buffers.

Apollo and Autoware have a similar scope, as both aim to provide an Software Development Kit (SDK) in conjunction with algorithms such as planning or data fusion. Raju *et al.* [14] have compared the functional components between Apollo and Autoware.AI and state that the functional modules in Apollo are more robust and performant at the cost of leaving the generic ROS environment. Apex.AI offers a middleware called Apex.OS that extends ROS 2 and promises hard real-time capability. However, studying and comparing potential advantages of the Autoware products over Apollo regarding hard real-time requirements were out of scope for this work.

We decided to deploy Apollo on our vehicle since Apollo already supports on-road driving, is developed rapidly, and follows a modern software design. We described our vehicle setup in [3]. We have since updated our system to Apollo v5.5.

Hardly any other publically available comprehensive stacks exist. NVIDIA's DriveWorks [15] is a SDK for developing driving functions. It also provides reference implementations for perception, localization, and planning. However, the stack is not open source and only supports NVIDIA hardware.

B. The Apollo Planning Module

Apollo's planning module offers several planning algorithms [16]. All are based on the same interface and take the vehicle ego state and the predicted positions or trajectories of obstacles as an input. A new planning is triggered for each new prediction message. The code is barely documented but follows modern and understandable coding patterns.

1) Public Road Planner: The public road planner is designed as a modular and holistic approach to generate suitable trajectories for various kinds of distinct driving situations by decoupling the selection of the current situation from the actual planning algorithm and its parametrization. Based on the scenario the vehicle is currently in, a predefined set of tasks is executed. This modular and flexible implementation comes at the downside of various switching and blending layers in the planner. Four classes of situations can be handled:

1) *Lane Following and Overtaking*: Track the desired velocity in a particular lane or follow the vehicle in front, perform lane changes and overtaking of slow vehicles.
2) *Intersections*: Handle signalized and unsignalized intersections with or without traffic signs according to the (American) traffic rules. When approaching a stop sign or performing an unprotected turn at a signalized intersection, the vehicle first comes to a complete stop and then creeps in the intersection to find a safe gap.
3) *Parking*: Maneuver from a lane into a parking spot or reverse. Here the *Open Space Planner* described in Sec. II-B2 is triggered.
4) *Emergency*: In case of failures, two emergency maneuvers are possible. Either the Open Space Planner drives the vehicle to the curb, or the vehicle stops in the current lane while preventing rear collisions.

The actual motion planning uses path-velocity decomposition. First, the vehicle state is translated into the Frenet frame. For this, a smooth reference line (up to the curvature derivative) is generated from the reference line using Quadratic Programming (QP). Second, the lane boundaries and static obstacles in front of the vehicle are converted into the Frenet frame. Selecting a homotopy to either pass an obstacle on the right or left side is usually a non-trivial task [17]. Apollo solves the homotopy class selection problem with a heuristic search. Third, a QP for the lateral deviation from the reference line is defined. The longitudinal interval on the reference line is set to a fixed resolution. The optimization problem models constraints to be kinematically feasible collision-free. It minimizes lateral reference deviation and lateral movement in a piecewise-jerk fashion. For longitudinal movement along the reference line, a similar QP is defined. Moving obstacles are considered by setting an appropriate reference speed for following a vehicle or merging in. Longitudinal and lateral trajectories are then combined and transformed back from the Frenet frame to the Cartesian space [18]. The algorithms have proven to master various on-road driving scenarios and compute smooth and collision-free results. The computation time is comparably low on standard hardware.

2) Open Space Planner: The Open Space Planner is a dedicated module for maneuvering in free-space scenarios without a lane structure, such as parking. As a basis, the Hybrid-A* path planning algorithm is used. In a subsequent step, the curvature of this collision-free path is smoothed using the Dual-Loop Iterative Anchoring Path Smoothing algorithm [19]. Afterward, a speed profile along the path is generated by formulating a constraint QP that minimizes the jerk [19, 20]. The planning algorithm shows a fast, reliable performance in simulated and real-road scenarios.

3) Limitations of the Apollo Planner: The planning module produces good baseline results in most of the tested scenarios. Nevertheless, we identified the following shortcomings of Apollo's current planning module:

1) Lack of interactive or cooperative planning,
2) Lack of transferability to arbitrary situations,
3) Trajectories are not $\mathcal{C}^3$-smooth.

We now discuss them in more depth.

The trajectory planners have an excellent performance in terms of evaluation time. A planning frequency of $100\,\mathrm{ms}$ is easily achieved. Due to this property, the planner is reactive to dynamic obstacles around the vehicle. However, it is barely interactive as it avoids the predicted trajectories of dynamic obstacles without the intention to influence or negotiate with others. The applied path-velocity decomposition in the planner also prevents the creation of interactive trajectories.

Apollo's rule-based planning module is designed for a specific set of driving maneuver classes, such as lane following or lane changing. For each of those, the detected traffic participants are processed differently before being fed to the planner. Designing this and tuning the triggering and translating between those requires significant engineering effort. No template or generic methodology is (yet) available to enhance the number of supported maneuver classes.

Apollo's trajectory planner imposes constraints on the controller stabilizing the trajectory. For example, the trajectories are not always $\mathcal{C}^3$-smooth, and therefore, the tracking controller has to smooth the trajectory to achieve a smooth driving behavior. This smoothing, however, modifies the planned trajectory and thus might not satisfy all objectives and constraints of the planner anymore.

The shortcomings of the Apollo planner motivate us to replace it with a more generic planning implementation that can generate interactive plans for multiple agents.

C. Interactive Planning

Game-theoretic approaches offer an elegant way to model interactions and dependencies between agents [21–24]. Based on the collaborative premise, the ego agent can incorporate future interactions with human-driven vehicles into its planning scheme. A decision tree often realizes the game-theoretic setting by sampling primitive actions, which has the disadvantage that the search-space discretization may eliminate solutions in convoluted solution spaces.

With Reinforcement Learning (RL) interactive behavior does not need to be modeled explicitly using models or multi-agent settings but can be learned implicitly from data. RL can be considered an offline planning algorithm and thus inherently will be tasked during execution to generalize to unforeseen environments. Past works employed prediction or formal methods for safe exploration [25–27], counterfactual reasoning for safe execution [28], learning on abstract semantic representations [29], or fusing learning with search-based techniques [30–32]. However, safety, non-interpretability, and efficiency of training data are just some of the problems that remain before their usage in safety-critical applications.

Optimal control approaches have employed Mixed-Integer Progamming (MIP) for cooperative multi-agent planning [33–35], as it can find an optimal solution to the usually combinatoric problem. However, some approaches [33, 34] only work for straight driving on straight roads, as they cannot encode the non-holonomy in the optimization problem. Similar problems can be expected from [35], where the abstracted discrete lateral action and state-space will complicate applying this approach in reality. Our previous work [4, 5] mitigates these shortcomings and can be applied to arbitrary roads or driving situations, which we will apply to reality in this work.

III. The MINIVAN Model

In the following, we briefly describe the Mixed Integer Interactive Planning (MINIVAN) algorithm that is used for motion planning in this work. The model is solved using Mixed-Integer Quadratic Programming (MIQP). For a detailed model formulation, we refer to our previous works [4, 5]. The implementation is available as open-source software [6].

1) Linear Motion Model: We model the vehicle as a third-order point-mass system in a global Cartesian frame with positions p_x, p_y, velocities v_y, v_y, and accelerations a_x, a_y as states. Direction-wise jerk in both directions j_x and j_y are the inputs of the model. We approximate the vehicle's shape using circles for the collision checking. However, the actual front axle position is not directly known from the linear model as the vehicle yaw angle is not part of the state space. Its computation is highly non-linear and cannot be used in MIQP. We thus define *regions* in the (v_x, v_y) plane, allowing us to obtain an approximation of the orientation based on linear inequalities only dependent on v_x and v_y. We impose region-dependent limits on acceleration and jerk to model the vehicle dynamics. Computing the upper and lower bounds for sine and cosine functions of the orientation leads to upper and lower bounds for the front axle position. For obtaining the lower and upper bounds for sine and cosine functions of the actual orientation, we fit linear polynomials on v_x and v_y.

2) Modeling the Non-Holonomics: To ensure the non-holonomics of a vehicle for a point-mass system, we need to constrain the curvature, which again is highly non-linear. We thus model the non-holonomics as linear constraints on a_y. For this, the curvature is approximated using a linear combination of v_x, v_y, a_x, and is solved for a_y.

3) Road Environment and Obstacle Avoidance: A polygonal representation of the environment is used and deflated with the radius of the collision circles. Non-convex environment polygons need to be split into several convex sub-polygons. We enforce the vehicle to be in at least one of these convex sub-polygons. This way, the vehicle-to-environment-polygon collision check becomes a point-to-polygon check for each circle used to approximate the vehicle. Likewise, we ensure that the vehicle does not collide with an arbitrary number of static or dynamic convex obstacle polygons, such as those obtained from a prediction.

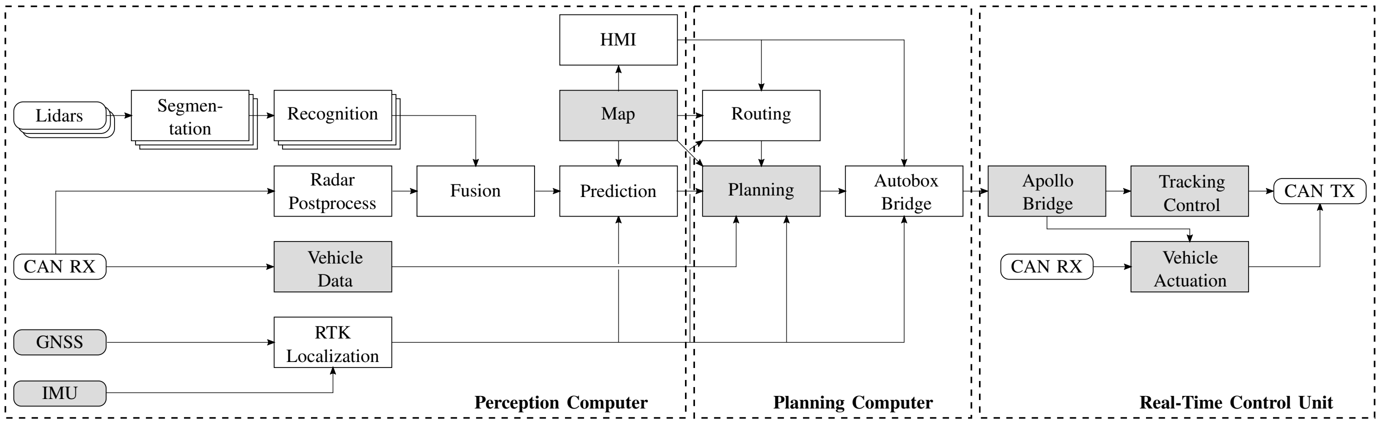


Fig. 2. Architecture overview of the setup we chose to operate the vehicle. Driver components are visualized using smaller boxes with rounded corners. The components with main contributions of this work are shown in gray. The object lists from Radar sensors remain unused in this work.

4) Cost function: We choose the weighted sum of position and velocity distance to the reference trajectory as a cost function for the ego vehicle. For acceleration and jerk, we aim to minimize the squared values.

5) Multi-Agent Planning: To facilitate interactive planning, we define the state space as the set of interacting agents. Each agent in the optimization problem satisfies the constraints described before. We employ linear constraints for agent-to-agent collision checking and a leveraged joint cost function to model cooperative, egoistic, or altruistic behavior.

IV. Adaptions to the Driving Stack Apollo

This section focuses on the extensions we implemented to run Apollo on our vehicle. These extensions are available as open-source software [7]. Apollo provides a variety of prebuilt functionality and a well-structured codebase. We aimed to introduce minimal changes to the Apollo stack to maintain the original functionality. Besides parameterization for our vehicle, we used the perception pipeline, the prediction module, the routing modules, and the Human Machine Interface (HMI) as implemented in Apollo. We modified the sensor drivers, implemented a different controller module, and developed several adapters and software components to connect the vehicle hardware with the Apollo stack. Each of these modifications served a specific need that we elaborate on in this section. The fact that we were able to introduce these changes into the existing system rather shows the flexibility of Apollo than its shortcomings. The customization of the stack to operate on a different vehicle platform using a modified planning and control system will be described in this section. Due to the different sensor sets and driving use-cases, we do not present a quantitative comparative study of the off-the-shelf Apollo system and our modifications.

A. System Overview

Apollo follows a classical sense-plan-act architecture pattern. Each component is started as a separate process, and the messages are exchanged among the components using Apollo's custom publish-subscribe middleware Cyber RT. This middleware makes the integration of customized components straightforward. We followed Apollo's architecture pattern as sketched in Fig. 2 but replaced some components with custom implementations. We also reflected the architecture by separating each architectural layer on an individual hardware computer. For perception and planning, we used two standard in-vehicle computers. The most significant architectural change is the separation of the vehicle controller from Apollo, placing the control layer on a dedicated real-time control unit.

B. Perception Pipeline and Obstacle Motion Prediction

Our sensor setup is different from the recommended Apollo hardware setup, which led us to implement an adapted perception pipeline. As we focus on the planning and control layer here, we only used a minimal set of sensors, mainly a 32-layer lidar on the roof center, and the differential GNSS system for localization. The radar and vision pipelines are not used in this work. The sensor data fusion component thus only post-processes the objects detected from lidar.

1) Ego Localization: Since our localization hardware is different from the Apollo reference vehicle, we had to develop a customized localization adapter. As the differential GNSS is supposed to provide highly accurate and consistent measurements, we decided to feed an already filtered GNSS position and velocity plus the IMU data from the same device into the stack without changing Apollo's standard messages. Doing so, we did not introduce changes in the build-in Real-Time Kinematic (RTK) localization module that combines GNSS and IMU messages to a consistent localization message and provides the dynamic transformation from the world frame to the vehicle reference frame.

2) Lidar Pipeline: Apollo provides a pipeline starting from the lidar drivers for the identification of objects in the point-cloud [36, 37]. The vehicle's motion during the point cloud scan is compensated. Distinct objects segmented are from the point cloud data by projecting all scan points into the two-dimensional Cartesian plane, quantizing them into grid cells, and applying a fully convolutional neural network to predict geometry and class per cell. From these cells, polygonal objects that have a valid orientation are created. Whereas the networks' implementation is provided, the training data for the neural nets are not available open-source. Our setup relied on the trained network for a 64-layer Velodyne lidar to process the data of the 32-layer Velodyne lidar, which implies an adaption in the segmentation process that is hard to verify. A recognition component tracks the segmented obstacles and associates each obstacle with a track in a subsequent step. The result is a list of objects with kinematic states, shapes, and types per lidar sensor. We only introduced parameter changes.

3) Prediction: We also did not introduce implementation changes to the prediction component [38]. It takes the objects from the sensor data fusion and generates an estimated future motion based on the object state, type, and the ego-motion. Different machine learning-based algorithms are executed according to the type of object and its proximity/probability to interact with the ego vehicle. An evaluator predicts path and velocity individually for each object, followed by a predictor that generates trajectories.

C. Map Processing

Apollo heavily relies on an HD-Map, which provides geometric information and a topology graph of how the lanes are interconnected. The routing component selects the shortest path on the network using the A* algorithm [39]. This map is stored in a customized format extending the OpenDrive format [40]. While standard OpenDrive maps describe the geometric shape of a road and lane using mathematical functions, Apollo-style maps use linestrings instead, a discretized variant of the functions. We did not modify Apollo's map format but had to provide maps from our test area in Munich. We thus developed a converter that translates standard-conform OpenDrive maps into custom-Apollo OpenDrive maps.

Gran [41] provides a thorough analysis of the differences between standard OpenDrive and Apollo's custom format. Standard OpenDrive maps define the centerline of each lane as a sequence of functions along the longitudinal road coordinate. On top, lanes are then defined as functions defining the lateral offset from the centerline or the previous lane in a street-local coordinate system. Each road has a unique ID, and within each road, each lane has a unique ID inside this road. The linkage of roads is achieved by specifying successor and predecessor lanes on other roads and junctions.

In contrast, Apollo's adaption of OpenDrive requires each geometric function to be provided as a set of discretized points in an appropriate geo-coordinate system. Our converter performs this discretization into WGS84 points and the respective associations. It also resolves discretization errors at lane boundaries. Also, lane linkages are implemented slightly differently, which the converter resolves. Furthermore, Apollo-specific map content such as the area of an intersection is generated. Having this custom OpenDrive map in XML format, we use Apollo's toolchain to translate the map to a Protocol Buffers-based internal map format. In this format, each lane is specified by its centerline and boundaries and a unique ID. A road only contains topology information, i.e., how lanes are connected. Alongside, a downsampled version of the map is generated for visualization purposes and a graph-based representation of the map for routing purposes.

D. Planning

We integrated our planner as a native component inside Apollo's Docker container. We aimed to reuse as many of the service functionalities already implemented. Therefore, we decided to derive from Apollo's LatticePlanner class and integrate into the existing planner.

As the implementation of our planner relies on C++ language and Standard Template Library features that are not available with GCC 4.8.5, which is the compiler used in Apollo's Docker container, we did not compile our planner alongside Apollo. Instead, we compile it into a library that is then linked to the Apollo planning module. This procedure also enables us to use third-party dependencies incompatible with Apollo's third-party dependencies, such as more recent versions of the Boost and Eigen libraries. With Apollo, we only compile a wrapper class that calls this library. Furthermore, this wrapper class gathers data, helps to assemble the optimization problem, and post-processes the output. Sec. V will elaborate the interfaces and software structure in detail.

E. Control

The trajectory tracking controller stabilizes a given trajectory and generates acceleration and steering commands passed to the vehicle interfaces. Although Apollo would serve this need, we chose to replace the controller used by Apollo and run it on a separated real-time platform instead. This separation facilitates higher safety and reliability in our prototype setup by separating control tasks in real-time execution for other software applications on different hardware platforms. Therefore, we developed a bridge communicating between the Apollo stack and the trajectory tracking controller on the real-time system. Our bridge implementation is based on Apollo's bridge component but is rewritten in ANSI C to decode and encode Apollo's Protocol Buffers-based messages directly on the real-time device. The software architecture of the controller has been published in [3]. We observed that this separation yields more stable system performance and optimally uses the different benefits of the modular computing hardware setup.

V. Integration of the MINIVAN Planner in Apollo

This section describes how we integrated MINIVAN, described in Sec. III, into Apollo. The architectural overview of the software components of the proposed planner is shown in Fig. 3. Besides the planner's core, solving the MIQP problem, that is described in [4, 5], various other functionalities are involved, e.g., in processing the input data, generating the problem in the appropriate coordinate system, and post-processing the solution. In the following, we describe the main software components involved.

We do not use MINIVAN in driving situations at very low speeds. As elaborated in [4], the main reason for this is the singularity of the $\arctan$ function at very low speeds. The $\arctan$ function is needed to translate the absolute vehicle speed to the decomposed speed vectors in x and y direction. These low-speed tasks are performed by our nonlinear trajectory smoothing module, as described in Sec. V-C. We introduced a minimum speed threshold $v_{\mathrm{miqp,min}}$ below which we do not

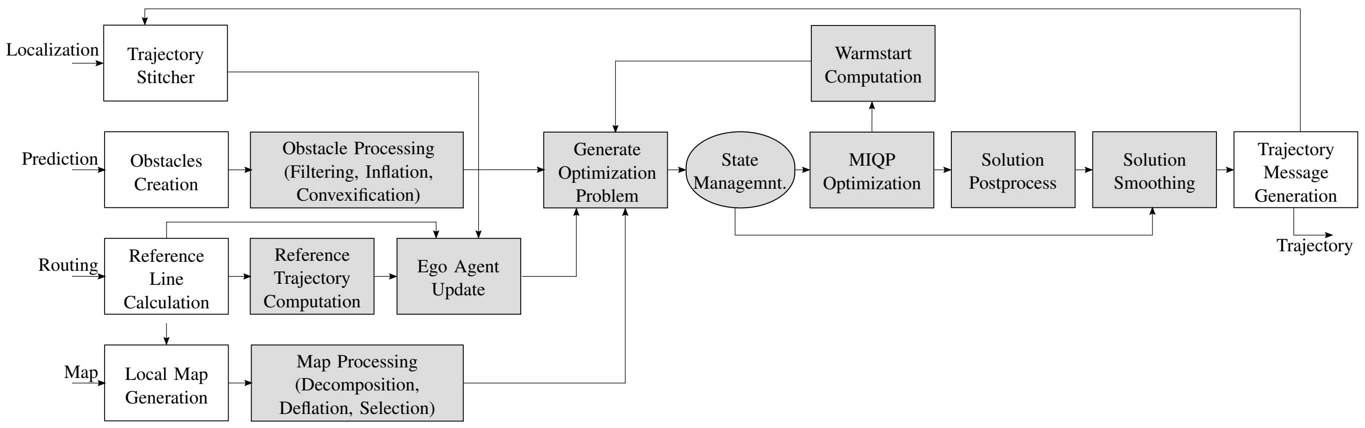

Localization
Trajectory Stitcher
Prediction
Obstacles Creation
Obstacle Processing (Filtering, Inflation, Convexification)
Routing
Reference Line Calculation
Reference Trajectory Computation
Ego Agent Update
Map
Local Map Generation
Map Processing (Decomposition, Deflation, Selection)
Generate Optimization Problem
State Managemnt.
Warmstart Computation
MIQP Optimization
Solution Postprocess
Solution Smoothing
Trajectory Message Generation
Trajectory

Fig. 3. Overview of the software components of our planner. Our contributed components are shown in gray. The other components are reused from Apollo. Functional building blocks are represented as boxes, the execution control is represented as an ellipse.

use MINIVAN. Trajectory points from MINIVAN below this speed and all subsequent ones are considered invalid. Through empirical experiments, we identified $v_{\mathrm{miqp,min}}$ as $0.7\,\mathrm{m/s}$ and used $1.0\,\mathrm{m/s}$ as a conservative upper bound.

A. Usage and Modification of Apollo Functionalities

From the routing request, Apollo creates a reference line for planning. This line generally consists of the stacked centerlines of the lanes on the routing corridor. Also, relevant obstacles are extracted from the environment model and a local except of the map. We have not changed these functionalities.

Apollo implements a trajectory stitching module to compute a reasonable initial point for the next planning cycle. Three cases are handled: static, moving without or moving along a valid trajectory. If the vehicle is static, the initial point is the current position. If the vehicle is moving, but no old trajectory is available (or the old trajectory is invalid), the trajectory stitcher extrapolates the current vehicle position into the future with the current speed vector for an estimated planning cycle duration. The default behavior of Apollo when moving along a trajectory is to select the next initial planning point as the point on the existing trajectory that the vehicle should reach in one planning cycle. While Apollo's planner has a rather deterministic low runtime, MINIVAN is terminated at an upper runtime bound. This bound is usually only reached in complicated planning situations, where mostly a feasible but sub-optimal solution is already available. In typical situations, this worst-case threshold is far from the mean runtime. We modified the trajectory stitcher to shift this extrapolation time to the worst-case computation time-bound. This way, we always have a valid trajectory as the stitching point can always be reached but have introduced additional inertia into the system. For the duration of the worst-case evaluation time, the planner cannot react to newly introduced obstacles, as the trajectory leading to the point at this time is fixed. This modification is, as such, no improvement of Apollo but a necessity to apply a planning concept with a non-deterministic runtime. Once a new valid trajectory is computed, it is appended to the old trajectory at the previously computed initial point. To interpolate the trajectory in the tracking controller, a sufficient backward horizon consisting of the old trajectories is kept.

We also changed the behavior if the planning module fails and cannot create a valid trajectory. In this case, Apollo would trigger an error state and compute an error trajectory, e.g., bring the vehicle to a stop. However, we often observed in experiments that if one planning cycle failed, e.g., due to an unfavorable prediction of one obstacle, the next planning cycle produces a valuable result, resulting in unnecessary error trajectories. Therefore, we decided not to trigger the error state with the first failing planning cycle but at the time instance when the vehicle has reached the end of the current planning horizon. This planner typically recovers to a normal state in a couple of planning cycles. We ensure the old trajectory is still safe by checking this old trajectory for collision with all obstacles. This modification yields more robust and stable overall system behavior without modifying components the stability issue comes from, which we wanted to avoid.

B. Pre- and Post-Processing the Optimization Problem

We first switch from the coordinate system in the driving direction of the vehicle to the global Cartesian coordinates

$$\begin{pmatrix} p_x & p_y & \theta & v & a & \kappa \end{pmatrix} \longrightarrow \begin{pmatrix} p_x & p_y & v_x & v_y & a_x & a_y \end{pmatrix} \quad (1)$$

with velocity v and acceleration a in the direction of the vehicle orientation θ and curvature κ. With velocities and acceleration in x and y direction v_x, a_x and v_y, a_y the transformation is then defined as:

$$v_x = v\cos(\theta), \quad (2a)$$
$$v_y = v\sin(\theta), \quad (2b)$$
$$a_x = a\cos(\theta) - v^2\kappa\sin(\theta), \quad (2c)$$
$$a_y = a\sin(\theta) + v^2\kappa\cos(\theta). \quad (2d)$$

The transformation is based on a kinematic single track model [42]. To map the accelerations (2c) and (2d), we derive the respective velocities and use the relation of yaw rate and curvature $\dot{\theta} = v\tan(\delta)/l = v\kappa$ (with steering angle δ and wheel base l).

Algorithm 1 Obstacle Consideration

```
Input: obstacles $\mathcal{O}_{\mathrm{Apollo}}$, region of interest $P_{\mathrm{roi}}$, convex deflated road cells $\mathcal{C}_{\mathrm{d}}$, collision radius $r_{\mathrm{c}}$, safety margin $r_{\mathrm{s}}$
Output: $\mathcal{O}_{\mathrm{MINIVAN}}$
1: for each $o \in \mathcal{O}_{\mathrm{Apollo}}$ do
2:     if type($o$) is static then
3:         if BBox$(o, 0) \cap \mathcal{C}_{\mathrm{d}} \cap P_{\mathrm{roi}} \neq \emptyset$ then
4:             $\mathcal{O}_{\mathrm{MINIVAN}} \leftarrow \mathrm{Inflate}(o, r_{\mathrm{c}} + r_{\mathrm{s}})$
5:     else                                        ▷ dynamic obstacle
6:         for each $t \in 0...T$ do
7:             if BBox$(o, t) \cap \mathcal{C}_{\mathrm{d}} \cap P_{\mathrm{roi}} \neq \emptyset$ then
8:                 $\mathcal{O}_{\mathrm{MINIVAN}} \leftarrow \mathrm{Inflate}(o, r_{\mathrm{c}} + r_{\mathrm{s}})$
```

1) Obstacle Processing: The processing of obstacles is shown in Algorithm 1. We first inflate the shape of each obstacle o by the collision circle radius r_{c} plus an appropriate safety margin r_{s}. This margin can be selected differently for different object types.

We then compute the minimum bounding rectangle $\mathrm{BBox}(o, t)$ for each object o at time t as the shape to check against collision in the MINIVAN optimization. Theoretically, every convex shape would be possible, but rectangles have proven to be a reasonable compromise between accuracy, flexibility, and runtime. We compute a translated and rotated shape for each state on the predicted trajectory for dynamic obstacles.

Obstacles are only considered if they intersect with the convex deflated road cells $\mathcal{C}_{\mathrm{d}}$ and the Region of Interest (ROI) P_{roi}. For dynamic obstacles, the shape of at least one prediction step has to intersect with the environment. The ROI P_{roi} is computed as a very simplistic reachable set of the vehicle forming a square. It filters out obstacles that do not interfere with the vehicle, such as objects behind the vehicle.

2) Reference Computation and Ego Agent Update: We get the reference line for planning alongside the destination point on this line. We use this line to create a linear reference speed profile with a constant acceleration phase up to the target speed add a constant velocity phase at reference speed. The reference decelerates with a linear speed profile to zero velocity when approaching the goal point. We scale the target speed with the curvature and a maximum accepted lateral acceleration threshold in the curves.

As a start point for the reference, we select the initial point for planning. The second point of the reference is selected as the point on the reference line with minimal Euclidean distance in front of the vehicle. The initial velocity is chosen as the velocity of the initial point for planning. Note that this straightforward reference generation strategy can yield suboptimal results for starting positions far off the reference line, as possibly the optimization tracks an unreachable reference. We do not adapt the reference line when intersecting with obstacles or the environment but expect MINIVAN to resolve these conflicts. From this reference line and speed profile, we sample an interpolated reference trajectory as an input to the planning step.

3) Map Processing: MINIVAN relies on a convex approximation of the available free-space for navigation. Nearly all

Algorithm 2 Map Processing

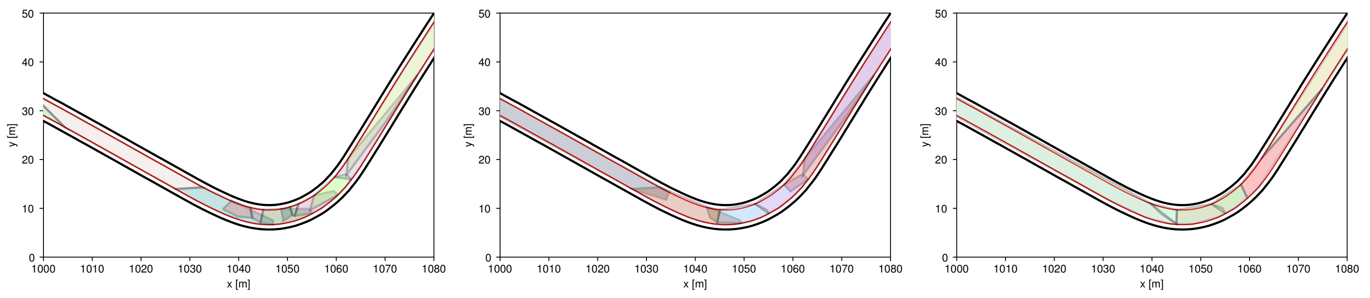
```
Input: road polygon $P_{\mathrm{road}}$, simplification distance $\delta_{\mathrm{s}}$, circle radius $r$, number of merging iterations $n$
Output: convex deflated road cells $\mathcal{C}_{\mathrm{d}}$
1: $P_{\mathrm{s}}$ = Simplify$(P_{\mathrm{road}}, \delta_{\mathrm{s}})$
2: $P_{\mathrm{s,d}}$ = Deflate$(P_{\mathrm{s}}, r)$
3: $\mathcal{C}_{\mathrm{voronoi}}$ = VoronoiDiagram$(P_{\mathrm{s,d}})$
4: $\mathcal{C}_{\mathrm{voronoi} \cap \mathrm{road}} = \{\}$
5: for each $C \in \mathcal{C}_{\mathrm{voronoi}}$ do
6:     if $C \cap P_{\mathrm{s,d}} \neq \emptyset$ then
7:         $\mathcal{C}_{\mathrm{voronoi} \cap \mathrm{road}} \leftarrow C \cap P_{\mathrm{s,d}}$
8: $\mathcal{T}$ = Triangulate$(\mathcal{C}_{\mathrm{voronoi} \cap \mathrm{road}})$
9: $P_{\mathrm{d}}$ = Deflate$(P_{\mathrm{road}}, r)$
10: $\mathcal{C}_{\mathrm{d}} = \mathcal{T}$
11: for each $i \in 1...n$ do
12:     $\mathcal{C}_{\mathrm{d}}$ = MergePolygons$(\mathcal{C}_{\mathrm{d}}, \tau_{\mathrm{m}}, P_{\mathrm{d}})$
```

road shapes are non-convex and therefore have to be decomposed into several convex sub-polygons. A naive triangulation is possible but undesirable, as the number of constraints scales linearly with the number of environment polygons. Therefore, we aim for an efficient convexification algorithm that outputs as few polygons as possible while still covering the whole area. Algorithm 2 shows the pseudocode of this convexification, where P denotes an arbitrary-shaped polygon, C a convex polygon, T a triangle, and $\mathcal{P}, \mathcal{C}, \mathcal{T}$ the respective sets.

We construct the (non-convex) road polygon P_{road} from the left and right road boundaries and simplify it using the Ramer-Douglas-Peucker algorithm. We deflate the environment polygon with the radius of the circles that approximate the vehicle and construct the Voronoi diagram on this polygon using a sweep line algorithm [43]. The Voronoi cells $\mathcal{C}_{\mathrm{voronoi}}$ represent a segmentation of the available space into polygons. In our case, all cells are polygons with straight edges. We intersect the deflated original polygon with the set of Voronoi cells and drop all cells outside the original polygon. We decompose non-convex Voronoi cells into triangles.

Finally, we iteratively merge the resulting set of polygons into bigger convex polygons in a greedy manner. We check if the convex hull of this union lies within the original (non-convex) road polygon or the union is itself convex and merge further triangles if yes, and start a new sub-polygon if not. This procedure does not guarantee an optimal result. To cope with this, it has proven to be beneficial for the road shapes used in this work to iterate twice over the resulting list of convex sub-polygons to reduce the number of polygons further and find larger convex shapes.

The convexity checks in the merge step are performed with a particular error margin that two shapes are merged, even if the union is marginally non-convex. Then the convex hull is used. This robust convexity check avoids numerical errors and helps to create significantly fewer sub-polygons. However, it introduces errors in the road approximation, which we add as a safety margin to the collision shape of the agents. As the last merging step, we simplify each polygon in the set of convex-sub polygons to avoid very small edges. We inflate



(a) $\delta_\mathrm{s} = 0.1\,\mathrm{m}$, $\tau_\mathrm{m} = 0.1\,\mathrm{m}$ (yields 12 polygons) (b) $\delta_\mathrm{s} = 0.1\,\mathrm{m}$, $\tau_\mathrm{m} = 0.5\,\mathrm{m}$ (yields 7 polygons) (c) $\delta_\mathrm{s} = 0.5\,\mathrm{m}$, $\tau_\mathrm{m} = 0.5\,\mathrm{m}$ (yields 6 polygons)

Fig. 4. Results of the convexification of a lane corridor forming a curve as an example environment. We vary the simplification distance δ_s and the convex merging tolerance τ_m. The original lane boundary is depicted in black, and the deflated lane corridor, which shall be decomposed, is depicted in red. The resulting convex polygons are shown in various colors.

each polygon by the maximum simplification error again to avoid holes between polygons.

The number and size of the sub-polygons are highly dependent on the two introduced parameters: the simplification distance δ_s and the convexity threshold τ_m for merging two polygons. We can find a balanced approximation accuracy and sub-polygons size with an appropriate setting. In Fig. 4, we compare three different settings and show the effect of the parameters. We observe that the number of polygons is reduced with increased merging tolerance. With low values, the approximation of the (red) deflated environment shape is accurate, whereas a notable approximation error at the inner part of the curve occurs with higher values. The strategy described here has proven to generate a sufficiently accurate environment decomposition with a fast runtime. Still, we rely on a greedy heuristic, and theoretically, shapes can be constructed where the algorithm yields poor results. We did not observe such problems in the tested road geometries.

4) Optimization Problem Generation: This stage collects all previously generated data and assembles the optimization problem. We compute values specific to the MINIVAN optimization, followed by various consistency checks to avoid infeasible optimization problems, e.g., ensuring to start inside the lane polygon. Here we also load the warmstart solution from the previous planning cycle.

5) Warmstart Computation: An existing solution can be provided to aid the MIP solver in finding a solution faster. Warmstarting works best if the solution is already primal feasible (fulfilling the non-integer constraints), but non-feasible or incomplete solutions can also serve as a warmstart. With a solution available at the start of the branch-and-cut process, non-optimal parts of the branch-and-cut tree can be eliminated, and a lower solution bound exists. However, the effectiveness of the warmstart is influenced mainly by the solution quality of the warmstart solution, which is unknown.

Therefore, we reuse all integer-feasible solutions from the last receding horizon instance as warmstarts. We input these solutions without modifications, such as shifting from the last to the current timestep. Using this strategy, we have the previous optimal solution available for warmstarting and non-optimal but integer-feasible solutions that lead to the current optimal solution faster as the integer decision variables are

$$\begin{tikzcd} & \text{Starting} \arrow[dr, "v > v_\mathrm{miqp,min}"] \arrow[dd, "s_\mathrm{rem} < s_\mathrm{stop}"] & \\ \text{start} \arrow[r] & \text{Standstill} \arrow[u, "(v > v_\mathrm{standstill}) \wedge (s_\mathrm{rem} > 0)"] & \text{Driving} \arrow[dl, "(v < v_\mathrm{miqp,min}) \wedge (s_\mathrm{rem} < s_\mathrm{stop})"] \\ & \text{Stopping} \arrow[ul, "v < v_\mathrm{standstill}"] & \end{tikzcd}$$

Fig. 5. Finite state automaton controlling the planning sequence.

initialized better. This approach, in practice, has a significant effect on the runtime.

The less the topology of the problem has changed, the more significant is the effect of the warmstart. Suppose environment sub-polygons, agents, or obstacles are added to the problem. In that case, the decision variables for those cannot be taken from the previous solution, which lowers the efficiency of the warmstart. Also, if the problem drastically changes, an old solution can guide the solver in the wrong direction. In Sec. VI and Sec. VII-D, we show the speedup from warmstarting.

6) Planner State Management: Our planner implementation is controlled by a finite state automaton with four states representing different driving situations:

- *Standstill*: We do not compute a driving trajectory but maintain the current position.
- *Starting*: We omit the MINIVAN planning step but compute a trajectory along the reference path and speed profile using the trajectory smoothing module.
- *Driving*: We set up and solve the MINIVAN planning problem and smooth the resulting trajectory.
- *Stopping*: We omit the MINIVAN planning step but compute a trajectory along the reference path and a speed profile ramped down to zero using the trajectory smoothing module.

Fig. 5 shows the resulting automaton and its respective transitions, where $s_\mathrm{stop} := \min(s_\mathrm{goal}, s_\mathrm{blocking})$. The automaton transitions from Standstill to Starting if the velocity is greater than a standstill threshold and the distance to the routing destination s_{goal} is greater than zero, and the path is not blocked by an obstacle right in front of the vehicle (s_{blocking}). The automaton transitions from *Starting* to *Driving* if the current velocity is greater than $v_{\text{miqp,min}}$, and from *Starting* to *Stopping* if the vehicle approaches the destination point. A transitions is triggered from *Driving* to *Stopping* if the vehicle approaches the destination point and the current speed falls below $v_{\text{miqp,min}}$. Once the velocity drops below $v_{\text{standstill}}$, the automaton transitions from *Stopping* to *Standstill*. In the *Starting* and *Stopping* state, we only consider obstacles that are immanently in front of the vehicle.

7) Post-Processing of the MINIVAN Solution: After a successful MINIVAN optimization, we post-process the result. First, we compute and store the warmstart for the next planning cycle according to the warmstart strategy described in Sec. V-B5. We then transform the solution trajectory back from global Cartesian coordinates into the vehicle-relative frame having absolute values in vehicle orientation

$$\begin{pmatrix} p_x & p_y & v_x & v_y & a_x & a_y \end{pmatrix} \longrightarrow \begin{pmatrix} p_x & p_y & \theta & v & a & \kappa \end{pmatrix} \tag{3}$$

using trigonometric formulas and appropriate derivations

$$\theta = \arctan(\frac{v_y}{v_x}), \tag{4a}$$
$$v = \frac{v_x}{\cos(\theta)}, \tag{4b}$$
$$a = \cos(\theta)a_x + \sin(\theta)a_y, \tag{4c}$$
$$\kappa = \frac{v_x a_y - a_x v_y}{(v_x^2 + v_y^2)^{3/2}}. \tag{4d}$$

To ensure that no approximation errors have been introduced, we check the resulting trajectory for collisions with the unmodified obstacles and environment. In the final step, we perform a trajectory smoothing as described in Sec. V-C.

C. Trajectory Smoother

MINIVAN does not allow putting direct costs on absolute acceleration change and curvature change. However, both are desirable to generate a comfortable driving experience. Also, due to the approximation of the orientation into a set of discrete regions, the vehicle orientation at the boundaries of regions can be inaccurate, and the orientation change can be non-smooth. Furthermore, in MINIVAN, we want to use a more extensive time discretization to achieve a longer planning horizon that we subsample in the smoother to hand over a smooth and more fine-grained trajectory representation to the controller. Note that due to the overapproximation of the vehicle shape, this is not a safety issue when avoiding obstacles. Nevertheless, it can lead to uncomfortable driving behavior.

Therefore, we introduce a trajectory smoothing step to find $\boldsymbol{T}_{\text{out}}$ as a nonlinear Model-Predictive Control (MPC) optimization in the vehicle frame to stay as close as possible to the original trajectory $\boldsymbol{T}_{\text{in}}$. Obstacles are not directly used in the smoothing optimization but only indirectly through the input trajectory. We implicitly perform interpolation by increasing the number of support points from $\boldsymbol{T}_{\text{in}}$ to $\boldsymbol{T}_{\text{out}}$.

We define that state space as

$$\boldsymbol{x}^k := (p_x^k, p_y^k, \theta^k, v^k, a^k, \kappa^k) \tag{5}$$

and the input as

$$\boldsymbol{u}^k := (j^k, \dot{\kappa}^k), \tag{6}$$

where superscript k denotes the discrete timestep. We use Heun's method to integrate the single-track model equations. The discretized system dynamics can then be formulated as

$$\begin{aligned}
p_x^{k+1} &= p_x^k + \frac{1}{2}\Delta t v^k \cos(\theta^k) & \text{(7a)}\\
&\quad + \frac{1}{2}\Delta t(v^k + \Delta t a^k)\cos(\theta^k + \Delta t v^k \kappa^k), \\
p_y^{k+1} &= p_y^k + \frac{1}{2}\Delta t v^k \sin(\theta^k) & \text{(7b)}\\
&\quad + \frac{1}{2}\Delta t(v^k + \Delta t a^k)\sin(\theta^k + \Delta t v^k \kappa^k), \\
\theta^{k+1} &= \theta^k + \frac{1}{2}\Delta t v^k \kappa^k + \frac{1}{2}\Delta t(v^k + \Delta t a^k)(\kappa^k + \Delta t \dot{\kappa}^k), & \text{(7c)}\\
v^{k+1} &= v^k + \frac{1}{2}\Delta t a^k + \frac{1}{2}\Delta t(a^k + \Delta t j^k), & \text{(7d)}\\
a^{k+1} &= a^k + \Delta t j^k, & \text{(7e)}\\
\kappa^{k+1} &= \kappa^k + \Delta t \dot{\kappa}^k. & \text{(7f)}
\end{aligned}$$

We place bounds on the states

$$v_{\max} < v(k) < v_{\max}, \tag{8a}$$
$$\kappa_{\max} < \kappa(k) < \kappa_{\max} \tag{8b}$$

and inputs

$$j_{\max} < j(k) < j_{\max}, \tag{9a}$$
$$\dot{\kappa}_{\max} < \dot{\kappa}(k) < \dot{\kappa}_{\max}. \tag{9b}$$

We compute a quadratic cost function

$$J(\boldsymbol{u}_1, ..., \boldsymbol{u}_N, \boldsymbol{x}_1, ..., \boldsymbol{x}_N) = \sum_{k=1}^{N} \Big((\boldsymbol{x}^k - \boldsymbol{x}_{\text{in}}^k)^\intercal \boldsymbol{Q}_1^k (\boldsymbol{x}^k - \boldsymbol{x}_{\text{in}}^k) + \boldsymbol{x}^{k\intercal} \boldsymbol{Q}_2^k \boldsymbol{x}^k + \boldsymbol{u}^{k\intercal} \boldsymbol{Q}_3^k \boldsymbol{u}^k \Big),$$

where $\boldsymbol{x}_{\text{in}}$ denotes the state of the input trajectory to the smoother module, which may not be available at some time instances due to the wider sampling of the input trajectory $\boldsymbol{T}_{\text{in}}$. Therefore, we select the weight matrices $\boldsymbol{Q}_1^k, \boldsymbol{Q}_2^k, \boldsymbol{Q}_3^k$ in a way to not penalize deviations of subsampled states. $\boldsymbol{Q}_1$ ensures minimal deviations from the input trajectory and $\boldsymbol{Q}_2$, $\boldsymbol{Q}_3$ ensure a smooth motion.

VI. Evaluating the Problem Complexity

This section analyzes the evaluation time with the growing complexity of the optimization problem, namely, more obstacles and agents. The scenarios were randomly generated. We simulated 100 variants of the same scenario and averaged the results to counteract randomness and runtime non-determinism. We set the maximum solution time to 30 s to find solutions even in complicated scenarios. We filter out all failed optimizations since these correspond to unsolvable scenarios, e.g., a randomly placed obstacle inside the vehicle. Fig. 6 shows the results of this analysis, which we now discuss in detail.

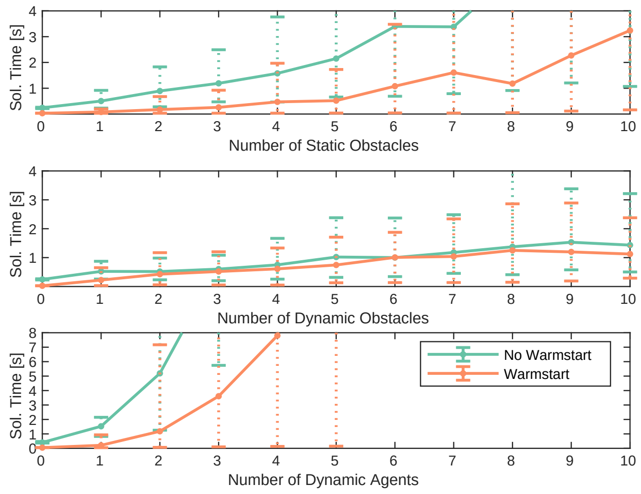


Fig. 6. Simulative complexity analysis for a varying number of static obstacles, moving obstacles, and moving agents around the ego vehicle. We also show the effect of an available warmstart solution and whether objects are modeled as agents or obstacles. The vertical dotted lines show the 5% and 95% quantile from 100 scenario variants. The warmstarted model on average and in nearly all scenarios has a lower, or at least equal runtime.

A. Static Obstacles Scaling

As a scenario, we randomly placed static obstacles close to the vehicle around a straight reference line the planner intended to track with a constant speed. With more obstacles placed on the reference, the trajectories became more evasive. The random placement of obstacles yielded scenarios with varying complexity that range from simple scenarios with obstacles only relatively far from the vehicle to complex ones with only obstacles close to the vehicle. The obstacle placement, however, led to more optimization problems becoming infeasible. The variance in the runtime reflects the scenario complexity.

Without a warmstart, we observe a linear scaling with the number of obstacles. The solution was found faster with an available warmstart computed from the last solution.

If all obstacles were placed very far from the reference, the planner's evaluation time did not scale with the number of obstacles.

B. Moving Obstacles and Agents Scaling

As a scenario, we randomly placed objects around the ego vehicle and let them drive in the same direction as the ego vehicle with a randomly assigned speed difference ($\pm 20\%$ of the ego vehicle speed). These objects can either be modeled as dynamic obstacles that need to be avoided or as agents for which the model plans an interactive trajectory.

If other objects were modeled as moving obstacles, we observed a good performance until around ten obstacles. The effect of warmstarting was not significant. Generally, warmstarting helped to lower the variance in the solution time. Again, the random placement of other objects yielded easy and hard-to-solve scenarios, which explains the high variance.

Our analysis shows that the problem becomes intractable in real-time for more than two other objects if modeled as interacting agents. This effect is also visible in the exponential growth in the problem size. Beginning with six other dynamic agents, the solver cannot find a solution in the given time in most scenarios. We show the evaluation until five other dynamic agents. With more agents, the runtime bound is reached in a majority of cases non-trivial cases, which distorts the evaluation.

If we placed all objects modeled as obstacles very far from the reference, we observed that the planner's evaluation time did not scale with the number of obstacles. Warmstarting and the number of regions barely affected the runtime. In contrast, we observed a comparable scaling effect in the multi-agent case regardless of the object distances.

C. Discussion and Findings

For a low runtime, a warmstarted solver is essential in complex situations. In some situations, warmstarting can also reduce the variance in the solution time. However, warmstarting can also negatively affect solution time in certain constellations, and the question remains if these scenarios can be distinguished in advance. Thus warmstarting does not fully mitigate the high solution time variance. This effect was expected in the setup of this analysis. The effect that the evaluation time is unstable remains a severe challenge in real-time application. All in all, the solver can handle a realistically high number of static or dynamic obstacles. However, more than two other interacting agents are not tractable in a real-time setting.

VII. Evaluation in Real-Road Scenarios

To evaluate the capabilities of the proposed planner, we performed several experiments on a real road; specifically,

- *Static Vehicle Avoidance*: avoiding a static vehicle on the right side.
- *Vehicle Following*: approaching a slower preceding vehicle, eventually slowing down.
- *Pedestrian Avoidance*: avoiding a static pedestrian blocking parts of the road.

We chose to analyze the planner's performance and overall system setup in these three benchmarks scenarios and not in arbitrary road situations to have reproducible results and examine specific strengths and weaknesses in detail.

Fig. 7 shows the evolution of the static vehicle avoidance scenario at three representative timesteps. In Fig. 8, we plot the evolution of the scenario over time, with the inflated obstacle shapes for each timestep and the environment representation deflated. The resulting trajectory is collision-free due to inaccurate obstacle measurements, which we will discuss in Sec. VII-A, it appears colliding.

A. Perception

To study Apollo's off-the-shelf object detection quality, we consider only the two scenarios with static obstacles. We analyze the deviation $(\cdot) - \overline{(\cdot)}$, where $\overline{(\cdot)}$ denotes the mean

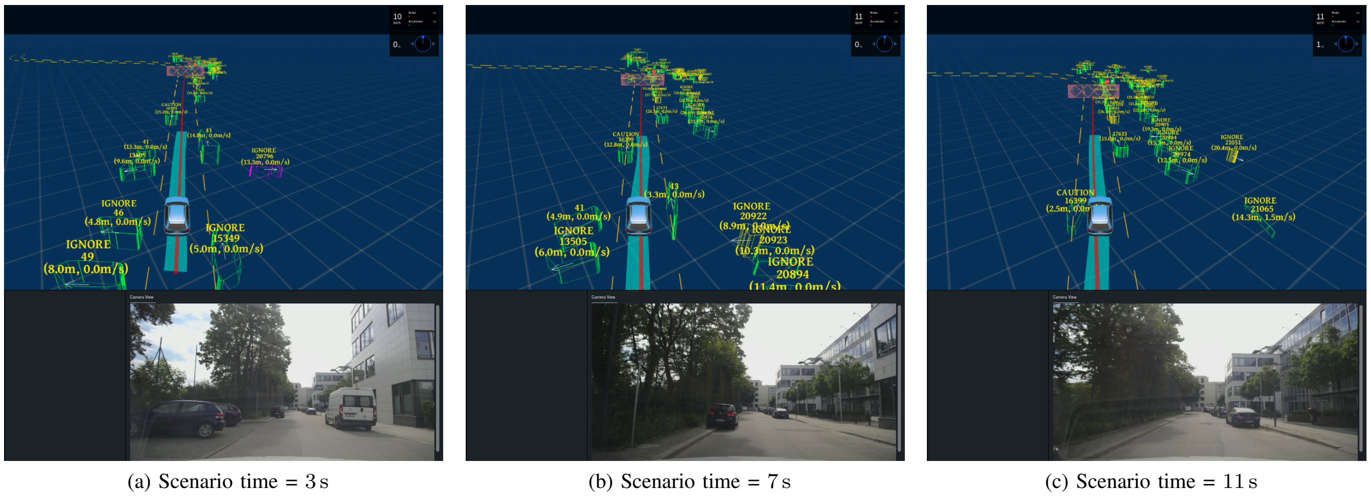

(a) Scenario time = 3 s (b) Scenario time = 7 s (c) Scenario time = 11 s

Fig. 7. Visualization of the static vehicle avoidance scenario.

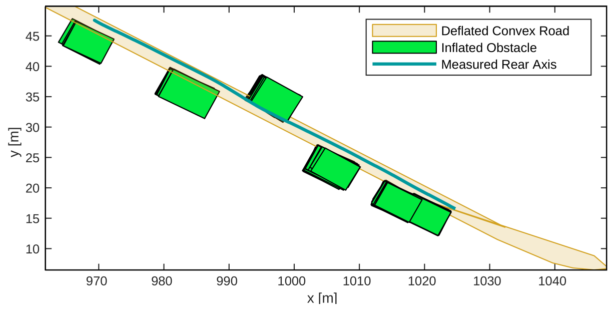


Fig. 8. Observed data from the static vehicle avoidance scenario accumulated for all timesteps with the deflated road polygon, the inflated obstacles, and the driven path. The inflated obstacles are shown throughout the scenario. The resulting trajectory is collision-free, but due to inaccurate obstacle measurements, it appears to be colliding.

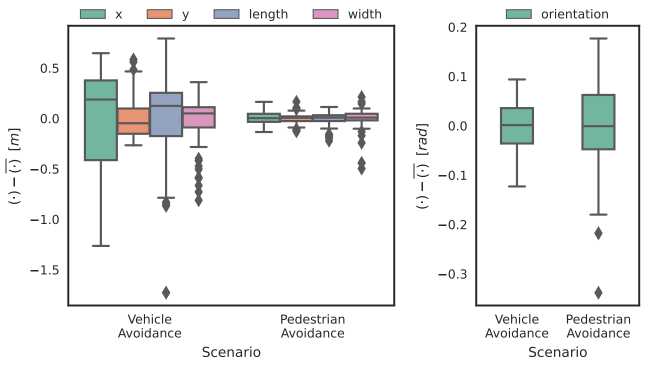


Fig. 9. Distribution of the deviation from the mean value of pose and bounding box dimension of the respective static obstacle.

value. We perform the analysis for the object's center position (x, y), orientation, and the bounding box width and length. Fig. 9 shows the resulting box plot. Extreme outliers (outside of the whiskers) are rare but may happen with deviations of more than $1.5\,\mathrm{m}$. Comparing the detection of vehicle and pedestrian, the errors regarding position and bounding box of the pedestrian are significantly smaller. On the other hand, the orientation of the pedestrian is much noisier than the orientation estimation of the vehicle. We expect that to improve when fusing with other sensors, e.g., cameras. However, poor orientation estimation of pedestrians may harm the prediction quality. As a takeaway, we can conclude that our planner needs to cope with noise within the standard outliers. We have thus added safety margins to the obstacles processed within the planning problem.

B. Planning

We first analyze the smoothness and consistency of the generated trajectories and the resulting tracking quality. Fig. 10 shows the trajectories generated from the planner for the vehicle avoidance scenario. Each line represents one trajectory with start and end times. In the ideal case without perception and tracking errors, these trajectories should form one continuous line. During the evasive maneuver from 8 s to 13 s, we observe the effect of the perception and perception error in the orientation plot. When further away from the obstacle, the planned trajectories start the evasive maneuver earlier than necessary as the obstacle's position is estimated closer to the vehicle than it actually is. When steering back to the reference line, the planner first generates trajectories taking a wider curve than necessary as the length estimation gets more accurate when driving parallel to the obstacle. These effects can also be observed in Fig. 9. When accelerating from standstill, the vehicle after activation of the low-level interfaces, accelerates slightly slower than the trajectory indicates. As we find the initial point for the next planning step on the old trajectory (if the tracking errors are small), the trajectories are not affected by localization or measurement errors. We observe that the velocity and orientation profiles are smooth and tracked smoothly. As the command to stop the vehicle is triggered based on the distance to the goal, one trajectory is visible that keeps the constant velocity longer. In the acceleration profile, we observe a typical noise along the reference. The curvature is computed from the steering

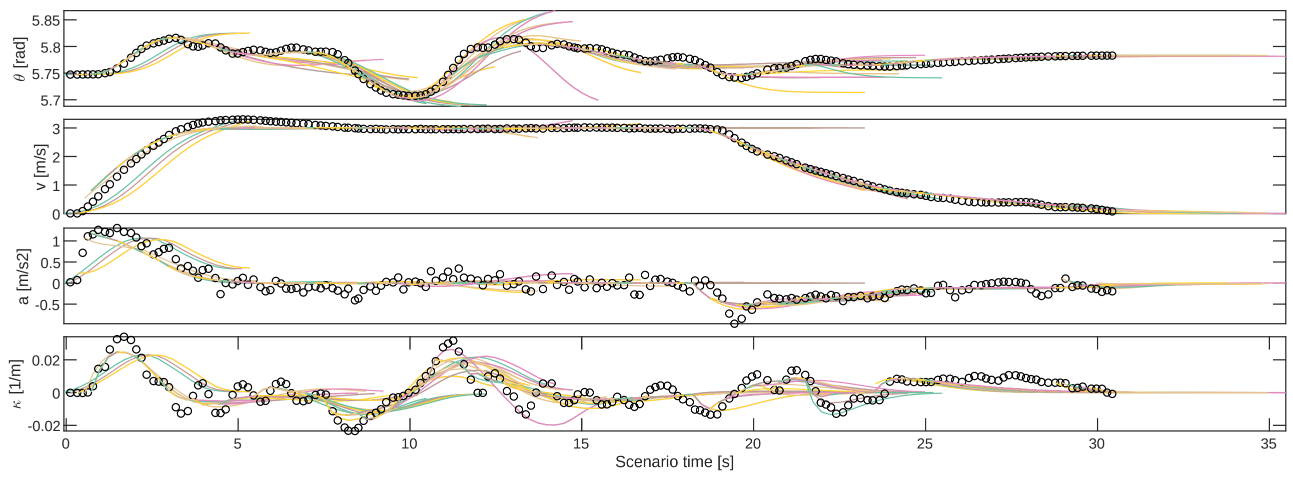


Fig. 10. Trajectories generated for the vehicle avoidance scenario. ○ denotes the measurement of the respective data. For clarity purposes, only every 5th trajectory is displayed.

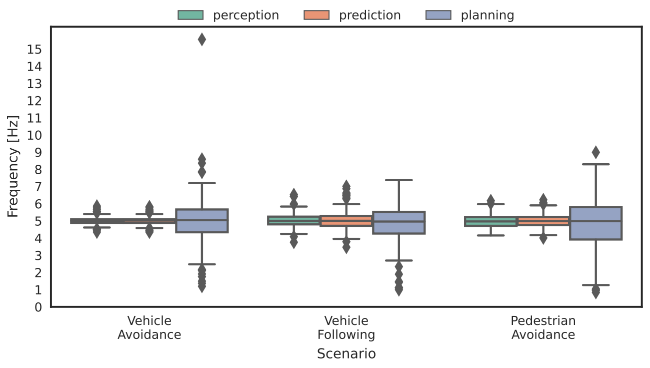


Fig. 11. Distribution of the frequency of the perception, prediction, and planning modules in the three studies scenarios.

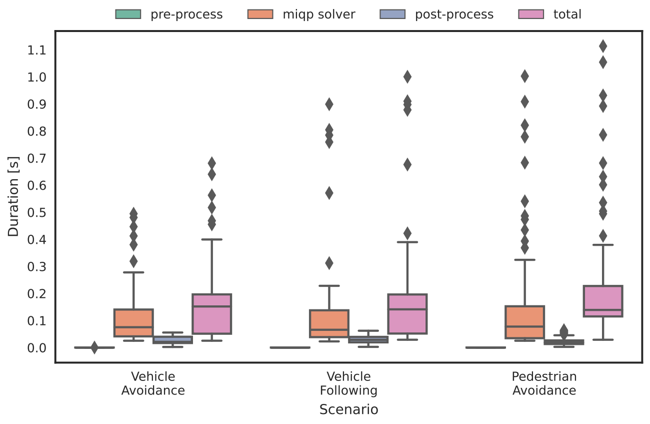


Fig. 12. Distribution of the planning times of the respective parts of the planning module in the three studied scenarios.

wheel angle measurement without smoothing, rate limitations, or stabilization terms. Therefore, minimal control interactions on the steering wheel result in visible oscillations here. Still, we clearly see the evasive maneuver and an initial steering command towards the reference line.

To answer the question of MINIVAN's real-time capability, we have analyzed the frequency of the perception, prediction, and planning modules. The goal for our planner is to come up with a new trajectory within at least $1\,\mathrm{s}$. Lidar sensor data triggers the perception, which triggers prediction, which triggers planning. Fig. 11 shows the results. We configured the lidar to publish sensor data at $5\,\mathrm{Hz}$. Both the perception and prediction components are capable of running at $5\,\mathrm{Hz}$ with some standard outliers down to $4\,\mathrm{Hz}$ and up to $6\,\mathrm{Hz}$ (to catch up). Increasing the lidar's publishing frequency would, at our planning frequency, only raise the system load. Our planner still runs at a mean frequency of $5\,\mathrm{Hz}$. However, the extreme outliers go down to $1\,\mathrm{Hz}$, which mostly happens when being close to an obstacle, where the solution space becomes narrow.

Second, we have analyzed the time consumption of the stages of the planning component, as sketched in Fig. 12: pre-processing, solving the MIQP problem, and post-processing. Pre-processing the input (e.g., decomposition of the map) is fast, as we are caching relevant data (e.g., decomposed parts of the map). Likewise, post-processing (including smoothing) is very efficient. The duration of pre- and post-processing is reliably below $0.06\,\mathrm{s}$ and does not have significant outliers. Most of the planning time is used by the MIQP solver. It has significant outliers of up to $1\,\mathrm{s}$ (when the solver is terminated), which consequently makes the total planning time range up to more than $1\,\mathrm{s}$.

The analysis shows that our intended general-purpose planning for arbitrary roads that avoids static and dynamic obstacles works smoothly at a replanning rate of $4\,\mathrm{Hz}$ to $6\,\mathrm{Hz}$ with outliers down to $1\,\mathrm{Hz}$.

C. Control

In this section, we describe the tracking performance of the controller in combination with trajectory planning. We show the results exemplary for the static vehicle avoidance scenario. In contrast to the other modules, the controller is executed on a dedicated hard real-time platform with a fixed cycle rate of

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIV.2022.3162671, IEEE Transactions on Intelligent Vehicles

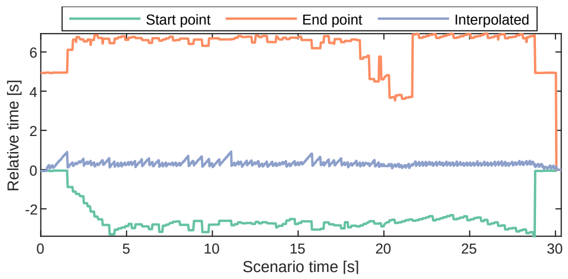


Fig. 13. Relative start time (zero or negative) and end times (positive) of the trajectory available to the controller measured on the real-time device and the controller's interpolated (matched) time for the vehicle avoidance scenario. We observe that at sufficient forward and backward horizon is available at every point in time, but the planning frequency varies.

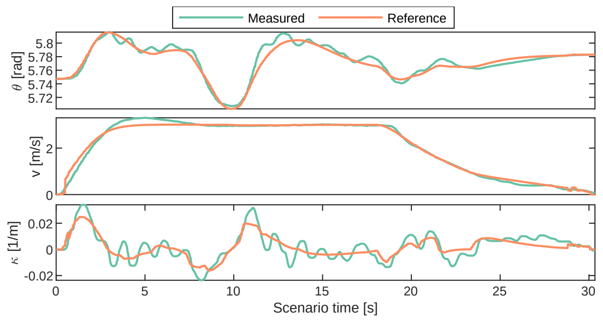


Fig. 14. Measured orientation, velocity, and curvature in comparison to reference signals for the vehicle avoidance scenario.

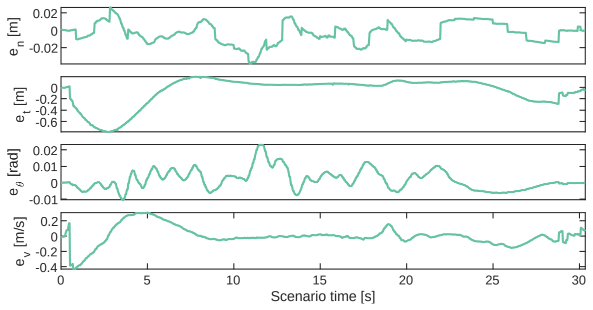


Fig. 15. Tracking errors of normal component e_n, tangential component e_t, orientation e_θ and velocity e_v for the vehicle avoidance scenario.

100 Hz. A trajectory with a sufficient time horizon must always be available to the controller. These trajectories are sent to the controller after each successful planning step, but our system has no guarantee that the controller receives a new trajectory in a given frequency. Therefore, we analyze in the following if a sufficient horizon is always available to the controller.

As the controller might need to interpolate between two trajectory points, a sufficient forward and backward horizon has to be available. Fig. 13 shows the relative start time and end time of the trajectory available to the controller and the interpolated time of the controller for the vehicle avoidance scenario. By *interpolated*, we denote the (relative) timestep that the controller currently tracks. We generally plan a trajectory for 6 s and the planner starts planning from a start point each time step, that is 1.25 s in the future (see Sec. V-A). With a maximum solution time of 1 s for the solver and some further delay in the system, a trajectory endpoint between 6 s and 7 s is therefore expected. At around 20 s, the trajectory is cut off at an intermediate point, as all subsequent points could therefore violate the non-holonomy assumption of the MINIVAN planner. Still, a sufficiently large horizon is available. Interpolation values greater than zero indicate that a point is interpolated in the future trajectory and not only the first point in time is tracked. We observe that the interpolated timestep is never greater than 1 s. Therefore the planning system operates without delays in real-time.

Fig. 14 shows the measured orientation, velocity, and curvature in comparison to reference signals for the vehicle avoidance scenario and Fig. 15 shows the corresponding tracking errors in the Frenet frame of normal component e_n, tangential component e_t, orientation e_θ and velocity e_v. We observe that the reference values from the trajectory planner are smooth but have to be stabilized by the controller in the lateral direction to achieve a only slight deviation from the trajectory in the normal (lateral) direction of at most $0.02\,\mathrm{m}$. Also, the orientation of the vehicle is stabilized up to an error of $0.02\,\mathrm{rad}$. This stabilization results in a solid tracking of the desired position, which we omit in the figures. In tangential (longitudinal) direction, we observe that the vehicle first accelerates too slowly, which results in velocity and tangential errors that are compensated in the first seconds. Due to the technical limitations of the prototype setup, tracking very low velocities are hardly possible. Therefore we initially accelerate from a standstill with a feedforward control law without tracking the reference. The effect can be seen in the very first second of the figures. Also, the evasive maneuver at around 10 s is observed.

From this analysis, we conclude that the whole chain of modules to compute the motion, starting from the generation of the reference line, the MIQP-based trajectory planning, the MPC trajectory smoothing, and real-time trajectory tracking yields a good and stable driving performance, not only in simulation but also on the road. The trajectories generated by MINIVAN can be executed by a standard control approach regarding timing and actual values. The parameterization was chosen for tracking performance rather than smoothness.

D. Parameterization Effects

The performance of MINIVAN can be improved by choosing an optimized set of parameters. We compare a general-purpose parameter set yielding good results in a variety of scenarios (denoted by *In car*), and tuned parameter sets created a-posteriori from a recorded test drive fitted to the specific needs (denoted by *Tuned*). Table I summarizes the mean runtime percentage improvements over all instances of one scenario compared to the default parameter setting of the solver without modification. Without a warmstart, the runtime increases, and therefore the number is negative. It can be observed that the parameters used for the test drives can be further improved when fitting the parameters for one specific scenario.

We also observe a significant effect for real-time execution in all scenarios when providing a warmstart solution. We

TABLE I
RUNTIME IMPROVEMENT IN PERCENT COMPARED TO THE DEFAULT PARAMETER SETTING OF THE SOLVER.

Scenario	Tuned	Without warmstart	In car
Vehicle avoidance	0.17098	-0.38412	0.17567
Vehicle following	0.29339	-0.33518	0.21956
Pedestrian avoidance	0.25172	-0.44545	0.12529

scarcely see disadvantages of the warmstart, and warmstarting is often essential to keep the runtime below 1 s. Still, few instances exist in the vehicle avoidance and following scenarios where warmstarting harms the runtime.

As expected, the default solver settings yield a higher runtime than a tuned parameter set. No clear conclusion can be drawn which parameter setting produces the fewest peaks in the runtime. These peaks are problematic for real-time evaluation and have to be avoided. While the tuned parameter set impedes such peaks in simulation studies, this effect cannot be clearly shown in on-road driving scenarios. Generally, with a sound warmstart solution, the evaluation time for the default and the tuned parameter set are often close. This effect can easily be seen in the fully static vehicle avoidance scenario. In a situation where more adaptions to the initial solution are necessary, such as in the second half of the vehicle-following scenario, the tuned parameter set is significantly faster.

VIII. Lessons Learned

After having described the Apollo-based software stack, the integration of MINIVAN and results with the system, we will summarize our experiences with this setup.

A. The Apollo Platform

Apollo's setup and build environment are easy to use for experienced developers after an initial hurdle, and the usage of one pre-configured Docker container eases this. Still, the pure size and number of dependencies, combined with the non-comprehensive documentation and hidden inter-dependencies, increases the risk of breaking an existing component when introducing changes in the build environment. Also, the usage of various outdated versions of packages and software tools (such as the Bazel build tool or the Boost C++ libraries) makes porting existing software into Apollo's Docker environment cumbersome.

The performance of the whole Apollo system heavily relies on the accuracy and level of detail of the HD-Map. For example, the planning component uses lane centerlines as a reference line the vehicle should drive on. The prediction component relies on the assumption that road users intend to stay close to the centerline of their respective lanes. Therefore, an inaccurate, unrealistic, or asymmetric true lane center yields unrealistic predictions or undesired ego planning maneuvers. Also, various ROI filters operate on lane geometries and benefit from maps with high accuracy. We decided not to activate the ROI filtering in the perception to not depend on the map. The benefit is that we do not lose any relevant obstacles, while the drawback is to have more objects in the system. The generation and maintenance of these maps in Apollo's customized format was an underestimated effort, as no standardized and open toolchain is available. Furthermore, the routing and maneuvering are purely map-based, which again poses high demands on the quality of the map topology. An inaccurate map is thus a very crucial issue.

The simulation engine shipped with Apollo is suitable for preliminary integration tests of a new planner but not comprehensive. For example, no obstacle simulation is available. Also, the vehicle motion is extracted from the planned trajectory position and not determined by a tracking controller component. Baidu offers a more comprehensive, cloud-based simulator that was not examined in this work, as the planner evaluations were executed in the BARK simulator [44]. BARK is designed as a benchmarking tool for planning algorithms, and it could be used for a simulative comparison of MINIVAN and Apollo's planner. We leave this to future work.

The Cyber RT middleware worked smoothly without introducing noticeable delay, jitter, latency, throughput bottlenecks, or hiccups. Distributing the components over multiple computers did not show any problem and only needed accurate time synchronization and appropriate configuration. Integrating additional components based on existing implementations is straightforward. The toolset provided with Cyber RT is sufficient but less comprehensive than the toolset of ROS. All basic functionalities for monitoring and debugging the system are available. For analysis, various scripting APIs exist.

Apollo's lidar pipeline proved to work well also with a 32-layer Velodyne lidar. Even though we do not have internal knowledge of the involved networks, i.e., how the networks were trained, and cannot judge the quality of the training data, the overall performance of the perception systems generalized to German roads. However, we only tested this for low-speed scenarios. Evaluating the accuracy of the estimated velocity in scenarios with higher speeds remains to be studied.

The data fusion component proved to be configurable for the usage in our setup with different sensor types and positions due to its configurable and modular code structure. Object recognition and segmentation provide state-of-the-art results that are sufficient for on-road driving experiments. The drawbacks are the high dependency of the relevant object-filtering algorithms to the HD-Map and the demands on the graphics card. In the setup with only lidar sensors, object type classification often showed unreliable results for non-vehicle objects.

Apollo's Car Area Network (CAN) bus component is extendable but based on low-level parsing of the CAN messages, which makes integration of custom CAN buses time consuming and requires testing effort. No parsing and automated analysis of CAN database files or similar are provided.

B. Our System Setup

To simplify the system setup, we chose to use the localization information and not the IMU information for control as our differential GNSS position is stabilized with three fiber-optical gyroscopes, an accelerometer, and a high-resolution wheel tick sensor. The assumption that tracking control is possible on localization data in such a setup turned out to be false. The data quality in terms of stability and absence of jumps is not high enough for robust trajectory tracking.

To maintain the experimental platform's stable setup, we reduced the system complexity at various points, such as only relying on lidar as sensor technology. These simplifications are not transferable to production-grade automated vehicles but proved to fit the needs of a research vehicle except for the high reliance on the GNSS.

The planning and prediction components in Apollo do not operate together interactively. Trajectories from prediction have to be post-processed in the planner. Otherwise, frequently errors occur, such as vehicles driving from behind into the ego vehicle or pedestrians walking on the side of the street being predicted to jump in front of the vehicle. This yields unnecessary planning errors. However, the prediction component yields a good baseline performance and reliable trajectories for vehicles moving on roads. Generally, the prediction system focuses on the lane centers and tends to project obstacles to follow these lane centerlines.

C. The MINIVAN Planning Approach

In theory and experiment, MINIVAN proved to operate reliably, reproducibly, and fast, even with a high number of obstacles present. This finding paves the path for its application in more complicated environments. However, more than two interacting agents are intractable with the current implementation. A reasonable trade-off must be found for objects to be treated as interacting agents or dynamic obstacles.

Fine-tuning the solver parameters and providing a warmstart solution, the runtime of the MIQP is sufficiently fast for real-time planning. Still, isolated peaks in the solution time can occur that are critical for the overall system performance but compensated by the postprocessing steps of MINIVAN. The mitigation of these is subject to future work.

MINIVAN is conceptionally unable to plan valid close-to-zero velocities (see Sec. V). The trajectory smoother handles situations where low velocities are necessary, such as starting and stopping. However, it does not account for interactivity. Therefore, the approach cannot plan a scenario where one agent has to come to a complete stop to solve the scenario.

IX. Conclusion and Future Work

We showed how we integrated Mixed Integer Interactive Planning (MINIVAN) [4, 5], a MIQP-based behavior and motion planning approach in the open-source fully autonomous driving stack Apollo. In three on-road benchmark scenarios, we demonstrated the real-time performance of the planner along with its limitations. Our in-depth analysis of relevant parts of Apollo, alongside the description of how we enhanced Apollo to integrate the MINIVAN planning approach, is concluded by a summary of the lessons learned on the way. We made MINIVAN, its integration into Apollo, and all additional modules available as open-source software [6, 7].

Apollo proved to be an excellent choice to base the experimental platform on due to the stable baseline performance and customizable setup. It comes at the price of a relatively closed environment where developers have to live with certain restrictions, such as a fixed development environment and the usage of a customized map format. Our modifications do not show drawbacks of the Apollo platform but motivate the applicability of a different planning concept and the flexibility of the stack. However, the development operations of working with an open-source stack should be improved by making deployment, testing, location adaption, and simulation open source. We demonstrated that MIP-based motion planning is real-time capable of considering a sufficiently high number of traffic participants around the ego vehicle. The employed MIQP-based planning approach with MPC post-smoothing step followed by a state feedback tracking control yielded stable and smooth performance on a distributed system.

Our extensions and experiences will allow research groups working on a similar setup not to have to adapt Apollo from scratch. While the employed minimal sensor set to reduce engineering maintenance served our needs, relying on the differential GNSS did not. We, therefore, encourage researchers to implement a sensor fusion for localization.

As a next step, we aim to accelerate MINIVAN further in terms of multi-agent cooperative planning to allow for real-time usage in complicated scenarios. We plan to implement an online verification safety layer to test scenarios with more complexity at higher speeds and conduct a study of the overall performance and reliability with a comprehensive sensor set.

Acknowledgment

This research was partly funded by the German Federal Ministry for Digital and Transport (BMDV) in the project *Cooperative Autonomous Driving with Safety Guarantees* (KoSi). The authors want to thank Patrick Hart for his support in conducting the driving experiments.

References

[1] Baidu, *Apollo: an open autonomous driving platform*, https://github.com/ApolloAuto/apollo, 2018 (accessed September 30, 2021).

[2] Autoware Foundation, *Autoware.AI*, https://www.autoware.org/autoware-ai, 2021 (accessed September 30, 2021).

[3] T. Kessler, J. Bernhard, M. Buechel, *et al.*, "Bridging the Gap between Open Source Software and Vehicle Hardware for Autonomous Driving," in *IEEE Intelligent Vehicles Symposium (IV)*, 2019.

[4] K. Esterle, T. Kessler, and A. Knoll, "Optimal Behavior Planning for Autonomous Driving: A Generic Mixed-Integer Formulation," in *IEEE Intelligent Vehicles Symposium (IV)*, 2020.

[5] T. Kessler, K. Esterle, and A. Knoll, "Linear Differential Games for Cooperative Behavior Planning of Autonomous Vehicles Using Mixed-Integer Programming," in *2020 59th IEEE Conference on Decision and Control (CDC)*, 2020.

[6] K. Esterle and T. Kessler, *Implementation of minivan (mixed integer interactive planning)*, https://github.com/bark-simulator/planner-miqp, 2021 (accessed September 30, 2021).

[7] ——, *Adaptions from fortiss to make apollo work on our prototype autonomous vehicle*, https://github.com/fortiss/apollo, 2021 (accessed September 30, 2021).

[8] S. Kato, S. Tokunaga, Y. Maruyama, *et al.*, "Autoware on Board: Enabling Autonomous Vehicles with Embedded Systems," *9th ACM/IEEE International Conference on Cyber-Physical Systems (ICCPS)*, 2018.

[9] Autoware Foundation, *Autoware.Auto*, 2021. [Online]. Available: https://www.autoware.org/autoware-auto.

[10] X. Wang, A.-K. Rettinger, and M. Althoff, "Coupling Apollo with the CommonRoad Motion Planning Framework," in *FISITA World Congress*, 2020.

[11] M. Althoff, M. Koschi, and S. Manzinger, "CommonRoad: Composable benchmarks for motion planning on roads," in *IEEE Intelligent Vehicles Symposium (IV)*, 2017.

[12] D. J. Fremont, E. Kim, Y. V. Pant, *et al.*, "Formal Scenario-Based Testing of Autonomous Vehicles: From Simulation to the Real World," 2020. [Online]. Available: http://arxiv.org/abs/2003.07739.

[13] Baidu, *Introduction into cyber rt*, https://github.com/fortiss/apollo/blob/r5.5.0/cyber/README.md, 2019 (accessed September 30, 2021).

[14] V. Raju, V. Gupta, and S. Lomate, "Performance of Open Autonomous Vehicle Platforms: Autoware and Apollo," *IEEE 5th International Conference for Convergence in Technology (I2CT)*, 2019.

[15] NVIDIA, *DriveWorks*, https://developer.nvidia.com/drive/driveworks, 2021, (accessed September 30, 2021).

[16] Baidu, *The apollo planning module*, https://github.com/fortiss/apollo/blob/r5.5.0/modules/planning/README.md, 2020 (accessed September 30, 2021).

[17] K. Esterle, P. Hart, J. Bernhard, and A. Knoll, "Spatiotemporal Motion Planning with Combinatorial Reasoning for Autonomous Driving," in *IEEE Conference on Intelligent Transportation Systems, Proceedings, ITSC*, 2018.

[18] Y. Zhang, H. Sun, J. Zhou, *et al.*, "Optimal Vehicle Path Planning Using Quadratic Optimization for Baidu Apollo Open Platform," in *IEEE Intelligent Vehicles Symposium, Proceedings*, 2020.

[19] J. Zhou, R. He, Y. Wang, *et al.*, "DL-IAPS and PJSO: A Path/Speed Decoupled Trajectory Optimization and its Application in Autonomous Driving," 2020. [Online]. Available: http://arxiv.org/abs/2009.11135.

[20] R. He, J. Zhou, S. Jiang, *et al.*, "TDR-OBCA: A Reliable Planner for Autonomous Driving in Free-Space Environment," 2020. [Online]. Available: https://arxiv.org/abs/2009.11345.

[21] M. Bahram, A. Lawitzky, J. Friedrichs, M. Aeberhard, and D. Wollherr, "A Game-Theoretic Approach to Replanning-Aware Interactive Scene Prediction and Planning," *IEEE Transactions on Vehicular Technology*, vol. 65, no. 6, pp. 3981–3992, 2016.

[22] D. Lenz, T. Kessler, and A. Knoll, "Tactical cooperative planning for autonomous highway driving using Monte-Carlo Tree Search," in *IEEE Intelligent Vehicles Symposium (IV)*, 2016.

[23] T. Kessler and A. Knoll, "Cooperative Multi-Vehicle Behavior Coordination for Autonomous Driving," in *IEEE Intelligent Vehicles Symposium (IV)*, 2019.

[24] K. Esterle, L. Gressenbuch, and A. Knoll, "Modeling and Testing Multi-Agent Traffic Rules within Interactive Behavior Planning," in *IROS Workshop Perception, Learning, and Control for Autonomous Agile Vehicles*, 2020.

[25] M. Bouton, A. Nakhaei, K. Fujimura, and M. J. Kochenderfer, "Safe Reinforcement Learning with Scene Decomposition for Navigating Complex Urban Environments," in *IEEE Intelligent Vehicles Symposium (IV)*, 2019.

[26] D. Isele, A. Nakhaei, and K. Fujimura, "Safe Reinforcement Learning on Autonomous Vehicles," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018.

[27] H. Krasowski, X. Wang, and M. Althoff, "Safe Reinforcement Learning for Autonomous Lane Changing Using Set-Based Prediction," in *23rd International Conference on Intelligent Transportation Systems (ITSC)*, 2020.

[28] P. Hart and A. Knoll, "Counterfactual Policy Evaluation for Decision-Making in Autonomous Driving," in *IROS Workshop Perception, Learning, and Control for Autonomous Agile Vehicles*, 2020.

[29] ——, "Graph Neural Networks and Reinforcement Learning for Behavior Generation in Semantic Environments," in *IEEE Intelligent Vehicles Symposium (IV)*, 2020.

[30] C. Paxton, V. Raman, G. D. Hager, and M. Kobilarov, "Combining neural networks and tree search for task and motion planning in challenging environments," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017.

[31] J. Bernhard, R. Gieselmann, K. Esterle, and A. Knoll, "Experience-Based Heuristic Search: Robust Motion Planning with Deep Q-Learning," in *21st International Conference on Intelligent Transportation Systems (ITSC)*, 2018.

[32] P. Weingertner, M. Ho, A. Timofeev, and G. Pita-gil, "Monte Carlo Tree Search With Reinforcement Learning for Motion Planning," in *23rd International Conference on Intelligent Transportation Systems (ITSC)*, 2020.

[33] C. Frese and J. Beyerer, "A comparison of motion planning algorithms for cooperative collision avoidance of multiple cognitive automobiles," in *IEEE Intelligent Vehicles Symposium (IV)*, 2011.

[34] C. Burger and M. Lauer, "Cooperative Multiple Vehicle Trajectory Planning using MIQP," in *IEEE Conference on Intelligent Transportation Systems, Proceedings, ITSC*, 2018.

[35] F. Fabiani and S. Grammatico, "Multi-vehicle automated driving as a generalized mixed-integer potential game," *IEEE Transactions on Intelligent Transportation Systems*, vol. 21, no. 3, 2020.

[36] Baidu, *Apollo 3d obstacle perception*, https://github.com/fortiss/apollo/blob/r5.5.0/docs/specs/3d_obstacle_perception.md, 2019 (accessed September 30, 2021).

[37] ——, *The apollo perception module*, https://github.com/fortiss/apollo/blob/r5.5.0/modules/perception/README.md, 2019 (accessed September 30, 2021).

[38] ——, *The apollo prediction module*, https://github.com/fortiss/apollo/blob/r5.5.0/modules/prediction/README.md, 2019 (accessed September 30, 2021).

[39] S. LaValle, *Planning Algorithms*. Cambridge University Press, 2006.

[40] ASAM, *OpenDrive*, https://www.asam.net/standards/detail/opendrive/, 2021 (accessed September 30, 2021).

[41] C. W. Gran, "HD-Maps in Autonomous Driving," M.S. thesis, Norwegian University of Science and Technoloy, 2019.

[42] R. Rajamani, *Vehicle Dynamics and Control*. Springer Science & Business Media, 2012.

[43] S. Fortune, "A Sweepline Algorithm for Voronoi Diagrams," in *Proceedings of the Second Annual Symposium on Computational Geometry*, 1986.

[44] J. Bernhard, K. Esterle, P. Hart, and T. Kessler, "BARK: Open Behavior Benchmarking in Multi-Agent Environments," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020.

Tobias Kessler received his diploma degree in mathematics from the Technical University of Munich, Germany in 2011. After a professional career as a real-time simulation specialist and consultant, he joined fortiss as a research scientist and is currently working towards his Ph.D. He is co-leading the group on Autonomous Systems. His research interests include cooperative and interactive behavior planning for autonomous vehicles and the application of this research to prototype vehicles.

Klemens Esterle received his diploma degree in mechatronics engineering from the Dresden University of Technology, Germany, in 2014 and his Ph.D. degree in computer science from the Technical University of Munich, Germany, in 2021. He was with fortiss from 2017 until 2021. His research interests include autonomous vehicles, motion and behavior planning.

Alois Knoll received his diploma degree in electrical/communications engineering from the University of Stuttgart, Germany, in 1985 and his Ph.D. degree in computer science from the Technical University of Berlin, Germany, in 1988. After his habilitation in 1993, he joined the University of Bielefeld, as a Full Professor and Director of the research group Technical Informatics. Since 2001, he has been a Professor of Computer Science at the Technical University of Munich. His research interests include cognitive, medical and sensor-based robotics, multi-agent systems, data fusion, adaptive systems, multimedia information retrieval, model-driven development of embedded systems with applications to automotive software and electric transportation, as well as simulation systems for robotics and traffic.