SN Computer Science (2024) 5:375
https://doi.org/10.1007/s42979-024-02732-5

ORIGINAL RESEARCH

MPC-Based Routing and Tracking Architecture for Safe Autonomous Driving in Urban Traffic

Mostafa Emam[1] · Matthias Gerdts[1]

Received: 27 September 2022 / Accepted: 18 February 2024

Abstract
This paper presents a configurable routing and tracking architecture that uses multi-objective Model Predictive Control (MPC) as its driving algorithm to guarantee safe autonomous driving of different vehicle types. The architecture consists of three main components and primarily relies on labeled map data to generate optimal path and velocity trajectories in accordance with the vehicle type and the desired control objectives. We begin with introducing the overall system architecture and its different inputs, outputs, and components. We also briefly explain the open-source services utilized in this work for trajectory generation, namely OpenStreetMap and GraphHopper. We then focus on formulating the multi-objective MPC problem and its vehicle-specific constraints, which is solved offline to generate the reference path and velocity trajectories. Afterwards, we discuss some adaptions to the system model and the controller operating strategy to incorporate real-time tracking of these trajectories while guaranteeing collision avoidance. Finally, we successfully demonstrate the system's feasibility by numerically evaluating its performance in a typical urban driving scenario for different vehicles.

Keywords Autonomous vehicles · Route planning · Trajectory following · Multi-objective MPC · GraphHopper · OpenStreetMap

Introduction

Nowadays, road traffic accidents represent a major public health concern as they are the primary cause of death and disability worldwide, resulting in over a million deaths annually and between 20 and 50 million injuries, some of which can lead to permanent disabilities. Moreover, expenditure on traffic accidents can account for up to 3% of a country's GDP, which is a significant financial burden, especially for developing countries [1, 2]. Therefore, it is imperative to explore innovative solutions that can create a safer driving and transportation environment from both a humanitarian and pragmatic perspective.

Autonomous Driving is a promising solution to this problem, as it has the potential to eliminate traffic accidents induced by human-driver error, such as speeding, drink-driving, delayed reaction time, and other forms of inattentive or aggressive driving [3]. In addition, it contributes to optimizing traffic flow, reducing fuel consumption, and enhancing passenger comfort. However, since it requires collaborative research across several domains, including computer vision, sensor data fusion, networks, and control theory, high-level automated driving has yet to be fully realized [4]. In this paper, we focus on the control aspects of the autonomous vehicles (AV) and present our research contribution to further advance the state-of-the-art in AV control, i.e., a generic, highly compatible architecture that can be applied for a variety of control systems and vehicle types with minor configurations.

Model Predictive Control (MPC) has recently become one of the most prevalent control methods in autonomous driving applications, owing to its flexibility, reliability, and effectiveness when dealing with multi-objective problems [5]. For example, it has been used in path following and lane

This article is part of the topical collection "Vehicle Technology and Intelligent Transport Systems" guest edited by Oleg Gusikhin and Markus Helfert.

Mostafa Emam
mostafa.emam@unibw.de

Matthias Gerdts
matthias.gerdts@unibw.de

1 Department of Aerospace Engineering, Institute of Applied Mathematics and Scientific Computing, University of the Bundeswehr Munich, Werner-Heisenberg-Weg 39, Neubiberg, 85579 Munich, Germany

Published online: 29 March 2024

keeping systems to promote driver safety and comfort while guaranteeing efficient fuel consumption, thus improving traffic flow and reducing air pollution [6, 7].

In a previous study [8], we introduced a deterministic MPC-based path tracker with collision-avoidance capabilities, in which multiple controllers were developed and fine-tuned for generic urban driving situations. Moreover, a Finite-State Machine (FSM) was implemented to analyze the current driving situation and appropriately activate the most suitable controller, as well as ensure smooth switching between the different controllers. This approach required modeling the path to be followed as a spline curve $\gamma_{ref} : [s_0, s_f] \to \mathbb{R}^2$ that spans from the desired start location A to the destination B and passes through the way-points $\gamma_{ref}(s) := [x_{ref}(s), y_{ref}(s)]^{\mathrm{T}}$. Accordingly, the control system was responsible for traversing this path as fast as possible while fulfilling some objectives (like minimizing the system controls) and adhering to some constraints (like speed limits).

Here, we continue to explore this idea by following a more standardized approach, in which we construct a configurable routing and tracking architecture that can control different vehicles in real-life urban driving scenarios. We take inspiration from sources like [9, 10] to understand how a typical autonomous driving architecture is designed, including expected system inputs and outputs and the characterization and responsibilities of each of the system components. To improve the applicability of our approach, we utilize the open-source geographic database OpenStreetMap (OSM) [11] and the open-source routing library GraphHopper [12] to construct feasible travel paths between desired locations across real-world street maps. Moreover, we consider sources like [13–15] to model the ego-vehicle and appropriately specify the system dynamics and constraints. Accordingly, we build vehicle- and objective-specific optimal reference path and velocity trajectories, which can be used as a baseline for online tracking. Finally, with the aid of [16, 17], we make some adaptations to our controller to create a real-time capable control system, which tracks the previously computed reference path and velocity trajectories while guaranteeing collision avoidance. This rounds up our proposed architecture, as it includes both path routing and real-time trajectory tracking while being configurable for different vehicle types and control objectives.

This paper is structured as follows. In “Control Architecture” section, we describe the overall system architecture and the inputs, outputs, and functionality of each of its components. We also briefly introduce the open-source services used in this study, namely OSM and GraphHopper. In “Trajectory Generation” section, we focus on the trajectory planning components, where we explain the algorithm that generates the reference path to be followed formulate the offline control problem that computes the vehicle-specific optimal velocity trajectory across the reference path. Afterwards, we discuss the tracking component in “Trajectory Tracking” section, with an emphasis on the modifications done to the system model and constraints to support real-time tracking of the reference trajectories. Finally, we evaluate the proposed architecture in numerical simulations and display the achieved results in “Numerical Simulation and Results” section, then summarize our work and ideas for future work in “Conclusion and Future Work” section.

Control Architecture

Autonomous driving has been the subject of extensive research over the past few decades, yet some challenges remain unsolved that hinder the development of fully autonomous vehicles (AVs) to this day, such as legal, technical, and ethical issues [18]. For example, one of the more controversial topics is the creation of a standardized development architecture for AVs, as it has been proven difficult to construct a single architecture suitable for all possible AV applications. Therefore, we can find multiple architectures that were developed to accommodate a specific scope, objective, or field of application [9, 10, 19], but on inspecting these architectures conjointly with an industrial approach to this problem [20], we realize that there are some characteristic components that recurrently appear in every architecture, such as the trajectory generation and trajectory execution components-albeit under different names. So in our proposed architecture, we primarily focus on the implementation of these two components, as they represent cornerstone elements that can be easily integrated into other existing architectures.

Other notable components are the environment- and self-perception modules, such as cameras, radar sensors, and data fusion modules. These modules may operate in a localized or cooperative manner with neighboring AVs, leveraging the vehicle’s sensor set and communication devices, as well as the available Internet of Vehicles (IoV) environment [21]. However, the configuration and selection of perception modules as well as their cost represent separate research topics that are beyond the scope of this study. Similarly, the actuation modules are typically vehicle and model-dependent and require considerable scientific contribution [22], so they will only be briefly discussed in the sequel.

Figure 1 illustrates a comprehensive summary of the typical components present in AV system architectures [19] from a control perspective, and can be explained as follows:

- *Perception* In this stage, the AV uses various sensors and algorithms to gather information about itself and its environment [23]. For example, camera modules

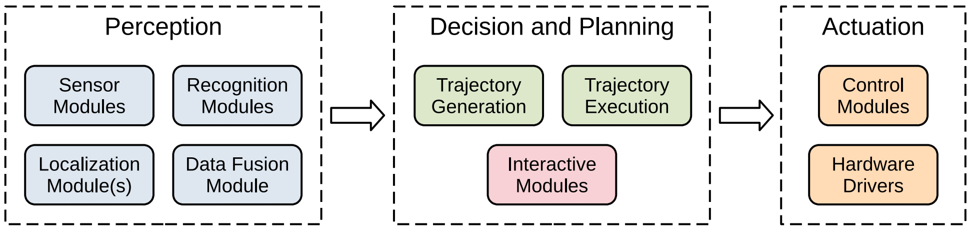


Fig. 1 System architecture of AVs with generic components

produce images that, when processed by the lane marks recognition module, yield feature points about the admissible driving area. By combining this with a localization algorithm, we can accurately determine the position of the ego-vehicle with respect to its travel lane to ensure safe driving [19]. The data fusion module is reserved for more complex functions, like pedestrian recognition and tracking, which require information from multiple sensors (e.g., cameras and LiDAR) to reliably extract the desired features [10].

- *Decision and planning* Here, the AV uses the acquired information from the perception modules to plan and decide the motion and behavior of the ego-vehicle, which includes trajectory planning, obstacle and collision avoidance, action prediction, and so on [24]. We can categorize these components into two groups: trajectory generation, i.e., the high-level path planning, and trajectory execution, i.e., the low-level real-time path tracking. Furthermore, some interactive modules exist that enable the driver to influence the decision-making process of the AV, e.g., to set a desired travel speed or to override a planned autonomous maneuver [9].
- *Actuation* After determining the desired driving maneuvers, the control modules convert these actions into physical controls of the ego-vehicle, such as steering and accelerating/braking. Consequently, they are transferred to and executed by the hardware drivers, as they directly interface with the chassis components of the AV (e.g., the steering wheel motor and the accelerator pedal motor) [10]. In some cases, the actuation modules may also be used to alert an inattentive driver with the aid of vibrant lights or loud sounds if the system detects an emergency [23].

In order to maximize compatibility with existing architectures, we propose a simplistic architecture to implement the decision and planning components, in which we assume prior knowledge of the required road data (e.g., driving lanes), ego data (e.g., speed and steering angle), and traffic data (e.g., positions and velocities of other traffic participants). Moreover, we configure our system to compute generic control actions, such that they can be converted later into the specific controls of different vehicles. Thus, the proposed architecture is displayed in Fig. 2 and the components of interest are highlighted in gray.

The implementation of these components will be discussed in detail in the upcoming sections, but first we will give a brief description of the open-source services we intend to use, namely OSM and GraphHopper. It is pertinent to mention that these services offer free solutions for academic purposes and were accordingly used at no cost in this study, so further research may be necessary to assess their financial viability in commercial applications.

OpenStreetMap

OSM [11] is a collaborative, volunteer-driven project that was launched in 2004 with the aim of creating a freely available geographic database of the world, that can be used in mapping, navigation, or similar applications without being limited by proprietary or monetary restrictions [25]. Since then, it has become one of the largest and most recognized Volunteered Graphical Information (VGI) projects, with millions of registered contributors and billions of data points covering countries across the entire world [25, 26]. Moreover, the OSM datasets are constantly increasing both in quality and magnitude, since their contributors include not only volunteers and non-profit organizations, but also professional editors from major tech corporations such as Facebook, Amazon, and Apple [27]. Finally, OSM datasets have already been utilized in navigation systems [26]

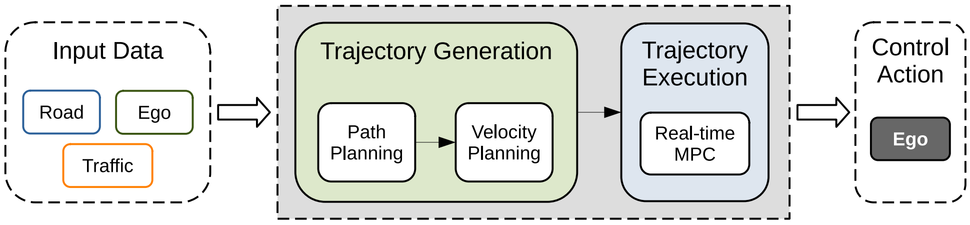


Fig. 2 Proposed routing and tracking architecture

and autonomous vehicle applications [27], which further supports our decision to use OSM in the path planning components of this study.

GraphHopper

GraphHopper is an efficient, open-source routing engine that offers multiple free and commercial solutions for traffic routing, such as turn-by-turn navigation, route planning and optimization, geocoding, and map matching [28]. It primarily utilizes OSM datasets for its road networks, and it can be configured to use different path-finding algorithms, such as Dijkstra, A*, and their bidirectional variations, to determine the fastest/shortest path between two given locations. GraphHopper works pretty efficiently, where it can calculate routes across hundreds of kilometers in mere seconds [29]. Additionally, it analyzes the map routes' metadata (e.g., road type, speed limits, and road dimensions), and thus can be configured for custom routing of different vehicles, cyclists, or pedestrians [26]. It can also be adapted to offer custom routing for specific driving objectives, such as finding the shortest path while optimizing nitrogen oxide emissions [30]. Overall, GraphHopper presents us with a free, fast, and customizable method for route planning, which perfectly fits our current problem.

Trajectory Generation

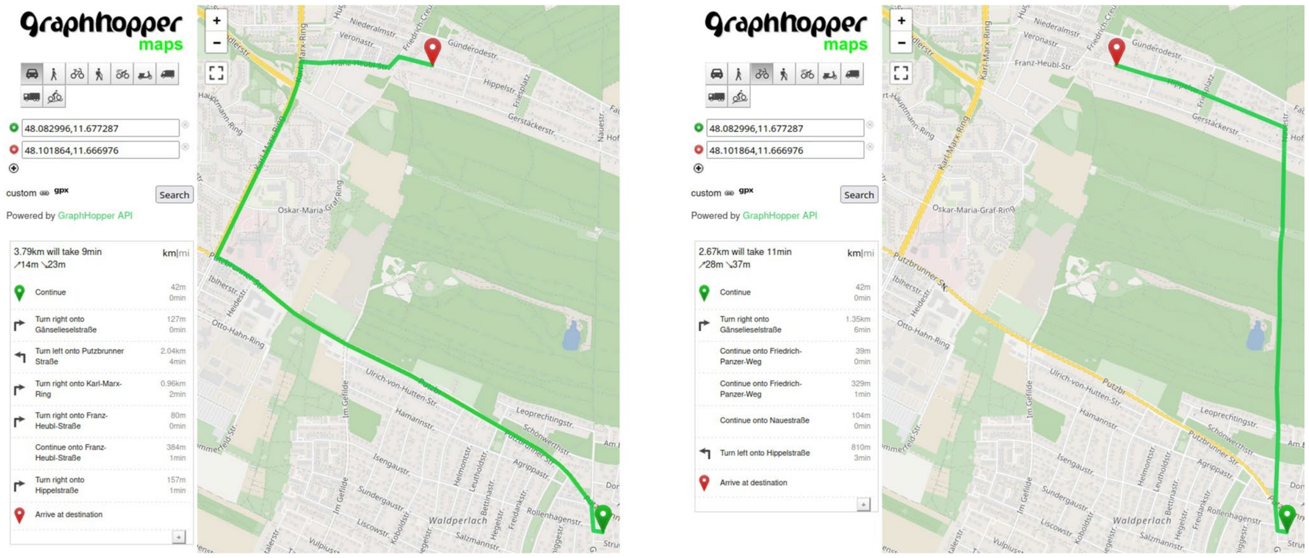
As previously discussed, we need to develop two components to generate the reference path and velocity trajectories that will be tracked by the ego-vehicle, which we will discuss in the following sections.

Path Planning

We start with the path planning component, which, when given a start location A and a destination B, must generate a feasible travel path γ_{ref} between them through the road network. To achieve this, we write the locations A and B in terms of their geographic coordinates $(lat_{A/B}, lon_{A/B})$ and use the GraphHopper Routing service to calculate the fastest route connecting them. If needed, we may also use the GraphHopper Geocoding service to convert desired addresses into (lat, lon) points. Figure 3 displays a sample of the different vehicle-specific fastest routes generated by GraphHopper between given start and end locations.

The web interface offers basic routing options, such as setting multiple stops between A and B and optimizing for a selected traffic participant type (e.g., car, truck, pedestrian, etc.). Accordingly, the generated route is a set of labeled geographic way-points with navigational instructions, as well as some rudimentary information like traveled distance and elevation data. However, the GraphHopper Routing API offers more capabilities, such as specifying a custom routing vehicle profile and selecting the language for navigational instructions, but more importantly, it can embed additional information in the metadata of the generated way-points, like road class, number of lanes, maximum speed, street name, and many more [12].

This API can be called over HTTP, where it responds with a JSON formatted data file containing the generated routing way-points with the requested metadata. To simplify the API calling process, we used MATLAB to develop the GUI depicted in Fig. 4, where the user can specify the routing options and desired metadata. Then, the GUI calls the API and exports the received generated route either to an



48.082996,11.677287
48.101864,11.666976

3.79km will take 9min
↗14m ↘23m

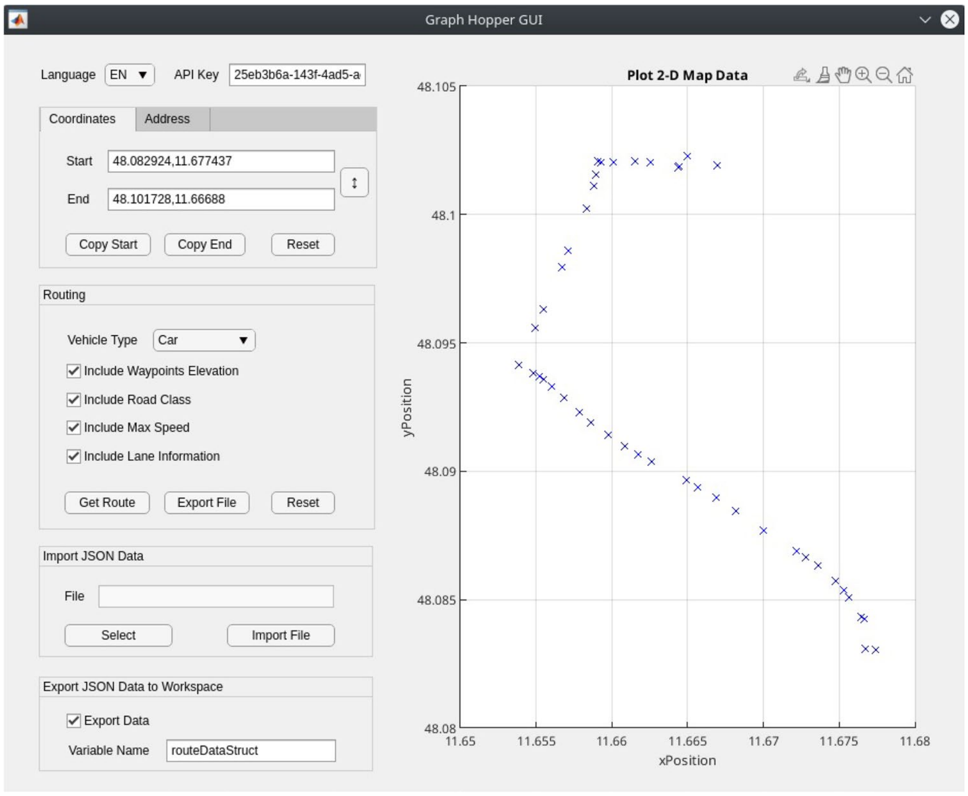
Continue	42m 0min
Turn right onto Gänselieselstraße	127m 0min
Turn left onto Putzbrunner Straße	2.04km 4min
Turn right onto Karl-Marx-Ring	0.96km 2min
Turn right onto Franz-Heubl-Straße	80m 0min
Continue onto Franz-Heubl-Straße	384m 1min
Turn right onto Hippelstraße	157m 1min
Arrive at destination	

48.082996,11.677287
48.101864,11.666976

2.67km will take 11min
↗28m ↘37m

Continue	42m 0min
Turn right onto Gänselieselstraße	1.35km 6min
Continue onto Friedrich-Panzer-Weg	39m 0min
Continue onto Friedrich-Panzer-Weg	329m 1min
Continue onto Nauestraße	104m 0min
Turn left onto Hippelstraße	810m 3min
Arrive at destination	

Fig. 3 Generated routes for a car (left) and a bicycle (right) between the same points using the GraphHopper web interface [12]


Graph Hopper GUI

Language EN ▼ API Key 25eb3b6a-143f-4ad5-a

Coordinates | Address

Start 48.082924,11.677437

End 48.101728,11.66688

Copy Start | Copy End | Reset

Routing

Vehicle Type Car ▼

☑ Include Waypoints Elevation

☑ Include Road Class

☑ Include Max Speed

☑ Include Lane Information

Get Route | Export File | Reset

Import JSON Data

File

Select | Import File

Export JSON Data to Workspace

☑ Export Data

Variable Name routeDataStruct

Fig. 4 Developed GUI to facilitate calling the GraphHopper Routing API

external data file or to the MATLAB workspace for further data manipulation. Also, we added a simple map displaying the generated way-points data in the GUI for clarity purposes.

On observing the generated route data, we notice two main issues: first, the way-points are specified in the geographic coordinates system, which is unsuitable for local positioning and trajectory following. To tackle this problem, we must redefine the 3D geodetic data in a local Cartesian coordinates system, which is achieved by using a map projection (e.g., the Universal Transverse Mercator) to transform the route map from ellipsoidal into plane coordinates [31]. We can then create a local Cartesian map with the projected way-points, which, in accordance with a GPS sensory module and an adequate vehicle model, enables system localization and tracking [4]. Luckily, MATLAB readily offers the function `latlon2local` that directly converts (*lat*, *lon*, *alt*) data points into (x, y, z) coordinates, where the local coordinate system is anchored around a desired origin point.

The second issue is that the received way-points resemble simple positions on a coordinates system, but we need a spline curve γ_{ref} for the path tracking component. We can solve this by constructing continuous natural cubic splines between the way-points to create γ_{ref} from A to B, which theoretically works, yet may yield infeasible routes due to the nature of the received way-points. As GraphHopper attempts to minimize the number of returned data points, it omits sequential points in a straight line, which results in having poorly defined road segments with few way-points, such that generating splines for these road segments may result in routes that span outside the allowed road network. We address this by analyzing the locations of each pair of sequential way-points and if their inter-distance is less than a specific threshold, we use interpolation to insert additional way-points between them. Thus, we can ensure that all of our road segments are adequately defined.

Algorithm 1 Compute Reference Path

```
Require: way-points in geo-coordinates
    Step 1: Map Projection
 1: Set the first way-point as origin for local coordinates
 2: for all way-points in geo-coordinates do
 3:     Calculate projected way-points in local Cartesian coordinates
 4: end for
    Step 2: Add Missing Way-points
 5: Create list of way-points in Cartesian coordinates
 6: for all $i$ with $1 \leq i <$ number of way-points do
 7:     AddExtraPoints(point($i$),point($i+1$))
 8: end for
 9: Sort the list of way-points
    Step 3: Adapt Location of Way-points
10: Create preliminary spline from sorted list of way-points
11: for all $i$ with $1 \leq i <$ number of way-points do
12:     if number of lanes at point($i$) $> 1$ then
13:         Use spline to project point($i$) to the right with offset $\triangleq$ lanes
14:     else if Navigation instructs to turn left/right at point($i$) then
15:         Use spline to project point($i$) to the left/right with constant offset
16:     end if
17: end for
18: Create final spline from adapted way-points
19: return Reference path as a spline

 1: procedure AddExtraPoints($A, B$)
 2:     if distance between $A$ and $B <$ threshold then
 3:         Calculate midpoint $M$ of $A$ and $B$
 4:         Add midpoint $M$ to the list of way-points
 5:         AddExtraPoints($A$, $M$)        ▷ recurse for points $A$ and $M$
 6:         AddExtraPoints($M$, $B$)        ▷ recurse for points $M$ and $B$
 7:     end if
 8: end procedure
```

Additionally, GraphHopper returns way-points exactly in the center of road segments, which may yield incorrect paths in case of multi-lane roads. So, by checking the number of lanes (from the metadata) in all road segments, we can appropriately shift the way-points' locations to favor traveling in the rightmost lane to promote safe driving. Similarly, we can analyze navigational instructions to identify turning maneuvers to the right/left and correspondingly shift the way-points to produce smoother travel paths. Lastly, we create continuous splines across the adapted way-points to form our reference path γ_{ref}. The complete process of way-points manipulation is summarized in Algorithm 1 and an example of the resulting spline is demonstrated in Fig. 5.

It is important to mention that the generated reference path serves only as a baseline for the ego-vehicle and that during real-time tracking, the sensor modules will provide the AV with additional data about the lane constraints, which will be used to ensure driving in admissible areas.

Velocity Planning

After generating the reference path, we now move unto calculating the optimal velocity trajectory corresponding to this path. We take the road data and the vehicle-specific dynamics and constraints into account and accordingly

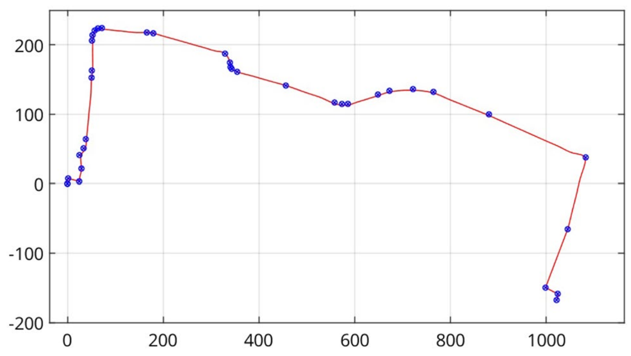


Fig. 5 Original way-points (blue) versus final spline from adapted way-points (red)

compute an explicit baseline for the velocity trajectory across this path. This ensures a safe, efficient, and comfortable trip, as well as simplifies the tracking problem, which facilitates achieving real-time trajectory tracking. We formulate this as an optimal control problem and solve it using Nonlinear MPC to guarantee feasible and smooth trajectories [32]. Since we intend to create generalized components for modeling various AVs, we need to use a generic model that describes the dynamic motion of the ego-vehicle while being independent of vehicle-specific parameters. So, we utilize the kinematic vehicle model [13] in coordination with a curvilinear coordinate system to describe the AV's motion across a reference curve $\gamma_{ref}: [0, L] \rightarrow \mathbb{R}^2$ as follows:

$$s'(t) = \frac{v(t)\cos\chi(t)}{1 - d(t)\cdot\kappa_{ref}(s(t))} \tag{1a}$$

$$d'(t) = v(t)\sin\chi(t) \tag{1b}$$

$$\chi'(t) = \psi'(t) - \psi'_{ref}(t) = v(t)\kappa(t) - s'(t)\kappa_{ref}(s(t)) \tag{1c}$$

$$\kappa'(t) = u_1(t) \tag{1d}$$

$$v'(t) = u_2(t) \tag{1e}$$

This yields the system state vector $x^T = [s, d, \chi, \kappa, v]$, where s is the arc length of a reference point on the ego-vehicle relative to γ_{ref}, d is the normal distance between this point and γ_{ref}, χ is the deviation in orientation with respect to γ_{ref}, and κ and κ_{ref} represent the curvature of the vehicle's trajectory and reference trajectory respectively. Another benefit of using the kinematic model is that there exists a subset of the system states $y^T = [d, \chi]$ that represent tracking errors to the reference trajectory, such that the path tracking problem equates to stabilizing $y(t)$ to 0. In addition, this model offers generic quantities for the system inputs (κ', v'), and they can later be converted to the controls of different vehicles by using the mapping $(x, u, X) \mapsto U = \mu(x, u, X)$ [8].

To define the system constraints, we first need to revisit the metadata of the way-points we received from GraphHopper, as they include information on the speed limits at the start and end of each of the road segments in the reference path. We use a simple heuristic to create continuous functions for the speed limits v_{min} and v_{max}, in which we assume constant values across a road segment if the speed limits remain unchanged, and otherwise use interpolation to determine the intermediate speed limit values. Consequently, we can impose the following restriction on the vehicle's velocity:

$$v_{min}(s(t)) \leq v(t) + \eta_v \leq v_{max}(s(t)) \tag{2}$$

such that $v_{min}(s(t))$ and $v_{max}(s(t))$ are computed with respect to the traveled arc length, and η_v is an additional slack variable for constraint relaxation, which increases the chances of finding feasible solutions without sacrificing system stability [33]. Second, we consider the AV's physical construction that may limit the behavior of its components, e.g., by imposing restrictions on its maximum acceleration/deceleration or its driving curvature. So, we add:

$$u_{min} \leq u(t) \leq u_{max} \tag{3a}$$

$$\kappa_{min} \leq \kappa(t) \leq \kappa_{max} \tag{3b}$$

which are vehicle-specific constraints that are determined and set for different vehicles independently [15]. Finally, we introduce a safety constraint that restricts the vehicle's lateral acceleration to a maximum allowed value $a_{n,max}$, in order to avoid excessive lateral forces when making sharp turns [13]:

$$-a_{n,max} \leq \kappa(t)v(t)^2 \leq a_{n,max} \tag{4}$$

As for the MPC cost function, we use the weighted multi-objective function:

$$min \int_0^T \omega_y||y|| + \omega_s\left(\frac{s_f - s}{s_f - s_0}\right)^2 + \omega_u||u|| + \omega_v{\eta_v}^2 dt \tag{5}$$

in which the first term incorporates minimization of the tracking errors to the reference trajectory, the second is a normalized incentive to minimize the difference between the vehicle's traveled distance s and the destination s_f [8], the third is a traditional objective to reduce energy consumption and maximize passenger comfort, and the fourth minimizes the slack variable for velocity relaxation η_v. By solving the problem, we get the desired reference path and velocity trajectories that will be later used in online tracking.

Trajectory Tracking

Nonlinear system models are suitable for describing the motion and dynamical behavior of complex systems, yet they impose an additional layer of complexity that optimal solvers may struggle against, which limits their usability in real-time applications [34]. Therefore, it is often beneficial to linearize the system dynamics around an operating point and attempt to find an optimal solution for the linearized model instead, as the problem structure can then be exploited to reach feasible solutions faster [35].

As the ego-vehicle is expected to closely follow the reference path, we can safely assume that its deviation in orientation χ with respect to γ_{ref} is small enough, such that $sin(\chi) \approx \chi$ and $cos(\chi) \approx 1$. Moreover, we can assume that its lateral deviation d is small with respect to the reference curve (which is defined as $r_{ref} = \frac{1}{\kappa_{ref}}$), such that $\frac{d}{r_{ref}} = d \cdot \kappa_{ref} \ll 1$ [16]. This simplifies the model introduced in Equation 1 and yields:

$$s'(t) = v(t) \tag{6a}$$

$$d'(t) = v(t)\chi(t) \tag{6b}$$

$$\chi'(t) = v(t) \cdot (\kappa(t) - \kappa_{ref}(s(t))) \tag{6c}$$

$$\kappa'(t) = u_1(t) \tag{6d}$$

$$v'(t) = u_2(t) \tag{6e}$$

Nevertheless, this model still includes some nonlinearities due to the coupled system dynamics in d and χ. To address this problem, some approaches proposed to assume a constant velocity profile [36], others suggested decoupling the longitudinal and lateral system dynamics and solving them in two sequential phases [37], however we argue that this can only yield sub-optimal trajectories. A more interesting idea would be to locally linearize the coupled system model using Taylor expansion at each time step and then solve for the fully linearized model, which may yield better results provided that the model is discretized with a small enough time step T_s, but we leave this for future studies. For the sake of this study and the application at hand, we proceed with the simplified model in Eq. 6 as it is sufficient to achieve acceptable real-time results.

For trajectory tracking, we use the constraints introduced in Eqs. 2 and 3, but ignore the constraint for $a_{n,max}$ as it has already been considered in computing the optimal velocity trajectory v_{ref} to simplify our problem further. In addition, we primarily reuse the constraints for safe following and driving within the permissible area previously proposed in [8], which we summarize in the following.

We define the occupancy region of the ego-vehicle as a rectangular area encompassing its complete 2-D footprint and optimally cover it with circular disks, then use a simple heuristic for collision detection, i.e., a collision occurs if an object (or lane markings) overlaps with the footprint of any of the covering disks. Accordingly, we guarantee driving within the admissible area by adding a set of clearance constraints $clear_{LN}$ that restrict the lateral deviation of the disks covering the ego-vehicle as demonstrated in Fig. 6. These constraints are defined as $d_{min,i} + r \leq d_i \leq d_{max,i} - r, \forall i \in 1, \dots, nrDisk_{ego}$, where d_i is calculated using trigonometric functions for each disk and r is the common disk radius. Determining $d_{max,i}$ and $d_{min,i}$ depends on the data we receive from the sensory modules [24], e.g., on receiving way-points corresponding to the lane markings, we may create the splines γ_{min} and γ_{max}, from which we can compute the minimum/maximum allowed lateral deviation with respect to arc length $d_{min/max}(s(t))$. Nonetheless, this is a rather irrelevant task to the MPC as it can be completely realized by an independent component that takes s and returns the corresponding $d_{min/max}$ values. So, in this work we rewrite the constraints to guarantee driving in one permissible lane with width w_{LN}, such that γ_{ref} lies in the middle of it, which yields the linear constraints:

$$r - \frac{w_{LN}}{2} \leq d \leq \frac{w_{LN}}{2} - r \tag{7a}$$

$$r - \frac{w_{LN}}{2} \leq d + \frac{l_d}{2}\chi \leq \frac{w_{LN}}{2} - r \tag{7b}$$

$$r - \frac{w_{LN}}{2} \leq d + l_d\chi \leq \frac{w_{LN}}{2} - r \tag{7c}$$

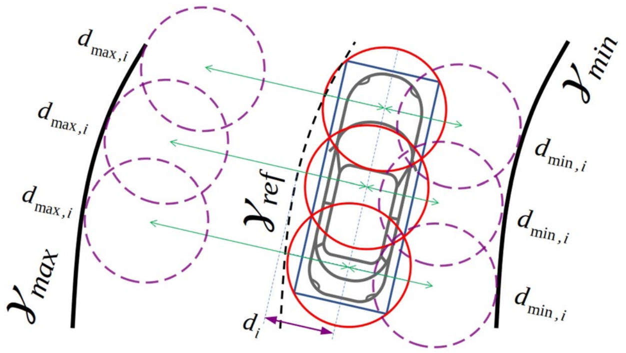


Fig. 6 The clearance constraints for driving inside the permissible area [8]

Moreover, we introduced a probabilistic component that determines the closest road user driving in the same lane as the ego-vehicle in [8]. We employ this component in accordance with a modified constant time headway policy to specify the safety distance $s_{SF}(t) = max(s_{SF,min}, v(t)t_h)$, which must always be kept to avoid collisions. Subsequently, we add a constraint for safe following:

$$s_{rel} \geq s_{SF} + \eta_{SF} \tag{8}$$

in which s_{rel} is the relative distance between the arc length of the leading traffic participant and the ego-vehicle respectively, and η_{SF} is the slack variable for safe following. Finally, we formulate the modified MPC objective function as:

$$\begin{aligned} &min(v(t_f) - v_{ref}(s(t_f)))^2 + \int_{t_0}^{t_f} \omega_y||y|| \\ &\quad + \omega_u||u|| + \omega_{SF}{\eta_{SF}}^2 dt \end{aligned} \tag{9}$$

which operates in the time span $t \in [t_0, t_f]$ and consists of a terminal term for following the reference velocity trajectory v_{ref}, in addition to a running cost for tracking the path trajectory as well as minimization of system controls and slack variables. We now solve the optimal control problem iteratively, where we start at $t = t_0$ and solve for the states, constraints, and cost function on the interval $\Delta T = t_f - t_0$ to determine the optimal control u^*, where u^* is applied for one time step $T_s = \frac{\Delta T}{N}$. Afterwards, we shift the prediction horizon by one time step and measure the system states, then solve the problem again for the updated states, constraints, and cost function. This is repeated until an exit condition is met, which, in our case, is reaching the destination. Notice that the terminal term $v(T) - v_{ref}(s(T))$ in Eq. 9 now replaces the normalized incentive $\frac{s_f - s}{s_f - s_0}$, as the vehicle shall continue to move with the optimal velocity trajectory until it finally stops at its destination.

FSM Operating Strategy

Similarly, we revise the FSM architecture previously discussed in [8] and add some modifications to incorporate the changes we made to our control problem due to introducing the velocity planning component. We present the modified configuration in Fig. 7 and switch conditions in Table 1, as we briefly discuss the formulation of the transition conditions in the following equations. For more details on the FSM modes, we refer to "Appendix".

On switching from one state to the other, we may have multiple exit conditions and in this case, we prioritize the conditions according to their indices, i.e., τ_1 is evaluated before τ_2. In case no exit condition is satisfied, the active state remains

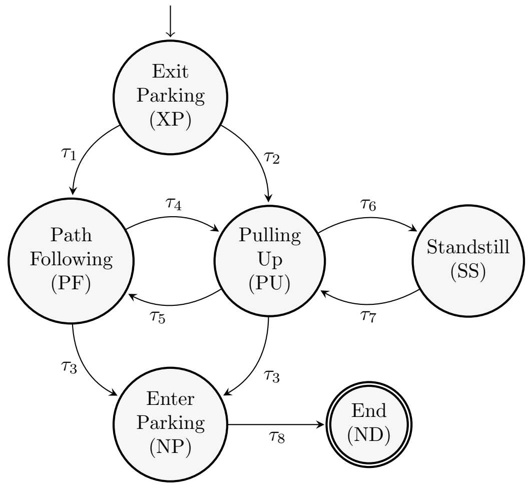


Fig. 7 FSM state configuration and possible transitions

unchanged, which is denoted by *OW* in Table 1. Moreover, we primarily use sigmoid functions $f(x) = \frac{1}{1+e^{\alpha+\beta x}}$ to calculate smoothing factors ω_τ for state exit conditions, such that we prevent aggressive switching from one state to the other, as well as guarantee smooth reversible switching if a condition is no longer satisfied. Notice that we will switch from one state to the other only when a condition is fully satisfied $\tau = 1$, otherwise, we remain in the same state and calculate a transition factor ω_τ to gradually modify the system objectives and constraints, then accordingly formulate a new control problem to find the optimal system controls [8].

For the first state XP, we have two exit conditions: τ_1 to PF and τ_2 to PU. Both are initially triggered on leaving the initial parking area (i.e., $s \geq s_{XP}$), in which s_{XP} is a traveled distance threshold for the parking area that is determined based on geographic data in accordance with γ_{ref}. Likewise, both transitions are completed after the ego-vehicle passes the relaxed threshold $s \geq s_{XP,\eta}$. The main difference between τ_1 and τ_2 is the maximum speed limit in each of their respective states, which appropriately affects the acceleration and deceleration limits for driving comfort. We define τ_1 as:

$$\tau_1 := \begin{cases} 1, & (s \geq s_{XP,\eta}) \land (v_{max} \geq 13.5[\mathrm{m}/s]) \\ \omega_{\tau_1}, & (s_{XP} \leq s < s_{XP,\eta}) \land (v_{max} \geq 13.5[\mathrm{m}/s]) \\ 0, & s < s_{XP} \end{cases} \tag{10}$$

where $\omega_{\tau_1} := f(s)$ is a sigmoid function, such that $\lim_{s\to s_{XP}} \omega_{\tau_1} = 0$ and $\lim_{s\to s_{XP,\eta}} \omega_{\tau_1} = 1$. Note that τ_2 is defined in a similar manner by using ω_{τ_1} as the smoothing function, but with a different speed limit $v_{max} \geq 8[\mathrm{m}/s]$. Next, we have the common exit condition τ_3 for both states PF and PU, which is defined in a complementary manner to τ_1 as:

$$\tau_3 := \begin{cases} 1, & s \geq s_{NP} \\ \omega_{\tau_3}, & s_{NP,\eta} \leq s < s_{NP} \\ 0, & s < s_{NP,\eta} \end{cases} \tag{11}$$

with s_{NP} and $s_{NP,\eta}$ being the distance threshold and relaxed threshold for the final parking area respectively. We now move unto the transition condition τ_4 from PF to PU, which is triggered either on following/approaching a slow road user, or when the road regulations specify a lower maximum speed limit than the current, i.e., 30 [km/h] instead of 50 [km/h]. Subsequently, we set τ_4 as:

$$\tau_4 := \begin{cases} 1, & \neg\tau_3 \land (v \leq 8) \land ((O_{inLN} \neq \emptyset) \lor (v_{max} \leq 8.4)) \\ \omega_{\tau_4}, & \neg\tau_3 \land ((v_{max} < 13.5) \lor ((O_{inLN} \neq \emptyset) \land (v < 13.5))) \\ 0, & \text{otherwise} \end{cases} \tag{12}$$

with the velocity-dependent sigmoid function $\omega_{\tau_4} := f(v[\mathrm{m}/s])$, such that $\lim_{v\to 13.5} \omega_{\tau_4} = 0$ and $\lim_{s\to 8} \omega_{\tau_4} = 1$. Note that we use marginally lower values for the speed limits (8 [m/s], 13.5 [m/s] instead of 30 [km/h], 50 [km/h]), since it promotes safe driving without sacrificing speed, while being slightly easier to formulate programmatically. Complementary to τ_4, we define the return transition τ_5 from PU to PF as:

Table 1 Conditions for switching between the different FSM modes

S_0	f					
	XP	PF	PU	NP	SS	ND
XP	*OW*	τ_1	τ_2	–	–	–
PF	–	*OW*	τ_4	τ_3	–	–
PU	–	τ_5	*OW*	τ_3	τ_6	–
SS	–	–	–	τ_7	*OW*	–
NP	–	–	–	*OW*	–	τ_8

$$\tau_5 := \begin{cases} 1, & \neg\tau_3 \wedge (v_{max} \geq 13.5) \wedge ((O_{inLN} = \emptyset) \vee (v \geq 13.5)) \\ \omega_{\tau_5}, & \neg\tau_3 \wedge (v_{max} > 8.4) \wedge ((O_{inLN} = \emptyset) \vee (v \geq 8)) \\ 0, & \text{otherwise} \end{cases} \tag{13}$$

where $\omega_{\tau_5} = 1 - \omega_{\tau_4}$. Afterwards, we examine the transitions for the utility state SS, which can be sufficiently implemented as hard constraints due to the nature of the state itself. The transitions for entering and exiting the state are:

$$\tau_6 := \neg\tau_5 \wedge (v \leq 0.5[\text{m}/s]) \wedge (a \leq 0[\text{m}/s^2]) \wedge (s_{rel} \leq s_{SF}) \tag{14}$$

$$\tau_7 := (O_{inLN} = \emptyset) \vee (s_{rel} \geq s_{SF} + s_{SF,\eta}). \tag{15}$$

Finally, we define the final transition τ_8 by specifying some threshold s_{ND} for reaching the destination s_f, such that:

$$\tau_8 := (s_f - s \leq s_{ND}) \tag{16}$$

after which the ND state is fully activated and promptly decelerates the ego-vehicle until it comes to a complete halt, thus completing the trip.

Numerical Simulation and Results

Next, we will assess the performance of the proposed architecture under the following assumptions: first, that the ego-vehicle is equipped with the necessary sensor set to accurately determine the vehicle's position and obtain the control-relevant states, such as velocity and curvature. Second, that the sensor data is high-fidelity, reliable, and instantaneously available without any data loss or delay after being filtered (similarly for the communication with the virtual traffic light). Finally, we assume that the vehicle control is consistent and prompt with no delays.

As a compromise between simplicity and performance, we implemented the proposed architecture as a combination of MATLAB and FORTRAN code, in which the software OCPID-DAE1 [38] is used to solve the optimal control problems, namely velocity trajectory planning and real-time tracking. The simulation results were obtained on a Linux system with the processor i5-5200U of 2.20 GHz and 8 GB of RAM. For the MPC parameters, we selected a prediction horizon of $\Delta T = 3.0\,[\text{s}]$ with $N = 15$ control points and a time step of $T_s = 0.2\,[\text{s}]$ for the velocity planning and $\Delta T = 2.0\,[\text{s}]$, $N = 10$, and $T_s = 0.2\,[\text{s}]$ for the real-time tracking. Model parameters were primarily defined using a variety of sources [15, 39] to guarantee feasibility and maximize passenger comfort, where we created two different sets of parameters for the kinematic vehicle to resemble a car and a small truck. Furthermore, the weights in the MPC objective function in Eqs. 5 and 9 were reused from [8], then slightly modified to improve performance based on trial and error.

Using the knowledge of local streets around the University of the Bundeswehr Munich, we formed a basic scenario with a route of about 1.5 [km] and a lane width of $w_{LN} = 3.25$ m that ensures activation of the main FSM states. This route is displayed in Fig. 8, where it is obvious that the path planning for both vehicles yields very similar routes, yet we get different velocity trajectories based on the model parameters and system constraints. This is especially true for the constraint due to $a_{n,max}$, which is responsible for the sharp dip in the velocity of both vehicles at around $s \approx 215$ m and $s \approx 520$ m. For evaluating the performance of the real-time tracking controller, we augment the scenario with a virtual traffic light that is located at $s = 1270$ m and gets activated (turns red) during the time interval $t = [150\,\text{s}, 180\,\text{s}]$, so as to verify the approach in a typical urban environment.

As shown in Fig. 9, both vehicles are able to travel across the reference path with minimal deviation from start to finish. In addition, they travel with smooth velocity trajectories that generally adhere to their respective maximum and optimal velocity trajectories as depicted in Fig. 10. Note that

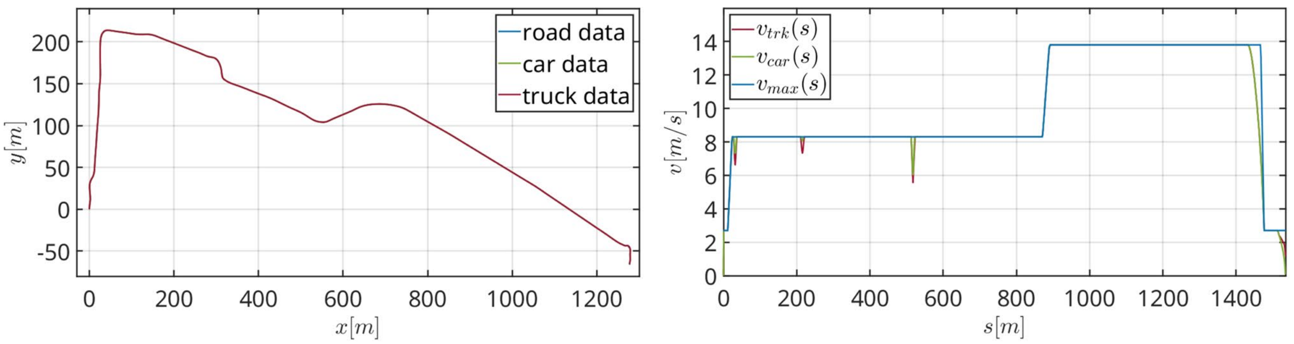


Fig. 8 Reference path (left) and velocity (right) trajectories generated for the different vehicle types

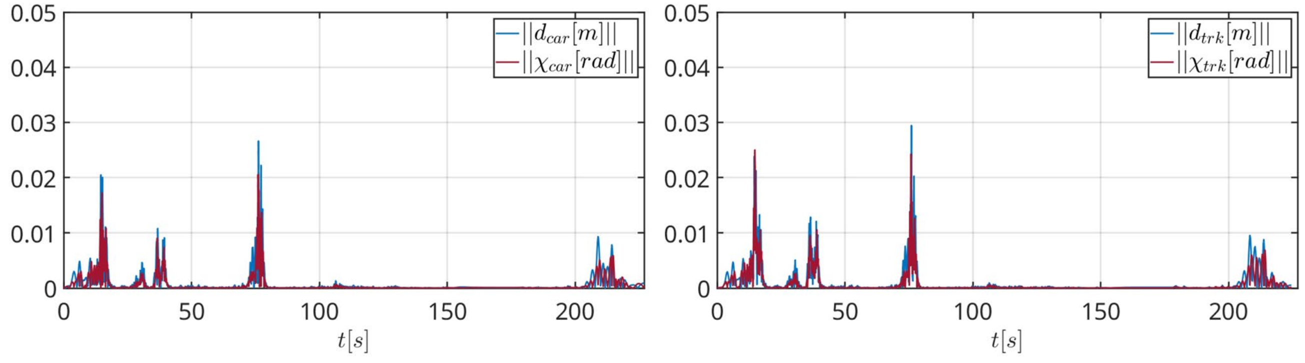


Fig. 9 Error states d, χ for both vehicles in the test scenario

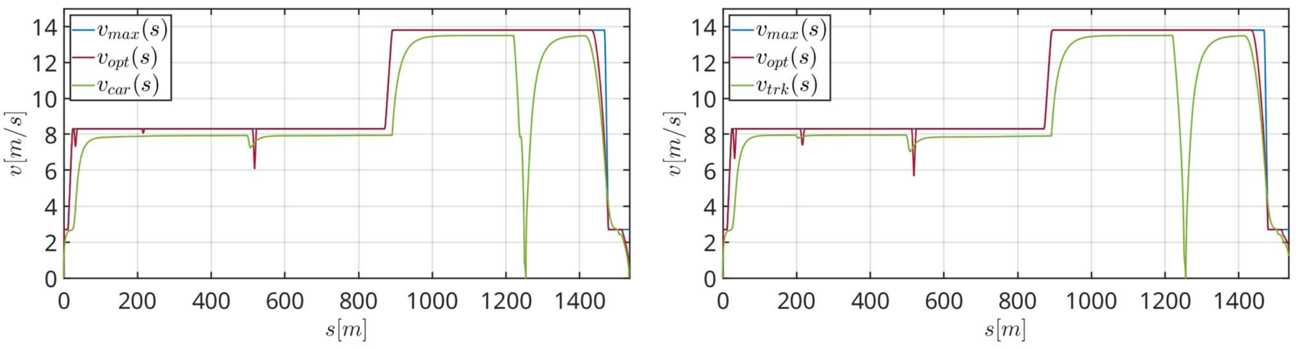


Fig. 10 Velocity trajectories for both vehicles with respect to their maximum and optimal velocities

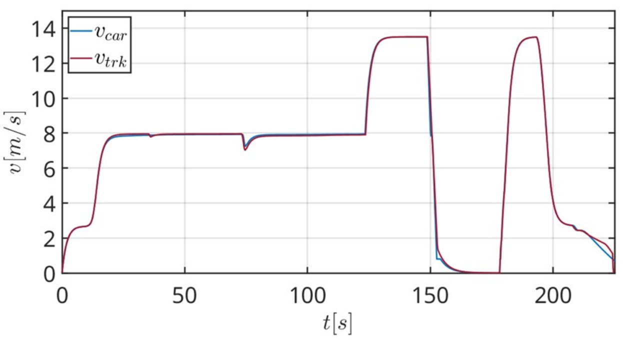


Fig. 11 The time-dependent velocity trajectories signify the effect of the traffic light in our test scenario

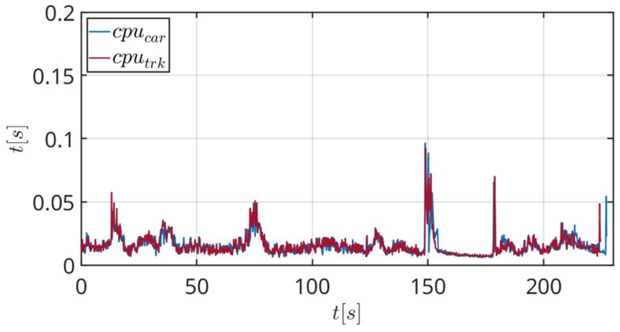


Fig. 12 Execution time for solving the online tracking control problem

these figures are displayed with respect to the arc length s and not time t in order to present a meaningful comparison.

Subsequently, Fig. 11 provides additional insight into the vehicles' real-time motion, as we see that both vehicles halt in the time period when the traffic light is activated. Finally, we can see in Fig. 12 that trajectory tracking problem was successfully solved in real-time, in which a solution was always found before our specified time step $T_s = 200\,[\text{ms}]$ had passed. To be precise, the average CPU time for solving the car problem was 15.33 [ms] with a maximum of 96.2 [ms], and the average CPU time for the truck problem was 15.53 [ms] with a maximum of 92.03 [ms], which solidifies the validity of our proposed approach for real-time applications.

Conclusion and Future Work

In this paper, we introduced a multi-component flexible architecture for autonomous driving, which integrates path routing, velocity planning, and real-time tracking of pre-computed path and velocity trajectories for different vehicle types. We utilized open-source services, i.e., GraphHopper and OSM, to receive preliminary routing way-points between desired start and end locations, and applied a manipulation algorithm that analyzes the way-points' meta-data to construct an initial continuous spline that can be followed by a traditional MPC controller. Moreover, we divided the optimal control problem into two steps, the first of which is solved offline to calculate the optimal path and velocity trajectories for a nonlinear model, and the second is solved online for a simplified model to follow the trajectories in real-time while guaranteeing safe following of leading traffic participants. The architecture was tested successfully and it shows promising results with respect to real-time autonomous driving.

Ideas for future studies include more comprehensive simulations in a virtual environment with realistic sensor models and system delays, as well as optimization of the system models to achieve improved real-time capabilities, e.g., support longer prediction horizons and shorter time steps. Ultimately, the architecture shall be tested on physical hardware to verify its applicability in real-life, for instance on scale RC cars initially, then on actual test vehicles. Screenshots in Fig. 3 are excerpted from the web interface of GraphHopper https://graphhopper.com/maps. The developed routing Algorithm 1 and corresponding GUI in Figs. 4, 5 are publicly available at the following repository: https://doi.org/10.5281/zenodo.10882122. Moreover, this repository includes the code and data files used in the test scenario demonstrated in Figs. 8, 9, 10, 11, 12, namely: the FORTRAN code files for velocity planning and trajectory tracking using the FSM approach, as well as the generated data files.

Appendix: Description of FSM Modes

The FSM modes were first introduced in [8] and they correspond to typical urban driving sequences, with the idea that different controllers may be developed and fine-tuned to better fit each driving sequence, and the FSM will optimally switch between these modes/controllers while guaranteeing smooth system controls. The FSM modes are defined as follows:

- *Exit Parking (XP)* The initial state that is activated on starting the trip. It signifies that the ego-vehicle is driving inside a parking area and imposes a restriction on the maximum velocity to be no faster than walking speed.
- *Path Following (PF)* The most common urban driving state, which corresponds to traveling with a maximum speed of $13.5\,[\mathrm{m/s}] \approx 50\,[\mathrm{km/h}]$ inside the city. This is either activated on exiting the parking area unto a main road, or when the road specifies the aforementioned speed limit and there are no slower traffic participants (or traffic lights) that hinder moving with this speed.
- *Pulling Up (PU)* The second most common urban driving state, which corresponds to traveling with a maximum speed of $8\,[\mathrm{m/s}] \approx 30\,[\mathrm{km/h}]$. This state is either activated when there are roadworks (or similar restrictions) that limit the maximum speed inside the city, or when the ego-vehicle is traveling behind a very slow leading road user. This state allows more aggressive controls as the ego-vehicle is traveling at a much slower speed than PF.
- *Stand Still (SS)* A utility state that temporarily stops the ego-vehicle behind a stationary object (or at a traffic light). This state acts as a buffer that prevents unnecessary start-stop maneuvers behind a very slow leading user, e.g., in a traffic jam, or when waiting at a traffic light to minimize fuel consumption.
- *Enter Parking (NP)* Similar to XP, with the difference being that it gets activated when the ego-vehicle enters the parking area close to its destination.
- *End (ND)* The final state, which gets activated on approaching the destination and is responsible for rapidly decelerating the ego-vehicle until it reaches a complete stop at its destination. Also, this state acts as a flag to notify the user that the trip has been concluded.

Acknowledgements This research paper is part of [project MORE - Munich Mobility Research Campus] and is funded by dtec.bw—Digitalization and Technology Research Center of the Bundeswehr. dtec.bw is funded by the European Union—NextGenerationEU. Map data copyrighted OpenStreetMap contributors are available from https://www.openstreetmap.org.

Funding Open Access funding enabled and organized by Projekt DEAL.

Data availability Source code and data files are now available at the repository: https://doi.org/10.5281/zenodo.10882122.

Declarations

Conflict of interest On behalf of all authors, the corresponding author states that there is no Conflict of interest.

References

1. Mohammed AA, Ambak K, Mosa AM, Syamsunur D. A review of the traffic accidents and related practices worldwide. Open Transp J. 2019;13(1):65–83. https://doi.org/10.2174/1874447801913010065.
2. Cruz OGD, Padilla JA, Victoria AN. Managing road traffic accidents: a review on its contributing factors. IOP Conf Ser Earth Environ Sci. 2021;822(1):012015. https://doi.org/10.1088/1755-1315/822/1/012015.
3. Winkle T. Autonomous driving. In: Maurer M, Gerdes JC, Lenz B, Winner H, editors. Safety benefits of automated vehicles: extended findings from accident research for development, validation and testing. Berlin: Springer; 2016. p. 335–64. https://doi.org/10.1007/978-3-662-48847-8_17.
4. Yurtsever E, Lambert J, Carballo A, Takeda K. A survey of autonomous driving: common practices and emerging technologies. IEEE Access. 2020;8:58443–69. https://doi.org/10.1109/ACCESS.2020.2983149.
5. Luo L, Liu H, Li P, Wang H. Model predictive control for adaptive cruise control with multi-objectives: comfort, fuel-economy, safety and car-following. J Zhejiang Univ Sci A. 2010;11:191–201. https://doi.org/10.1631/jzus.A0900374.
6. Musa A, Pipicelli M, Spano M, Tufano F, De Nola F, Di Blasio G, Gimelli A, Misul DA, Toscano G. A review of model predictive controls applied to advanced driver-assistance systems. Energies. 2021;14(23):7974. https://doi.org/10.3390/en14237974.
7. Yu S, Hirche M, Huang Y, Chen H, Allgöwer F. Model predictive control for autonomous ground vehicles: a review. Auton Intell Syst. 2021;1(1):4. https://doi.org/10.1007/s43684-021-00005-z.
8. Emam M, Gerdts M. Deterministic operating strategy for multi-objective NMPC for safe autonomous driving in urban traffic. In: Proceedings of the 8th international conference on vehicle technology and intelligent transport systems—VEHITS; 2022. p. 152–161. https://doi.org/10.5220/0011115400003191
9. Nolte M, Rose M, Stolte T, Maurer M. Model predictive control based trajectory generation for autonomous vehicles—an architectural approach. In: 2017 IEEE intelligent vehicles symposium (IV); 2017. https://doi.org/10.1109/ivs.2017.7995814
10. Behere S, Törngren M. A functional architecture for autonomous driving. In: Proceedings of the first international workshop on automotive software architecture; 2015. https://doi.org/10.1145/2752489.2752491
11. OpenStreetMap contributors: Planet dump. Retrieved from https://planet.osm.org. https://www.openstreetmap.org; 2017.
12. GraphHopper GmbH: GraphHopper Directions API. https://github.com/graphhopper/graphhopper
13. Burger M, Gerdts M. Applications of differential-algebraic equations: examples and benchmarks. In: Campbell S, Ilchmann A, Mehrmann V, Reis T, editors. DAE aspects in vehicle dynamics and mobile robotics. Cham: Springer; 2019. p. 37–80. https://doi.org/10.1007/11221_2018_6.
14. Polack P, Altche F, d'Andrea-Novel B, de La Fortelle A. The kinematic bicycle model: a consistent model for planning feasible trajectories for autonomous vehicles? In: 2017 IEEE Intelligent Vehicles Symposium (IV); 2017. https://doi.org/10.1109/ivs.2017.7995816
15. Bokare PS, Maurya AK. Acceleration-deceleration behaviour of various vehicle types. Transp Res Proc. 2017;25:4733–49. https://doi.org/10.1016/j.trpro.2017.05.486.
16. Gutjahr B, Gröll L, Werling M. Lateral vehicle trajectory optimization using constrained linear time-varying MPC. IEEE Trans Intell Transp Syst. 2017;18(6):1586–95. https://doi.org/10.1109/TITS.2016.2614705.
17. Chen C, Guo J, Guo C, Chen C, Zhang Y, Wang J. Adaptive cruise control for cut-in scenarios based on model predictive control algorithm. Appl Sci. 2021;11(11):5293. https://doi.org/10.3390/app11115293.
18. Kröger F. Autonomous driving. In: Maurer M, Gerdes JC, Lenz B, Winner H, editors. Automated driving in its social, historical and cultural contexts. Berlin: Springer; 2016. p. 41–68. https://doi.org/10.1007/978-3-662-48847-8_3.
19. Zong W, Zhang C, Wang Z, Zhu J, Chen Q. Architecture design and implementation of an autonomous vehicle. IEEE Access. 2018;6:21956–70. https://doi.org/10.1109/access.2018.2828260.
20. ISO Central Secretary: Intelligent transport systems—Reference model architecture(s) for the ITS sector—Part 5: Requirements for architecture description in ITS standards. Standard, International Organization for Standardization, Geneva; 2020. https://www.iso.org/standard/73746.html
21. Cui G, Zhang W, Xiao Y, Yao L, Fang Z. Cooperative perception technology of autonomous driving in the internet of vehicles environment: a review. Sensors. 2022;22(15):5535. https://doi.org/10.3390/s22155535.
22. Balerna C, Neumann M-P, Robuschi N, Duhr P, Cerofolini A, Ravaglioli V, Onder C. Time-optimal low-level control and gearshift strategies for the formula 1 hybrid electric powertrain. Energies. 2020;14(1):171. https://doi.org/10.3390/en14010171.
23. Zanchin BC, Adamshuk R, Santos MM, Collazos KS. On the instrumentation and classification of autonomous cars. In: 2017 IEEE international conference on systems, man, and cybernetics (SMC); 2017. https://doi.org/10.1109/smc.2017.8123022
24. Ahangar MN, Ahmed QZ, Khan FA, Hafeez M. A survey of autonomous vehicles: enabling communication technologies and challenges. Sensors. 2021;21(3):706. https://doi.org/10.3390/s21030706.
25. Neis P, Zielstra D. Recent developments and future trends in volunteered geographic information research: the case of OpenStreetMap. Future Internet. 2014;6(1):76–106. https://doi.org/10.3390/fi6010076.
26. Samah KAFA, Ibrahim S, Ghazali N, Suffian M, Mansor M, Latif WA. Mapping a hospital using OpenStreetMap and GraphHopper: a navigation system. Bull Electr Eng Inf. 2020. https://doi.org/10.11591/eei.v9i2.2082.
27. Anderson J, Sarkar D, Palen L. Corporate editors in the evolving landscape of OpenStreetMap. ISPRS Int J of Geo-Inf. 2019;8(5):232. https://doi.org/10.3390/ijgi8050232.
28. Karich P. Flexible Routenplanung mit GraphHopper. FOSSGIS e.V.; 2016. https://doi.org/10.5446/19732. https://av.tib.eu/media/19732
29. Johnson I, Henderson J, Perry C, Schöning J, Hecht B. Beautiful... but at what cost? Proc ACM Interact Mobile Wearable Ubiquitous Technol. 2017;1(2):1–21. https://doi.org/10.1145/3090080.
30. Engelmann M, Schulze P, Wittmann J. Emission-based routing using the GraphHopper API and OpenStreetMap. In: Progress in IS; 2019. pp. 91–104. https://doi.org/10.1007/978-3-030-30862-9_7
31. Usery E, Finn M, Mugnier C. Chapter 8—Coordinate Systems and Map Projections; 2009. p. 87–112.

32. Grüne L, Pannek J. Nonlinear model predictive control: theory and algorithms, communications and control engineering. London: Springer; 2011.
33. Vu TM, Moezzi R, Cyrus J, Hlava J. Model predictive control for autonomous driving vehicles. Electronics. 2021;10(21):2593. https://doi.org/10.3390/electronics10212593.
34. Falcone P, Tufo M, Borrelli F, Asgari J, Tseng H.E. A linear time varying model predictive control approach to the integrated vehicle dynamics control problem in autonomous systems. In: 2007 46th IEEE conference on decision and control; 2007. p. 2980–2985. https://doi.org/10.1109/CDC.2007.4434137
35. Huber AK-P. Methoden zur berechnung optimaler rennlinien im dynamischen grenzbereich. PhD thesis, Universität der Bundeswehr München; 2020. https://athene-forschung.unibw.de/135181
36. Pereira GC, Svensson L, Lima PF, Martensson J. Lateral model predictive control for over-actuated autonomous vehicle. In: 2017 IEEE Intelligent Vehicles Symposium (IV); 2017. p. 310–316. https://doi.org/10.1109/ivs.2017.7995737
37. Turri V, Carvalho A, Tseng H.E, Johansson K.H, Borrelli F. Linear model predictive control for lane keeping and obstacle avoidance on low curvature roads. In: 16th international IEEE conference on intelligent transportation systems (ITSC 2013); 2013. https://doi.org/10.1109/itsc.2013.6728261
38. Matthias Gerdts: OCPID-DAE1 : Optimal Control and Parameter Identification with Differential-Algebraic Equations of Index 1. https://www.unibw.de/ingmathe/teaching/prakopt/2018/ocpiddae1.pdf
39. ISO Central Secretary: Intelligent transport systems—Adaptive cruise control systems—Performance requirements and test procedures. Standard, International Organization for Standardization; 2018. https://www.iso.org/standard/71515.html