Universität der Bundeswehr München
Fakultät für Luft- und Raumfahrttechnik
Institut für Technik Autonomer Systeme

Towards Smoother and More Precise Autonomous Driving

Dipl.-Ing. Benjamin Christian Heinrich

Vollständiger Abdruck der von der Fakultät für Luft- und Raumfahrttechnik der Universität der Bundeswehr München zur Erlangung des akademischen Grades eines

Doktor-Ingenieurs (Dr.-Ing.)

genehmigten Dissertation.

Gutachter: 1. Univ.-Prof. Dr.-Ing. Hans-Joachim Wünsche
2. Univ.-Prof. Dr.-Ing. Prof. h.c. Dr. h.c. Torsten Bertram

Diese Dissertation wurde am 06.02.2023 bei der Universität der Bundeswehr München eingereicht und durch die Fakultät für Luft- und Raumfahrttechnik am 28.06.2023 angenommen. Die mündliche Prüfung fand am 17.07.2023 statt.

Preface

Danksagung

2022–02–06

Liebe Lesende,

ich freue mich an dem Punkt angekommen zu sein, an dem ich meine Arbeit durch das Schreiben dieses Grußwortes abschließen kann — und das selbiges, wie hoffentlich auch die Arbeit selbst, gelesen wird. Wie jedes Dokument, ist dieses im Kontext der Zeit zu sehen, in der es enstanden ist, und sollte entsprechend eingeordnet werden.

Die folgende Arbeit entstand während meiner Zeit am Institut für die Technik Autonome Systeme (TAS) an der Universität der Bundeswehr, München, zwischen Oktober 2012 und Dezember 2018. Um genauer zu sein, ihr Inhalt entstand während meiner Arbeit als wissenschaftlicher Mitarbeiter dort. Das Zusammenschreiben war ein längerer Prozess, da ich, seitdem ich in die Industrie gewechselt bin, zum zweiten und dritten Mal Vater wurde sowie ein Haus erbaut habe. Zusätzlich stehen wir seit nun mehr als zwei Jahren die COVID-19 Pandemie durch; und jeder, der Kinder hat, weiß was das für die persönliche Zeiteinteilung bedeutet.

Zum Glück bin ich in meiner Arbeit dem Thema „autonomes Fahren“[1] treu gelieben und kann somit weiterhin den Stand der Technik beurteilen und guten Gewissens meine Dissertation abgeben. Glücklich ist auch, dass mein Arbeitgeber —die Robert Bosch GmbH— mich durch eine Freistellung im Dezember 2021 unterstützt hat, während derer der Großteil der Schreibarbeit fertiggestellt werden konnte. An dieser Stelle möchte ich mich hierfür bei meinem Arbeitgeber bedanken.

Großer Dank geht an Herrn Prof. Dr.-Ing. Prof. h.c. Dr. h.c. Torsten Bertram, der sich bereit erklärt hat diese Arbeit als Zweitgutachter mitzubetreuen. Der Austausch mit ihm und seinen Doktoranden auf verschiedenen Konferenzen war stets wertig.

Größter Dank geht natürlich an meinen Betreuer Herrn Prof. Dr.-Ing. Hans-Joachim Wünsche. Die Ausbildung die ich an seinem Institut genossen habe, mit dem Fokus darauf, wissenschaftliche Erkenntnisse auch immer direkt ins Fahrzeug zu bringen, war die perfekte Vorbereitung für meine derzeitige Arbeit. Die Industrie würde sich wünschen, dass es mehr Institute gäbe, bei denen echte Erfahrung im Fahrzeug und mit einer großen Codebase in C++ gesammelt werden kann.

[1] In dieser Arbeit wird vom ‘Autonomen Fahren’ gesprochen, obwohl Autonom aus $\alpha\upsilon\tau o\varsigma$ (autós: selbst) und $\nu o\mu o\varsigma$ (nómos: ‘Gesetz’) für ein Fahren nach *eigenen* Gesetzen steht. Korrekter Weise müsste man vom hochautomatisierten Fahren sprechen; das klingt aber natürlich nicht so gut und ist deshalb eher in Fachkreisen und weniger in den Medien oder in wissenschaftlichen Veröffentlichungen zu finden.

Was die Arbeit am Institut so wertvoll gemacht hat, waren auch in erster Linie die Kollegen mit denen ich Tage und Nächte am Versuchsträger oder vor dem Bildschirm (oder beides auf einmal) verbracht habe.
Danke Dennis, für die enge Zusammenarbeit. Unser Aufeinandertreffen, also das von Informatiker und Ingenieur bzw. von Pragmatiker und Perfektionist, hat die Ergebnisse in dieser Arbeit erst möglich gemacht. Besten Dank auch, dass Du diese Arbeit korrekturgelesen hast!
Danke Hanno, dass Du mit mir die tiefen Abgründe von C++ erkundet hast und mir auch immer wieder geholfen hast Probleme zu lösen, die vielleicht garnicht unbedingt nötig gewesen wären, aber dennoch gelöst werden wollten!
Danke Thorsten, dass Du den Laden immer zusammengehalten hast und mit unermüdlichem Einsatz dafür gesorgt hast, dass wir doch immer, allen Widerständen zum Trotz, wieder ins Ziel gekommen sind!

Last but not least danke ich meiner Familie und allen voran meiner Frau Franziska. Danke, dass Du diesen langwierigen Prozess der Finalisierung der Promotion neben der Arbeit, neben den Kindern, neben der fehlenden Kinderbetreuung, neben dem Hausbau, und neben all dem Anderen mitgemacht hast.

—Benjamin Heinrich

Abstract

Autonomous driving evolved into a vast field of research. While it started already in the 80s of the last century (see Dickmanns and Zapp [1987]), it was really kicked off with the DARPA Challenges (Grand and Urban, see Buehler et al. [2007, 2009], respectively). Now, fully automated *Highway Pilots* and *Urban Automated Taxis* seem within reach as different major players in the industry announce these regularly. Yet, as regularly, these plans are postponed which shows that there are still more than enough problems to solve. In this work, planning and control problems for autonomous driving in unstructured environments, e. g., gravel roads through the woods or very dense urban scenarios, are considered. The goal is to achieve improved driving performance in terms of smoothness and precision especially in these challenging settings.

Since every part of the control chain —from perception over fusion and planning to actuation— can only be as strong as its weakest link, this work starts with the system architecture. A new architecture is proposed which, by providing the latter parts of the control chain with more information, enables better performance of the planning and control part and thus improves the overall performance. The new architecture also aims at facilitating development and maintenance. Simple practical examples show improved driving performance, even with the same controllers in place.

One main focus in this work is motion planning. The backbone of TAS (Institute for Autonomous Systems Technology) motion-planning framework is an extension of the so-called hybrid-state A* algorithm (see Dolgov et al. [2008]) where a guided exploration is used. Further, the classic path-velocity decomposition (see Kant and Zucker [1986]) is used and extended by using it in conjunction with modern trajectory-planning and optimization techniques. For free driving, i. e., without lead vehicle, state-of-the-art collision-checking methods are improved. For convoy driving, i. e., one vehicle following another, TAS's own award-winning following performance is improved upon. For platooning, i. e., multi-vehicle convoys, the state of the art is defined in a novel setting: without any lateral guidance from lane markings and with only a shared bandwidth of 19.2 $\frac{\text{kbit}}{\text{s}}$ for all inter-vehicle communication.

The other main focus in this work is motion control. The new architecture has a vehicle-specific low-level part and a vehicle-unspecific high-level part. While the former is described mainly for completeness, the latter features several contributions. The main mode of operation is trajectory following, where the importance of localizing the ego vehicle in both time and place consistently over time is stressed. This allows, together with the consistency of the trajectory generation introduced in the former chapter, to handle delays in the control chain efficiently. Further, extensions to the trajectory following —especially for the convoying use-case— are discussed. Finally, it is described how a normal-sized (electric) vehicle is guided and positioned with sub-centimeter precision in outdoor scenarios.

Zusammenfassung

Das Thema *Autonomes Fahren* hat sich zu einem immensen Forschungsgebiet entwickelt. Erste große Schritte wurden zwar bereits in den 80er Jahren des letzten Jahrhunderts gemacht (siehe Dickmanns and Zapp [1987]), größere Popularität aber erhielt das Thema erst nach den sogenannten *DARPA Challenges* (*Grand* und *Urban*, siehe Buehler et al. [2007] und Buehler et al. [2009]). Heutzutage liest man regelmäßig Ankündigungen, dass der nächste *Highway Pilot* oder das nächste *Autonome Taxi* nur noch wenige Jahre (oder Monate) entfernt sei. Genauso regelmäßig werden diese Ankündigungen der großen Technologiekonzerne und Autobauer dann allerdings wieder verschoben, was zeigt, dass es noch genug ungelöste Probleme gibt. In dieser Arbeit wird das Thema *Autonomes Fahren im unstrukturierten Gelände*, also zum Beispiel auf Waldwegen oder in beengten städtischen Szenarien, behandelt. Ziel dieser Arbeit ist es, in diesen anspruchsvollen Fällen das Fahrverhalten zu verbessern, insbesondere was den Komfort und die Präzision angeht.

Da jede Kette nur so stark ist wie ihr schwächstes Glied —in diesem Fall ist die Wirkkette von der Wahrnehmung über Fusion und Planung zur Aktuation gemeint— wird zuerst die Systemarchitektur betrachtet. Die vorgeschlagene neue Architektur gewährleistet einen besseren Informationsfluss zu den hinteren Teilen der Wirkkette, was nötig ist um die Leistung dieser, und somit zugleich die Gesamtleistung, zu verbessern. Zusätzlich vereinfacht die vorgeschlagene Architektur auch den Entwicklungsprozess und die Wartbarkeit der Software. An einfachen praktischen Beispielen wird gezeigt, dass sich durch die getroffenen Maßnahmen das Fahrverhalten selbst unter Verwendung der gleichen Regler verbessert.

Der eine thematische Schwerpunkt dieser Arbeit ist die Bewegungsplanung. Das Rückrad des TAS Planungsframeworks bildet eine erweiterte Version des *hybrid-state A** Algorithmus (siehe Dolgov et al. [2008]), also eine durch eine Heuristik geführte Suche. Des Weiteren wird die klassische Pfad-Geschwindigkeits-Dekomposition (siehe Kant and Zucker [1986]) genutzt und erweitert indem sie mit modernen Trajektorienplanungs- und Optimierungsmethoden zusammengebracht wird. Für die Freifahrt, d. h. ohne Führungsfahrzeug, wird der Stand der Technik in der Kollisionsprüfung übertroffen. Für die Folgefahrt, d. h. ein Fahrzeug folgt einem anderen, wird die TAS-eigene, preisgekrönte Performance verbessert. Für Mehrfahrzeugkonvois wird die Grenzen des Machbaren unter erschwerten Bedingungen ausgelotet: Ohne Querführung durch etwa Straßenmarkierungen und unterstützt durch eine Interfahrzeugkommunikation mit einer geteilten Bandbreite von nur 19.2 $\frac{\text{kbit}}{\text{s}}$.

Der andere thematische Schwerpunkt ist die Fahrdynamikregelung. Es wird eine Architektur vorgestellt, die aus einem fahrzeugspezifischen *low-level* und einem fahrzeugunspezifischen *high-level* Teil aufgebaut ist. Während Ersterer hautpsächlich der Vollständigkeit halber beschrieben wird, beinhaltet Zweiterer mehrere Neuerungen. Der primäre Arbeitsmodus ist die Trajektorienfolgeregelung. Hierbei wird auf die Wichtigkeit einer Lokalisierung des Fahrzeugs auf der Trajektorie sowohl im Raum als auch der Zeit eingegangen. Diese erlaubt es, zusammen mit den Garantien zur Glattheit, Kontinuität usw. der Trajektorienplanung aus den vorderen Kapiteln, Totzeiten effizient zu kompensieren. Des Weiteren werden Erweiterungen zur klassischen Trajektorienfolgeregelung diskutiert, insbesondere für Folgefahrt-Szenarien. Schließlich wird ein Ansatz beschreiben, der es ermöglicht ein normalgroßes (Elektro-) Fahrzeug in Outdoor-Konditionen mit Sub-Zentimeter-Präzision zu positionieren.

Contents

1 Introduction

The significance of the topic of autonomous driving in our time has been stressed numerous times (e. g., see Bertoncello and Wee [2015], Maurer et al. [2016], Trommer et al. [2016], Hörl et al. [2016], Duarte and Ratti [2018]). In order for cars to drive autonomously, a considerable number of challenges are still to be overcome. Topics range from sensor-data interpretation through scene understanding and decision-making to actually taking action. This work focuses on control-related aspects, i. e., the part that actually makes the car drive and hence interact with its environment.

The presented contributions were tested extensively under real-world conditions on the test vehicles MuCAR-3 (Munich Cognitive Autonomous Robot Car 3rd Generation) and MuCAR-4 (see Figure 1.1) of the Institute for Autonomous Systems Technology (TAS) as well as cars from cooperations with industrial partners. The research focus at TAS is on driving in unstructured environments, i. e., no scenarios with clear structures such as line markings and very foreseeable road geometries, like on the highway, but rather gravel roads or even off-road scenarios. These scenarios have a completely different set of requirements than the aforementioned highway driving; for instance, both the range of the available information on the environment and, consequently, also the driven speeds are much smaller.

This chapter is structured as follows:
First, an overview of system capabilities when this work started in 2012 is given in Section 1.1. Second, the state of the art in autonomous (off-road) driving is presented in Section 1.2. Then, a short summary of the author's original contributions is given in Section 1.3. Finally, the further structure of this work is laid out in Section 1.4.

Figure 1.1: MuCAR-4 and MuCAR-3

1.1 State of the MuCAR

The MuCAR (Munich Cognitive Autonomous Robot Car) has a long history in the field of autonomous-driving research. Its first two iterations, the so-called test vehicle for autonomous mobility and computer vision, *german: Versuchsfahrzeug für autonome Mobilität und Rechnersehen* (VaMoRs) and VaMoRs-PKW (PKW), performed state-of-the-art feats, such as driving most of the 1000 km from Munich to Copenhagen in normal traffic at speeds up to 180 km/h almost autonomously[1] already in the 1990s (see Dickmanns and Graefe [1988]). Since 2006, its 3rd iteration —a VW Touareg I[2] — is the main test vehicle at TAS. The 4th iteration, a VW Tiguan I, started as a lead vehicle for convoy scenarios but got robotized over time.

Both test vehicles feature full drive-by-wire capabilities, i. e., steering, accelerator, brake and gearshift are actuated and controllable by the computer hardware and thus software. Additionally, they are equipped with a wide array of sensors, as is laid out in more detail in Section 2.2. The autonomous capabilities of MuCAR-3 were demonstrated during international competitions, namely the ELROB (European Land Robot Trial) in most years since 2007 and the EURATHLON in 2013[3]. Additionally, TAS was part of Team AnnieWAY, which competed in the DARPA (Defense Advanced Research Projects Agency) Urban Challenge 2007 Finals (see Kammel et al. [2008]).

1.1.1 The 4D Approach

Research at TAS is strongly influenced by Ernst D. Dickmanns' *4D Approach*[4]. In very short, the idea is to use a world model (3 dimensions) plus knowledge of one's own and other objects' dynamics over time (the 4th dimension). This dynamic model allows for predicting one's environment and hence to focus one's attention to the relevant part of the scene. For instance, if a road marker was at a certain place one camera frame ago, it is searched for at that very place again — modified by the known motion of the camera in the scene (see Figure 1.2).

This paradigm was extensively used for the perception of the MuCARs. However, the planning and/or control parts of the software were designed to be *reactive*. A focus of this work is to incorporate *predictive*/*proactive* structures also in those areas. This is discussed in more detail in Section 2.4.

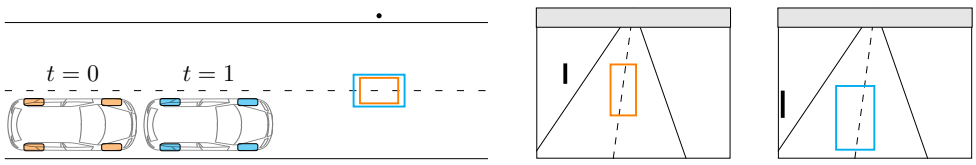


Figure 1.2:
A simple 4D application: using the known egomotion and thus camera movement, features can be found faster in the next frame since their relative position is predicted.

[1] Maneuver such as *overtaking* needed approval by the safety drivers

[2] initially modeled after the DARPA Grand Challenge 2005 Winner "Stanley" (see Thrun et al. [2006])

[3] Results for both can be found at `www.elrob.org`

[4] For a comprehensive overview see Dickmanns [2007]

1.1.2 Leader–Follower Capabilities

A leader–follower system describes two entities, for instance: cars. The second follows the first, hence they are called follower and leader, respectively. Usually, the leader is steered manually while the follower operates autonomously.

The ability to follow a lead vehicle or human guide can be crucial in very unstructured environments. On the one hand, potential dangers may not be detectable by sensors, e. g., quicksands — so-called false negatives. On the other hand, dangers may be detected, where there are non, e. g., high grass as an obstacle — so-called false positives. Thus, the capability of following a leader was researched at TAS (see Manz [2013], later continued by Fries [2019]). Here, the focus was on tracking a lead vehicle, but it was extended for human guidance (see Ebert et al. [2018]).

As described in Manz [2013], tracked positions of a lead vehicle are saved and used in a second step, where a clothoid arc is fitted through them. This fit is filtered using an Extended Kalman Filter (EKF) where the positions' spatial uncertainty rises with their respective age. The state and hence result of the filter is a so-called *lane*.

Definition 1.1 (Lane)**.** Let the 4-tuple: curvature (c), (spatial) change in curvature[5] ($\mathring{c}$), displacement (Δy) and yaw error[6] ($\Delta\psi$) be called a *lane*.

This lane was fed to the motion control which then tried to minimize both the yaw error and the displacement. From a control point of view, this has the following drawbacks which are overcome in this work:

- The filter runs time-triggered, i. e., due to the filter update with the same measurements, the displacement and yaw error change even when standing. This implies that the lane is not earth-fixed but moves[7].
- The clothoid's change in curvature $\mathring{c}$ is a filter noise term and prone to changing erratically over time. Thus, $\mathring{c}$ could not be used as a feedforward for the steering rate as in earlier works (cf. Dickmanns et al. [1990]).
- Using a single clothoid does not allow for representing maneuvers with a change in the steering rate, e. g., s-curves.

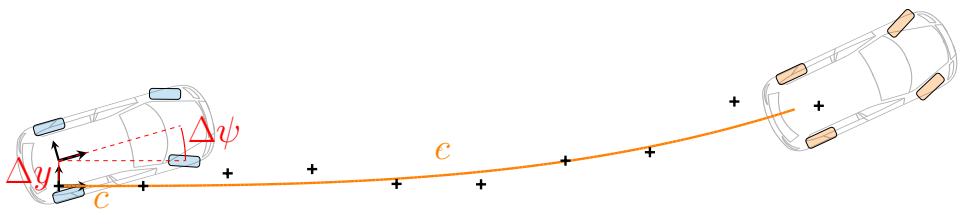


Figure 1.3:
Clothoid fit (orange) of given pose estimates (black) and derived errors (red). Dimensions are not to scale.

[5] Change in curvature over the length, and not over time. I. e., $\mathring{c} \stackrel{\Delta}{=} \frac{\partial c}{\partial l}$, not $\dot{c} = \frac{\partial c}{\partial t}$.

[6] It incorporates both the track to be followed and the momentary error of the ego vehicle.

[7] Old measurements with growing uncertainty become less relevant and hence the filter slowly converges to the current observation.

Note that these issues arise primarily when navigating unstructured environments, where precise maneuvering is of the essence. When following a highway with its very gradual changes in the road geometry and not doing safety-critical maneuvers, this is a valid approach.

1.1.3 Navigation Capabilities

In order to navigate in mostly unstructured environments, a dirt-track and a crossroad tracker were developed (see Manz [2013], later continued by Bayerl and Wuensche [2014], Bayerl et al. [2015]). Multiple features were used to detect a drivable track. For instance, the color saturation from the camera images turned out to be very valuable. Additionally, the "color" of the ground on which the ego vehicle is momentarily driving on was evaluated. Successfully driving on it is a strong indication that further driving on it is beneficial. This approach had the added benefit of working in different scenarios, e. g., a brown track is to be followed in snowy conditions, while a gray gravel track is to be preferred over brown mud next to it. In addition to camera information, also LiDAR (Light Detection And Ranging) data was used, e. g., for detecting slopes. See Figure 1.4 for some impressions.

Once a path was detected, positions along its center were extracted. These were used as virtual lead-vehicle poses to generate the *lane*, as described in Section 1.1.2.

A drawback of this approach was that there is no active obstacle avoidance, but rather only lane centering. Nevertheless, it enabled following dirt roads smoothly, even if there is, for instance, high grass in its center. I. e., it can be used as an SAE L2 assistance system (see Appendix A.1), where the driver constantly has to monitor the path ahead. Yet, in order to drive autonomously, this is not a viable approach and led to the development of the motion-planning framework described in this work.

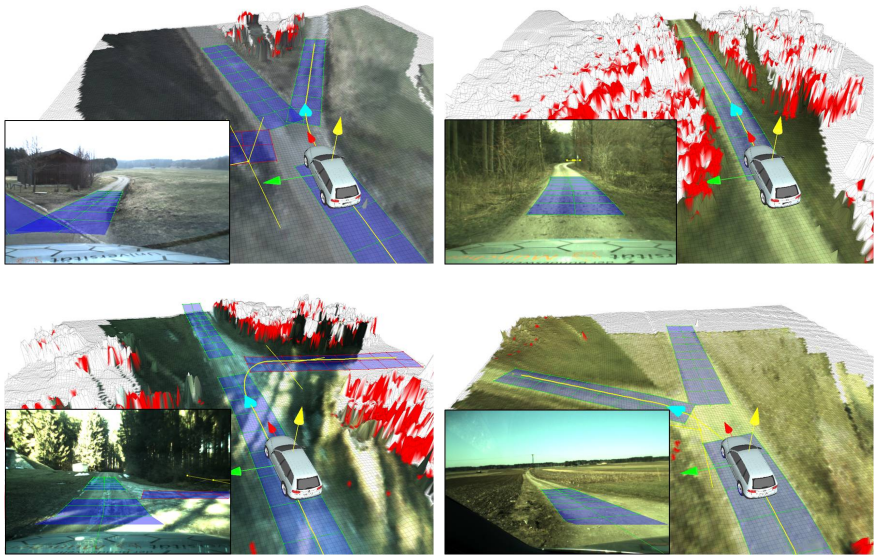


Figure 1.4:
Examples for the road and crossroad tracking. Pictures taken from Manz et al. [2011].

1.1.4 End-to-end Capabilities

The so-called *tentacles* navigation (see von Hundelshausen et al. [2008], Himmelsbach et al. [2011][8]) is a reactive biologically inspired approach where a batch of motion primitives is assessed periodically. In the initial publication, these were fixed steering angles, driven for a limited amount of time. Later these were replaced by steering rates, see Figure 1.5.

The tentacles are ranked based on a multitude of features. Occupancy, slopes and closeness to a target path (from global route planning), among others, are evaluated in the grid-based environment representation, see Jaspers et al. [2017]. Initially, also features in the camera image, where the tentacles were back-projected into, were utilized (see Manz et al. [2009]). Much like in a receding-horizon framework, the best tentacle (which corresponds to a certain steering/acceleration change) is executed until the next cycle. I. e., only a tiny part of the tentacle is actually driven before new information is considered. Note that this approach is highly parallelizable.

Because sensor feedback is directly mapped to control outputs this is an early, very deterministic, i. e., explainable, way of implementing end-to-end navigation in real-time. Complex behavior can arise from iteration: executing a fraction of the best tentacle and then generating and evaluating a new set of tentacles, where again, a fraction of the best one is executed etc. However, behaviors such as executing a three-point turn, or parking the ego vehicle require careful guidance by a higher-level planning instance (see Luettel et al. [2011]). Additionally, oscillatory behavior may arise from noisy input data. The capabilities of the tentacles approach is exceeded when there is delay in the control chain, a continuous reference is required for control or complex maneuvers need to be planned — which is why the motion-planning and -control framework in this work was developed.

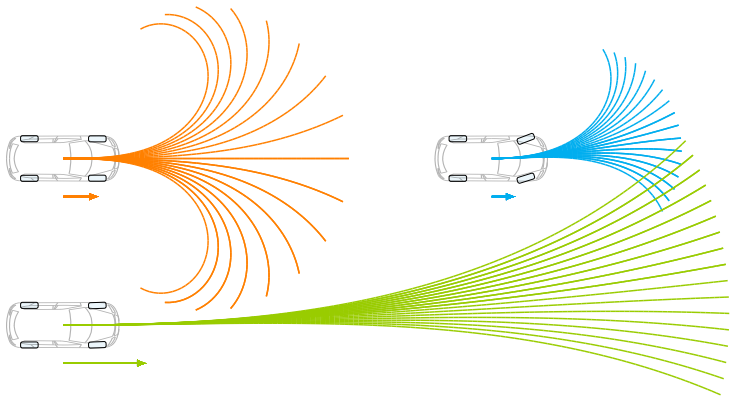


Figure 1.5:
Simplified illustration of tentacle sets for different speeds and start steering angles. Each line represents a simplistic possible future trajectory, dependent on the current state. Note that due to, e. g., steering-rate or lateral acceleration constraints, the possible change in curvature becomes smaller with higher speeds, but the tentacles become longer.

[8] This approach was well received by the scientific community and numerous modifications/extensions were published, making it one the most-cited works at TAS at the time of writing.

1.1.5 Summary and Shortcomings

All in all, the navigation capabilities of MuCAR were tailored for simple behaviors and focused on being reactive. Each behavior featured a separate control chain, which led to a robust set of capabilities in many real-world scenarios. The contribution of a fall-back layer for Team AnnieWAY (DARPA Urban Challenge) and TAS's own results in a number of international competitions proved the validity of this concept.

However, while especially the monocular vision and LiDAR perception performance was state of the art, the later parts of the control chain had a lot of unused potential. Additionally, while the different reactive abilities were well-performing in the respective niches they were built to excel in, the arbitration between those abilities proved to be non-trivial[9]. Once they need to be combined, oscillatory behavior, inconsistencies, and in general the curse of dimensionality —due to the consequent combinatorial explosion of transitions— are the resulting impediments.

Moreover, the later parts of the control chain were impeded by the lack of foresight. Given the time constants and delays involved in autonomous driving, there is only so much any algorithm (or its implementation in a software component) can do to achieve reasonable performance. The resulting decision is then usually, either to follow the reference precisely, or to drive smoothly.

The stated goal of this work is to facilitate both: smooth *and* precise driving. Hence, the architecture was to be overhauled as a first step. Further, a motion-planning framework needed to be created, capable of planning at least a limited time ahead. Only then, an overall increase in general performance was achievable.

Before elaborating how the mentioned shortcomings were addressed, the state of the art at the time of writing is examined in the upcoming Section 1.2.

[9] Note that different behaviors were generated by different modules that were not aware of each other's existence.

1.2 State of the Art

Autonomous-driving research has a long history, especially at the TAS. However, the global development made a huge leap after the DARPA created a series of well funded challenges in the early 2000s. There, sizable amounts of money were rewarded to the teams that could fulfill certain driving tasks fastest.

It started with the DARPA Grand Challenge 2004, where one million dollar was awarded for driving a 150 miles dessert track. No team made it even 5 percent of that distance. The second installment, the DARPA Grand Challenge 2005, featured a two-million dollar prize for a slightly shorter track (132 miles) and was actually finished by 5 teams.

The winning team —consisting primarily of Stanford, Volkswagen and Intel Research— describes their approach in Thrun et al. [2006]. In its over 30 pages, a thorough overview from their design philosophy "*treat autonomous navigation as a software problem*" through system architecture down to implementation details, e. g., of their Unscented Kalman filter (UKF)-based ego-pose estimation, is given. Even though many points are specific to the challenge's track, rules and data formats, a lot can be and was learned from this. At the time of writing this thesis, the paper has over 2500 citations already.

Two years later, in 2007, the DARPA Urban Challenge was held. Here, the participants had to obey traffic rules and interact with other traffic on a 60 miles course on an air-force base.

A collection of the finalists teams' learnings can be found in Buehler et al. [2009]. It includes papers describing the teams' approaches (e. g., for the podium teams: Boss, Junior and Odin (see Urmson et al. [2008], Montemerlo et al. [2008], Bacha et al. [2008], respectively)) akin to the Stanley paper mentioned above. Additionally, special issue papers like an investigation of the MIT–Cornell collision (see Fletcher et al. [2009]) or technical details like TAS' own tentacles approach (see von Hundelshausen et al. [2008]) are included. All in all it contains over 600 pages of insightful first experience in urban autonomous driving, which can today be assumed basic knowledge to autonomous driving researchers.

While conducting this work, the Bertha Benz Memorial route drive by Daimler and the Karlsruhe Institute for Technology (KIT) (see Ziegler et al. [2014b]) was a public event that spawned numerous publications. Also worth mentioning was the so-called Grand Cooperative Driving Challenge in 2016, where the participants had to drive in a platooning highway setup and their influence on the performance of the whole platoon was measured and rewarded.

But of course, there were also many noteworthy publications independent of public events. There is a dedicated state-of-the-art section for each of the major three chapters of this work —system architecture, motion planning and motion control— see Sections 2.1, 3.1 and 4.1, respectively.

1.3 Contributions

The main goal of this thesis is to work towards smoother as well as more precise autonomous driving. This is achieved by improving the whole system's performance at multiple bottlenecks rather than developing, e. g., one intricate novel algorithm. The original contributions in this work are the following:

- Restructuring of information flow from perception to control (see Chapter 2)
- Improvement and post-processing of trajectories, namely:
  - faster collision checks (see Section 3.4)
  - smoother speed profile generation (see Section 3.5) and
  - a novel spatiotemporal two-step speed profile generation algorithm (see Section 3.7)
- Safer ACC (see Section 4.3.4)
- Stable Platooning[10] control (see Sections 3.6 and 4.3.4)
- Improved delay handling (see Section 4.3.2)
- Unprecedented precise positioning for autonomous vehicles (see Section 4.3.5)

In accordance with the *Doctoral Degree Regulations of the Bundeswehr University, Munich (February 2012)*, parts of this thesis were pre-published in

- workshops (see Heinrich and Wuensche [2017]),
- peer-reviewed conference papers (see Heinrich et al. [2016, 2017, 2018a,b], Fassbender et al. [2016b, 2017b])
- and journals (see Heinrich et al. [2018c]).

1.4 Structure

This work is structured as follows:

- First, the system architecture is described in Chapter 2, where the former and new architecture are compared. The improvements are shown quantitatively using simple experiments.
- Second, a short overview of the motion-planning framework is given, and it is shown how its speed and quality were improved in Chapter 3. There, the focus lies on improving the temporal components of a spatially given path.
- Third, in Chapter 4, a trajectory controller is presented and scenario-driven solutions for vehicle following, platooning and positioning are presented. An additional key point here is the handling of delays in the control chain.
- Finally, the work is concluded by summing up the results, stating open issues and giving possible future directions in Chapter 5.

[10] While Adaptive Cruise Control (ACC) considers only the immediate lead vehicle, the platooning case considers multiple vehicles following each other.

2 System Architecture

System architecture comprises both the hardware and the software aspects of a system and considers especially the interplay between them. The hardware components are the sensors used to perceive the environment, the computation hardware that runs the software, and the actuators, which are used to interact with the environment. Software components in robotics are often divided into perception, planning and control. The used terminology is visualized in Figure 2.1.

All live information available to the system is perceived by sensors. The raw data provided differs vastly between the sensors. For instance, a camera perceives accumulated colored light while a LiDAR (Light Detection And Ranging) perceives runtime from its emitted laser beams. It is the task of the perception modules to bring this information into an internal representation which can be reasoned and acted upon. In turn, this internal representation, or *world model*, helps, e. g., tracking modules, see Section 1.1.

The system interacts with the environment using its actuators. This interaction comprises not only the manipulation of other objects but also —and in this work primarily— the movement of the system within the environment. This movement will cause parts of the environment to react (e. g., other traffic participants) and also influence what can be perceived by the sensors. I. e., the complete feedback loop is to be considered for any action taken.

The ultimate goal when designing a system architecture is to provide a framework to streamline the interaction between all modules. This is done by carefully specifying interfaces and controlling the information flow.

In the following, a short overview of the state of the art and the existing hardware architecture is given in Sections 2.1 and 2.2, respectively. Then, the former system architecture and its successor are presented and compared in Sections 2.3 to 2.5.

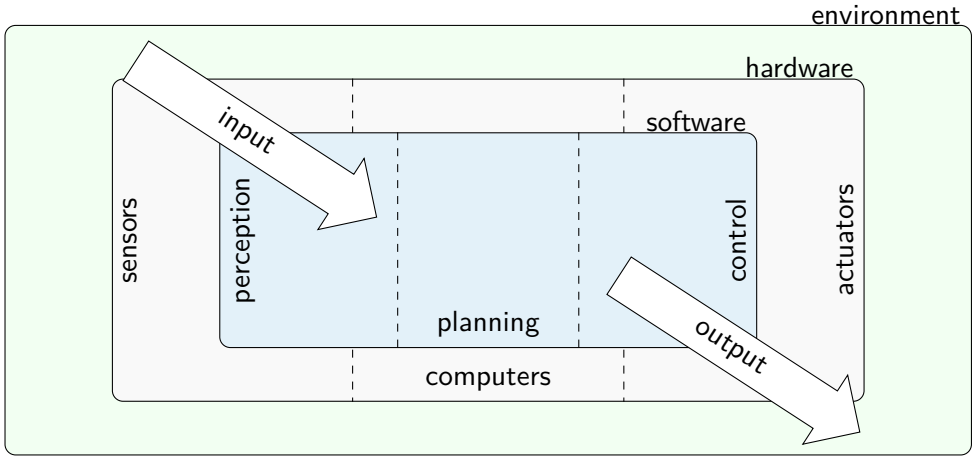


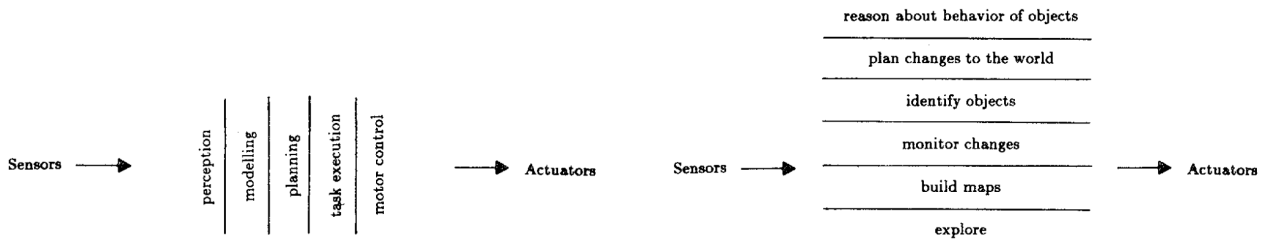
Figure 2.1: System architecture terminology

2.1 State of the Art

Already Thorpe et al. [1991] found that architectures *trying to encompass all possible systems [...] lose their prescriptive power* and that their main contribution is to be *descriptive: providing a common vocabulary in which to discuss the differences between architectures*. Accordingly, there is no agreed-upon state of the art in robotics system architecture.

In general, there are two classes of system architecture in robotics. The classic *sense→plan→act* methodology and the reactive *sense↔act* pattern, as proposed by Brooks [1986]. Where classically, there would be *components*, like a perception module, a fusion module, etc., in the latter there are *functions* (see Figure 2.2). For instance, an evasion function would directly map sensor input to an actuator reaction. This would be a lower layer to a more goal oriented other steering function. Prevalence handling of different functions is called *subsumption*. The advantage of Brook's approach is that each function is easier to understand/maintain/implement and potentially faster due to not being blocked by (for this function useless) other activities — of course, assuming parallel processing of all functions, a flawless arbitration between all functions and no counter-acting functions (due to different size of scope). Note that this function-based architecture in its pure form also does not utilize any internal state, i. e., no memory of the past.

Both classes have their benefits, and no modern architecture purely follows one or the other. The "Stanley" architecture (see Thrun et al. [2006]), for instance, refers to Gat et al. [1998]. There, the Brooks' subsumption architecture is critically evaluated and the *three-layer architecture* is proposed. The layers are: *a reactive feedback control mechanism, a reactive plan execution mechanism, and a mechanism for performing time-consuming deliberate computations*. Stanley's architecture is said to use both the feedback-control and the reactive-plan-execution mechanism, but not the long-term planning. The first part aims at generating stateless Primitive Behaviors (e. g., follow the wall) that may fail, but have to be able to detect their failure, i. e., *fail cognizantly*. The second part, also called a Sequencer, evaluates and chooses the best Primitive Behavior. The third part, not present in Stanley, would predict the future. Given how simple the task was (follow a road), it was superfluous.



(a) The classic control chain

(b) Function-based decomposition

Figure 2.2: Original depictions taken from Brooks [1986].

Taking only the first two parts still results in a Brooks-like architecture. However, in its implementation there are no single functions, but components (modules) that, for instance, try to identify a road given multiple sensors, or a path planner, that consumes input from various environment modeling modules, which is indeed the classic way in robotics. To sum it up, it is hard to generalize an actual working architecture in purely theoretical terms.

Communication is the key to architecture. Note that, much like in the modern Robot Operating System (ROS), all modules in Stanley already worked on a publish–subscribe base. The Urban-Challenge-winning "Boss" architecture (see Urmson et al. [2008], Baker and Dolan [2008]) incorporates four major areas: Perception, Mission Planning, Behavioral Execution and Motion Planning. Externally, the modules communicate on a publish–subscribe base, but internally, a so-called observer pattern is used, where multiple sub functions can observe a subject and react to changes. The authors themselves write that *the inability to fully understand the complex notification patterns among the Observers led to many unexpected and erroneous results*. Nevertheless, their architecture was able to outperform the rest of the competitors.

Architecture is always specific to the system's use case. The Bertha Benz (see Ziegler et al. [2014b]) architecture has four major components: Perception (including Fusion), Motion Planning (including Behavior Generation), Trajectory Control and additionally, Localization. The addition of the latter as a major building block is due to the system's use case: driving exactly one well-known route with a high dependency on high-definition map data of that route. Note that additionally, a Reactive Layer, namely the vehicle's AEB (Automatic Emergency Brake) system, was active. I. e., both architecture classes, the component- and the function-based one, are utilized.

Another more recent stream tries to eliminate the need to understand the system's internal interaction altogether: After the advent of deep neural nets for specific components, like perception (e. g., see Redmon et al. [2016]), there is now the attempt to solve the whole chain with so-called *end-to-end* networks (e. g., see Grigorescu et al. [2019]).

Additionally, the industrialization of the autonomous-driving research is under way. Thus, the arduous part of making the approaches safely and reliably work together —not only in specific test scenarios but on the road in real traffic situations— is handed over to the industry, which usually does not publish their trade secrets. Nevertheless, due to the topics complexity and importance as well as the complex supply chains in the automotive industry, there are ongoing efforts to establish standards and even to open-source them.

This is done in the so-called AUTOSAR (AUTomotive Open System ARchitecture), which was launched in 2003 (see Fennel et al. [2006]). Note that this gives a framework to build one's architecture in, not the architecture itself. The early versions are nowadays called *classic* version. In 2017, the so-called *adaptive* version was released (see Reichart and Asmus [2021]). The former is a very rigid standard for resource-limited microcontroller that are statically configured and work on bare metal in C. In contrast, the adaptive version allows for more flexible load balancing on POSIX basis in modern C++.

2.2 Hardware

This section gives a brief overview of the platform (i. e., the vehicles), the sensors, the computers and the actuators. A condensed into signal-flow schematic for both main test vehicles can be found in Appendix A.6.

2.2.1 Platforms

The vehicles used to conduct the majority of the tests in this work were MuCAR-3 and MuCAR-4 (see Figure 1.1). The former is a VW Touareg I that was initially modeled after the DARPA (Defense Advanced Research Projects Agency) winner "Stanley" (see Thrun et al. [2006]) and incrementally improved ever since. The latter is a VW Tiguan I which started as a lead vehicle, but slowly became fully robotized in the later years. Results in Section 4.3.5 were generated using a Streetscooter Work Box (B14) (see Figure 2.4). Parts of the software stack also ran on a heavy six-wheel-drive Rheinmetall MAN HX 58 truck, the so-called TULF (Technologieträger Unbemanntes Landfahrzeug) (see Fassbender et al. [2014a]).

2.2.2 Sensors

Both main vehicles feature

- a high-definition 64-beam LiDAR sensor which provides 360° field of view,
- a MarVEye-8: an actuated camera platform for actively focusing their attention; for a depiction, see Figure 2.3 for more details, see Unterholzner [2016], and
- a high-precision inertial navigation system (INS) which utilizes Real time kinematic (RTK) and the wheel-pulse-counter signal for determining a ground-truth-level egomotion estimation

Exclusive to MuCAR-3 are

- an automotive LiDAR (front),
- an automotive Radar (Radio Detection And Ranging) (front),
- a low-light camera, mounted behind the windshield, as well as
- a FIR (Far-InfraRed) camera, mounted on the roof.

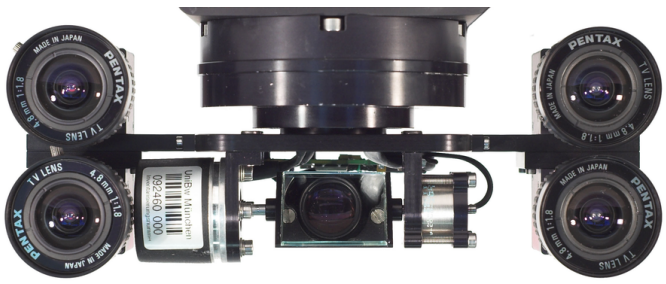


Figure 2.3:
MarVEye-8 camera platform which can rotate around the z-axis and, additionally, has a downward-facing camera (inside the vertical tube) which can rapidly compensate for vehicle-chassis pitch motion by rotating its mirror around the y-axis.

Figure 2.4: Streetscooter Work Box (B14) (taken from Heinrich et al. [2018c])

Exclusive to MuCAR-4 are multiple stereo systems, including a unique triple-stereo setup (see Kallwies and Wuensche [2018], Kallwies et al. [2020]).

The B14 featured the same INS setup and

- three LiDARs ($2\times32 + 1\times16$ beams) instead of one 64-beam sensor plus
- a side-mounted camera for target tracking as the singular video sensor

2.2.3 Computers

In terms of computer hardware, all vehicles feature

- the so-called *high-level system*, a state-of-the-art personal computer where sensor-data processing and decision-making takes place and
- the so-called *low-level system*, one (MuCAR-4, B14) or two (MuCAR-3) real-time computer(s) which handle(s) hardware i/o, safety and low-level control.

The software running on the *high-level system* is not vehicle-specific, apart from some run-time parameters. I. e., exactly the same code runs on all vehicles, which greatly facilitates maintenance. The *low-level system* runs vehicle- and actuator-specific code, e. g., incorporates the engine's characteristic performance map. Consequently, the interface between high- and low-level system is designed to be a universal one.

2.2.4 Actuators

All vehicles feature complete drive-by-wire actuation (steering, accelerator, brake, drive-mode). It should be noted that apart from the MuCAR-4 steering, no drive-by-wire capabilities are inherently available from the platforms themselves. The different functionalities were implemented by means of custom hardware such as custom throttle circuit boards, linear drives that move the drive-mode selector or even a chain drive for the MuCAR-3 steering wheel. This implies deficits in the low-level control performance when compared to modern test vehicles with direct interfaces to power steering and motor/brake torque. Nevertheless, unprecedented high-precision control was achieved, as will be shown later on.

2.3 Former Architecture

As noted in Section 1.1, MuCAR-3 was a successful international competitor. Its initial architecture was postulated in Goebl et al. [2008], where the CoTeSys (Cognition for Technical Systems) vehicle (Audi Q7) was used. Yet, the architecture used during all except the latest competitions evolved from Lüttel [2008]. Since there is no publicly available record —and it is important to understand the initial conditions, when this thesis started— an overview is published here. This section is split into the three main architectural building blocks: perception, planning and control.

2.3.1 Perception

Perception has always been a main focus at Institute for Autonomous Systems Technology (TAS). The perception of the environment was handled by several modules (for an overview, see Figure 2.5):

Egomotion Estimation (EME) The *egomotion estimation* relied on the inertial measurement unit (IMU) sensors and additionally provided a dead-reckoning estimate. No corrections by Global Navigation Satellite System (GNSS) are used in a dead-reckoning system, and thus a continuous position is provided. Note that there is no correlation between the dead-reckoning estimate and the actual position of the ego vehicle in the world.

Environment Mapping The *environment mapping* used a multi-layered local grid representation. The layers were generated from the LiDAR point cloud, which was colored by camera images, and accumulated over time, utilizing the EME. Layers hold, for instance, obstacle probabilities, slopes or color information. For more details, e. g., on how this is done memory-efficiently, see Jaspers et al. [2017].

Road Estimation Based on the colored grid, roads were detected. Here, roads mean one-lane sub-urban roads, gravel or dirt roads, or even tracks in fields or in the forest. A main feature was the color back-projection (the road most likely looks like the terrain below the ego vehicle) and the search in Hue saturation value (HSV)-space (tracks tend to exhibit less saturation). For more details, e. g., on its extension to night-drives, see Bayerl and Wuensche [2014], Bayerl et al. [2015].

Lead-Vehicle Tracking The *lead-vehicle tracking* primarily used a camera on the MarVEye-8 platform (though this was later extended also to using thermal images, see Fries and Wuensche [2015]) to track a known lead vehicle. It used a hand-crafted feature model (later also automatically generated, see Fries and Wuensche [2016]) as target for the particle-filter-based tracking. Here, according to the 4D approach, the future lead-vehicle position was predicted given the last tracking estimate and the Egomotion Estimate (EME) for determining where to sample new particles. A simple, i. e., not accumulated, LiDAR occupancy grid was used as additional input. Further, the INS data of the lead vehicle could be used, which was primarily done for evaluation purposes.

Multi-Object Tracking The LiDAR-based *multi-object tracking* (see Himmelsbach et al. [2010], Himmelsbach and Wuensche [2012]) used a bottom-up top-down approach to track multiple moving objects using the 64-beam laser scanner mounted on MuCAR's roof. The objects were predicted according to the 4D approach, utilizing the EME and different motion models for, e. g., kinematic bicycle model for vehicles and constant velocity for pedestrians.

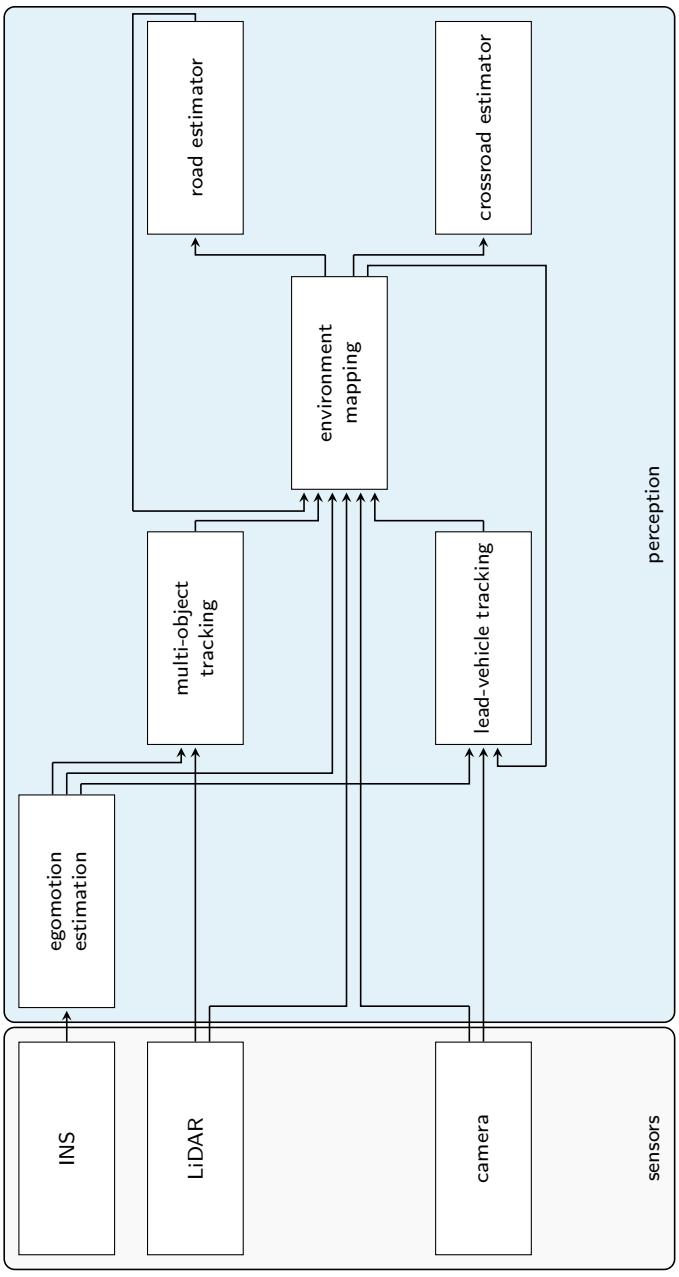


Figure 2.5:
Overview of signal flow between perception modules in the former architecture (simplified). For the sake of clarity, all outgoing signals are omitted. Note the central role of the environment mapping which is enriched by the modules reading it as an input as well.

2.3.2 Planning

The interface between planning and perception was the so-called *pathgeneration* module. It accumulated the estimates from the tracking modules: for objects their respective bounding-box centers and for roads their centers in a predefined look-ahead distance. Through these estimates, a sequential Kalman Filter (KF) fitted a clothoid arc which was used as the *lane* object (see Definition 1.1). As mentioned in Section 1.1.2, this is a suboptimal interface.

At the core of the former planning architecture stood a hierarchical finite state machine. A state machine is an abstract representation of conditional behavior. Each behavior is represented by a so-called *state*. In a hierarchical state machine, these states can also be nested. One of the states is the active state. Dependent on certain conditions or inputs, the *active state* may change. A change of active state is called a *transition*. For a simple illustration, see Figure 2.6.

To transition from one state to another, certain conditions have to be met. A condition can be either intrinsic (e. g., after 5 s transition to next state) or extrinsic (e. g., red button pushed). States can model *behaviors*, i. e., long-term actions like: follow the lane until further notice. They can also model parts of those actions, for instance: the initial gearshift to drive. This resulted in 22 states with 62 possible transitions, which are not enumerated here for lack of added value. For more details about the intended behaviors and subtasks, see Luettel et al. [2011].

The former state machine, along with many other components, was parameterized by the so-called *mission handler*. This software module communicates information such as the type of mission, e. g., *follow lane* or *follow vehicle*, and parameters such as the desired lead-vehicle type or the maximum speed. The mission handler was manually configured, either using a Graphical User Interface (GUI) or a mission file.

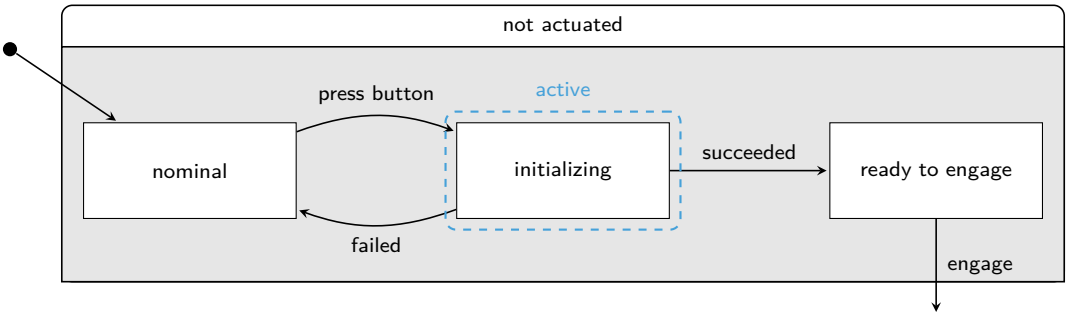


Figure 2.6:
Depiction of a part of a simple hierarchical state machine. The state *not actuated* has several sub-states, all of which share some properties. In this case: 'The autonomous vehicle is not actuated'. There are extrinsic conditions, like *press button*, and intrinsic ones (*failed, succeeded*) leading to state transitions between the substates. When the system is *engaged*, the state machine goes into the *actuated* state (not shown). That state typically has many more sub-states and potentially sub-sub-states with the according transitions, also back to *not actuated*.

The inputs to the state machine were, next to the mission, all perceived objects and lanes. Additionally, there was a tight coupling to the control module, which will be elaborated shortly. Information was processed every 50 ms, i. e., in a time-triggered manner.

The output of the state machine is a chosen *maneuver*, a *lane* (referenced by lane-id) and potentially a *lead vehicle* (referenced by object-id). A maneuver consisted of a lateral and a longitudinal part. Each part then had multiple components, as described in Section 2.3.3. Maneuvers are supposed to represent single abilities that are built from re-usable blocks and, what is more, can be combined to complex behaviors. To exemplify the design idea, consider the following hypothetical overtake behavior:

Example 2.1 (Overtake)**.** The state machine decides that ego vehicle E overtakes a lead vehicle L on a two-lane road. This action is split into three maneuvers:

(A) Change lane from right lane to opposing lane
(B) While not ahead of L, stay on opposing lane
(C) Change lane from opposing lane to right lane

Each maneuver is split into a lateral and a longitudinal component

(A.lat) Change lane from right lane to opposing lane
(A.lon) Stay behind L, match its speed
(B.lat) Stay in opposing lane
(B.lon) Overtake L by exceeding its speed
(C.lat) Change lane from opposing lane to right lane
(C.lon) Stay in front of L, exceed its speed, go to desired speed

Each component is split into a feedforward and a feedback part, for instance

(A.lat.ff) reference steering angle from look-up table based on desired lateral offset
(A.lat.fb) minimize error against reference steering angle feedforward
(A.lon.ff) go to desired speed
(A.lon.fb) keep distance to L
[...]

The state machine monitors the progress of the maneuvers and decides when to transit between them. □

2.3.3 Control

The architecture of the former vehicle control was as follows: There were two separate entities: one for the lateral and one for the longitudinal control. Each ran in a separate thread, making their execution theoretically independent. The idea behind this was that new sensor information can be processed independently, also at different rates. E. g., a radar would communicate the Adaptive Cruise Control (ACC) target for the longitudinal thread and a camera the lane-center deviation for the lateral one. This is the way SAE Level 1–2 (cf. Appendix A.1) systems work today, but it is not suited for higher-level autonomy.

Each thread received the desired *maneuver*. Each maneuver was composed of several building blocks. Input signals were processed sequentially, first by at least one pre-filter, then by any number of feedforward blocks and finally handed to the controller. In Figure 2.7, these blocks are shown simplified as *pre*, *ff* and *fb*, respectively. Typical input signals were a *lane* object for the lateral thread and either a tracked lead vehicle or a stopping distance for the longitudinal thread. The filter chain always started with one that read the required information (as referenced by the maneuver) from the KogniMobil Real Time Data Base (KogMo-RTDB) and converted data. Then, often additional low-pass filters were applied. Typical feedforward and feedback parts are seen in Example 2.1.

During the maneuvers, their respective progress was communicated back to the state machine and thus monitored. Note that the transition between maneuvers was not necessarily smooth since the whole controller was exchanged instantaneously. Further, the switching needed to be very precisely tuned. For instance, a three-point turn behavior required switching between forward and backwards maneuvers at the respective stop points. If the stopping controller would hold some centimeters before the stopping point and the progress was hence not complete, still a switch should occur — yet, if the controller was still approaching slowly, the maneuver should not switch, as this would result in very uncomfortable behavior.

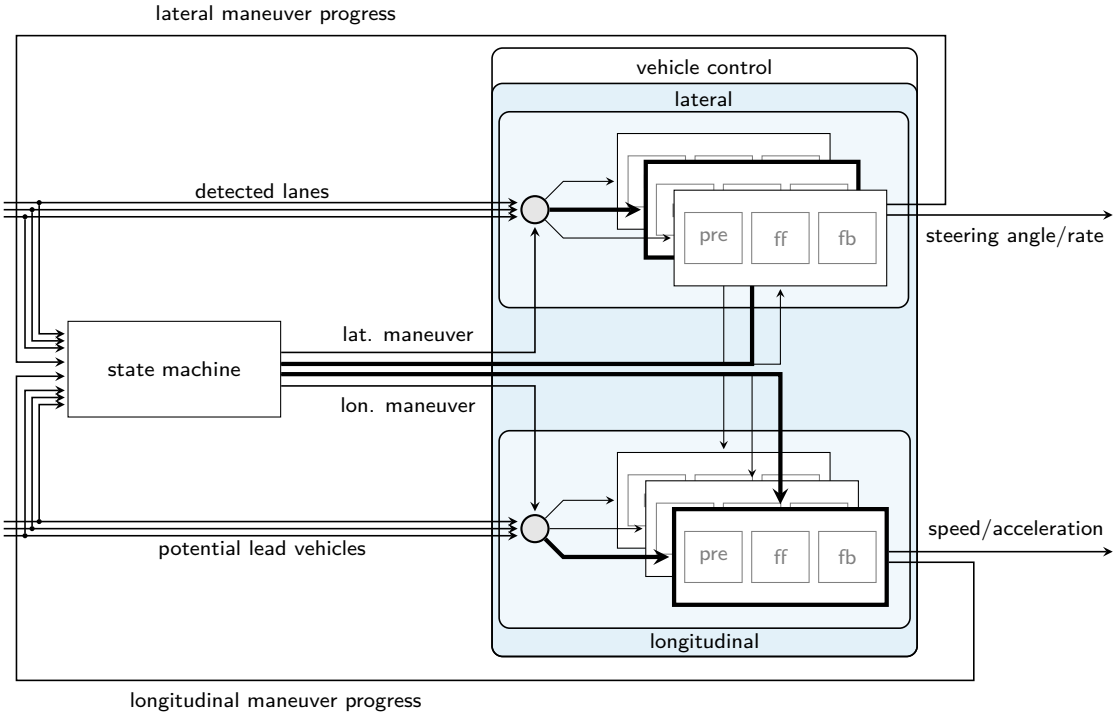


Figure 2.7:
Overview of signal flow between control modules in the former architecture (simplified). The lateral and longitudinal thread are controlled by the state machine. This includes the selection of the used maneuver and its respective input as well as parameterization with reference values, e. g., stopping distances.

2.4 New Architecture

In this section, the new system architecture is described. Like the previous section, this one is split into the three main architectural building blocks: perception, planning and control. In each, many lessons learned from the last years of applied research are implemented.

A main learning was that working on each module, one needs to have the whole functional chain in mind since every small change potentially has major implications for the whole system's behavior. E. g., an altered perception influences all following modules, which in turn —through the altered decision-making and resulting control— then change what can be perceived due to the vehicle moving differently.

A further learning was that maintainability is an often overlooked topic. Due to the ever-increasing complexity of the system (more sensors, more computers,...in general: more communication), it is paramount to ensure that the signal flow is clear (e. g., regarding re-use of already filtered information) and that dependencies are well understood (e. g., if one sensor breaks down, which modules can still run).

While this safety aspect is crucial for real Level 3+ systems (see Appendix A.1), in a development vehicle, where state-of-the-art algorithms are tested and tuned, there is always a safety driver as well as an operator present. Thus, the aspects of degeneration, fail-safe/fail-operational architecture etc. is not addressed here. Rather, the focus is on a way to facilitate the development process by added transparency.

The solution is to split the software into multiple small, specialized packages rather than few monolithic and thus inherently very complex ones. This makes sense from both a software and a personnel perspective: smaller pieces of software are both easier to maintain and to hand over (or replace).

2.4.1 Perception

As shown in Figure 2.5, each perception module ran independently of each other, even when (partly) the same information was processed. It is proposed[1] to use separate, specialized modules which run independently and optionally use data from a central fusion block. A depiction can be found in Figure 2.8. For the sake of brevity, only an object-level fusion architecture is described here. Note that it naturally extends to feature- or even low-level-fusion, which is also ongoing research at TAS (see, e. g., Michaelis et al. [2019]).

The perceived environment is split into two parts:

A) objects (moving or not) which can be detected and potentially classified and
B) everything else perceived, which cannot be reasoned about (yet).

Each specialized, sensor-specific module is able to detect objects, classify and track them over time. The distinguished object classes are to be chosen dependent on

[1] Note that the perception part of the new architecture was never implemented/tested on the real test vehicles and is thus only a concept. Regardless, it is presented here to paint the larger picture any architecture should provide.

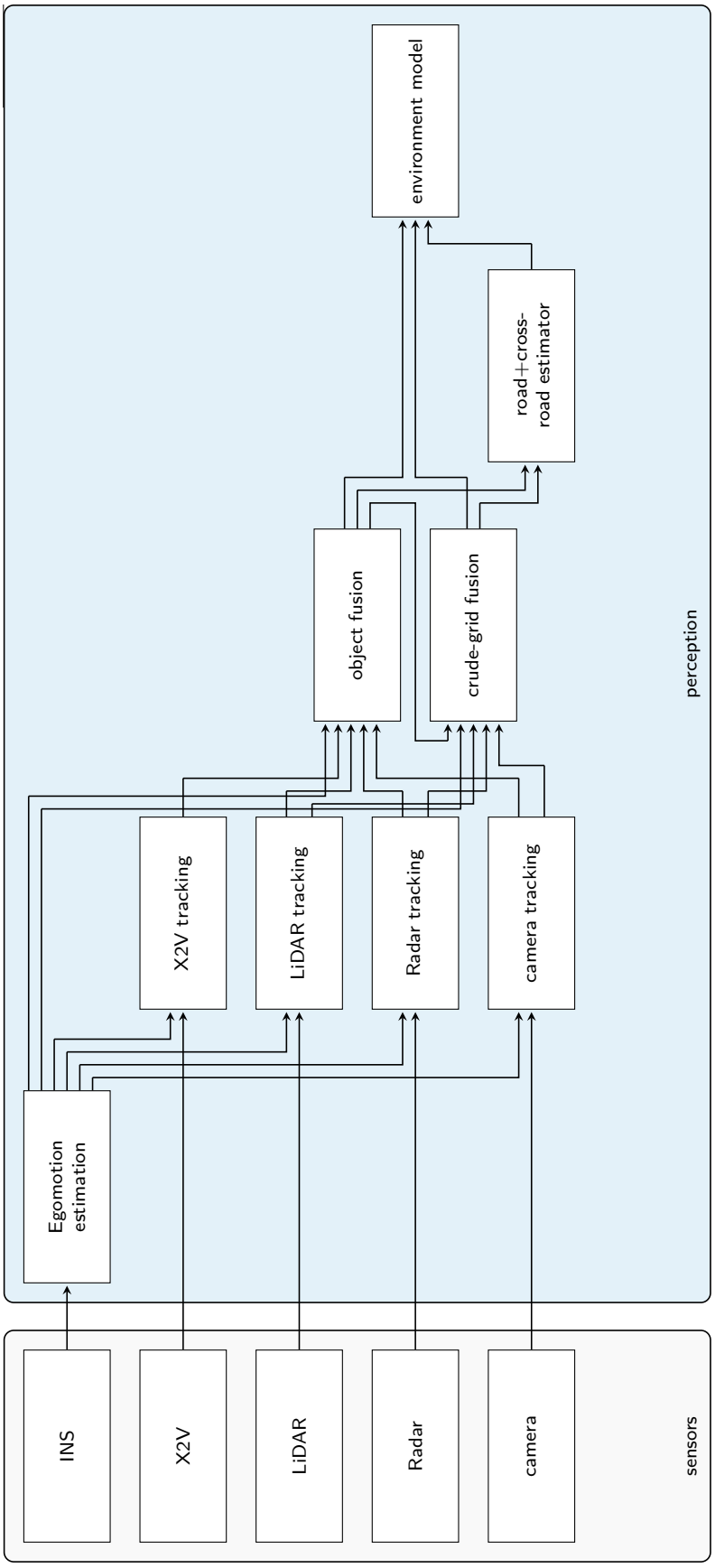


Figure 2.8:
Overview of signal flow between perception modules in the new architecture. For visual clarity, the back path (fusion hypotheses to each tracking modules) is omitted. Note that X2V encompasses Vehicle-to-Vehicle, Infrastructure-to-Vehicle, etc.

sensor capabilities and the specific operative design domain. The resulting objects are provided to the central fusion module. Optionally, the perception modules can utilize object hypotheses provided by the central fusion module (not shown in Figure 2.8 for visual clarity). Furthermore, detections that are (not yet) classified or do not belong to any track (yet) are provided to another module, which is introduced shortly.

The fusion is triggered on new input and its internal filter (e. g., PhD, see Vo and Ma [2006], or Multi-Bernoulli, see Reuter et al. [2014]) updates accordingly. Since the time between sensor estimates is relatively short, e. g., 100 ms for LiDAR and less for camera, simple motion models suffice for the prediction in-between updates and thus for the published central fused prediction.

Next to the (classified) objects, also everything that cannot be directly classified (yet) has to be represented. For this, a grid-based approach is used. For the evolution of the environment-mapping grid at TAS see Jaspers et al. [2017], Engler et al. [2018], Jaspers [2021].

While it is not the focus of this work, the idea behind the *crude grid* is shortly introduced: Perception modules need some sensor cycles in order to detect, track and classify objects (e. g., other traffic participants) with sufficient certainty.[2] For the initial sensor measurements and measurements which cannot be reasoned upon, for instance an irregular static object on road —a typical use case is lost cargo— another representation is necessary.

This representation needs to work with very cluttered and uncertain (both in existence and position) *things*. Those things can be anything from sensor noise to immature tracks of objects. Hence, a grid-based representation which works independently of any semantic information is used.

Recent papers (e. g., Steyer et al. [2017, 2020]) handle the issues arising from such a heterogeneous representation, i. e., the transition between grid- and object-world. However, often (e. g., Tanzmeister et al. [2014b], Yuan et al. [2015]), the grid is used for gathering information and then extract objects from it. Here, it is proposed to explicitly let each sensor use its best internal representation and only use the grid for accumulating what is left. I. e., the information flow is reversed: once an object track is matured, this information is provided to the crude grid which then removes the according artifacts.

The new framework is tailored for robustness. Each perception block is able to work on its own, but may benefit (through the fusion module) from all other sensors. While the fusion is a central part, enhancing the results, there are still objects the planning module can react to, if it stops working.

Next to the software-sided benefits, such as fewer dependencies and smaller easier-to-maintain programs, such system is also easier to develop. For instance:

- a changed sensor only affects one component, the interface remains stable
- not everybody needs in-depth knowledge of all sensors
- a change in staff does not necessarily mean a re-write of monolithic module

[2] There exist once-shot methods (e. g., Redmon et al. [2016]), which are good as detectors, but in most cases only tracking over time yields sufficient reliability.

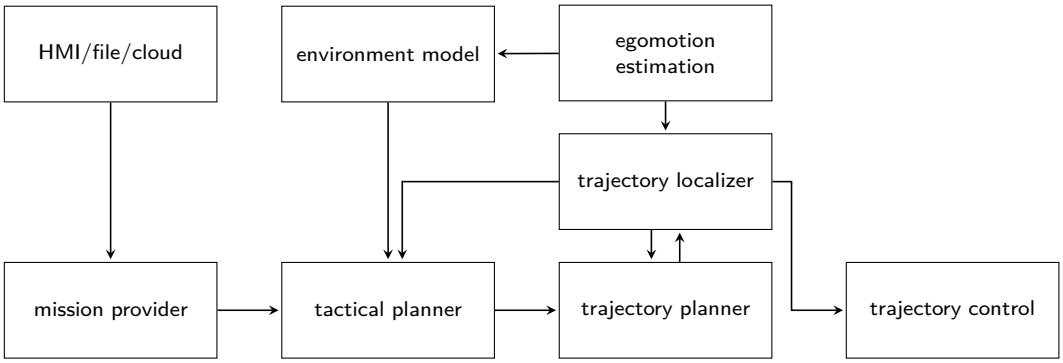


Figure 2.9:
Overview of new planning architecture. The classic cascade of strategical, tactical and operational planning is extended by a central trajectory localization instance. The output of the trajectory localizer holds the ego localization (and prediction) and the reference to the trajectory it localizes on. Further, the mission provider may utilize the egomotion estimation to a limited extend.

2.4.2 Planning

Formerly, the planning was done using a hierarchical state machine which reasoned upon the given scene (lanes and objects) and selected maneuvers according to a predetermined mission. In the new framework, the functionality of the state machine is split into two blocks:

- a *tactical planner*, which reasons upon the scene, augments it accordingly and thus imposes the desired behavior to
- a *trajectory planner*, which, given the enriched environment representation, plans the future ego states accordingly.

A depiction can be found in Figure 2.9. There are two further building blocks of the planning:

- a *mission provider* for top-level goals (e. g., routing to a selected destination) —this is more or less unchanged— and
- a central *trajectory localizer* that handles the current and predicted ego position w. r. t. the planned trajectory.

Thus, the hierarchical state machine is replaced by a hierarchical control architecture. Its components are described shortly below and a tabular view on the according time scales can be found in Table 2.1

Mission Provider The mission provider does the *strategical planning*, e. g., it decides which route to drive. This can be arbitrarily complex (fleet management) or very simple. For this work, the latter suffices: A simple GUI (alternatively: file) to specify a goal on a given (rough) routing graph. Also, other parameters, like maximum mission speed, maximum mission lateral acceleration etc. are provided similarly.

Table 2.1: Tabular view on components of the new planning architecture.

component	trigger	cycle	purpose
mission provider	human interaction	acyclic	set goal, set parameters
tactical planner	environment model	100 ms	do reasoning, enrich environment data accordingly
trajectory planner	tactical planner	100 ms	iteratively replan smooth and safe ego trajectory
trajectory localization	egomotion estimation	10 ms	predict current and future relative ego pose
trajectory control	trajectory localization	10 ms	guarantee stability, communicate with low-level

Tactical Planner The tactical planner takes the parameters from the mission, potentially a map and the newest data from perception, i. e., the perceived static and dynamic environment as provided by the environment model, to analyze the current scene. Its task is to augment the environment representation such that the trajectory planner is able to find a trajectory suitable for the current situation. Here, knowledge from both the static and dynamic environment is contextualized to enable tactical decisions like: overtake the vehicle in front or stay behind it.

Consider this basic example to illustrate a simplified tactical decision-making:

Example 2.2 (Tactical Planner)**.** When driving along a road, it would be more time-efficient to use the oncoming lane during a left curve. However: even if that lane is momentarily free, the ego vehicle is expected to stay in its lane. Thus, the tactical planner would usually tag the middle lane boundary as a hard constraint and the space in the opposing lane as an obstacle. Additionally, any non-suspicious dynamic agents on the oncoming lane and on the far side of it are tagged as unimportant. If, however, a very slow vehicle (or a poorly parked car) is in front of the ego vehicle, and there is clearly no oncoming traffic, this constraint can be temporarily removed and the space of the oncoming lane is merely tagged as undesirable to drive upon. □

In order to provide the trajectory planner with a tactical search space, this planner needs to predict dynamic agents in the scene for a planning horizon larger than the perception is able to provide. Additionally, due to the larger planning horizon, it is paramount to also consider multiple intents, i. e., multiple possible cases like:

- does the pedestrian use the crosswalk or go on, or
- which exit of the roundabout is the car going to take?

have to be considered. For computational reasons, it is usually necessary to use simpler motion models to handle this multi-modality.

Next to imposing traffic rules (e. g., prohibiting forbidden and/or undesirable areas) and object filtering, a task the tactical planner should handle is *visibility reasoning*: From the environment representation, it can be deduced which areas are observable and which areas are not. These areas change with the movement of the ego vehicle and all other agents in the scene. Only when this information is observed over time and reasoned upon, can the ego vehicle drive safely in highly unstructured or densely populated, i. e., urban scenes (e. g., see Yu et al. [2019]). However, since this work focuses on rural environments, considering the occlusion from the static world is sufficient.

Trajectory Planner The trajectory planner uses the augmented static and dynamic environment representation and plans the future position in time and space of the ego accordingly. The new approach is described in detail in Chapter 3. In short: it uses a search-based algorithm rather than popular optimization-based or sample-based algorithms. The used algorithm works very well in unstructured, i. e., non-convex, environments and independently of prior map information. The earliest point where last cycle's trajectory may be altered is determined by the trajectory localizer.

Trajectory Localizer Centralizing the localization of the ego w. r. t. the trajectory is necessary to resolve any potential ambiguity between the planning modules. While for normal driving and maneuvering small ambiguities usually pose no issues, if sub-centimeter precision is required, any unnecessary source of error needs to be eliminated. Note that the localization here is the ego position relative to the planned trajectory and its current closest point on the trajectory, not the localization in the sense of SLAM (Simultaneous Localization and Mapping). For more details, see Section 4.3.1.

2.4.3 Control

In contrast to the former framework, the new framework generates actual trajectories, i. e., reliable information on the planned future movement of the ego vehicle for a certain time horizon. Thus, the controllers can rely on important properties, for instance: there are guarantees on the

- spatial continuity, i. e., there are no abrupt changes in position
- geometric continuity, i. e., there are no abrupt changes in curvature and
- temporal continuity, i. e., there are no abrupt changes of the point in time at which a point in space should be reached.

Of course, the planned trajectories need to change over time due to changes in the environment or due to an improved perception of the same. However, up to a certain point in time in the near future the trajectory is kept fixed — for more details, see Section 4.3.2. After this point, the trajectory may be exchanged and a new trajectory part may be stitched to it. Note that the same continuity constraints hold for the stitching point as for the rest of the trajectory. I. e., the trajectory is always continuous, everywhere, in every iteration.

Consequently, the controller can be tuned to follow its reference very precisely and still provide a smooth ride. Without these guarantees, this would be a conundrum as one could either react smoothly, e. g., by low-passing / low-gain control, or minimize the control error at the cost of comfort and the risk of overshot.

The separation between high-level non-vehicle-specific control and low-level vehicle-specific control remains as before, since this concept proved to work well and facilitates the work on different test vehicles.

2.4.4 Introspection

When developing autonomously driving test vehicles, it is paramount for the people on board the vehicle (or remotely controlling it, or later analyzing recordings of incidents) to know

- what is happening now,
- why it is happening and
- what is likely to happen in the near future.

A first step is to visualize the current environment representation and the currently planned trajectory. This gives visual information on what the car perceives and what it plans to do next. Due to the centralized environment representation and the introduced trajectories (which give guarantees on its nearest-future continuity) this was easy to implement.

Nevertheless, if the system gets into trouble, e. g., the environment perception becomes inconsistent or the trajectory planner malfunctions, further tools are needed. For this, an introspection system was developed where each process actively monitors its

- inputs: which data is expected in which intervals, how old may data be, and is it mandatory or optional
- outputs: which data is expected to be provided in which intervals
- internal status: major process steps are monitored (also profiling their runtime) and short free-form messages for information or warnings are published.
- CPU load and memory usage[3]: with the multitude of processes and the growing usage of parallelization in each, bottlenecks needed to be detected early

This monitoring system was added into both application frameworks at TAS. There, the introspection data was recorded together with all other data to the middleware (KogMo-RTDB).

Having all information available helps to detect trouble during testing and competition (which process/data is missing? Which application does one have to restart?) but also facilitates forensics. I. e., after tests/competitions have been driven, an in-detail view on the system's performance is available. Here, non-phenomenological issues can be scrutinized, among others: load spikes or processes that froze but recovered.

Further, it helps to track performance gain/loss over time. For instance, it enabled to run an updated software on the same data and compare the impact on the system load. Thus, gains or losses that are typically not measured (in contrast to, e. g., tracking quality) can be quantified long before issues due to load spikes occur. Naturally, this enables an automatic continuous evaluation of the software quality in a CI/CD (continuous integration/continuous deployment) manner.

[3] This extension was implemented by Thorsten Lüttel.

2.5 Comparison

In the following, the former and the new system architecture are compared both qualitatively and quantitatively. Note that parts of the latter were pre-published in Heinrich et al. [2017].

2.5.1 Qualitatively

The former architecture was composed of components which were designed and written by former PhD students in the scope of their respective research. Each component was built to optimize its own performance and in general succeeded to do so. This means, individual fusion chains had great metrics, but the overall performance/maintainability was not in the focus.

The system architecture introduced in this thesis has the goal of considering the complete control chain, form input to output. This is necessary to improve the system's output (movement) which formerly lacked input data due to the loss of information along the functional chain. See Table 2.2 for an overview of the steps that were therefore taken:

Motion Control The input to control was improved from the former so-called *lane* struct (holding only momentary information, see Definition 1.1) to a trajectory plus localization interface. This provides the possibility to improve the controller performance and creates room for modern control approaches. I. e., smoother and more precise driving performance are now achievable. For more details, see Chapter 4.

Motion Planning In order to provide a real trajectory with continuity guarantees, the reactive tentacles-based and/or clothoid-based lane-follow approach where replaced with an actual trajectory planner. Apart from the egomotion estimation (and thus localization) and lead-vehicle tracking —which due to being filter-based, are prone to noise— this now yields a smooth and continuous input. Actual trajectory planning now allows performing complex maneuvers in challenging unstructured environments. For more details, see Chapter 3.

Behavior In order to be able to utilize the new planning capabilities, the hierarchical finite state machine was replaced by a hierarchical planning architecture. Now, different tasks can be handled in different abstraction levels and frequencies. Additional benefits are that

- the tight coupling between the state-machine and vehicle-control components is lifted, allowing for easier maintenance
- low-level tasks (like shifting gears) no longer need to be handled in high-level decision-making
- there are no hard state transitions anymore, potentially causing undesired transient behavior

- this solution is scalable, i. e., extendable for more complex scenarios with reasonable effort

Note that the behavior layer was only implemented prototypically, though in a major project, and showed promising results.

Fusion A centralized fusion architecture was proposed, which uses both object fusion and grid fusion. Each is used for the benefits in their respective domain. This, and the split into single-purpose perception modules, should alleviate the issues from having large single-purpose applications which are hard to maintain and to hand over. The interplay between different levels of fusion is ongoing research at TAS and out of the scope of this work. The fusion changes are presented primarily to give a whole-system view of the necessary changes on the way to improve the overall performance of the TAS test vehicles.

Monitoring Introspection data was added to the TAS application frameworks. Thus, it is possible to explicitly track performance live in the vehicle and also over time, given the recordings from test drives / competitions. This does not directly benefit the driving performance but facilitates development, testing, bug fixing, performance tracking and the performance in competitions. It is thus hopefully a valuable contribution.

Note that this is only the first step towards error management in the TAS prototypes. The next step after monitoring is automated reactions to certain errors, e. g., degradation modes that bring the ego vehicle to a safe state, even if some components fail — a task the safety driver is currently responsible for. This is a large field on its own which is out of scope of the research at TAS and hence this work.

Table 2.2: Qualitative comparison between former and new system architecture

	former	new	maturity
motion control	reactive	predictive/proactive	ready
	separate lat and lon	unified trajectory	ready
motion planning	reactive	predictive/proactive	ready
	no guarantees	smooth and continuous	ready
behavior	hierarchical states	hierarchical architecture	prototype
	scales badly	scales reasonably	prototype
fusion	application driven	centralized	concept
monitoring	no	introspection	visual+logs

2.5.2 Quantitatively

For a quantitative comparison, a simple test scenario is used. Both the trajectory-tracking precision and the driving comfort in terms of lateral acceleration is measured. For more details on the underlying trajectory-following controller, please see Section 4.3.3.

A simple leader-follower setup is used which allows to showcase the benefits of the streamlined information flow and thus information gain in the controller. The convoy lead vehicle starts by driving straight at a constant $3\,\frac{\mathrm{m}}{\mathrm{s}}$. Then, it performs a left turn after which it accelerates to $6\,\frac{\mathrm{m}}{\mathrm{s}}$ before it comes to a stop.

The experiments were conducted both in simulation and in the real world using MuCAR-3 and a simulated lead vehicle. The architecture allows running exactly the same high-level system software in both tests. In simulation, only the low-level system and the egomotion are replaced.

The results are shown in Figure 2.10. In the top row, the bird's-eye view in UTM (Universal Transverse Mercator) coordinates is shown. The lead vehicle starts at the top left and drives to the bottom right (black dashed line). The follower vehicle is once simulated using the previous framework (orange) and once using the new framework (blue). Note that there are slight differences in heading between simulation and real world and between two real-world runs. This is due to the simulated lead vehicle being placed relatively to the current ego position. Its relative trajectory, however, is exactly the same (in an earth-fixed reference frame).

There is a sizable lateral offset using the previous framework, both in simulation (see Figure 2.10a) and in the real-world (see Figure 2.10b) experiment. Even quantitatively, the results are similar. Though, of course, the real-world deviations from the reference path are slightly larger.

Since the new framework has information on the future curvature and thus the expected lateral accelerations, the comfort limit of $1.5\,\frac{\mathrm{m}}{\mathrm{s}^2}$ is kept, as can be seen in Figure 2.10c. In contrast, the lateral acceleration exceeds the safety limit of $3\,\frac{\mathrm{m}}{\mathrm{s}^2}$ before the longitudinal controller is able to react and to slow the car down in the former framework. There, lateral and longitudinal control run separately and only the current *lane* (see Definition 1.1) is available.

The real-word tests validate the simulation results. Again, the measured values are slightly higher than the simulated ones. Note that the plotted curves in Figure 2.10d were already low-pass filtered; the actual measurements are also shown with less opacity.

The experimental data was recorded when the test site was icy. Hence, the deviations from the reference path and the measured accelerations are larger than usual. This dataset was chosen intentionally to showcases the robustness of the new framework.

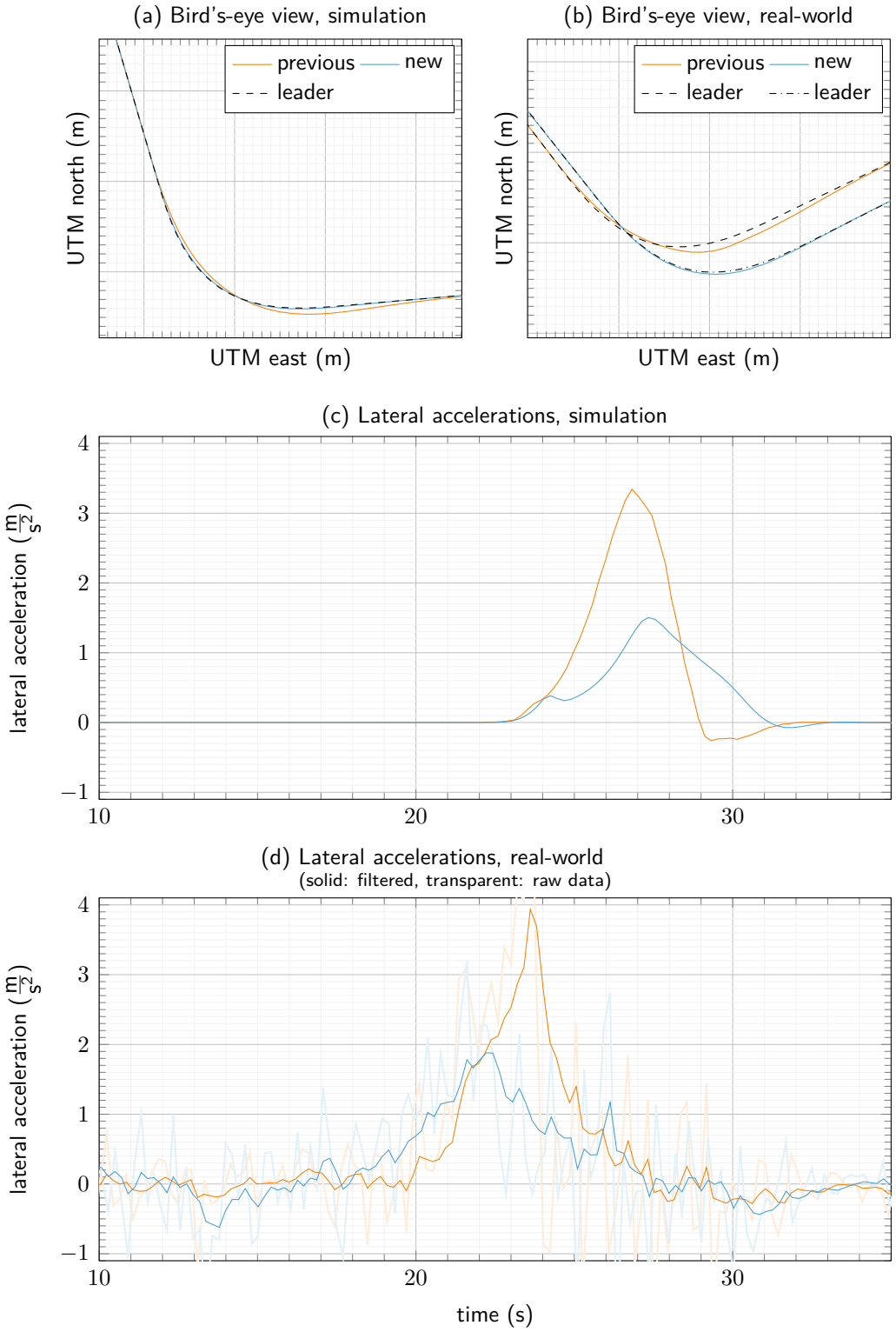


Figure 2.10:
The top row shows the bird's-eye views of the test scenario, both in simulation and in real testing. The lead vehicle (black) starts at the top left and then does a left turn. The coordinates are in UTM, thin lines and thick lines represent 1 m and 10 m, respectively. Below, there is a comparison between the simulated and the actual lateral acceleration. In all plots, the follower vehicles using the previous and new framework are depicted in orange and blue, respectively.

2.6 Conclusion

The goal of this work is to improve both the smoothness and the precision of an autonomous driving experience. A re-design of the system architecture was necessary to achieve those goals. Additionally, measures to improve system stability and to facilitate system analysis were taken.

While at the time of writing, some parts are still in the concept phase (perception) and other parts are only implemented prototypically (tactical planner), the core of this work, the step from a reactive controller to a predictive actual motion planning and control was made and tested and refined on various test vehicles and even in international robotics competitions.

Due to the architecture's scalability and modularity it was used for tasks from driving in constraint (sub-)urban scenarios to precise vehicle positioning, off-road vehicle following and more. The added introspection, while not benefiting the system performance directly, facilitates the development process and enables the test drivers —especially during competitions— to react faster to any issue by providing crucial information at a glance.

3 Motion Planning

Motion planning is a major task in autonomous driving. It describes the act of defining the desired future positions of the ego vehicle in space and time, i. e., the generation of the future ego trajectory (see Definition 3.3). Recalling the main goal of this thesis: improving both the precision and the smoothness of the driving performance, this is clearly the key element.

Throughout this work, the assumption is that a two- (or two-and-a-half-) dimensional representation of the environment suffices for the task of autonomous driving. This was found to be viable even in unstructured terrain. How, e. g., slopes, are handled as another two-dimensional layer in the environment representation is laid out in Jaspers et al. [2017].

A prerequisite for motion planning is that the resulting trajectory shall be physically drivable by the vehicle. In autonomous driving, typically non-holonomic systems are used. I. e., the next ego state is dependent on its current state and a set of constraints. For example, a car cannot move laterally without moving longitudinally. At low speeds, the so-called kinematic bicycle model (see Appendix A.10) was found to be sufficient for motion planning and generating a smooth riding experience, even in off-road scenarios.

Another prerequisite for motion planning is that the resulting trajectory shall be free of obstacles. Here, an obstacle means any kind of dangerous environment. This includes static dangers —e. g., trees, bushes or so-called negative obstacles, like: holes— and dynamic dangers — for instance, cars, bicycles or pedestrians. In order to master challenging scenarios, the motion planning (and execution) needs to be very accurate.

This chapter is structured as follows:
After a short overview regarding the state of the art in Section 3.1, motion-planning-relevant notation and definitions are given in Section 3.2. Then, in Section 3.3, a short summary of the used motion-planning framework is given. The aspect of collision checking is briefly explained and a contribution is presented in Section 3.4. Thereafter, two kinds of trajectory generation are discussed: First, a classic path-velocity decomposition in Section 3.5. There, different kinds of speed-profile generation and contributions in that area are provided. Additionally, contributions to mitigating the effects of multiple vehicles following each other (lateral offset, longitudinal oscillation) are presented in Section 3.6. Second, a novel two-step trajectory generation method is given in Section 3.7. Finally, the results are summarized in Section 3.8.

3.1 State of the Art

Motion planning for robotics in general and for car-like vehicles is a prolific subject with many facets. This work primarily investigates algorithms that were shown to actually work in real-world scenarios or competitions.

In the Bertha Drive, for instance, a local continuous method was used (see Ziegler et al. [2014a]). Continuous methods usually optimize a cost functional using variational approaches. These are typically local since they use (in the broadest sense) gradient descent to converge to an optimum.

A predominant method for constrained optimization-based trajectory generation is Model Predictive Control (MPC). Since its early days (see Mayne et al. [2000]), its various branches continue to evolve, e. g., linear MPC (see Turri et al. [2013]) or tube MPC (see Mayne et al. [2011], Rakovic et al. [2012]) and toolboxes for solvers are created (for instance, see Domahidi et al. [2012], Zanelli et al. [2020]).

MPC is more a scheme than an actual algorithm. Its idea is to find an optimal trajectory for a finite horizon. Only the initial part of that trajectory is traversed. During that time, a new optimal trajectory from the end of that initial part to a now shifted horizon is calculated, which is why it is also called a *receding horizon* method. Utilizing Bellmann's principle of optimality (Bellman [1957], oversimplified: parts of an optimal solution form an optimal solution) the resulting trajectory is still optimal. The basic MPC principle is utilized in multiple ways in this work, even though no classic MPC is implemented.

In the Bertha case, the optimization was conducted in Cartesian coordinates. In on-road scenarios, however, the broad majority of algorithms work in Frenet coordinates (see Frenet [1852]). This moving reference frame allows formulating the lateral planning (or control) problem in curvilinear coordinates such that any road curvature is abstracted away (see Figure 3.1), i. e., a coordinate frame is used that is relative to a reference path — typically the lane center.

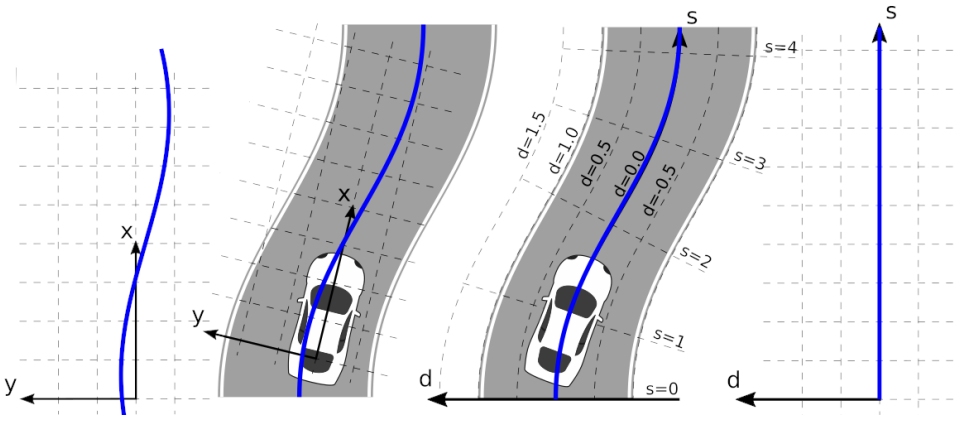


Figure 3.1:
Same street scene represented in Cartesian (left) and Frenet (right) frame. Note that, in the latter, the curvature is abstracted away in the planning frame.
Taken and modified from Pucher [2018].

Another variational method are the so-called elastic bands (see Quinlan and Khatib [1993]). Initially purely a path-planning approach where intermediate poses were modified such that obstacles were avoided, it was extended to incorporate time information and temporal constraints (see Rösmann et al. [2013]) and was used in a MPC-scheme (see Rösmann et al. [2015]). This, finally, led to a sparse optimization problem subject to kinodynamic constraints in addition to the initial obstacle avoidance (see Rösmann et al. [2017]).

Next to variational methods, there are methods trying to circumvent staying in local minima. One of them is so-called sampling-based motion planning. They were used by most teams in the DARPA (Defense Advanced Research Projects Agency) challenges. Werling et al. [2010] presented a direct and concise follow up: In a Frenet frame, goal positions are generated (displacement from center at different lengths at different points in time). The optimal one-dimensional trajectory connecting a start to a goal position is very cheap to generate since there exists an analytical jerk-optimal solution in the form of a 5^{th}-order polynomial (see Takahashi et al. [1989]). Lateral and longitudinal trajectories are combined and result in thousands of solution candidates. As these still need to be checked for kinodynamic constraints, collisions, etc. this is a typical *generate and evaluate* scheme. While at first glance, this approach seems wasteful and brute force, it has the major advantage of being independent of the topology of the problem. This means, within its sample range and discretization the sampling-based method finds the global semi-optimum. This is achieved within nearly constant runtime for potentially highly non-convex problems, if collision checking is excluded.

Apart from the deterministic sampling, there is also a branch in robotic motion planning that utilizes probabilistic sampling. The classic foundation is Rapidly-Exploring Random Tree (RRT) (see LaValle [1998]), where a path is planned by connecting randomly chosen nearby poses using a *steering function*. This function considers kinodynamic constraints and can potentially also check for collisions. The randomness in the target selection has a major advantage: if a solution exists, the algorithm will eventually find it.

Again, there are numerous sub-branches of RRT. In general, the issue of the convergence speed is addressed, for instance, by using bi-directional search (see Kuffner and LaValle [2000]) or adding heuristics (see Urmson and Simmons [2003]). An optimal version was published in Karaman and Frazzoli [2011] — note that the classic approach does not asymptotically converge to the optimal solution! This was also extended to bi-direcitonal search (see Jordan and Perez [2013]), enriched with heuristics (e. g., see Gammell et al. [2014]) and any combination of those (e. g., see Burget et al. [2016]).

Batch Informed Trees (BIT*) (see Gammell et al. [2015, 2020]) promise to unify the advantages of classic probabilistic sampling with the efficiency of heuristic-based algorithms, such as A*. Here, multiple trees are generated, one in each step (called *batch*). First, random samples are placed globally and a search area (based on a heuristic) is expanded until a solution has been found, i. e., a connection from start to goal state. This concludes a batch. The next batch then randomly samples the last search area and, again, repeats the expanding search, now within the smaller

area. Thus, through repeated refinement, a converging solution can be found which can be shown to be asymptotically optimal.

Yet another sampling-based direction is *search-based* planning. This branch was mainly driven by the successful motion planner used in the DARPA Urban Challenge by Dolgov et al. [2008, 2010]. He extended the classically graph-based A*-algorithm (see Hart et al. [1968]) to generate the search graph on the fly. Here, each node was not on any fixed raster, but rather defined by expanding the currently most promising node with a *motion primitive*, i. e., something kinodynamically feasible. These motion primitives were sampled, typically representing an uneven number of steering rate alternatives. Note that Ziegler et al. [2008] was also inspired by the kinematic bicycle model, but did only plan for constant speeds.

Other works try to combine the strengths of different branches. Kunz and Dietmayer [2016], for example, utilized the strength of a sampling-based algorithm to find a good homotopy, i. e., a solution close to the global optimum. This result was used as starting solution for an optimization-based (MPC) motion planner. The latter was able to run at a higher frequency and provided a smooth locally optimal trajectory. So the strength of the former (finding the correct homotopy) is combined with the strength of the latter (finding a smooth trajectory, fast) while the weaknesses (slower, rougher solution and staying in a local optimum, respectively) are mitigated.

3.2 Notation

The following notation for points and sequences is used in this chapter. The smallest building block, a point $\mathbf{p}$, is defined as:

Definition 3.1 (Point)**.** A point $\mathbf{p}$ is a tuple of information. There are

- *spatial points*: $\langle l, x, y, \ldots \rangle$ providing running length l and x, y coordinates
- *temporal points*: $\langle t, v, \ldots \rangle$ providing a timestamp t and a speed v
- *spatiotemporal points*: a combination of the two above

Note the ellipsis '…': it means that dependent on the transition on the sequence, further information can be necessary.[1] Throughout this work, the type of transition $\mathrm{T}(\cdot)$ will define how to connect two given points $\mathbf{p}$:

Definition 3.2 (Transition)**.** A transition $\mathrm{T}(\cdot)$ describes how to interpolate[2] between two points $\mathbf{p}_i$. There are

- *spatial transitions*: $\mathbf{p}(l) \leftarrow \mathrm{T}(\mathbf{p}_i(l_i), \mathbf{p}_{i+1}(l_{i+1}), l)$, where $l \in [l_i, l_{i+1}]$
- *temporal transitions*: $\mathbf{p}(t) \leftarrow \mathrm{T}(\mathbf{p}_i(t_i), \mathbf{p}_{i+1}(t_{i+1}), t)$, where $t \in [t_i, t_{i+1}]$
- *spatiotemporal transitions*: where either l or t is given and the other is deduced

For a very simple transition, no further information is needed per point (see Example 3.1). In order to interpolate using constant jerk, for instance, a temporal point would need to incorporate the acceleration, too.

Example 3.1 (Temporal Transition)**.** Given start and end point, $\mathbf{p}_0 = \langle 10\,\mathrm{s}, 3\,\frac{\mathrm{m}}{\mathrm{s}} \rangle$ and $\mathbf{p}_1 = \langle 13\,\mathrm{s}, 2\,\frac{\mathrm{m}}{\mathrm{s}} \rangle$, respectively, and a time $t = 12\,\mathrm{s}$, using the constant-acceleration transition $\mathrm{T}_{\mathsf{ca}}() : \begin{cases} \mathbf{p}_{[v]} \leftarrow \mathbf{p}_{0,[v]} + \dfrac{\mathbf{p}_{1,[v]} - \mathbf{p}_{0,[v]}}{\mathbf{p}_{1,[t]} - \mathbf{p}_{0,[t]}}(t - \mathbf{p}_{0,[t]}) \\ \mathbf{p}_{[t]} \leftarrow t \end{cases}$, it follows that $\mathbf{p} = \mathrm{T}_{\mathsf{ca}}(\mathbf{p}_0, \mathbf{p}_1, t) = \langle 12\,\mathrm{s}, 2\tfrac{1}{3}\,\frac{\mathrm{m}}{\mathrm{s}} \rangle$. □

Using the last definitions, the following types of sequences are defined:

Definition 3.3 (Sequence Types)**.** Combining Definitions 3.1 and 3.2, define the following ordered sequences $\mathbf{P}$:

- *path*: spatial points, combined by a spatial transition
- *speed profile*: temporal points, combined by a temporal transition
- *trajectory*: spatiotemporal points, combined by a spatiotemporal transition

Note that the given transition $\mathrm{T}(\cdot)$ allows interpolating arbitrary points $\mathbf{p}$ from given length coordinates l (path/trajectory) or given time coordinates t (speed profile/trajectory).

[1] For instance, z information, if the spatial points are three-dimensional.
[2] This is not limited to linear interpolation but depends on the underlying function.

3.3 Framework

The proposed motion-planning framework is based on the so-called hybrid-state A* algorithm (hA*, see Dolgov et al. [2008, 2010]). The algorithm is particularly well suited for navigating in unstructured terrain. It was first published in Fassbender et al. [2014b], but extended and improved since then.

3.3.1 Roots

The proposed motion-planning framework is based on hA*, which is based on A* (see Hart et al. [1968]) which is based on Dijkstra's algorithm (see Dijkstra [1959]). Dijkstra's algorithm, as well as A*, finds the shortest path on a graph.

A graph consists of *nodes* which are connected by *edges*. Each edge has a cost associated to it. A typical example would be a road map: intersections (nodes) are connected by streets (edges), which are rated by their traveling time (cost).

Dijkstra's algorithm, without going into the details, works as follows:

1. Pick the cheapest known node (in the beginning: the start node, it's free)
2. Evaluate all neighboring nodes: Annotate them with the expanded node's cost plus the connecting edge's cost. Let it be called the node's *cost-until*.
3. Check if the goal node is the cheapest. If so: the path is found, else: repeat from 1.

Note that in the background, the paths are stored and sorted.

A* extends this algorithm as follows: Before evaluation, apply a guiding heuristic. For each node, an approximated *cost-ahead* is annotated according to the heuristic. Then, when picking the cheapest node, the sum of the accumulated cost-until and the approximated cost-ahead is considered.

In the road-graph example, the heuristic could be the Euclidean distance to the target node. Thus, the graph would be explored with bias towards the goal. Dependent on the graph's topology (e. g., a lot of dead ends) the heuristic can also be misleading, but as long as it is *admissible*, the algorithm is still guaranteed to find the optimal solution.

The guiding heuristic is *admissible* if it never overestimates the real cost-until. Intuitively this is clear, since otherwise it could ignore nodes due to too high approximated cost-ahead. For a formal proof, cf. Hart et al. [1968].

While, for routing problems, there is usually a graph of possible transitions available, for motion planning, the graph of all possible motions would be prohibitively large. One possibility is to use a so-called grid graph (also known as lattice graph, see Figure 3.2a). A grid graph distributes nodes evenly spaced in Euclidean space and connects a fixed number of neighbors (typically 4 or 8), forming a regular mesh. This mesh, dependent on its granularity, then allows to find paths in any environments.

However, the resulting path moves along the edges and hence consists of a number of instantaneous (typically 90° or 45°) turns. This requires a holonomic system, i. e.,

(a) A* on a grid graph (b) Hybrid-state A*

Figure 3.2:
Comparison of grid-graph A* and hybrid-state A*. Taken from Dolgov et al. [2008].

one that can move in any direction instantly. For autonomous driving, and especially if a smooth ride is desired, the resulting path is not viable for driving[3]. This is where hA* comes into play.

In hA*, the edges are drivable *motion primitives* (see Figure 3.2b). Since a graph of all future drivable motion primitives is infeasible to generate a-priori[4], the search graph (or rather search tree) is built on the fly. For this, extend the base algorithm as follows:

1. Pick the cheapest known node: Consider its accumulated cost-until and the heuristic cost-ahead.
2. Expand the node: Generate a number of neighboring nodes using a set of motion primitives as edges.
3. Evaluate all neighboring nodes: Annotate them with the cost of the current node plus the connecting edge's cost plus the heuristic cost-ahead.
4. Check if a node within the goal manifold is the cheapest. If so: the path is found, else: repeat from 1.

Note that when building the graph incrementally, it is not possible to define a goal node. Thus, a goal manifold needs to be designed carefully. Note further that it can be beneficial to relax the abort condition such that the search continues, e. g., for a limited amount of time/expansions, in order to find a better path.

Dolgov solved this problem by adding so-called analytical expansions to this hA*. In those, he used the well-known Reed–Shepp paths (see Reeds and Shepp [1990]) to connect his current node directly to the goal. While this is the optimal (shortest) path and leads perfectly to the goal node, it is not a directly drivable path due to being only $\mathcal{C}^0$ continuous. Therefore, Dolgov used a further optimization step after he found the initial path candidate. This also potentially mitigates the issue with the optimality of the solution. The optimizer will definitely only find a local optimum. Practically, however, this will often be the global optimum, since the search algorithm has a good chance of finding the best homotopy.

[3]Alternatively, the resulting path can function as a start solution for a trajectory smoother or some different motion planner.

[4]Note that state-lattice methods (see Likhachev and Ferguson [2009]) manage to sample with fixed discretization by utilizing pre-calculations with the caveat of restricting their solution space to the predefined primitives.

This already addresses one of the main concerns with hA*: it does not guarantee to find the globally optimal solution. While theoretically true, in practice often a good solution suffices, especially in unstructured environments where the environment changes (potentially drastically) every planning cycle.

The other concern with hA* is, that dependent on the discretization of the motion primitives, it is not guaranteed to find an existing solution. However, practical experiments at Institute for Autonomous Systems Technology (TAS) confirm Dolgov's observation that in practice these theoretical disadvantages over, e. g., RRT*, are outweighed by the speed and flexibility of hA*. Further, the original algorithm was modified and extended, as is shown in the next subsection.

3.3.2 Modifications

Note that the original author of the motion-planning framework at TAS was Dennis Faßbender. Since there is currently no plan for a separate publication of the motion-planning framework, a condensed overview is given here. Note that only the last-mentioned extension is a contribution for this work. Parts of the extensions were pre-published jointly in Fassbender et al. [2014b, 2016b], Heinrich et al. [2016].

Nodes and Edges While Dolgov et al. [2008] uses only the Cartesian position and orientation (in Dolgov et al. [2010], also the direction: forward or backward) as nodes, the dynamic state here is the kinematic bicycle model in Euclidean space (x, y, ψ, c, v and l/t see Appendix A.10). Note the step from pure spatial path planning to real spatiotemporal trajectory planning.

The edges are constant-sharpness paths (clothoids, see Appendix A.3) with an initially constant-acceleration speed profile. Note that the viable sharpness range, i. e., minimum/maximum rate of curvature change, is speed-dependent, see Example 3.2.

Speed-profile generation Since it is computationally infeasible to expand both spatial and temporal transitions, the following technique is used:

- When expanding, constant-deceleration speed profiles are used, down to a given minimum crawl speed. This way, the viable curvature change rate grows in the future.
- When the best trajectory candidate has been found, a maximum speed profile from the given path and kinematic as well as dynamic constraints of the system is calculated. Details about the generation of the speed profiles are given in Section 3.5.1.

Trajectories are thus planned without the typical prohibiting complexity. The widening of the search space in the further lookout is necessary for allowing to brake for sharp turns without sacrificing performance and comfort in the close vicinity. Note that due to the constant re-planning and the quality of the environment perception, the planned slow-downs and sharper turns are usually only used when they are actually necessary.

Example 3.2 (State-dependent constraints)**.** Considering a maximum lateral acceleration, determine a speed-dependent max curvature:

$$a_{\text{lat}} = \frac{v^2}{r} = cv^2 \quad \Rightarrow \quad c_{\text{max}} = a_{\text{lat,max}} v^{-2} \,. \tag{3.1}$$

Given the edge's length Δl and root node's initial curvature c_0, there is a viable $\hat{c}$ range of

$$\hat{c} \in \left[\frac{-c_{\text{max}} - c_0}{\Delta l}, \frac{c_{\text{max}} - c_0}{\Delta l} \right] , \tag{3.2}$$

within which one could sample, e. g., uniformly.[5] □

Pruning In order to keep the search graph small, all nodes that are within one cell of the grid-based environment representation are stored. If a new node gets expanded into an already occupied cell and both nodes are sufficiently similar, the more expensive one is removed — together with all its children in all queues.

Heuristic The heuristic includes two terms

- The shortest $\mathcal{C}^1$-steady path, found using Reeds and Shepp's circle-and-straight connection (see Reeds and Shepp [1990]). This is an admissible heuristic since it underestimates the real path length due to its instant changes in curvature. Further, it does not take obstacles into account, which would lead to detours.
- The shortest obstacle-free path, found using A* on the grid graph assuming holonomic motion. The cost-ahead for the heuristic is the Euclidean distance to the goal, its cost-until the path length. This is also admissible, since it underestimates the real path length due to its instant changes in heading.

As was shown in Dolgov et al. [2008], combining two admissible heuristics results in an admissible heuristic if one takes the $\max$ of both. This is intuitively clear when considering that each heuristic underestimates the true cost, but each one does so more or less conservatively. I. e., one can always take the less conservative estimate and has thus found a better guiding heuristic.

Target Manifold In contrast to other works in this field, here it is not required to reach a goal position exactly. This is due to the fact that the algorithm was developed with real-world off-road conditions in mind, where neither the maps nor the Global Navigation Satellite System (GNSS) can be completely relied upon. E. g., when entering/leaving forests, spontaneous shifts in the global ego position happen frequently.

Consequently, the target manifold is a broad corridor in the positive-x half plane starting at the goal position on the road graph[6]. As mentioned earlier, the search is not necessarily aborted after the first path is in the goal manifold, but after a predetermined period of time. When the available planning time is up, the cheapest

[5] This is a simplified depiction for the sake of brevity. Additionally, the maximum steering angle and rate are considered.

[6] Note that the road graph used is usually a very low-fidelity map and the goal is extracted without any matching to sensor data.

path is chosen not only according to the accumulated cost-until but also considering the terminal cost of the final posture.

Continuous Re-planning The trajectory is re-planned at least every 100 ms. It is evaluated whether the current trajectory (planned in the last cycle) is still collision-free and whether the target manifold shifted. If the target manifold merely shifted forward due to progression on the route, the graph is simply extended further. If a collision is imminent, it is re-planned from a point on the current trajectory which is far enough away from the current ego position that it does not violate the continuity assumption from motion control and sufficiently far away from the detected collision to maximize the chance to prevent the collision.

While this is a tuning issue, it was found to perform superior to applying repairing algorithms (e. g., see Koenig et al. [2004]) or re-using the later parts of the search tree from the last cycle in any other way. A main reason for this is that in unstructured environments, the perceived environment in the later trajectory parts is prone to drastic changes.

Flexible Expansions Use cases such as exactly following a road graph (see Fassbender et al. [2016b]) or exactly parking at a specific location (see Heinrich et al. [2016]) are handled by adding additional types of expansions to the algorithm.

For following a road graph exactly without meandering left and right due to the discrete sampling, *pure-pursuit* expansions are used. I. e., a pure-pursuit path-following control (see Snider [2009]) is iteratively simulated forward, resulting in short clothoid arcs, e. g., a third of the usual expansion length, such that a sequence of three is comparable with a normal expansion.

For exactly reaching a location on a road graph, *one-shot* expansions are used. I. e., computationally efficient steering functions that directly connect a start to a goal pose are used as another expansion. Both numerical-optimization-based (see Kelly and Nagy [2003]) and spline-based (see Walton and Meek [2005]) methods are implemented.

While Fraichard and Scheuer [2004] or Walton and Meek [2005] (see Figure 3.3) generate optimal, constrained, $\mathcal{C}^2$-continuous curves (i. e., clothoid), they were not able to handle starting curvature. This is mitigated as follows (see Figure 3.4):

- Add a circle segment to the current posture (only if $c \neq 0$)
- Add a closing clothoid segment, i. e., one that reduces the curvature to zero, to the intermediate posture
- Use the method described in Walton and Meek [2005] to analytically find the optimal connection between the second intermediate pose and the goal pose.

The best parameters for the circle-segment length and the closing clothoid's curvature-change rate are then searched for using a simple two-step (first coarse, then fine) parameter search. The cost is simply the resulting overall path length. This approach was benchmarked against cubic curvature polynomials (used e. g., in Heinrich et al. [2015]) and found to be $2.33\times$faster and yielding 4219 vs 56 (of 5445 possible) actually feasible trajectories. For more details, see Heinrich et al. [2016].

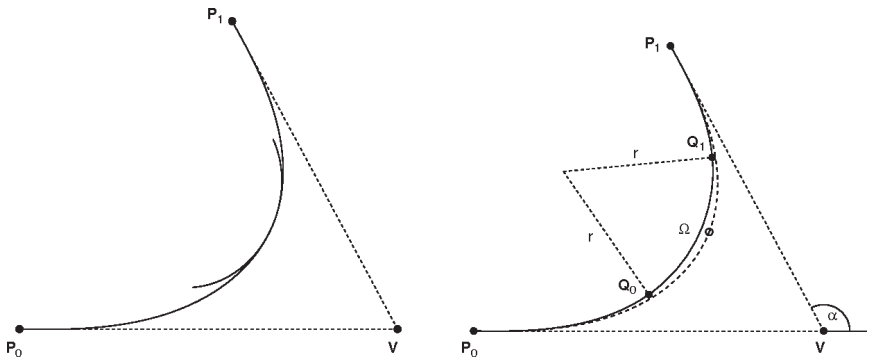


Figure 3.3:
Method to generate a symmetrical $\mathcal{C}^2$-continuous path between two poses: (P_0 and P_1). The formulation allows to specify a constraint on the maximum curvature. If it is violated, an intermediate circle segment is added. The left path has no intermediate circle segment, the right path has a maximum-curvature circle inserted. The former is shorter, while the latter fulfills the constraints. Taken from Walton and Meek [2005].

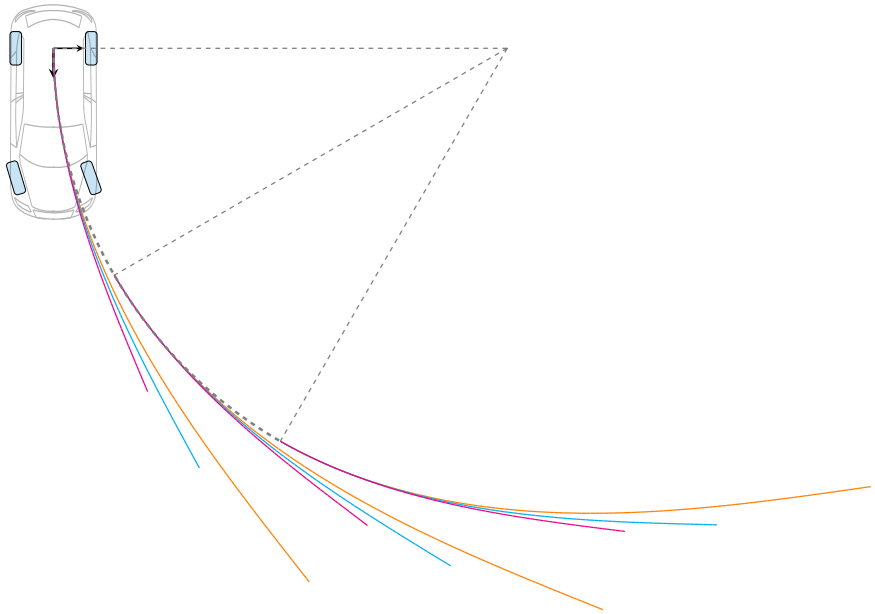


Figure 3.4:
Extension to Figure 3.3: Initial curvature is handled by prepending an (optional) circle with radius according to current steering angle and closing clothoid (to zero curvature) from the initial pose. For illustrative purpose exaggerated, circle-segments of three different lengths (gray, dashed) and end poses resulting from three different (color-coded) $\mathring{c}$ are shown.

3.4 Collision Checking

Trajectories for autonomous driving are required to avoid any detected obstacle. This is true both for on-road driving and driving in unstructured environments. Though the requirements for both kinds of environments differ, they share a computational burden: checking whether a planned trajectory is (still) collision free, or not.

The contribution of this section —the speedup of the collision-checking algorithm— is tailored for, but its use is not limited to, the motion-planning framework presented in Section 3.3. As pointed out, e. g., by Tanzmeister et al. [2014a], the computation of collision checking is still a major burden in motion planning. Note that parts of this section were pre-published in Heinrich et al. [2018a].

3.4.1 Introduction

Autonomous vehicles move non-holonomically, i. e., their future movement is largely constrained by their current state. This poses a challenge for motion planning, but also yields possibilities for performance optimization when checking for collisions, as will be shown in this section.

Bounding boxes are commonly used as approximation of vehicle shapes (see Figure 3.5).A better approximation than a rectangular box might be desired and circles provide this to some extent. Utilizing the discs' curve, the vehicle's contour may be matched better (see Fassbender et al. [2014b]). However, for the considerations in this section using a shorter bounding box is equivalent.

In order to obtain the area which is to be checked for collision, a given trajectory is usually sampled. A polygon is then built by joining the ego shapes which are placed at the sampled points. This is done since the analytic description of the area given the trajectory is usually infeasible. A comparison between the planned trajectory and the actual area that needs to be checked —further called *occupancy*— can be seen in Figure 3.6.

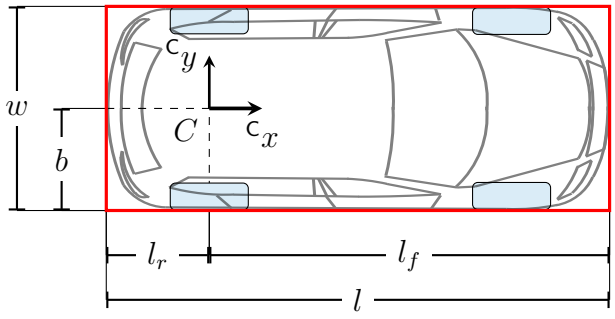


Figure 3.5:
A bounding box (red, length l, width w) for a vehicle (drawn as contour). The planning frame C is defined at the vehicle's center of rotation. Dependent key geometric dimensions are: length to rear, length to front and half width, denoted l_r, l_f and b, respectively. C is assumed to be centered regarding w.

Different approaches to speed up collision checking have been published. Ziegler and Stiller [2010]'s method is considered as the baseline here. It approximates the vehicle's shape by using discs. This is efficient due to the fact that given a distance-transformed grid, an obstacle check is reduced to a single look-up against the respective disc's radius. It is used as baseline since, on the one hand, classic robotics one-disc approximations (see Qu et al. [2004]) are generalized by it, and on the other hand, many authors directly use it or slight variations of it (e. g., see Geiger et al. [2012a], Kunz et al. [2015], Gutjahr et al. [2017], Klaudt et al. [2017]).

This section is structured as follows:
First, the baseline algorithm is explained in Section 3.4.2. Then, a faster method is proposed in Section 3.4.3. Both methods are compared in a number of scenarios in Section 3.4.4.

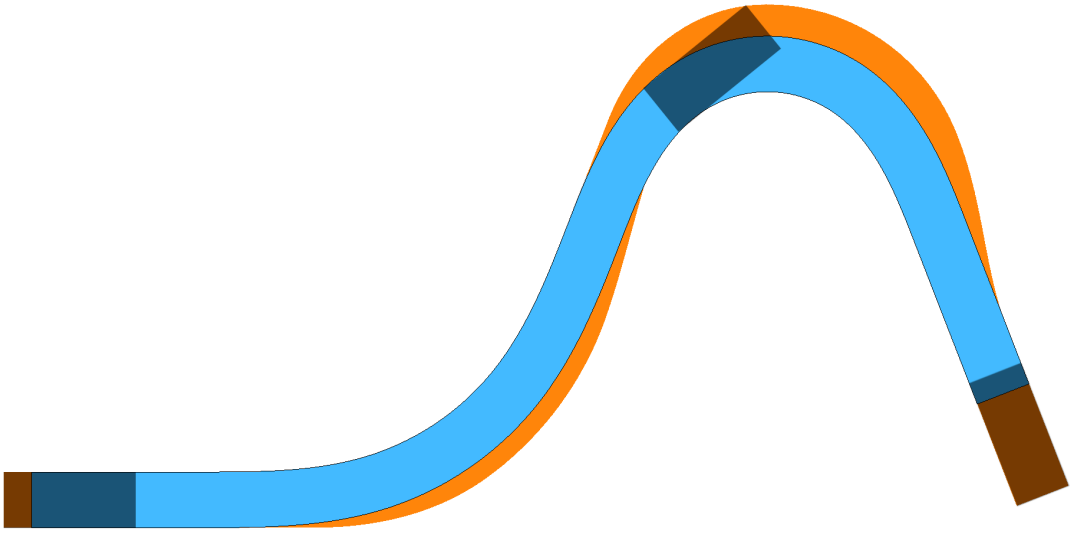


Figure 3.6:
Comparison of a planned trajectory in center-of-rotation coordinates (blue, rear-axle with respective width) and the encompassing joined polygon of bounding boxes sampled every centimeter, i. e., the actual occupancy (orange). Three example bounding-box samples are shown (start, between, end). Note especially the growth of the occupancy with increasing curvature.

3.4.2 Baseline Collision Check

The considered baseline collision-checking method was published in Ziegler and Stiller [2010]. It is parameterizable by the positive uneven number $n \in \mathbb{N}$ of discs. Given n, the discs' radius r and distance constant d are defined as

$$r = \frac{1}{2}\sqrt{\frac{l^2}{n^2} + w^2} \tag{3.3}$$

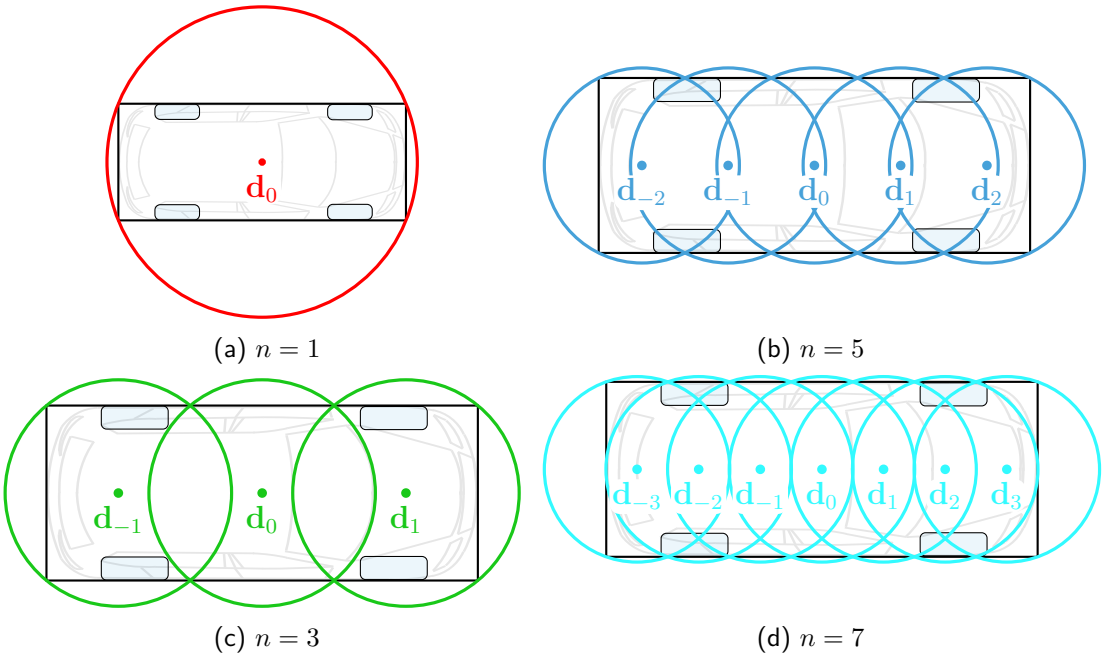
$$d = 2\sqrt{r^2 - \frac{w^2}{4}}\,. \tag{3.4}$$

The disc centers are placed relative to the bounding-box center at $i \cdot d$ along the vehicle's longitudinal axis, where $i_{\max} = \left\lfloor \frac{n}{2} \right\rfloor$ and the index $i \in \{-i_{\max}, -i_{\max}+1, \ldots, i_{\max}\}$. In the center-of-rotation frame C, where motion planning usually takes place, the disc center $\mathbf{d}$ is then given by

$${}^{C}\mathbf{d}_{i,[x]} = \frac{l}{2} + l_r + i \cdot d \tag{3.5}$$

$${}^{C}\mathbf{d}_{i,[y]} = 0\,, \tag{3.6}$$

where l_r is the length from C to the rear end of the bounding box, i. e., it is negative in frame C. A depiction for $n \in \{1, 3, 5, 7\}$ can be found in Figure 3.7.



(a) $n = 1$ (b) $n = 5$

(c) $n = 3$ (d) $n = 7$

Figure 3.7:
Baseline algorithm for the case of $n \in \{1, 3, 5, 7\}$ discs, shown in red, green, blue and cyan, respectively. Note that for one sample, the bounding box (black) is always completely covered. The lateral overestimation shrinks with n, the longitudinal increases.

3.4.3 Faster Collision Check

The main assumption of the proposed method is that the ego vehicle moves momentarily on a circle with radius r_m. The overall traversed ground can then —momentarily— be described by an inner and an outer circle with radii

$$r_i = r_m - b \quad \text{and} \tag{3.7}$$
$$r_o = \text{hypot}(r_m + b, l_f) \ , \tag{3.8}$$

respectively (see: Figure 3.8, red and green lines). In the kinematic case, the inner circle is tangential to the vehicle chassis at its rear axle, the outer circle has the same center and goes through the outer front corner of the vehicle's bounding box.

Given the inner and outer circle radii, the central-circle radius is

$$r_c = \frac{r_o + r_i}{2} \tag{3.9}$$

(see: Figure 3.8, blue line). There, the frontal collision-checking disc center $\mathbf{f}$ is placed with radius

$$r_p = \frac{r_o - r_i}{2} \tag{3.10}$$

at angle

$$\alpha = \arctan\left(\frac{l_f}{r_m + b}\right) \ , \tag{3.11}$$

along the central circle, starting at the negative y-axis (the point marked with a cross in Figure 3.8), resulting in

$$\begin{aligned} {}^{C}\mathbf{f}_{[x]} &= r_c \sin(\alpha) \\ {}^{C}\mathbf{f}_{[y]} &= b - r_p + r_c(1 - \cos(\alpha)) \, . \end{aligned} \tag{3.12}$$

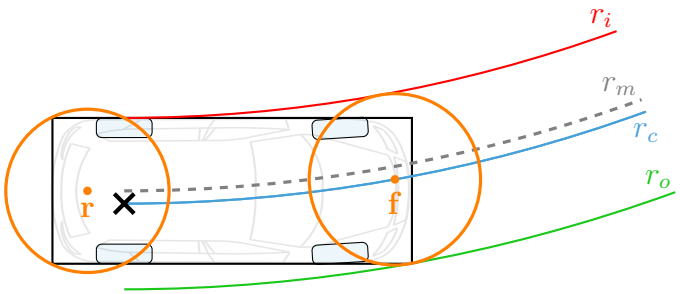


Figure 3.8:
Discs from the proposed algorithm are shown in orange. An inner (red) and outer (green) circle describe the area that is checked for collision, assuming the vehicle's center of rotation travels along the dashed trajectory. The frontal proposed disc is placed on the (blue) central circle such that the out-most point of the contour is covered. The proposed rear disc covers the rear bounding-box corners and the inner circle.

For significant performance improvements, apply the trigonometric identities

$$\begin{aligned} \sin(\arctan(x)) &= \frac{x}{\sqrt{1+x^2}} \\ \cos(\arctan(x)) &= \frac{1}{\sqrt{1+x^2}}\,. \end{aligned} \tag{3.13}$$

In addition to this frontal disc, a second disc is placed at $\mathbf{r}$, covering the vehicle's rear end. Its radius is chosen such that it includes the rear corners of the bounding box as well as the inner circle, i. e.,

$$r_r = \operatorname{hypot}\left(b, \frac{l_r}{2}\right)\,. \tag{3.14}$$

It is placed at

$$\begin{aligned} {}^{C}\mathbf{r}_{[x]} &= l_r + \sqrt{r_r^2 - b^2} \\ {}^{C}\mathbf{r}_{[y]} &= 0\,. \end{aligned} \tag{3.15}$$

Note that (3.12) is not defined for driving exactly straight, as the circle radii tend to infinity. Hence, when $r_m > 10000$, similarly to (3.15), define

$$\begin{aligned} {}^{C}\mathbf{f}_{s,[x]} &= l_f - \sqrt{r_r^2 - b^2} \\ {}^{C}\mathbf{f}_{s,[y]} &= 0\,. \end{aligned} \tag{3.16}$$

Tuning Like the baseline method, the proposed method also has a tuning factor $s' \in \mathbb{R}, s' \geq 1$. Here, s' directly scales both disc radii and thus increases the overall checked occupancy. Additionally, the disc centers are moved such that the outer corners of the bounding box remain exactly on the edge of the respective disc.

The proposed frontal disc's center remains on the central circle, but its angle α changes to $\alpha' = \alpha - \Delta\alpha$, where

$$\begin{aligned} \Delta\alpha &= \arccos\left(\frac{r_c^2 + (r_c + r_p)^2 - r_p^2 s^2}{2r_c(r_c + r_p)}\right) \\ &= \arccos\left(1 + \frac{r_p^2(1 - s^2)}{2r_c(r_c + r_p)}\right) \end{aligned} \tag{3.17}$$

by the law of cosines. The frontal disc center coordinates (3.12), are thus

$$\begin{aligned} {}^{C}\mathbf{f}'_{[x]} &= r_c \sin(\alpha') \\ {}^{C}\mathbf{f}'_{[y]} &= b - r_p + r_c(1 - \cos(\alpha'))\,. \end{aligned} \tag{3.18}$$

Given the structure of

$$\alpha' = \arctan\left(\frac{l_f}{r_m + b}\right) - \arccos\left(1 + \frac{r_p^2(1 - s^2)}{2r_c(r_c + r_p)}\right)\,, \tag{3.19}$$

the trigonometric identities

$$\begin{aligned}\sin(\arctan(x) - \arccos(y)) &= \frac{xy - \sqrt{(1-y^2)}}{\sqrt{1+x^2}} \\ \cos(\arctan(x) - \arccos(y)) &= \frac{y + x\sqrt{(1-y^2)}}{\sqrt{1+x^2}}\end{aligned} \tag{3.20}$$

can be applied for significant performance improvements.

The proposed rear disc's center is shifted along the x-axis such that it always exactly includes the rear corners of the bounding box. Modifying (3.15) and (3.16), this leads to

$$\begin{aligned}{}^{C}\mathbf{r}'_{[x]} &= l_r + \sqrt{s'^2 r_p^2 - b^2} \\ {}^{C}\mathbf{f}'_{s,[x]} &= l_f - \sqrt{s'^2 r_p^2 - b^2}\,.\end{aligned} \tag{3.21}$$

For notational convenience, define $s = 1000(s' - 1)$, where $s \geq 0, s \in \mathbb{R}$. A comparison for $s = 0$ and $s = 10$ is depicted in Figure 3.9.

Discussion With the proposed disc placement and radii, the true occupancy is approximated with only two discs. The computation time for the proposed disc-center placement is similar as for the 3-disc baseline case due to the more complicated formulas. Nevertheless, this is compensated for by the lower number of actual grid evaluations that have to be carried out.

Additionally, the proposed approach yields better results when incrementally building a trajectory. This is due to the fact that the frontal disc already covers the area a baseline rear disc would cover on one of the next samples, assuming near-constant curvature. I. e., potentially dangerous cells are checked beforehand.

While this is conservative for trajectories with decreasing absolute curvature, it usually helps to speed up the search measurably by reducing the number of expansions. Note that checking the rear disc is indeed necessary as especially in cases with increasing absolute curvature the back of the car would not be covered sufficiently. Generally,

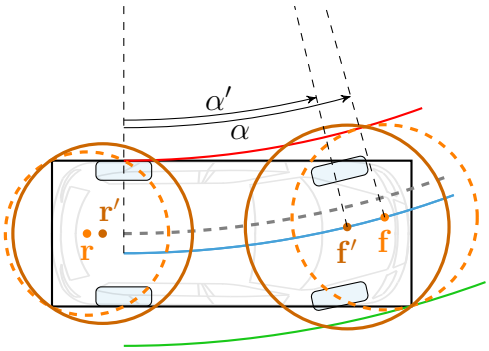


Figure 3.9:
Visualization of the influence of the tuning factor s. The base case $s = 0$ is shown dashed, the case for $s = 10$ solid. Note the shifted circle centers.

the errors due to the constant-curvature assumptions are comparably small, as is shown in Section 3.4.4.

In very tight scenarios, especially when forward-backward maneuvers are necessary, it is recommended to use a hierarchical approach where tighter discs are checked, should the search using the proposed method not yield any result. This is assumed feasible here, since high curvature changes at limited steering rates require lower speeds, and lower speed means more planning time is available. This effect is even more pronounced when changing the driving direction.

The proposed method is not suited for checking single positions, as the shape's center is not covered. This, however, is in most cases not a problem for planning trajectories as it will be checked by later samples. Figure 3.10 shows the trajectory to be completely covered after 3 m. This, of course, is dependent on the geometric parameters of the bounding box and the tuning factor s.

The proposed method is tailored for *normal-sized* vehicles driving comfortably. As the requirements and safety margins depend strongly on the scenario, quality of environment representation etc., no generic evaluation was carried out. Instead, scenarios and environment representations as experienced in off-road scenarios using the geometry of the TAS test vehicles are evaluated.

Note that driving backwards was not explicitly covered here but works analogously.

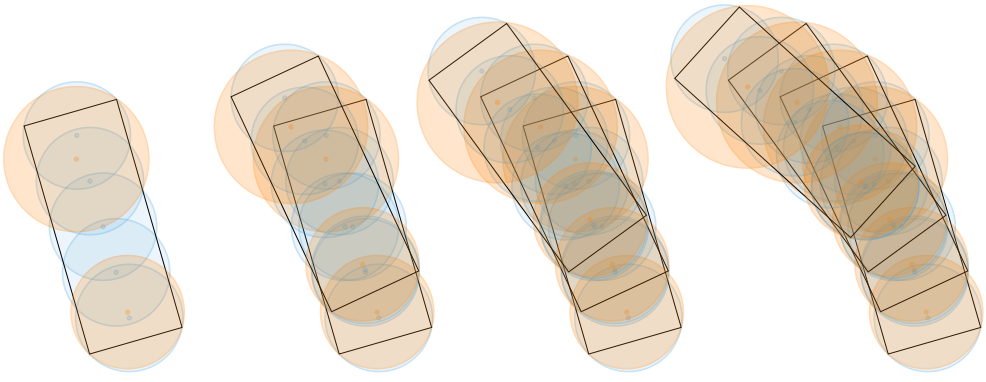


Figure 3.10:
Validity for proposed method with very short trajectories. The vehicle makes a sharp left turn, starting at a curvature of 0.140 $\frac{1}{\text{m}}$ finishing at 0.219 $\frac{1}{\text{m}}$ within 4 m. The baseline method is shown for $n = 5$ case in blue, the proposed for $s = 5$ in orange, the sampling distance D is set to 1 m. From left to right one to four samples are evaluated.

3.4.4 Results

For the evaluation of the algorithms, the performance metrics are introduced first. Let the true area to be checked and its estimates be denoted A and $\hat{A}$, respectively. Furthermore, as checking is often done in discretized grids, let ΔA and $\Delta\hat{A}$ denote their 0.2 m-rastered version.

- *Oversampling* describes the area covered by discs, but not the occupancy. For algorithm j it is $O_j \triangleq \hat{A}_j \setminus A$. Analogously for ΔO_j.
- *Undersampling* describes the area covered by the occupancy, but not the discs. For algorithm j it is $U_j \triangleq A \setminus \hat{A}_j$. Analogously for ΔU_j.
- *Computation Time* comprises the calculation of the disc centers and the evaluation of the obstacle distances.
  For algorithm j it is denoted T_j and given in milliseconds.

Note that undersampling is a measure of the risk for undetected collisions and should consequently be kept as close to zero as possible. Oversampling, on the other hand, gives the conservativeness of the algorithm and is thus a measure of performance (smaller is better) in complex environments.

All results were produced on an Intel i7–3770K CPU @ 3.50 GHz with 24 GB of RAM. The code was written in C++ and compiled with GCC 8.1.1 with -02. No parallelization was used. All evaluation times include a logging overhead. Nevertheless, since both methods are affected the same way, the relative speed-up is not compromised.

The geometric parameters were taken from Munich Cognitive Autonomous Robot Car 3rd Generation (MuCAR-3) and are

$$\begin{aligned} l &= 4.754\,\text{m} & l_f &= 3.781\,\text{m} \\ w &= 1.928\,\text{m} & l_r &= -0.973\,\text{m}\,. \end{aligned}$$

The test vehicle is an SUV, hence its geometric parameters are representative for many autonomous-driving prototypes.

Generic Test For a generic test, the trajectory shown in Figure 3.6 is used. It consists of eight 6 m parts with the curvature changing linearly between the following curvatures:

$$0 \to 0 \to .1 \to .1 \to 0 \to -.2 \to -.2 \to 0 \to 0\,.$$

I. e., there are always two straight, constant-curvature, increasing and decreasing curvature clothoid segments. The performance metrics can be seen in Figure 3.11. Note that no pre-calculation took place, neither for the disc configurations for different trajectory-sample yaw angles nor, in the proposed case, for different curvatures.

Application in Planning Until now, a given complete trajectory was checked. Another use case is to generate a trajectory incrementally, e. g., by the hA* search, as described in Section 3.3. The proposed checking was engineered for this task and hence performs accordingly, as is shown below.

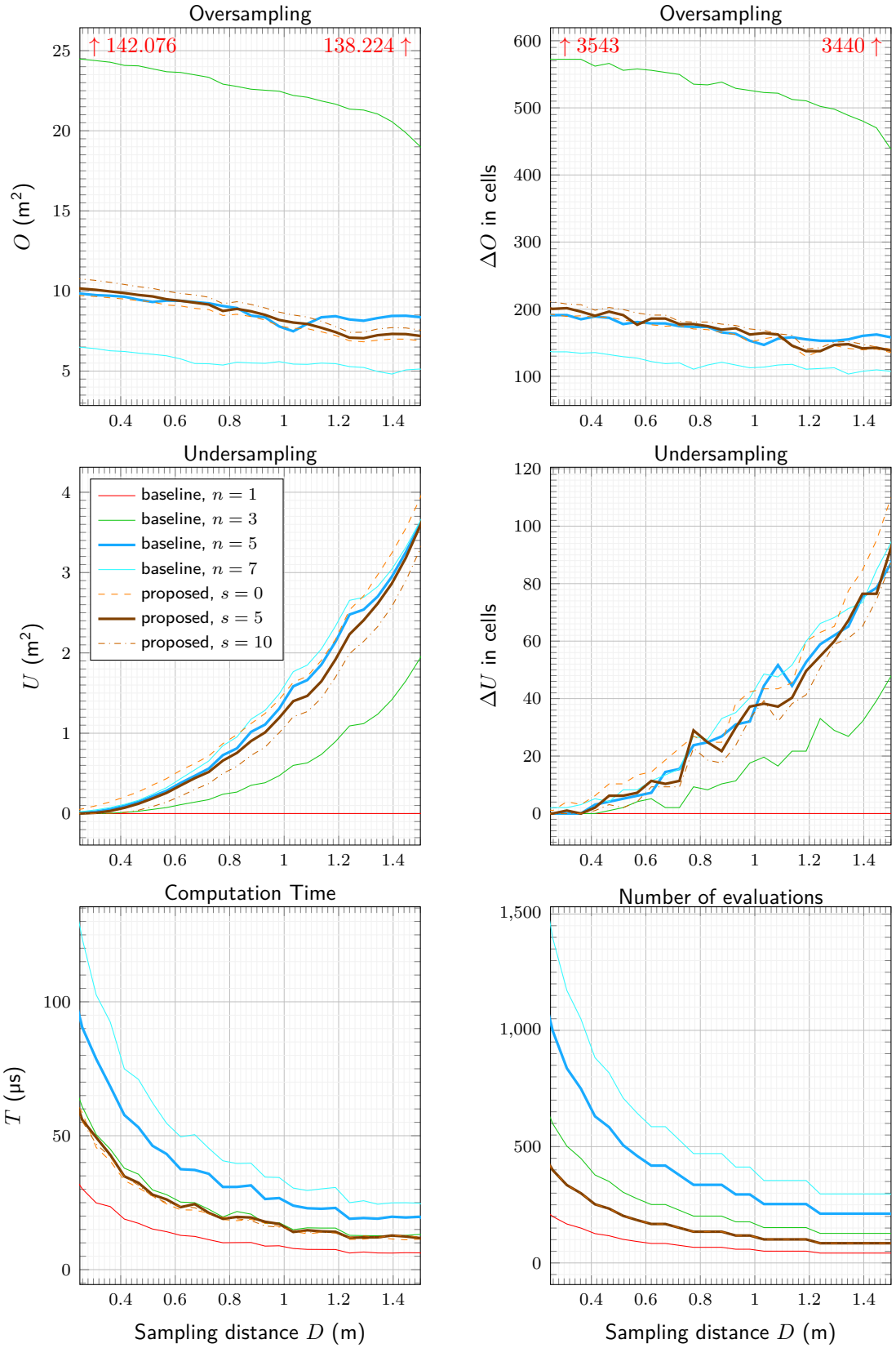


Figure 3.11:
Results regarding Figure 3.6; computation times are averaged over 30 runs. Note that additionally, a distance transform of the obstacle grid is calculated, which takes about 0.63 ms on a 60 m × 60 m grid with 0.2 m resolution. The actual checking then takes about 60 ns per circle center.

Table 3.1: Test results for Figure 3.12

	collision (ms)	expansions	cost
	Unstructured		
baseline, $n = 5$	7.369	1197	46.822
proposed, $s = 0$	4.767	1062	46.586
	Tunnel		
baseline, $n = 5$	104.965	20 888	99.212
proposed, $s = 0$	64.471	18 196	99.256
	Dead end		
baseline, $n = 5$	165.556	29 128	73.028
proposed, $s = 0$	118.044	27 453	72.819

Three test cases are depicted in Figure 3.12:

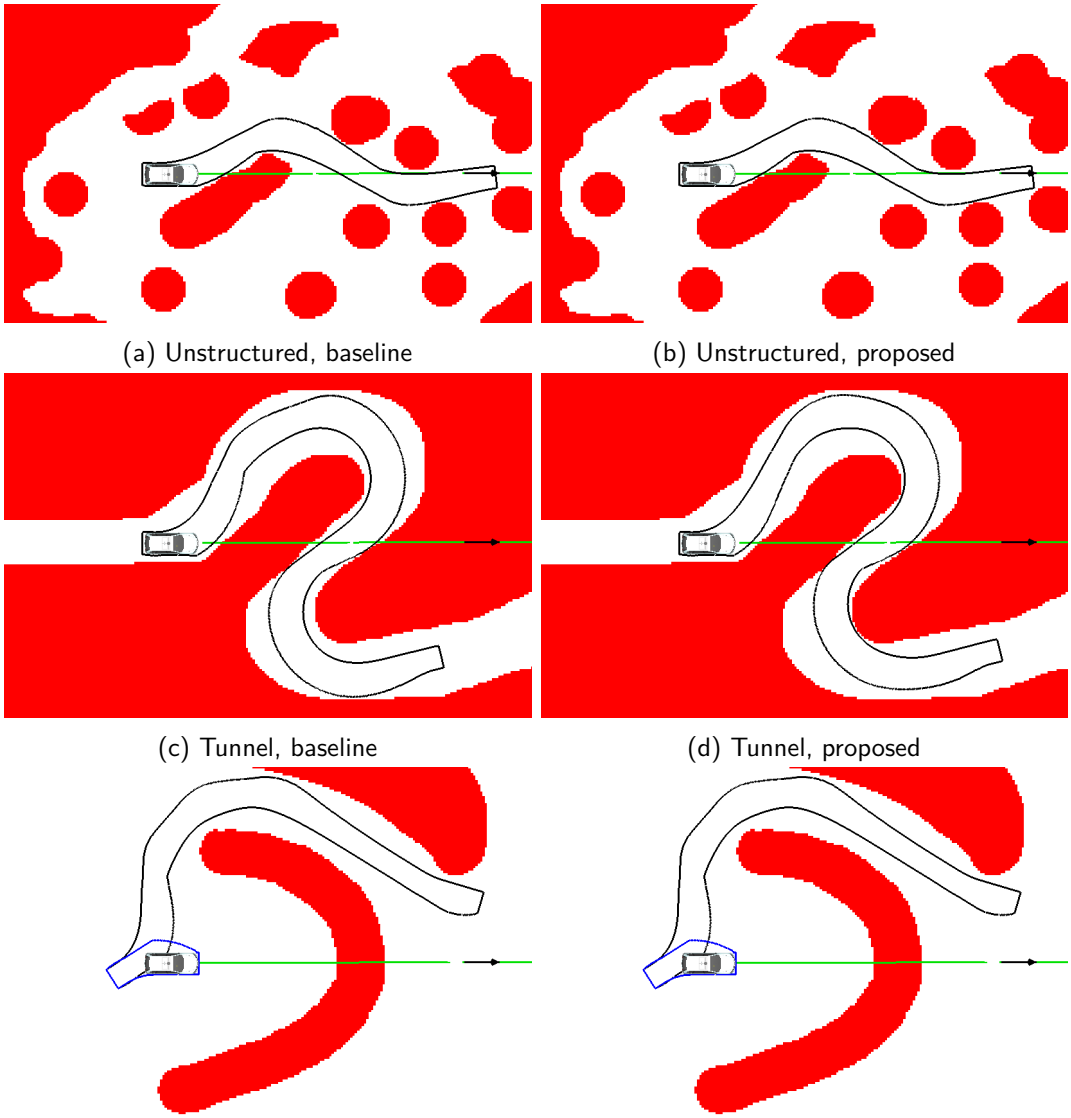
- The first scenario is in an unstructured environment.
- The second features a tight tunnel with an s-shape.
- In the last scenario, a dead end is handled.

The resulting trajectories for the baseline ($n = 5$) and proposed case are very similar, as can be seen visually or from the cost[7] in Table 3.1. As the distance to obstacles is also a factor in the calculation of the cost function, different discs necessarily mean different cost. However, the absolute costs are surprisingly even between both approaches. Note that in the tunnel case, where the proposed method has higher costs, the final solution comes 1.51 m closer to the goal pose.

The number of expansions used for all scenarios was 7 forward and potentially 3 backward. In all cases, the total number of expansions is roughly 10 % lower for the proposed method. This is due to the predictive nature of the disc placement, i. e., branches that are highly likely to lead to collisions will not get expanded as long as there are more promising ones. The smaller number of expansions together with the lower computational cost per expansion result in a net speed gain of 35.3 %, 38.6 % and 28.7 % for the three respective scenarios.

The large total number of expansions is due to the hA* planner being designed (and tuned) for sensor-based planning. I. e., application on an autonomous vehicle where obstacles are only detectable with direct line of sight and at a limited range. This means a planner designed for large a-priori grid maps will most likely outperform the absolute numbers given in Table 3.1. The relative performance gain, however, is not compromised by that.

[7] The cost term —in the context of discussing the performance gain by the collision-checking algorithm— is only an abstract value to indicate the similarity of the solutions on a better level than just visually. The assumption, which holds for the chosen planner, is that the cost encodes, among others, curvatures, curvature change rates and path length.



(a) Unstructured, baseline (b) Unstructured, proposed

(c) Tunnel, baseline (d) Tunnel, proposed

(e) Dead end, baseline (f) Dead end, proposed

Figure 3.12:
Three test cases (top to bottom) using the baseline (left) and the proposed (right) collision-checking algorithm. Obstacles are depicted red, a global path to follow in green. The accumulated occupancy of the planned trajectory is contoured in black (or blue for backwards). Only minimal differences in the resulting occupancies are detectable — if any. Nevertheless, the speed-up using the proposed algorithm is about 30 %, see Table 3.1.

3.5 Path–Velocity Decomposition

A classical approach to speed up the planning of trajectories is the so-called Path-Velocity Decomposition (PVD), postulated in Kant and Zucker [1986]. The procedure is to first plan a path, e. g., through an obstacle grid, using only spatial information and transitions. Then, in a second step, the velocities for each point are determined, i. e., temporal information is added.

This classic approach is applicable whenever the trajectory is highly constrained by the static environment. This could be the case in an urban environment, when the ego vehicle is required to stay in a narrow lane and is further constrained by badly parked cars on the side.

Its main disadvantage is that once the path is planned, it cannot be adapted to moving objects. For instance, emergency-evasion maneuvers or overtaking another vehicle which moves unpredictably is not possible, because this would require a change in the spatial component of the trajectory.

Nevertheless, the PVD lends itself very well for driving in off-road scenarios. Here, the static environment is arbitrarily complex and interactions with other dynamic objects are infrequent or even non-existent.

Another use case is *platooning*. Platooning describes one or more vehicles following a lead vehicle. This scenario under different circumstances (like at night, at high distances) is a major research topic at TAS (see Fries et al. [2012, 2013a,b, 2015], Fassbender et al. [2015, 2016a, 2017a]). Here, the goal is to exactly follow a lead vehicle in unstructured terrain where even the most advanced algorithms fail to distinguish between potentially dangerous and safe-to-drive areas, e. g., high grass hiding (negative) obstacles.

This section covers two variants of the PVD. Section 3.5.1 covers the algorithm used in combination with the introduced planning approach (see Section 3.3). Section 3.5.2 covers applications of the leader-follower case. The multi-vehicle case is considered separately in Section 3.6.

3.5.1 Speed Profiles

The contribution presented here is a very fast and reliable heuristic algorithm which generates (suboptimal) speed profiles given a path and a number of constraints. The original plan was to use it as a fall-back solution in case more advanced algorithms failed. However, the quality of the results turned out to be sufficient (and there were always more pressing matters). Hence, it was the sole algorithm used outside convoying, even where hard constraints due to tight maneuvering or high precision requirements were imposed as, e. g., in Heinrich et al. [2018c].

Problem The problem statement is: Generate a valid trajectory by finding a smooth[8] speed profile given a sequence of spatial reference points. Here, paths consisting of clothoid segments (see Appendix A.3) described by

$$\mathbf{r}_{[i]} = (l_i^*, c_i^*), \qquad i \in \{0, \dots, N\} \ , \tag{3.22}$$

are considered, where:

l_i^* length at beginning of i$^{\text{th}}$ segment,
c_i^* initial curvature of the segment.

A trajectory is valid if all kinematic and dynamic constraints are satisfied. The kinematic constraints are assumed to be already respected by the spatial path. Hence, only the dynamic constraints are considered when determining the temporal component.

Limits on jerk, acceleration and speed are considered. In their implementation, there is an absolute limit on the jerk and there are different limits for accelerating/decelerating which are all independent of the current speed and acceleration. This means the imposed limits are comfort limits — the algorithm is intended to provide smooth driving, not to win a race or handle emergency-critical maneuvers. Nevertheless, the heuristic is easily extendable to using a state-dependent set of constraints.

Algorithm The algorithm is described below and visualized in a flow chart in Figure 3.14. The pseudo code for the functions in the diagram can be found in Algorithm 1 and 2. Finally, a schematic result is shown in Figure 3.13.

In an initialization step, an upper speed limit is computed per path point. The maximal achievable speeds per path point ($\bar{v}_i$) given the maximal lateral acceleration ($\bar{a}_{\text{lat}}$) is calculated via

$$\bar{v}_i = \sqrt{\bar{a}_{\text{lat}} r_i} = \sqrt{\frac{\bar{a}_{\text{lat}}}{c_i^*}} \, . \tag{3.23}$$

They are depicted as black crosses (**+**) in Figure 3.13.

[8] The algorithm is posed generically and was implemented with both constant acceleration and constant jerk transitions. Some clothoid-segment borders are used as extremas where $a, v = 0$.

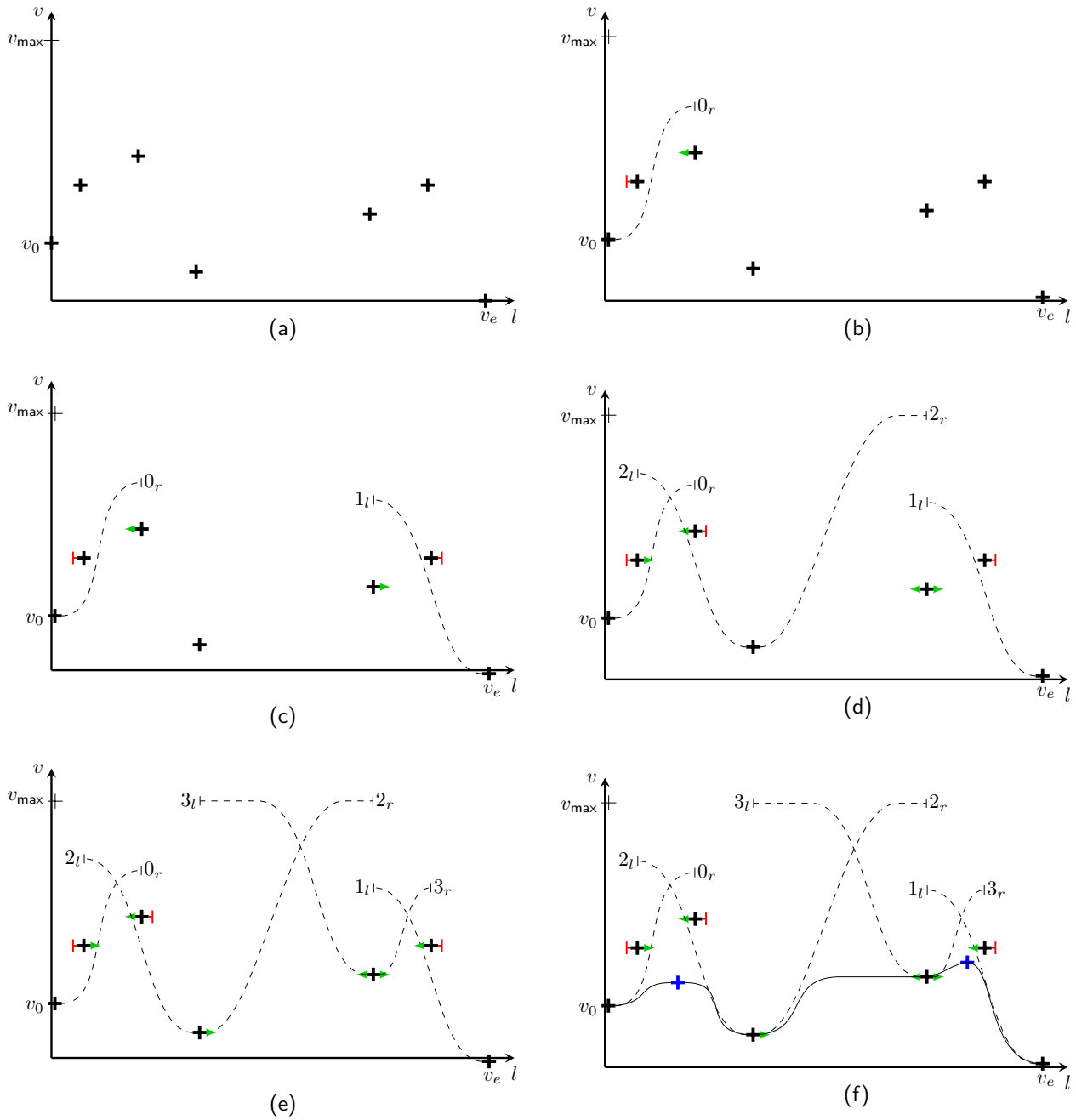


Figure 3.13:
Schematic example of the heuristic speed-planning algorithm. In (a), the maximum speeds are calculated for each path point $\mathbf{p}_i$. In (b), the start point is expanded, finding the first point unreachable (red) and the next point reachable (marked green). In (c–e), the other points are expanded in ascending order of maximum speed. In (f), maximum reachable intermediate points are found iteratively and the final speed profile is generated.

Then, a queue is generated comprised of all path points. The algorithm requires two relations:

- Priority: it is determined by their $\bar{v}$ value, with the lowest first.
- Spatial sequence: which point comes before/after on the path; it saved and denoted *left* and *right*, respectively.

Note that the first path point is considered fixed and thus always evaluated first. This is necessary since the algorithm allows to lower the speed per point, which is not viable for the initial condition. The order inside the queue is depicted by the number on each expansion in Figure 3.13.

For all points in the queue, iterate over the left and right neighbors until the next reachable point is found (see Figure 3.14): For each consecutive neighbor, check which speed is reachable from the current querying point, given the distance along the path between the two points and the currently applicable limits (see Code 1). If a neighbor j is not reachable, save the maximal reachable speed at its path length as $\bar{v}_j^{\{\leftarrow,\rightarrow\}}$, where the arrows denote if it was visited from left or right, respectively.

Algorithm 1: `transit(from` i`, to` j`)`

$\Delta l \leftarrow l_j - l_i$; `// ← or → based on sign of` Δl
if $\Delta l > 0$ **then**
  $\bar{v}_j^{\leftrightarrow} \leftarrow \bar{v}_j^{\leftarrow}$;
else
  $\bar{v}_j^{\leftrightarrow} \leftarrow \bar{v}_j^{\rightarrow}$;
end
$\bar{v}_j^{\leftrightarrow} \leftarrow$ `maxSpeedForDistance(`Δl`,` v_i`)`;
return $\bar{v}_j^{\leftrightarrow} > \bar{v}_j$;

If a neighbor is reachable from one side (▸), but not the other (⊣), An intermediate point can be used in its stead (see Algorithm 2). This is done by iterating over possible speeds for the intermediate point and checking whether it is reachable given the current temporal transition. The iteration range is between the maximal upper speed-limit ($\bar{v}$) of the points that are only reachable from one side and the maximal upper speed-limit of their lower neighbors ($\underline{v}$) which are guaranteed reachable. Intermediate points are depicted as blue crosses (**+**) in Figure 3.13.

Algorithm 2: `findIntermediate(left, right, nodes between` i, j`)`

$\underline{v} \leftarrow \max(\bar{v}_{\text{left}}, \bar{v}_{\text{right}})$;
$\bar{v} \leftarrow \min(\bar{v}_i^{\leftarrow}, \bar{v}_i^{\rightarrow}, \bar{v}_j^{\leftarrow}, \bar{v}_j^{\rightarrow})$;
$l_{\text{max}} \leftarrow l_{\text{right}} - l_{\text{left}}$;
do
  $\bar{v}_{\text{mid}} \leftarrow$ `nextSampleBetween(`$\bar{v}, \underline{v}$`)`;
  $l^{\leftarrow} \leftarrow$ `distanceForSpeedTransition(`$\bar{v}_{\text{left}}, \bar{v}_{\text{mid}}$`)`;
  $l^{\rightarrow} \leftarrow$ `distanceForSpeedTransition(`$\bar{v}_{\text{mid}}, \bar{v}_{\text{right}}$`)`;
while $l^{\leftarrow} + l^{\rightarrow} > l_{max}$;
$\mathbf{p}_{\text{mid}} \leftarrow$ `pointAtPathLengthWithSpeed(`$l_{\text{left}} + l^{\leftarrow} + \frac{l_{\text{max}} - (l^{\leftarrow} + l^{\rightarrow})}{2}, \bar{v}_{\text{mid}}$`)`;

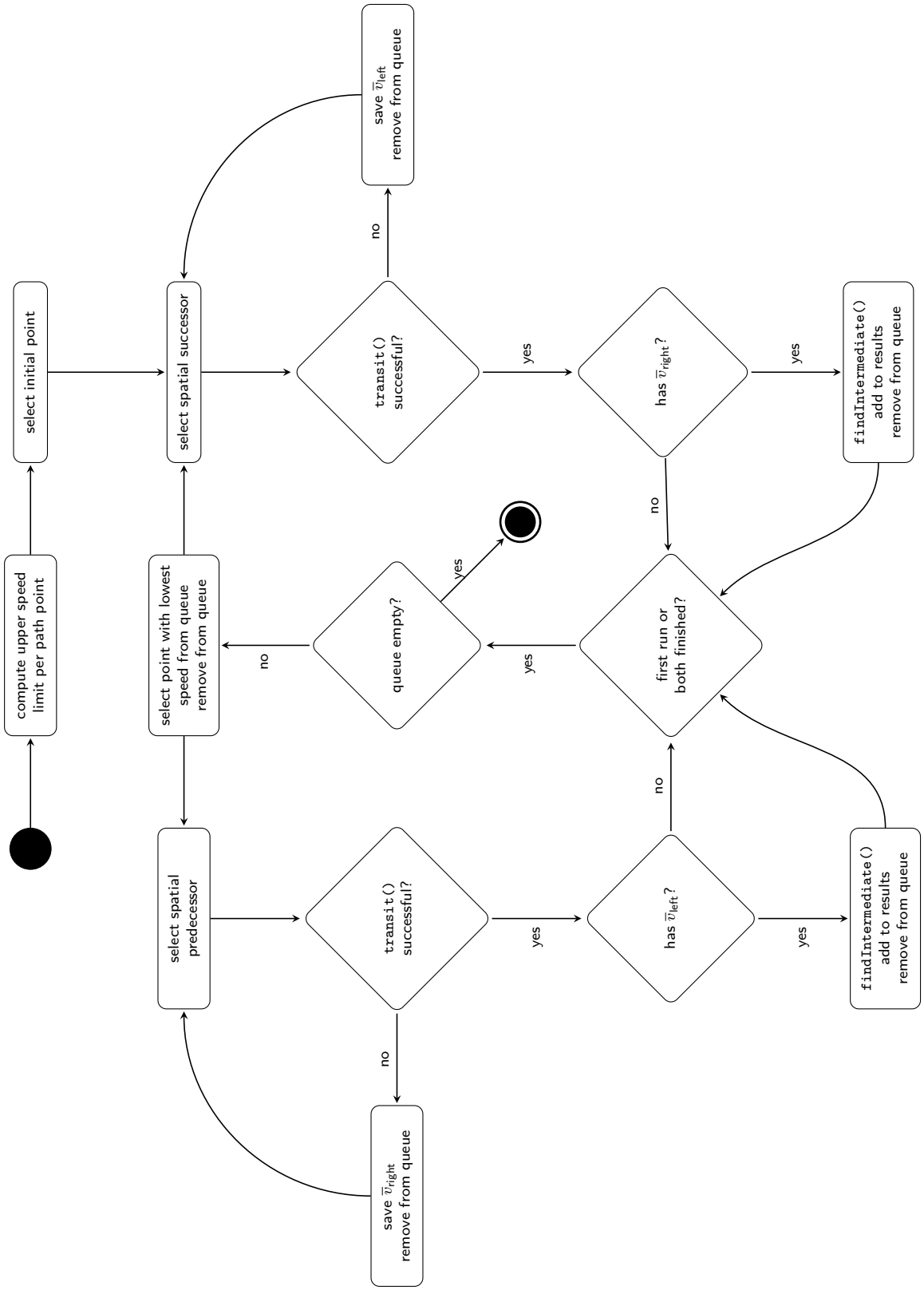


Figure 3.14: Heuristic speed-planning algorithm: flow chart (UML activity diagram)

3.5.2 Leader-Follower System

Here, a specialized motion planner for single leader-follower systems is discussed. An additional requirement is imposed: follow a lead vehicle as precisely as possible. Since the multi-follower case is a problem an order of magnitude harder, it is discussed separately in Section 3.6. This section was partly pre-published in Fassbender et al. [2017b].

This approach is in the path-velocity decomposition section even though, in a first step, a trajectory is generated. Yet, its speed profile is only used as an upper bound and the actual longitudinal control is done separately, as will be detailed in the following.

The initial trajectory generation by Fassbender et al. [2017b] works as follows: Consider a given path consisting of measurements of lead-vehicle positions (see Figure 3.15). As an initial guess, generate a trajectory using pure-pursuit control (see Snider [2009]), which is simulated forward along that path. The guess is then approximated by a fixed number of clothoid arcs N, because a fixed low number of optimization variables is beneficial for the latter optimization.

The trajectory generation is formulated as a Sequential Quadratic Programming (SQP) problem as follows:

$$\min \quad \sum_{i=0}^{N} k_{\dot{\delta}} \dot{\delta}_i^2 + k_{a,\mathsf{lat}} a_{\mathsf{lat},i}^2 + k_{a,\mathsf{lon}} a_{\mathsf{lon},i}^2 + k_{\Delta d} \Delta d_i^2 + k_{\Delta y} \Delta y_i^2 + k_{\Delta\psi,\mathsf{f}} \Delta\psi_{\mathsf{f},i}^2 \tag{3.24}$$

$$\text{s.t.} \quad \underline{a}_{\mathsf{lat}} < a_{\mathsf{lat}} < \bar{a}_{\mathsf{lat}} \tag{3.25}$$

$$\underline{a}_{\mathsf{lon}} < a_{\mathsf{lon}} < \bar{a}_{\mathsf{lon}} \tag{3.26}$$

$$\underline{\delta} < \delta < \bar{\delta} \tag{3.27}$$

$$\underline{\dot{\delta}} < \dot{\delta} < \bar{\dot{\delta}} \tag{3.28}$$

where

δ	steering angle (comfort term)
$\dot{\delta}$	steering rate (comfort term)
a_{lat}	lateral acceleration (comfort term)
a_{lon}	longitudinal acceleration (comfort term)
Δd	distance-keeping violation (tracking term)
Δy	path-tracking deviation (tracking term)
$\Delta\psi_{\mathsf{f}}$	final heading difference (tracking term)
$k_{(\cdot)}$	weighting factor for the respective variable
$\underline{(\cdot)}/\bar{(\cdot)}$	lower limit / upper limit of the respective variable

The objective incorporates both comfort terms and tracking terms. Constraints are imposed on kinematic and dynamic inputs and states of the ego vehicle: steering-angle, steering-rate and longitudinal- as well as lateral-acceleration are kept in a valid range.

Initial guess, objective and constraints are fed to a SQP solver, namely the model-based control library called "yane" (see Grüne and Pannek [2017]), where a solver by Schittkowski [2010] is utilized. It proved vital for the solver's stability and performance to provide the derivatives of the constraint and objective functions, which can be obtained analytically.

The result of the optimization is following the lead vehicle exactly while keeping constraints on the ego dynamics and providing the best-possible smoothness. The initial trajectory, however, has no safety guarantees since a minimum distance between the vehicles is not imposed as hard constraint. This is a conscious design decision.

The architectural idea and contribution in this work is as follows:

- Use the optimization to couple longitudinal and lateral motion and generate a smooth nominal trajectory. This is done in the motion-planning step, i. e., with a 100 ms cycle time.
- High-level motion control, however, directly uses the latest object information for pure longitudinal control in its 10 ms cycle time. This improves the system's reactiveness and thus passenger comfort. What is more, in edge cases, the best-possible safety is provided. Note that in order to not violate lateral constraints, the optimized speed profile is used as an upper bound for the control part.

For keeping both parts consistent, both utilize the same Adaptive Cruise Control (ACC) rule (see Section 4.3.4). However, the optimization is tuned slightly more aggressively. Thus, the trajectory always gives an upper bound on the speed that is considered comfortable.

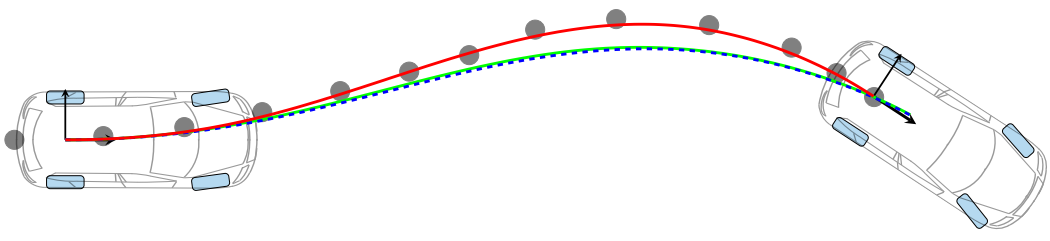


Figure 3.15:
The follower (left) follows a lead vehicle (right). Measured positions are shown as gray dots. The pure-pursuit (green) and approximated (blue, dashed) trajectories are virtually indistinguishable. Using the latter as an initial guess, the red trajectory is generated. Taken and slightly modified from Fassbender et al. [2017b].

Therefore, the disadvantages of trajectory control and the classical pure longitudinal control are overcome, leading to a superior driving performance.

An advantage over pure trajectory control is faster reaction times and the ability to violate comfort limits in edge cases (see Figure 3.16). No hard constraints on the critical distance-tracking term in the planning provides greater numeric stability, while safety is provided by the control part.

The advantage over pure longitudinal control is that lateral limitations are considered beforehand in the planning step. This improves the stability of the path-tracking since the control reference always remains within a comfortable lateral dynamics range, i. e., it is guaranteed drivable within the actuation limits. Consequently, braking happens proactively before reaching safety-critical regions (see Figure 2.10).

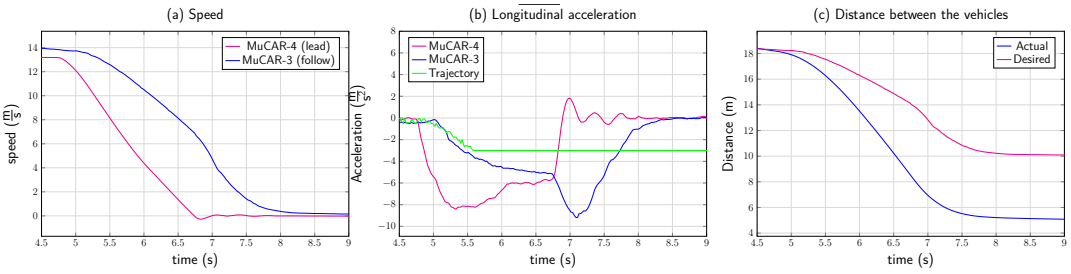


Figure 3.16:
Critical break scenario: Driving at 13.3 m/s (a), Munich Cognitive Autonomous Robot Car 4th Generation (MuCAR-4) initiates a hard braking maneuver (b). As a result, the actual distance between the vehicles falls below the trajectory planner's desired distance d_{des} (see Equation (4.27)), shown in (c). While this causes the planner to command its maximum allowed deceleration of $-3\,\text{m/s}^2$ from 4 s onward (b, green), MuCAR-3's controller overrides the trajectory's speed profile, decelerating at a rate of up to $-9.2\,\text{m/s}^2$ in order to keep a safe distance. As MuCAR-4 comes to a halt, MuCAR-3 keeps approaching it very slowly until the absolute minimum distance of 5 m is reached. Note that, for safety reasons, the leader's position and speed were obtained from ground truth (Real time kinematic (RTK)-GNSS) rather than tracking data in this scenario.
Taken and slightly modified from Fassbender et al. [2017b].

3.6 Platooning

In addition to the single-follower convoy case (section above), in this section, the behavior of multiple vehicles following each other is considered. The contribution here is the extension of this well-studied but still not completely solved problem (see Alfraheed et al. [2011], Bergenhem et al. [2012]) by providing a decentralized algorithm that can be used to ensure stability both longitudinally and —what is often disregarded— laterally. This is achieved without infrastructure, i. e., base stations for Vehicle-to-Vehicle (V2V) communication or lane markings for lateral stabilization. Note that this approach works with a manually driven lead vehicle which makes the problem harder since its future trajectory is unknown. This section was partly pre-published in Heinrich [2015], Heinrich and Wuensche [2017].

It has been shown (e. g., see Marsden et al. [2001], Kesting et al. [2007]) that chaining even well-designed stable leader-follower systems potentially leads to an unstable chain. However, with current sensor technology, only the information of the respective lead vehicle can be sensed reliably. Thus, information on disturbances ahead are only relayed through the action of the respective predecessors. This effect can be mitigated by using V2V communication, which is utilized here, albeit under severe limitations to the data rates.

This section is structured as follows:
First, an introduction to the overall decentralized global control scheme is given in Section 3.6.1. Then, the longitudinal and lateral stabilization are described in Sections 3.6.2 and 3.6.3, respectively. Finally, results are presented in Section 3.6.4.

3.6.1 Decentralized Global Control

Central control of a platoon is assumed to be infeasible in off-road settings. This is due to several reasons, for instance, bandwidth and/or coverage limitations. Additionally, it is known that local ACC control quality, i. e., the behavior of each successive platoon member, diminishes with growing platoon length if only local information is used. Hence, the proposed method is a *decentralized global control algorithm*.

The proposed method requires that each vehicle in a platoon is capable of following its respective lead vehicle. I. e., by means of on-board sensors, each vehicle is able to perceive its predecessor's speed, distance and relative position. Furthermore, it requires each vehicle to have at least limited V2V-communication capabilities. Then, a platoon can be stabilized as follows:

Let $\mathbf{x} \in \mathbb{R}^{\eta \cdot n}$ denote the state of the whole system. For η followers, it consists of substates $\mathbf{x}_{[i]} \in \mathbb{R}^n, i \in \{1, \ldots, \eta\}$. Each $\mathbf{x}_{[i]}$ comprises observations —such as the distance— by follower i and is broadcast to all other followers at a chosen time interval. Usually a fraction of the total bandwidth is allocated and split equally over members, resulting in how often each can send.

Assuming that given the full system state at time k, $\mathbf{x}^{\mathsf{k}}$, there exists an optimal solution for the system states for at least h time steps: $\mathbf{x}^{\mathsf{k}}, \ldots, \mathbf{x}^{\mathsf{k+h}}$. This solution

is saved in a state buffer $B \in \mathbb{R}^{\eta n \times \lceil \nu h \rceil}$, where $\nu > 1 \in \mathbb{R}$ is a tuning parameter dependent on the system and communication delays. The control law $f : \mathbb{R}^{\eta n} \to \mathbb{R}^{\eta n}$ leading to $\mathbf{x}^{\mathsf{k}+1} = f(\mathbf{x}^{\mathsf{k}})$ is assumed to be known by all convoy members.

Note that f depends on the individual capabilities of each follower. A heavier vehicle, for instance, is typically not able to accelerate as quickly as a lighter one. Hence, f has to be generated accordingly when the platoon is formed.

All received and transmitted substates $\mathbf{x}_{[i]}^{\mathsf{k}_i}$ are saved in a time-sorted update queue $Q \in \mathbb{R}^{n \times q_{\mathsf{max}}}$. Once a new substate $\mathbf{x}_{[j]}^{\mathsf{k}_j}$ is received, it is placed in Q. Then B is updated at k_j, replacing the predicted $\mathbf{x}_{[j]}^{\mathsf{k}_j}$ with the measured one.

Note that the substate is completely replaced and no form of probabilistic update is applied. This is due to the incoming message being the result of an already probabilistic tracking of the closest convoy member. It is hence the best guess available for that state and any further modification would mean filtering already filtered results.

From the time point k_j forward, B is updated/predicted step-by-step using the known control law f until the next queued substate — which is then used to update the state accordingly. This is repeated until the newest available update in Q was evaluated h times. A graphical representation of this update-prediction cycle can be seen in Figure 3.17, (simplified) pseudo code is given in Algorithm 3.

Algorithm 3: Reference Generation

```
while Q not empty do
    q ←top(Q) ;                                   // take the oldest update
    pop(Q) ;                                      // and remove it from queue
    if q_t > now −νh then
        pushOntoNextQue(q) ;                      // remember for next time
        x ← B(q_t) ;                              // init state from prediction
        i ← q_index ;                             // get index and update accordingly
        if i = 0 then
            v_lead ← u_v ;                        // convoy lead is observed by first follower
        else
            x_{v,[i]} ← q_v − v_lead;
            x_{d,[i]} ← q_d−desiredDistance(q_v);
        end
        h' ← min(h,top(Q)_t − q_t) ;              // number of steps to next update
        for τ = q_t + 1 ... q_t + h' do
            B(τ) ← f(B(τ − 1)) ;                  // iteratively predict state
        end
    end
end
```

In a nutshell, each convoy member calculates the global solution locally and uses the solution as control reference. Thus, all followers try to reach the global solution and error accumulation over the platoon is negated. Additionally, local measurements —which are available at a higher frequency/lower delay/lower discretization error—

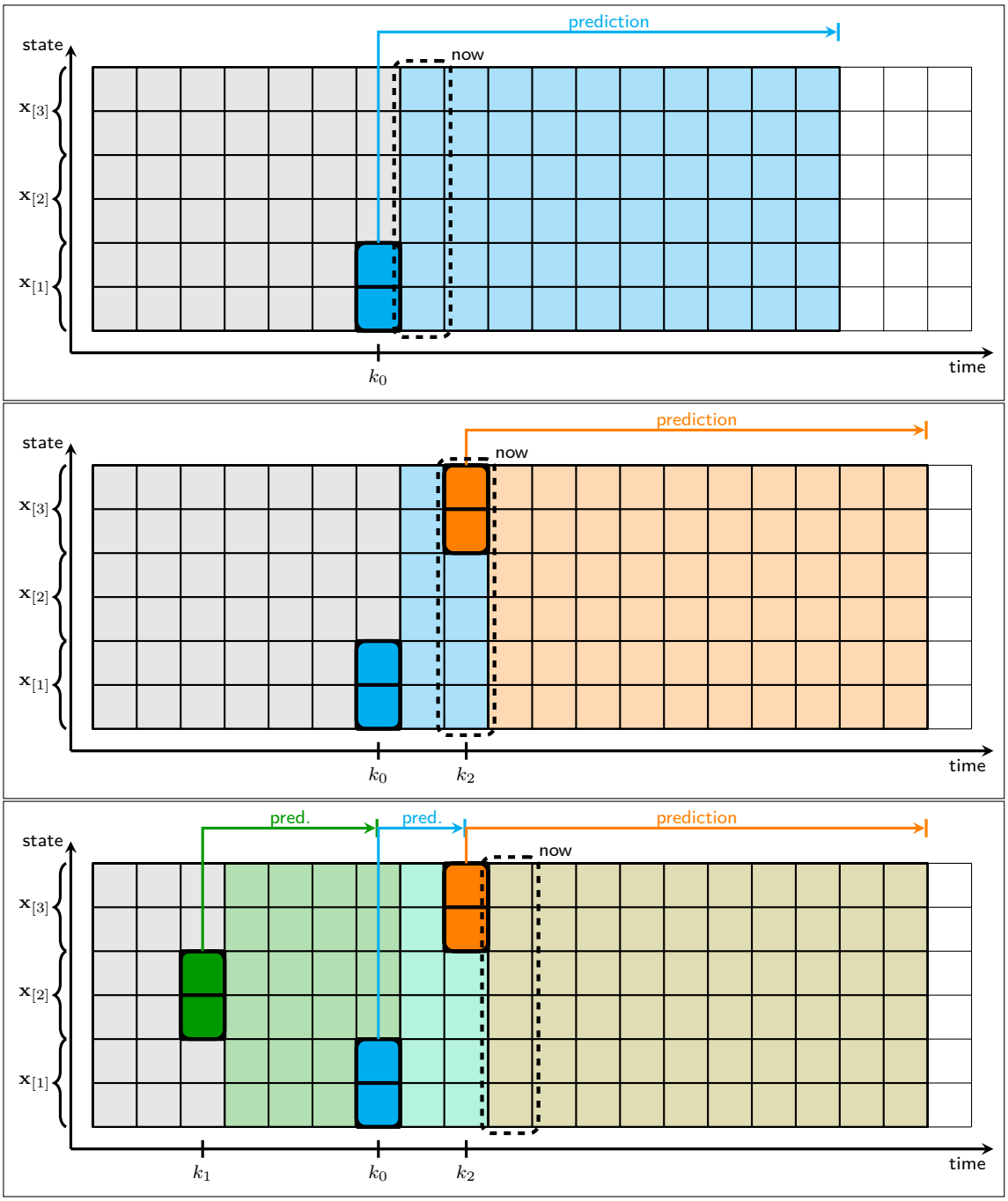


Figure 3.17:
Depiction of the update-prediction cycle of the state buffer (here $\eta = 3$). Three time steps where three measurements arrive are shown, one per follower. The measurements are depicted with bold borders, their respective prediction is color-coded to the measurement. While in the first two cases no additional prediction is necessary, the delayed measurement of vehicle 2 in the third case leads to three prediction steps.
Taken and slightly modified from Heinrich and Wuensche [2017].

are used for tracking this reference. Hence, local stability can be ensured as well. Details regarding delay stability are given in Sections 4.3.2 and 4.3.4.

Note that consistent timings are paramount for the proposed framework. Hence, all transmitted data carry timestamps, i. e., the points in time when the sensor reading leading to the transmitted estimates took place. In order to be independent of the local computer clocks, and thus the need to synchronize them, timestamps are published as GNSS time and converted to local time for each follower.

Since the system is required to work without infrastructure, it is important to minimize bandwidth usage. Some more technical details on the data structures and message routing used to make it work using only 19.2 $\frac{\text{kbit}}{\text{s}}$ can be found in Heinrich and Wuensche [2017].

3.6.2 Lateral-Offset Correction

Keeping a platoon laterally stable is a solved problem, as long as there are local features like lane markings/curbs or a precise and stable GNSS services available. In off-road scenarios, however, neither of those is reliable. The latter due to difficult GNSS conditions and/or potential jamming/spoofing. The former due to the terrain being potentially very unstructured. This leaves room for interpretation as to which is the best possible path. Note that it is assumed to be dangerous to stray from the exact path the lead vehicle drove.

Assuming that the on-board perception is able to provide a sufficient estimate of each predecessor's path, exact following should not pose an issue. However, a problem may arise once systematic errors in each follower's control accumulate. This can happen, for instance, due to difficult terrain conditions like lateral inclines or very muddy ground. From the motion-planing perspective, this challenge is solved as follows:

The convoy leader drives along a path $\mathbf{P}_0$. Each follower $i \in \{1, \ldots, \eta\}$ is assumed to be able to estimate its predecessor's path $\mathbf{P}_{i-1}$. For this, the predecessor's poses $\mathbf{p}_{i-1,[t]}$ are estimated using local sensors and tracked over time.

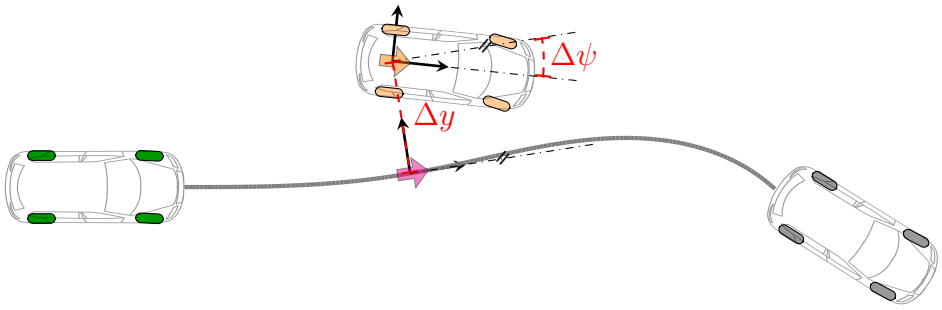


Figure 3.18:
Correcting the reference path by known lateral error $(\Delta y, \Delta\psi)$ of the predecessor. The ego vehicle, its predecessor and the convoy leader are shown as contours. The tracked and corrected predecessor poses are shown as arrows. The convoy leader's path is drawn in gray.

Assume follower i at time t has a lateral control error $\varepsilon = \begin{bmatrix}\Delta y & \Delta\psi\end{bmatrix}^\mathsf{T}$ in tracking this desired path $\mathbf{P}_{i-1}$, where

Δy is the lateral displacement and
$\Delta\psi$ is the yaw error, respectively.

This error is transmitted to its successor: follower $i+1$.

Follower $i+1$ then transforms its respective perceived predecessor's pose $\mathbf{P}_{i,[t]}$ by the inverse of ε. This is done by simply rotating each estimate by $-\Delta\psi$ and then shifting them along their y axes by $-\Delta y$. The resulting corrected path $\mathbf{P}_i^*$ thus resembles $\mathbf{P}_{i-1}$ (see Figure 3.18). I. e., observable control errors are not accumulated over the followers.

Nevertheless, path generation and trajectory control rely on the estimate of the (corrected) leader poses $\mathbf{P}_i^*$. If these estimates have a systematic error, this will —again— accumulate over the followers. A way to bound this would be to transmit GNSS positions.

However, GNSS positions from vehicles with different antennas/receivers often differ when traversing the same path. Hence,the relative GNSS offsets would need to be estimated and transmitted. This is possible, though not practicable, given the limited radio bandwidth.

Note that this framework facilitates obstacle avoidance. If one follower detects an obstacle on its lead-vehicle path estimate (e. g., lost cargo) and drives around it, this is not a control error. Hence, its successors will have the corrected path as reference rather than the one taken by the convoy leader.

3.6.3 Longitudinal Stabilization

The lead-vehicle trajectory is assumed unknown, either due to the vehicle being driven manually or because it is continuously changing due to adverse environmental conditions — as is often the case when driving off-road. The following time-discrete system model is used: For follower i, the absolute and the desired substate are denoted

$$\xi_{[i]} = \begin{bmatrix} d_i \\ v_i \\ a_i \end{bmatrix} \quad \text{and} \quad \xi_{\text{des},[i]} = \begin{bmatrix} d_{\text{min}} + t_{\text{gap}} v_i \\ v_0 \\ a_0 \end{bmatrix}, \tag{3.29}$$

where d_i, v_i and a_i denote the distance to its predecessor, the velocity and the acceleration of member i, respectively. Further,

d_{min} is the minimal desired distance (see Equation (4.27)) and
t_{gap} is a constant-time gap (see Equation (4.28)).

The substate i follows as

$$\mathbf{x}_{[i]} = \xi_{[i]} - \xi_{\text{des},[i]} = \begin{bmatrix} \Delta d_{\text{i}} \\ \Delta v_{\text{i}} \\ \Delta a_{\text{i}} \end{bmatrix}. \tag{3.30}$$

Note that the lead vehicle is assumed to drive with constant velocity, i. e., $a_0 = 0$.

For a single leader-follower ($\eta = 1$) system, the dynamics are described by

$$\underbrace{\begin{bmatrix}\Delta d^{k+1}\\ \Delta v^{k+1}\\ \Delta a^{k+1}\end{bmatrix}}_{\mathbf{x}^{k+1}} = \underbrace{\begin{bmatrix}1 & -\Delta t & -\frac{\Delta t^2}{2} - t_{\text{gap}}\Delta t\\ 0 & 1 & \Delta t\\ 0 & 0 & 1\end{bmatrix}}_{\mathbf{A}_1}\underbrace{\begin{bmatrix}\Delta d_i^k\\ \Delta v_i^k\\ \Delta a_i^k\end{bmatrix}}_{\mathbf{x}^k} + \underbrace{\begin{bmatrix}-\frac{\Delta t^3}{6} - t_{\text{gap}}\frac{\Delta t^2}{2}\\ \frac{\Delta t^2}{2}\\ \Delta t\end{bmatrix}}_{\mathbf{B}_1}\underbrace{j_i^k}_{\mathbf{u}^k}, \quad (3.31)$$

where Δt is the discretization time constant and the jerk j is chosen as input. Chaining multiple single convoys, i. e., considering multiple followers, results in coupling terms on the lower block diagonal. As an example for the two-follower case, see Equation (3.32), which is easily extended to the η-follower case.

$$\underbrace{\begin{bmatrix}\Delta d_1^{k+1}\\ \Delta v_1^{k+1}\\ \Delta a_1^{k+1}\\ \Delta d_2^{k+1}\\ \Delta v_2^{k+1}\\ \Delta a_2^{k+1}\end{bmatrix}}_{\mathbf{x}^{k+1}} = \underbrace{\left[\begin{array}{ccc|c} & \mathbf{A}_1 & & 0\\ \hline 0 & \Delta t & \frac{\Delta t^2}{2} & \\ 0 & 0 & 0 & \mathbf{A}_1\\ 0 & 0 & 0 & \end{array}\right]}_{\mathbf{A}}\underbrace{\begin{bmatrix}\Delta d_1^k\\ \Delta v_1^k\\ \Delta a_1^k\\ \Delta d_2^k\\ \Delta v_2^k\\ \Delta a_2^k\end{bmatrix}}_{\mathbf{x}^k} + \underbrace{\left[\begin{array}{c|c} & 0\\ \mathbf{B}_1 & 0\\ & 0\\ \hline \frac{\Delta t^3}{6} & \\ 0 & \mathbf{B}_1\\ 0 & \end{array}\right]}_{\mathbf{B}}\underbrace{\begin{bmatrix}\Delta j_1^k\\ \Delta j_2^k\end{bmatrix}}_{\mathbf{u}^k}. \quad (3.32)$$

Note that this is a linear system where the full state can be measured. Hence, a state-feedback control law $\mathbf{u}^k = -\mathbf{K}\mathbf{x}^k$, where $\mathbf{K}$ is calculated using a Linear Quadratic Regulator (LQR) can be utilized. Consequently, $\mathbf{x}^{k+1} = f(\mathbf{x}^k) \triangleq (\mathbf{A} - \mathbf{B}\mathbf{K})\mathbf{x}^k$ is found.

In order to obtain $\mathbf{K}$, a discrete-time algebraic Riccati equation needs to be solved once at the convoy initialization. The solution is time-invariant (for one convoy configuration) as it only depends on a set of convoy-specific but known parameters. Note that the computationally expensive part is done only once: at initialization. After that, the whole system prediction is only a single matrix multiplication, i. e., computationally very cheap.

3.6.4 Results

The proposed framework was tested both in real-world scenarios and in simulation. First, the lateral-offset correction using the TAS test vehicles is shown. Then, the longitudinal dampening is shown in simulation since the effect is more pronounced with more automated follower vehicles than the two at the institute's disposal. Note that the software was also demonstrated live as part of a large-scale project (see Heinrich [2015]).

Lateral Correction The setup consists of three vehicles. The manually driven lead vehicle is equipped with a mobile sensor box, capable of determining its velocity from a low-cost GNSS receiver and inertial measurement unit (IMU), and broadcasting it via narrowband radio. The followers are MuCAR-3- and MuCAR-4, two fully drive-by-wire-enabled cars.

For communication, a small-bandwidth radio ($19.2\,\frac{\text{kbit}}{\text{s}}$) is used between the lead vehicle and the first follower (MuCAR-4) and WLAN is used between first and second follower (MuCAR-3).[9] Due to fair conditions, all members sent at 10 Hz, resulting in 30% usage of the nominally available bandwidth. For hard off-road conditions, for instance signal dampening due to wet woods, and especially for longer platoons, it is not recommended to use more than 20%.

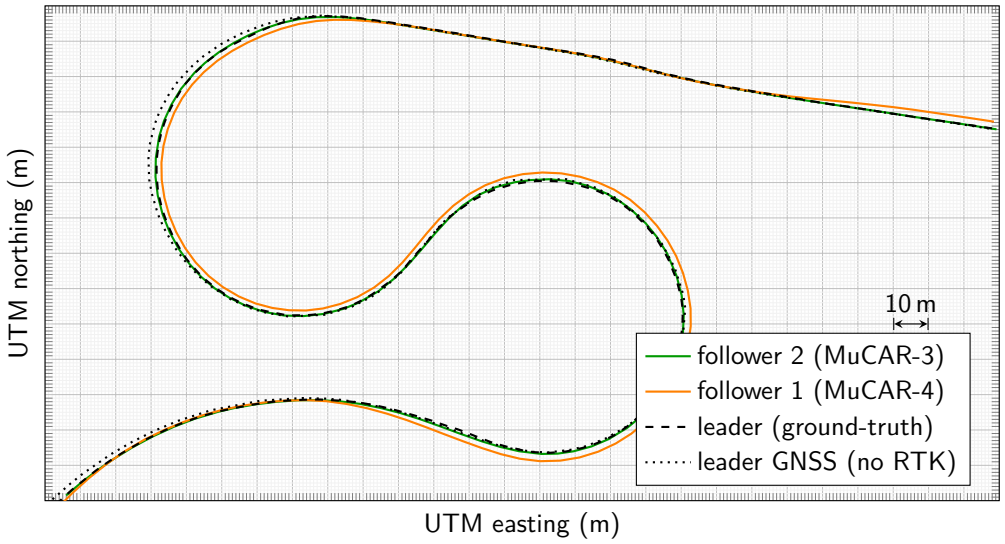


Figure 3.19:
Paths (bottom left to top right) of the manually driven lead vehicle and the two following vehicles. The purposely introduced lateral control errors of the first follower are disregarded by the second. This is RTK-GNSS ground-truth data. Additionally, the recorded GNSS track of the lead vehicle is shown dotted in order to demonstrate that control on a GNSS basis without offset correction from local sensors is infeasible.

[9] Note that the communication framework allows for inhomogeneous nets and forwards messages to convoy members that are not directly reachable.

The test scenario for the lateral-offset correction is depicted in Figure 3.19. There, the first follower purposely introduced a lateral control error, which the second follower was able to correct due to V2V communication. As a result, the second follower tracks the convoy leader's path, rather than the direct predecessor's path.

To show how important it is to use local perception, the GNSS path recorded by the lead vehicle is plotted additionally. Even with open-sky conditions, the offset to the ground truth recorded using RTK-DGNSS is sizable. Hence, GNSS signals cannot be used directly for off-road convoy applications.

Longitudinal Dampening Since only two automated test vehicles are available at TAS, the dampening effect in the longitudinal direction is shown in simulation. The TAS simulation environment utilizes exactly the same software as is running in the cars. The communication architecture/framework is also the same. Each simulated vehicle runs on a different computer, communicating via network and providing the vehicles' nonlinear dynamics by simulating low-level control, actuator dynamics, delays, etc. The main difference to the real-world setup is that an exact estimate of the lead vehicle is provided every 100 ms instead of estimated from a vehicle-tracking module at approximately 20 Hz camera frame rate.

A test setup with 6 simulated vehicles is shown where the convoy leader follows a challenging speed profile, see Figure 3.20. Again, using 30% of the theoretically available bandwidth, the lead vehicle send with 10 Hz and the 5 followers with 4 Hz. For comparison, the same scenario is simulated using a chain of ACC-controlled vehicles.

In Figure 3.20, it is shown that over the number of followers, the speed profile is dampened and the convergence behavior, especially when starting, is superior in the Cooperative Cruise Control (CCC) case. However, in the more dynamic middle part of the roughly 1 km long drive, there is more jitter in the speed profiles of the first followers. This is due to the slow updates from all convoy members and the unsteady speed profile of the convoy leader. However, this effect is declining for the later followers and could be mitigated by tuning the cost terms of the LQR or by slightly relieving the bandwidth constraints.

The tuning finds a balance between minimizing the RMSE of velocities and distances while maintaining good dampening. Note that achieving nice-looking curves on simple scenarios or with simpler underlying simulations is trivial and hence not reproduced here. The shown scenario is challenging and the bandwidth constraints are considerable, which results in challenging plots. Still, for example, comparing the RMSE of Δv_5 (0.64 for the CCC and 0.78 for ACC) and Δd_5 (2.59 and 3.87), a reduction of 17.9% and 33.1% for the 5th follower is achieved, respectively.

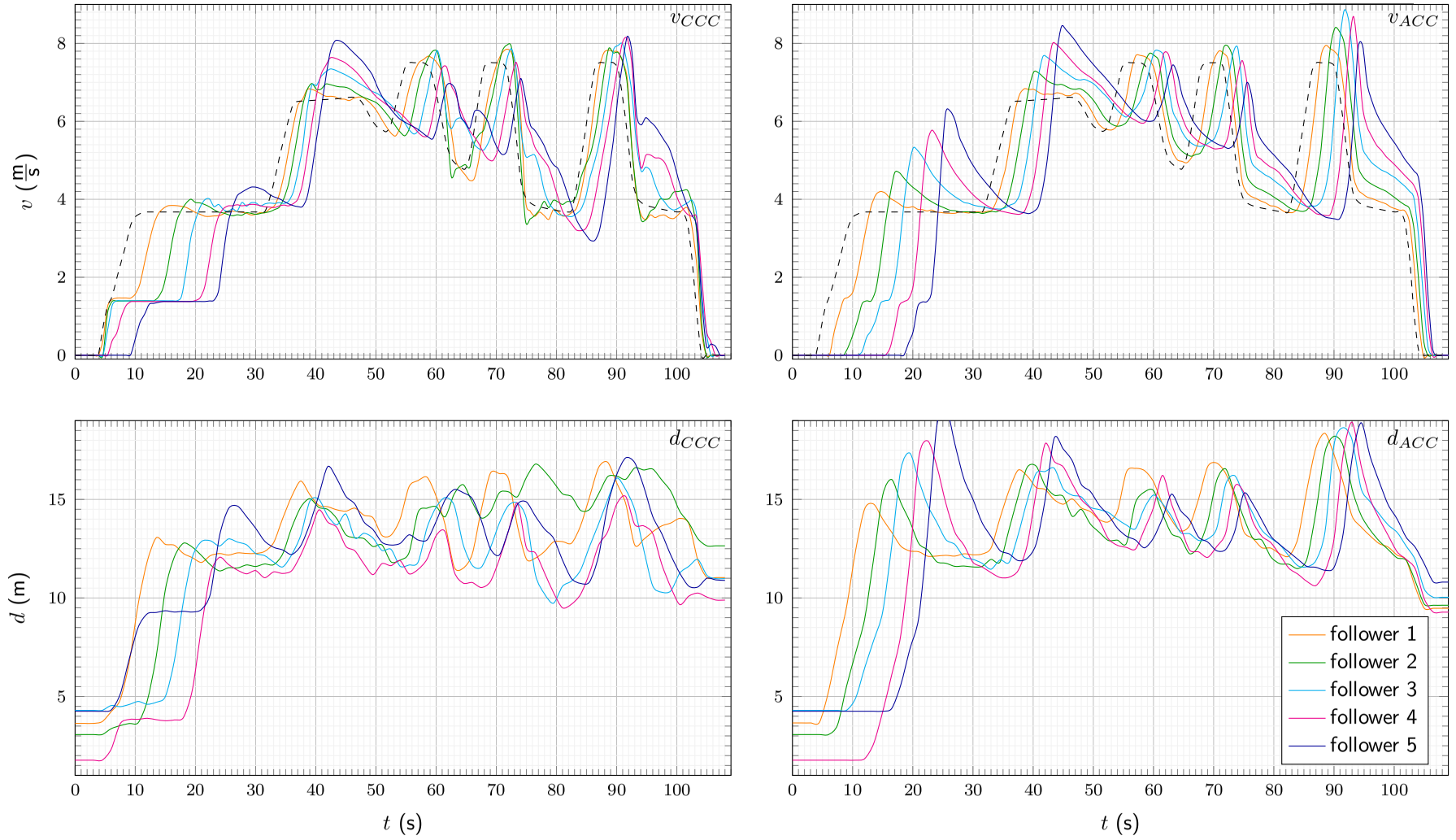


Figure 3.20:
Longitudinal performance for a simulated 6-vehicle setup using CCC (left-hand side) and ACC (right-hand side). The desired distance is $6 + 0.8 \cdot v_i$ (upper bound by 10) and was omitted for clutter reduction. Especially when the assumption of near-constant convoy-leader velocity holds, the CCC shows superior convergence behavior.

3.7 Two-step Trajectory Generation

Here, a novel method for motion planning is introduced that is related to the classic PVD as presented in Section 3.5. Like all contributions in this thesis, the main goal is to improve the actual behavior of the TAS test cars in real-world scenarios. Therefore, a method to plan trajectories in a two-step manner is proposed, where the first step already provides an initial trajectory and the second step improves its temporal component. This section was partly pre-published in Heinrich et al. [2018b].

3.7.1 Introduction

When other moving objects, e. g., traffic participants, have to be considered, motion planning becomes more complex since the search space is not static but changes over time. A major challenge is finding the right homotopy, i. e., a solution set. For instance: overtake before or after the oncoming car passed, or pass the crossing before or after the cross traffic. In this section, search-based algorithms are continued to be used since optimization-based algorithms tend to stay within their initial homotopy as, for instance, is shown in Kunz and Dietmayer [2016].

An issue arises when extending search-based algorithms from motion primitives which only adjust lateral changes (changes in the steering rate/angle) to also include longitudinal changes (accelerating or braking). Due to the combinatorial explosion, this yields an essentially infeasible problem:

Example 3.3 (Combinatorial Explosion)**.** Consider a search-based algorithm with $n = 5$ lateral samples. Assume a tree depth of $d = 10$. The search space then measures $n^d = 5^{10} = 9.765 \times 10^6$ possible combinations. This is large for brute-force, but still manageable for guided-search algorithms.

If $m = 5$ longitudinal samples are added, the search space grows to $(m \cdot n)^d = 25^{10} = 9.537 \times 10^{13}$, which is prohibitively large. □

The main idea of the proposed motion-planning algorithm is thus to split the trajectory planning into two parts:

- Find a trajectory using only very simple temporal planning and provide desired timestamps for key sample points.
- Ensure comfortable driving by adapting the speed profile without altering the given timestamps at those key points.

This allows to decrease the complexity of the motion-planning problem by relaxing the driving-comfort requirements on the initial creation and shifting it to a second smoothing step. Note that the safety guarantees of the initial trajectory are kept by staying *spatiotemporally consistent*. I. e., the times at which important path points are reached may not be changed. To the best of this author's knowledge, this has not been proposed before.

This section is structured as follows:
First, some prerequisites and a simple speed-planning heuristic as performance baseline is given in Section 3.7.2. Then, two optimization-based spatiotemporally consistent speed-profile planners are presented in Section 3.7.3. An extension that relaxes unnecessary constraints is given in Section 3.7.4. Finally, the given planners are evaluated and discussed in Section 3.7.5.

3.7.2 Prerequisites & Heuristics

In the following, a trajectory generated using the motion-planning framework presented in Section 3.3 is assumed given. I. e., it consists of N concatenated clothoid (see Appendix A.3) segments described by $N+1$ reference points. During its planning, a piece-wise constant acceleration profile is used. An illustration of two arcs can be seen in Figure 3.21.

Henceforth, let this *reference trajectory* be denoted as

$$\mathbf{r}_{[i]} = (l_i^*, t_i^*, c_i^*), \qquad i \in \{0, \dots, N\} \ , \tag{3.33}$$

where:

- l_i^* length at beginning of i^{th} clothoid segment,
- t_i^* time when to reach the beginning of the segment,
- c_i^* initial curvature of the clothoid segment.

The premise is that constant-acceleration profiles are not comfortable to drive due to their discontinuities in the acceleration. The objective is to generate a new smooth speed profile with at least continuous acceleration. For this, piece-wise polynomials are used: $l_i(t)$, $v_i(t)$ and $a_i(t)$ for the path length, speed and acceleration at time t on segment i, respectively.[10]

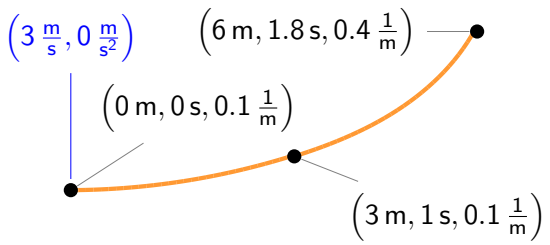


Figure 3.21:
Illustration of a reference sequence (orange). The spatiotemporal reference points incorporate *length*, *time* and *curvature* (black). Additionally, starting *speed* and *acceleration* are given (blue)

[10] Note the slight abuse of notation: On the one hand, v_i^* is the reference value of the speed at $\mathbf{r}_{[i]}$ and v_i the actual value there. On the other hand, $v_i(t)$ is the function of the speed on the i^{th} segment, i. e., for $t \in [t_i^*, t_{i+1}^*]$.

For continuity,

$$l_i(t^*_{i+1}) = l_{i+1}(t^*_{i+1}) = l_{i+1} \tag{3.34}$$
$$v_i(t^*_{i+1}) = v_{i+1}(t^*_{i+1}) = v_{i+1} \tag{3.35}$$
$$a_i(t^*_{i+1}) = a_{i+1}(t^*_{i+1}) = a_{i+1}\,, \tag{3.36}$$

needs to hold $\forall i \in \{0, \dots, N-2\}$. In other words: the end value on polynomial i has to be the same as the start value of polynomial $i+1$, where the transition happens at time t^*_{i+1}.

The provided reference trajectory $\mathbf{r}$ is planned to be collision-free considering the static as well as the dynamic objects in the environment. Thus, the resulting smoothed trajectory shall maintain the original spatiotemporal mapping

$$l(t^*_i) = l^*_i \quad \forall i \in \mathcal{K}\,, \tag{3.37}$$

where $\mathcal{K}$ is the set of all key indices. The key indices can be chosen use-case specific but typically are those before and after a dynamic object (or pseudo dynamic object such as a traffic light) crosses the ego vehicle's path, see Section 3.7.4.

Heuristic As a performance baseline, a spatiotemporally consistent speed-profile generation heuristic is introduced. Given the original trajectory $\mathbf{r}$, this suboptimal heuristic provides a continuous-jerk speed profile that adheres to the reference at virtually no computation cost. For notational convenience, let

$$\begin{array}{ll} \Delta l_i = l^*_{i+1} - l^*_i & \text{length of segment } i, \\ \Delta t_i = t^*_{i+1} - t^*_i & \text{duration of that segment and} \\ \tau_i(t) = t - t^*_i & \text{elapsed time on that segment.} \end{array} \tag{3.38}$$

Continuity in the acceleration $a(t)$ is guaranteed when using a polynomial for the jerk $j(t)$, and deducing $a(t)$, $v(t)$ as well as $l(t)$ from integration. Let

$$j_i(t) = \alpha_i \tau(t)_i + \beta_i \tau(t)_i^2 + \gamma_i \tau(t)_i^3\,, \tag{3.39}$$

where a constant term is omitted to enforce zero jerk at all segment borders, i. e.,

$$j_i(t^*_i) = j_i(t^*_{i+1}) = 0\,. \tag{3.40}$$

Consequently, the resulting speed profile is also continuous in the jerk. Using Equation (3.36) and Equation (3.40), and solving for the parameters $\alpha_i, \beta_i, \gamma_i$ results in

$$\alpha_i = -12\frac{-10\Delta l_i + \Delta t_i(4a_i\Delta t_i + a_{i+1}\Delta t_i + 10v_i)}{\Delta t_i^4} \tag{3.41}$$
$$\beta_i = 12\frac{-30\Delta l_i + \Delta t_i(11a_i\Delta t_i + 4a_{i+1}\Delta t_i + 30v_i)}{\Delta t_i^5} \tag{3.42}$$
$$\gamma_i = -12\frac{-20\Delta l_i + \Delta t_i(7a_i\Delta t_i + 3a_{i+1}\Delta t_i + 20v_i)}{\Delta t^6}\,. \tag{3.43}$$

Note that l_0, v_0 and a_0 are assumed given. Hence, starting at the 0^{th} segment, v_i can be deduced for $i \in \{1, \dots, N\}$, if a_i is chosen. Since there is one more parameter than condition, either v_i or a_i needs to be chosen to be able to compute the coefficients.

Given $\mathbf{r}$, the speeds a constant-acceleration profile would have is

$$\tilde{v}_i = 2\frac{\Delta l_i}{\Delta t_i} - \tilde{v}_{i-1}\,. \tag{3.44}$$

The acceleration follows as

$$a_i = \begin{cases} \frac{\tilde{v}_{i+1} - \tilde{v}_{i-1}}{\Delta t_{i+1} + \Delta t_i} & \forall i \in \{1, \dots, N-1\} \\ \frac{\tilde{v}_i - \tilde{v}_{i-1}}{\Delta t_i} & i = N \end{cases}\,. \tag{3.45}$$

A depiction of a typical baseline transition can be found in Figure 3.22.

A drawback of this approach is that it cannot be guaranteed that

$$\forall t : v(t) > 0\,. \tag{3.46}$$

I. e., in fringe cases, unintended back and forth maneuvering may be the result for, e. g., very small speeds and large desired acceleration transitions. Also, neither steering-rate nor lateral-acceleration restrictions are considered. In cases where the reference $\mathbf{r}$ is generated considering those constraints with an additional small margin, this is not an issue.

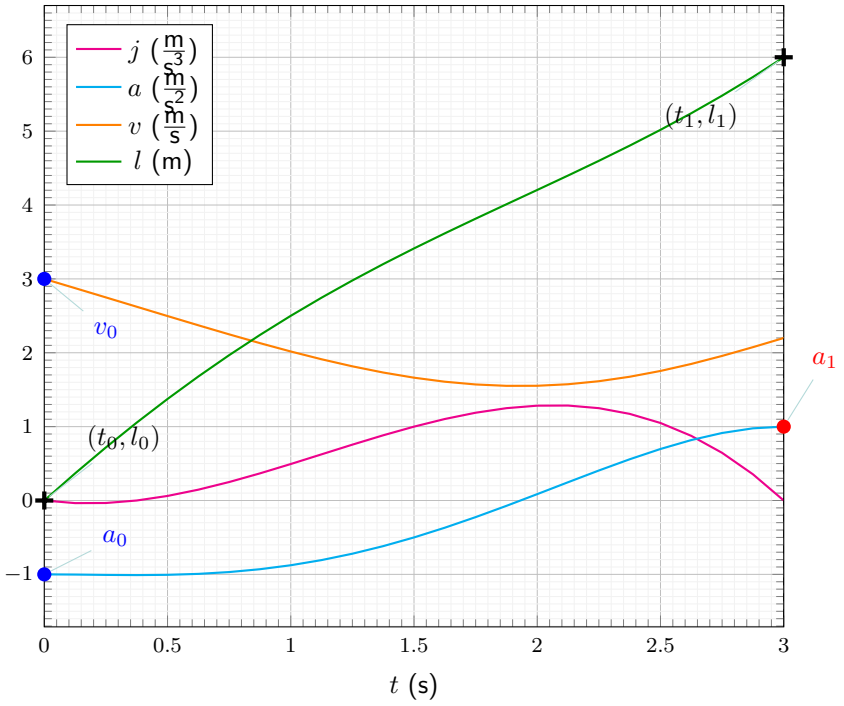


Figure 3.22:
Typical transition using proposed baseline. Given points from $\mathbf{r}$ are marked with black crosses. The starting condition v_0, a_0 are shown as blue dots. The heuristically chosen value a_1 is shown as red dot.

3.7.3 Optimization Approaches

In order to find an optimal spatiotemporally consistent speed profile, define the objective function as the normalized sum of squared jerks and steering rates over all segments, or more formally:

$$J = \frac{\sum_{i=0}^{N-1} \int_{t_i^*}^{t_{i+1}^*} k_j j_i^2(t, \mathbf{x}) + k_\delta \mathring{c}_i{}^2 v_i^2(t, \mathbf{x}) \, \mathrm{d}t}{t_N^* - t_0^*}, \tag{3.47}$$

where

k_j weighting factor for longitudinal jerk,
k_δ weighting factor for steering rate

and where the following steering-rate approximation was used: Utilizing the kinematic bicycle model (see Appendix A.10) and the small-angle approximation, the steering rate $\dot{\delta}$ for a vehicle moving along a path can be written as

$$\dot{\delta} = \mathring{c} v w \,, \tag{3.48}$$

where:

$\mathring{c}$ change of curvature c w. r. t. the curve's length,
w vehicle's wheelbase.

As the given trajectory consists of clothoid arcs, the change in curvature on each segment is constant and simply computed as

$$\mathring{c}_i = \frac{c_{i+1}^* - c_i^*}{l_{i+1}^* - l_i^*} \,. \tag{3.49}$$

In order to guarantee continuity, Equations (3.34) to (3.36) need to hold. The continuity constraints can be brought into the form

$$\mathbf{A}_{\text{eq}} \cdot \mathbf{x} = \mathbf{b} \,, \tag{3.50}$$

where $\mathbf{b}$ is filled with zeros and

$$\mathbf{A}_{\text{eq}} = \begin{bmatrix} 0 & 0 & \cdots & & \\ -I_3 & I_3 & 0 & \cdots & \\ 0 & -I_3 & I_3 & 0 & \cdots \\ & \ddots & \ddots & \ddots & \end{bmatrix} \in \mathbb{R}^{3N \times 3N} \tag{3.51}$$

with I_3, the unity matrix of dimension 3 and the zeros are filled accordingly.

Furthermore, positive speeds are enforced at all times, i. e.,

$$v(t_k, \mathbf{x}) \geq 0 \qquad \forall t \,. \tag{3.52}$$

This is necessary since sometimes it can be more 'optimal' (according to Equation (3.47)) to overshoot a certain length and drive backwards on the path. On the

one hand, this is physically not smoothly possible due to the necessary gearshifts. On the other hand, it endangers the ego vehicle since collision checking is only carried out up to the target length.

A closed-form solution for Equation (3.52) would be required to guarantee non-negative speeds all times. In order to remain as general as possible and computationally feasible, the polynomials are sampled at a fixed sampling interval Δt_{sample} and checked for Equation (3.52) which is laid out in more details below. The speed constraints are also brought into the form

$$\mathbf{A}_{\text{ineq}} \cdot \mathbf{x} \geq \underline{\mathbf{b}} \,. \tag{3.53}$$

Additionally, the steering rates are bound using

$$\left|\dot{\delta}(t_k, \mathbf{x})\right| \leq \dot{\delta}_{\text{max}} \tag{3.54}$$

$$\Rightarrow w \,|\mathcal{C}|\, v(t_k, \mathbf{x}) \leq \dot{\delta}_{\text{max}} \tag{3.55}$$

$$\Leftrightarrow v(t_k, \mathbf{x}) \leq \frac{\dot{\delta}_{\text{max}}}{w \,|\mathcal{C}|} \tag{3.56}$$

and lateral accelerations using

$$|a_{\text{lat}}(t_k, \mathbf{x})| \leq a_{\text{lat,max}} \tag{3.57}$$

$$\Rightarrow \; |c(l(t_k, \mathbf{x}))|\, v^2(t_k, \mathbf{x}) \leq a_{\text{lat,max}} \tag{3.58}$$

$$\Leftrightarrow v(t_k, \mathbf{x}) \leq \sqrt{\frac{a_{\text{lat,max}}}{|c(l(t_k, \mathbf{x}))|}} \,. \tag{3.59}$$

Equation (3.59) depicts a nonlinear inequality constraint, which would drastically increase the complexity of the optimization problem. Thus, the following simplification are used to linearize the problem:

$$v(t_k, \mathbf{x}) \leq \sqrt{\frac{a_{\text{lat,max}}}{|c_{i,\text{max}}|}} \,, \tag{3.60}$$

where $c_{i,\text{max}}$ is the maximum curvature of the i^{th} segment.

Note that (3.56) and (3.60) can be combined into one single inequality constraint

$$v(t_k, \mathbf{x}) \leq \min\left(\frac{\dot{\delta}_{\text{max}}}{w \,|\mathcal{C}_i|}, \sqrt{\frac{a_{\text{lat,max}}}{|c_{i,\text{max}}|}}\right) \,, \tag{3.61}$$

which can, similar to before, be written in matrix form as

$$\mathbf{A}_{\text{v}} \cdot \mathbf{x} \leq \overline{\mathbf{b}} \,, \tag{3.62}$$

where

$\overline{\mathbf{b}}$ is the stacked vector of constants and upper bounds from Equation (3.61)
$\mathbf{A}_{\text{ineq}}$ is generated using sub-sampling, just as in Equation (3.53).

Combining Equation (3.53) and Equation (3.62), gives

$$\underline{\mathbf{b}} \leq \mathbf{A}_{\text{ineq}} \cdot \mathbf{x} \leq \overline{\mathbf{b}} \tag{3.63}$$

and allows to finalize the statement of the optimization problem as

$$\begin{aligned} \min_{\mathbf{x}} \quad & J = \frac{1}{2}\mathbf{x}^{\mathsf{T}}\mathbf{H}\mathbf{x} = \frac{\sum_{i=0}^{N-1} \int_{t_i^*}^{t_{i+1}^*} k_j j_i^2(t,\mathbf{x}) + k_\delta \delta_i^2 v_i^2(t,\mathbf{x})\,\mathrm{d}t}{t_N^* - t_0^*} \\ \text{s. t.} \quad & \mathbf{A}_{\text{eq}}\mathbf{x} = \mathbf{b} \\ & \underline{\mathbf{b}} \leq \mathbf{A}_{\text{ineq}}\mathbf{x} \leq \overline{\mathbf{b}}\,, \end{aligned} \tag{3.64}$$

where, after evaluating the integral terms using the fixed reference points t_i^*, J can can be expressed in quadratic form with a Hessian $\mathbf{H}$ which is only a function of t_i^*. For simplicity, the weighting factors k were not tuned but set to one. For a fixed number of sample points t_i^*, the expressions for $\mathbf{H}$ can be derived offline. Therefore, the online evaluation of the cost functional, given $\mathbf{r}$, becomes very fast.

Optimizing the Heuristics Since Equation (3.45) is clearly suboptimal, Equation (3.64) is used to optimize its values. For this, define

$$\mathbf{x} = \begin{bmatrix} a_1 & a_2 & \cdots & a_N \end{bmatrix}^{\mathsf{T}}. \tag{3.65}$$

Note that since the heuristics already handles continuity, $\mathbf{A}_{\text{eq}}$ does not have to be included into the optimization problem. For the inequality constraints, however, the polynomials for $a_i(t)$, $v_i(t)$ and $l_i(t)$ have to be evaluated and checked at all sample times $t_{i,j}$. The segment's initial time t_i^* is checked and then every Δt_{sample}, i. e., each segment i has $n_i = 1 + \left\lfloor \frac{t_{i+1}^* - t_i^*}{t_{\text{sample}}} \right\rfloor$ samples. Thus, the dimension of $\mathbf{A}_{\text{ineq}}$ is a function of the number of segments N and the total sum of their respective samples $n_{\text{samples}} = \sum_i n_i$, i. e., $\mathbf{A}_{\text{ineq}} \in \mathbb{R}^{3N n_{\text{samples}} \times N}$.

For instance, when integrating the jerk heuristic Equation (3.39) and applying Equations (3.41) to (3.43) it results in the following formulation for the speed:

$$\begin{aligned} v_i(t) = {} & v_{i-1}\left(-\frac{12t^5}{\Delta t_i^5} + \frac{30t^4}{\Delta t_i^4} - \frac{20t^3}{\Delta t_i^3} + 1\right) \\ & + a_{i-1}\left(-\frac{21t^5}{5\Delta t_i^4} + \frac{11t^4}{\Delta t_i^3} - \frac{8t^3}{\Delta t_i^2} + t\right) \\ & + a_i\left(-\frac{9t^5}{5\Delta t_i^4} + \frac{4t^4}{\Delta t_i^3} - \frac{2t^3}{\Delta t_i^2}\right) \\ & + \frac{12\Delta l_i}{\Delta t^6}t^5 - \frac{30\Delta l_i}{\Delta t_i^5}t^4 + \frac{20\Delta l_i}{\Delta t_i^4}t^3\,, \end{aligned} \tag{3.66}$$

with $a_i = \mathbf{x}_i$.

The optimization of the heuristic is thus exactly in the form of the linear quadratic optimization problem in Equation (3.64). The results of the optimized heuristic are part of the comparison in the results Section 3.7.5.

Generalized Polynomial Parameter Optimization In a further step, the optimization is generalized by removing the suboptimality-introducing constraint $j(t_i^*) = 0$. Defining a polynomial of degree $M \geq 4$ for the length $l(t)$ in each segment i, speed, acceleration and jerk can directly be derived as:[11]

$$l_i(t) = \sum_{k=0}^{M} p_{i,k} \tau_i(t)^k \tag{3.67}$$

$$\Rightarrow v_i(t) = \sum_{k=1}^{M} k \cdot p_{i,k} \tau_i(t)^{k-1} \tag{3.68}$$

$$\Rightarrow a_i(t) = \sum_{k=2}^{M} k \cdot (k-1) \cdot p_{i,k} \tau_i(t)^{k-2} \tag{3.69}$$

$$\Rightarrow j_i(t) = \sum_{k=3}^{M} k \cdot (k-1) \cdot (k-2) \cdot p_{i,k} \tau_i(t)^{k-3} \,. \tag{3.70}$$

Let the objective function be the same as before, i. e., Equation (3.47). Only, in this case, the optimization variables are given as

$$\mathbf{x} = \begin{bmatrix} p_{0,0} & p_{0,1} & \cdots & p_{i,k} & \cdots & p_{N,M} \end{bmatrix}^\mathsf{T} \,. \tag{3.71}$$

In contrast to the previous approach, equality constraints have to be defined to satisfy the continuity requirements at the key times t_i (see Equations (3.34) to (3.36)) These can be written as a linear matrix equation block for each segment i:

$$\mathbf{A}_{\text{cont},i} \cdot \mathbf{x}_i = \begin{bmatrix} 1 & \Delta t_i & \Delta t_i^2 & \cdots \\ 0 & 1 & 2\Delta t_i & \cdots \\ 0 & 0 & 2 & \cdots \\ & & & \ddots \end{bmatrix} \cdot \mathbf{x}_i = \mathbf{x}_{i+1} \,. \tag{3.72}$$

In addition to the continuity constraints, the initial conditions (l_0^*, v_0 and a_0) as well as the key length values $l_i^* \ \forall i \in \{1, \ldots, N\}$ need to be fixed. This is achieved using

$$\mathbf{A}_{\text{initial}} \cdot \mathbf{x} = \begin{bmatrix} 1 & 0 & \cdots & & \\ 0 & 1 & 0 & \cdots & \\ & 0 & 1 & 0 & \cdots \\ & & 0 & 0 & \cdots \\ & & & & \ddots \end{bmatrix} \cdot \mathbf{x} = \begin{bmatrix} l_0^* \\ v_0 \\ a_0 \\ 0 \\ \vdots \end{bmatrix} = \mathbf{b}_{\text{initial}} \tag{3.73}$$

[11] Recall that τ is the elapsed time on a segment, see Equation (3.38), and hence its time derivative is 1

$$\mathbf{A}_{\text{key}} \cdot \mathbf{x} = \begin{bmatrix} \ddots & & & 0 \\ & \boxed{\begin{matrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{matrix}} & & \\ 0 & \mathbf{A}_{\text{key},i} & & \ddots \end{bmatrix} \cdot \mathbf{x} = \begin{bmatrix} \vdots \\ l_i^* \\ 0 \\ 0 \\ \vdots \end{bmatrix} = \mathbf{b}_{\text{key}}\,, \tag{3.74}$$

This is easily done since all values are optimization variables, i. e., the matrices $\mathbf{A}_{\text{initial}}$ and $\mathbf{A}_{\text{key},i}$ simply have ones on the diagonal at the first entry (all) and second and third (only initial), respectively.

Combining all equality constraints results in a block matrix in the form

$$\begin{bmatrix} \mathbf{A}_{\text{initial}} & 0 & \cdots & & & \\ 0 & \mathbf{A}_{\text{key},1} & \mathbf{A}_{\text{key},2} \cdots & & \cdots & \mathbf{A}_{\text{key},N} \\ \mathbf{A}_{\text{cont},0} & -I & 0 & \cdots & & \\ 0 & \mathbf{A}_{\text{cont},1} & -I & 0 & \cdots & \\ & & \ddots & \ddots & & \ddots \end{bmatrix} \cdot \mathbf{x} = \begin{bmatrix} \mathbf{b}_{\text{initial}} \\ \mathbf{b}_{\text{key}} \\ 0 \\ \vdots \\ 0 \end{bmatrix}. \tag{3.75}$$

Again, the minimum speed $v_i(t) > 0$, steering rate $\dot{\delta} < \dot{\delta}_{\text{max}}$ and the lateral acceleration $a_{\text{lat,max}}$ are constrained using Equations (3.56) and (3.60). Since all inequality constraints are only dependent on the current speed, the second row from Equation (3.72) is used to get

$$\underbrace{0}_{\underline{\mathbf{b}}_i} \leq \underbrace{\begin{bmatrix} 0 & 1 & 2\Delta t_i & \cdots \end{bmatrix}}_{\mathbf{A}_{\text{ineq},i}} \cdot \mathbf{x}_i \leq \underbrace{\min\left(\frac{\dot{\delta}_{\text{max}}}{w\,|\mathring{c}_i|}, \sqrt{\frac{a_{\text{lat,max}}}{|c_{i,\text{max}}|}} \right)}_{\overline{\mathbf{b}}_i}, \tag{3.76}$$

which, again, is combined to a diagonal block matrix

$$\underline{\mathbf{b}} \leq \mathbf{A}_{\text{ineq}} \cdot \mathbf{x} = \operatorname{diag}(\mathbf{A}_{\text{ineq},i}) \leq \overline{\mathbf{b}}\,. \tag{3.77}$$

The generalized polynomial parameter optimization is thus exactly in the form of the linear quadratic optimization problem in Equation (3.64). The results of the generalized polynomial parameter optimization are part of the comparison in the results Section 3.7.5.

3.7.4 Extension to Length Tubes

The proposed two-step trajectory generation keeps the exact spatiotemporal mapping of the initially planned trajectory, which, in the longitudinal domain, was only planned roughly. This subsection describes a natural extension to the proposed algorithm which keeps the ego vehicle in the found homotopy but disregards the exact spatiotemporal mapping in favor of so-called *length tubes*.

Consider a scenario where other vehicles cross the planned trajectory (see Figure 3.23). There, key points $\mathbf{r}_{[i]}$, or more precisely the corresponding key lengths l_i^* found in the first planning step, are relaxed to collision-free intervals $[\underline{l}_i^*, \overline{l}_i^*]$. If these intervals can

be provided by the initial motion planner, the modification of the proposed two-step algorithm is the following:

- In the first step, after finding the homotopy, determine the occupancy of dynamic objects at key times t_i^*
- Using the prediction uncertainty and some error margins, compute min and max lengths coordinates $[\underline{l}_i^*, \overline{l}_i^*]$ for the ego vehicle at key times t_i^*
- In the second step, the optimization problem (Equation (3.64)), remove $\mathbf{A}_{\text{key},i}$ from the equality constraints (Equation (3.75))
- Add $\underline{l}_i^* < l(t_i^*) < \overline{l}_i^*$ to the inequality constraints (Equation (3.77))

The extension is straight forward and introduces more room for the algorithm to optimize in.

The results of the extension to length tubes is also part of the comparison in the results Section 3.7.5.

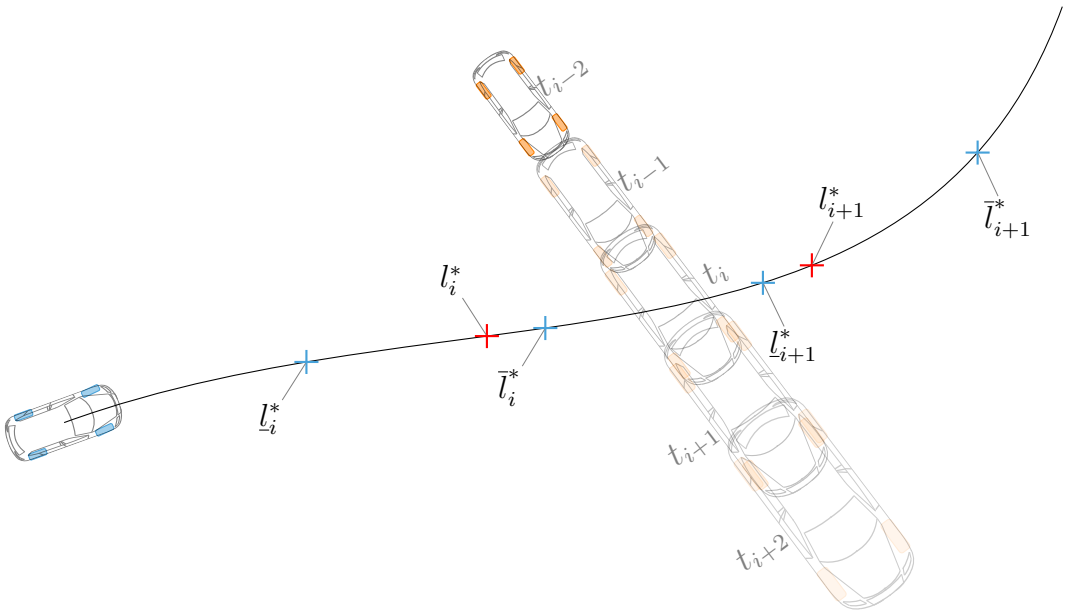


Figure 3.23:
Visualization of the extension of exact spatiotemporal mapping to length tubes. The planned trajectory (black) of the ego vehicle (blue) is crossed by another vehicle (orange). Consider a critical point in time t_i. The key lengths l_i^* which were determined in the first homotopy-finding step are relaxed to lower and upper bounds, $\underline{l}_i^*$ and $\overline{l}_i^*$, respectively. At the critical point in time, the algorithm guarantees that the ego vehicle will be within these bounds, but it is not predetermined where exactly. The increase in size of the crossing (orange) vehicle represents the prediction uncertainty.

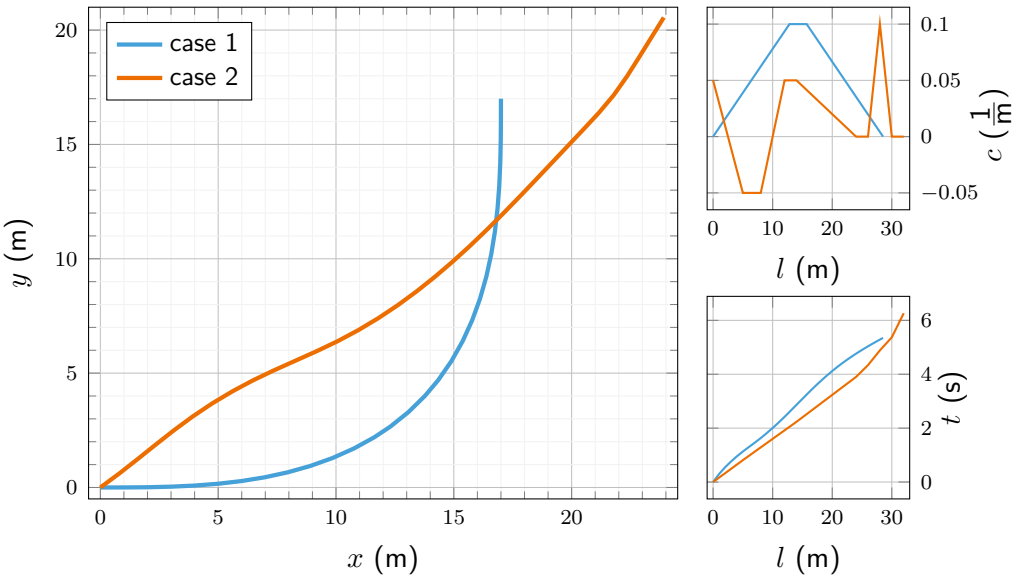


Figure 3.24:
Test scenarios 1 (blue) and 2 (orange), a left turn and a chicane, respectively. A bird's-eye view is given on the left, the right side shows the curvature and time profiles over the path length.

3.7.5 Results

All results were produced on an Intel i7–3770K CPU @ 3.50 GHz with 24 GB of RAM. The code was written and executed using MATLAB 2017b, the solver for the constrained optimization problem was `quadprog` with no further parameters.

The following parameters were used for all tests:

$$w = 2.855\,\text{m} \qquad a_{\text{lat,max}} = 2.3\,\tfrac{\text{m}}{\text{s}^2} \qquad k_a = 1.0$$
$$\Delta t_{\text{sample}} = 0.2\,\text{s} \qquad \dot{\delta}_{\text{max}} = 0.65\,\tfrac{\text{rad}}{\text{s}} \qquad k_\delta = 1.0\,.$$

Two scenarios (see Figure 3.24) are evaluated:

- In *case 1*, a simple 90° left turn, starting at 3 $\frac{\text{m}}{\text{s}}$ and trying to accelerate as much as possible.
- In *case 2*, a randomly generated more challenging chicane, starting at 6 $\frac{\text{m}}{\text{s}}$ and stopping at the end.

- In *subcase* a, the key points (forming $\mathbf{r}$) were chosen to be the trajectory kink points, i. e., the points where $\mathring{c}$ changes.
- In *subcase* b, the key points were sampled with a maximum Δl of 1 m.

Additionally, the computational cost of activating the inequality constraints, see Equations (3.56) and (3.60), is evaluated. Note that in this case, the reference already respects those values with a 15 % margin.

In Figures 3.25 to 3.27, plots for both cases are shown. Polynomial degrees greater 5 are not shown as the visible difference in the curves is minor. In Tables 3.2 and 3.3,

numerical results up to 6^{th} order are shown, as the performance improvement begins to stagnate there.

The following holds for all results: The black-dashed reference line shows the virtual constant-acceleration profile which is discontinuous in the acceleration domain. The blue plot shows the heuristics. At the key points, it always returns the jerk to zero and the acceleration to a value between the constant-acceleration reference values before and after. The green plot shows the optimization of the heuristically derived accelerations. Consequently, the acceleration values differ, but the jerk values always return to zero. The red and orange plots show the generalized polynomial optimization. While the 4^{th}-order polynomial is discontinuous in the jerk, the 5^{th}-order[12] are continuous in that domain, and of course, all lower derivatives.

The choice of the sampling distance Δt_{sample} and thus the number of sample points determines how tight the initial speed profile is followed. Unsurprisingly, the smoothing potential decreases with the density of sampling points (see Figures 3.26 and 3.27 for a side-by-side comparison). An increased number of samples is not only reflected in the computation time but can also introduce oscillatory behavior (see Figure 3.27, acceleration plot). Thus, it is recommended to only use critical points as key points, for instance those directly at an intersection, and to not 'oversample' uncritical parts of the trajectory. Sometimes, however, too few points can result in an infeasible problem, as can be seen from Table 3.2 (lower right quadrant). This is due to the inequality constraints using conservative approximations off the lateral acceleration.

Further experiments show that additional polynomial degrees sometimes allow for feasible solutions where lower degrees fail. However, in those cases, often very large accelerations are reached and hence it is preferable to just use a discontinuous acceleration profile.

The 5^{th}-order polynomial seems to be a good trade-off between optimality and computation time. Hence, it was implemented in C++ and tested on test case 2b with the result of a speed-up of an order of magnitude (see Table 3.4), as was expected.

In order to evaluate the performance impact of the extension to length tubes, 100 random scenarios of roughly 60 m trajectories with 4 crossing dynamic objects were generated: test case 3. The homotopy, i. e., if the ego vehicle goes before or after the crossing object, is chosen randomly.

Table 3.5 shows the results. An increase in the runtime by roughly a factor of three can be observed, but the cost —according to the cost functional— are one order of magnitude lower, i. e., the solutions are more comfortable. Over many more trials, the optimization never averaged over 3 ms runtime, which can be considered tolerable. In the length-tube case, the initialization seems to be more critical than before (higher optimality gain). The increase in runtime in the initialized length-tube case can be explained by the higher number of challenging cases that can be solved.

[12] and higher orders (not shown)

Table 3.2: Case 1, computation time vs optimality

	time (ms)	cost J	time (ms)	cost J
unconstrained	case $1a : N = 29$		case $1b : N = 4$	
heuristic	0.014	25.52	0.002	2.79
basic	2.002	19.70	2.000	0.84
poly_4	5.463	17.09	2.402	0.72
poly_5	5.736	16.90	2.584	0.50
poly_6	6.660	16.90	2.512	0.50
constrained	case $1a : N = 29$		case $1b : N = 4$	
basic	4.893	36.679	–	–
poly_4	11.123	20.367	–	–
poly_5	12.861	19.206	–	–
poly_6	15.684	18.309	–	–

Table 3.3: Case 2, computation time vs optimality

	time (ms)	cost J	time (ms)	cost J
unconstrained	case $2a : N = 33$		case $2b : N = 10$	
heuristic	0.015	307.24	0.004	49.74
basic	2.288	192.05	2.001	21.87
poly_4	3.989	170.85	2.527	20.24
poly_5	4.618	167.04	2.697	19.16
poly_6	5.082	167.04	2.836	19.16
constrained	case $2a : N = 33$		case $2b : N = 10$	
basic	4.714	210.48	3.464	21.87
poly_4	18.780	184.06	7.466	20.24
poly_5	20.660	174.65	7.728	19.16
poly_6	24.190	173.27	8.408	19.16

Table 3.4: Case 2b, computation time C++ vs MATLAB

	case 2b C++	case 2b MATLAB
poly$_5$	time (ms)	time (ms)
unconstrained	0.321	2.697
constrained	0.652	7.728

Table 3.5: Case 3: with vs without extension to length tubes

	with length tubes		without length tubes	
poly$_5$	time (ms)	cost J	time (ms)	cost J
uninitialized	0.659	0.748	0.932	8.372
initialized	2.522	0.516	0.884	6.071

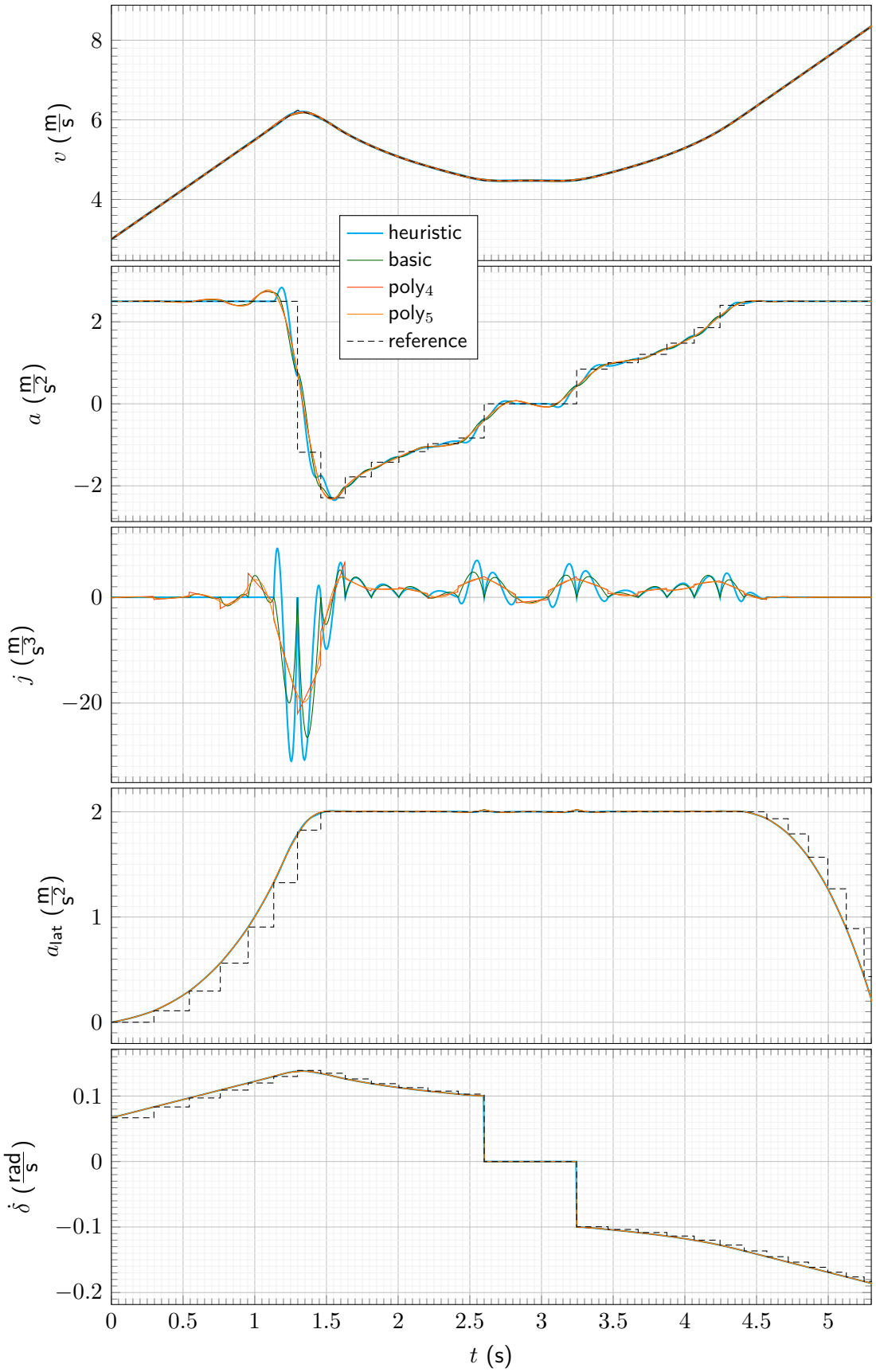


Figure 3.25: Results test case 1b

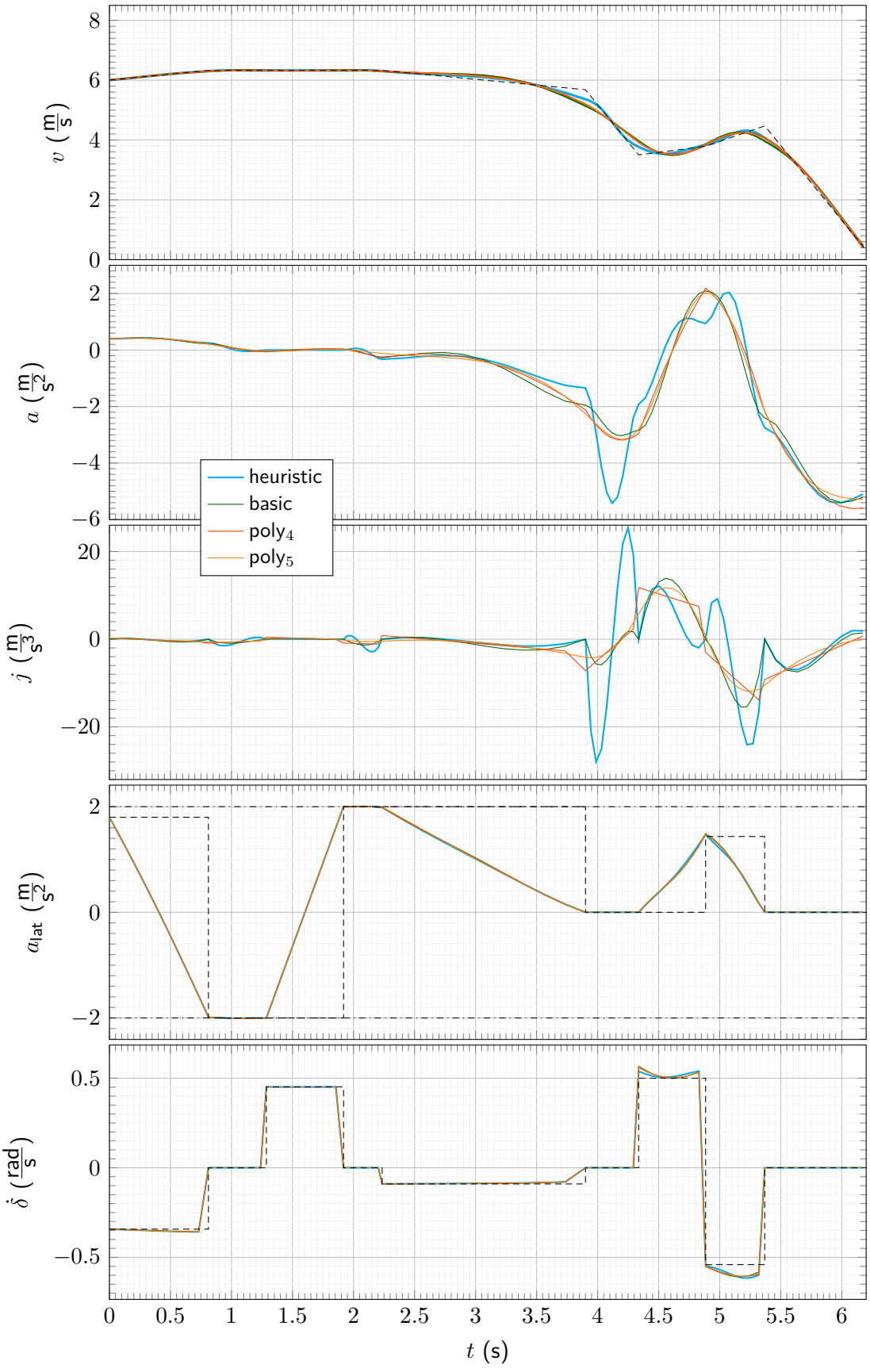


Figure 3.26: Results of test case 2, subcase a

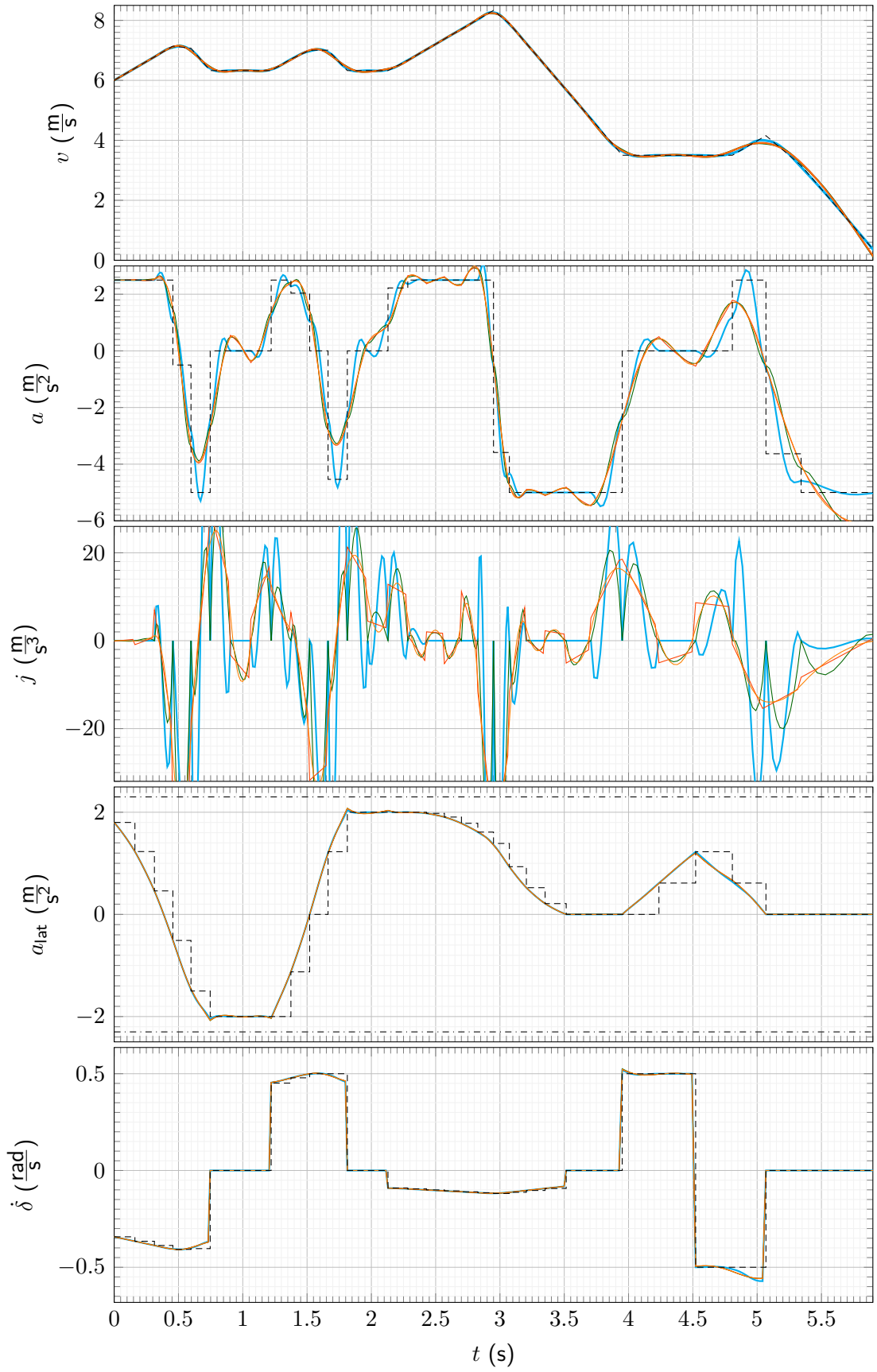


Figure 3.27: Results of test case 2, subcase b

3.8 Conclusion

In this chapter, a novel method to speed up collision checking in motion planning, custom-tailored variants of the classical path-velocity decomposition including an original take on convoying and a novel two-step trajectory generation paradigm, which can be seen as a variation of the former, were introduced. Additionally, platooning under off-road conditions, i. e., without any infrastructure was investigated. All of these are steps on the road to smoother and more precise driving.

Collision Checking Faster and more preemptive collision checking helps to speed up the spatiotemporal motion planning and thus results in safer —due to lower reaction times— or better —due to more time to explore— trajectories. Note that currently, the performance of the collision checking algorithms is still one of the limiting factors in autonomous-driving motion planning.

The proposed method places a fixed number of only two discs in contrast to the well-known baseline algorithm Ziegler and Stiller [2010], where their number n is a tuning factor which is usually set to 3 or 5. The proposed method also has a tuning factor s, which scales the trade-off between over- and undersampling of the true occupancy. The frontal disc of the proposed method already checks most of the area which will be covered by the vehicle contour assuming near-constant curvature and hence lends itself well for exploring algorithms, such as Hybrid-state A*. This preemptive checking can be interpreted as being predictive.

The comparison to the baseline showed comparable performance in terms of under- and oversampling (for the case of $n = 5$ and $s = 5$) while being faster to calculate than the $n = 3$ case. It was also shown that even though being conservative due to the assumption of near-constant curvature, even very tight passages (see Figure 3.12d) can be handled. The performance increase of about 30 % was achieved by using knowledge of the system's motion model.

Path-Velocity Decomposition Planning the spatial and the temporal component of a trajectory sequentially is a well-known practice. An efficient algorithm was presented that generates a trajectory given a path consisting of clothoid arcs and a set of constraints on, e. g., lateral accelerations, which was used extensively in real-world application.

Further, an optimization-based approach was presented that generates the speed profile in a holistic manner, i. e., taking the lateral component into account as well as tracking a desired speed-dependent distance to the lead vehicle, which generated the path. The speed profile is used as upper bound for a classical controller-based ACC longitudinal control. Here, the optimization and the ACC use a consistent control law, but the optimization is parameterized more aggressively w. r. t. distance keeping, giving the ACC room to work. The methods are proven in use and tailored to improve the actual behavior of the vehicles, leading to smoother and safer rides.

Platooning A novel approach for platooning was presented that is especially well suited for driving in unstructured terrain. This means, without any road markings for lateral lane keeping or the infrastructure for proper Vehicle-to-Vehicle (V2V) communication. Longitudinal as well as lateral stability were proven in live demonstrations (see Heinrich [2015]).

Longitudinally, the proposed Cooperative Cruise Control is a decentralized globally optimal low-bandwidth algorithm that works based on the assumption that the control laws of all convoy members are known. While the lead vehicle may be driven manually, all other convoy members drive autonomously utilizing their on-board perception to estimate each respective predecessor.

Laterally, the proposed offset correction allows compensating for *known* control errors, which could, for instance, be induced by terrain or weather conditions and would else destabilize platoons driving outside structured environments. Note that perception errors are not compensated and that, when systematic, lead to the same effect. It was out of scope for this work to utilize the GNSS signals. Using them would require offset correction which is infeasible due to the increase in necessary bandwidth. Recall that the shared bandwidth of $19.2\,\frac{\text{kbit}}{\text{s}}$ —of which experience dictates only a small fraction can be relied on— was already used completely.

Two-Step Trajectory Generation Three algorithms for spatiotemporally consistent speed-profile smoothing were presented. I. e., the smoothing of a speed profile which keeps the exact mapping of a given sequence of key length-time tuples. This way, safety is still guaranteed while the driving comfort can be increased at minimal computational overhead.

The method splits motion planning into two steps: A first phase, where a best trajectory (e. g., in terms of the best homotopy) is found using a search-based algorithm without emphasis on the driving comfort. A second phase, where the speed profile is smoothed using one of the proposed methods and thus achieve a more comfortable driving experience.

The methods are

- a simple heuristic as baseline, which is instantaneously calculable but suboptimal and does not respect constraints
- the optimization of the heuristic, which is a trade-off of speed vs optimality and
- a generalized polynomial parameter optimization, which provides an optimal solution w. r. t. the chosen cost functional.

The main intention was to find the best trade-off between comfort and computation time, where the latter ultimately is a measure for safety. The method's capabilities were shown on some sample scenarios where the importance of finding the correct key points in order to get the best possible result was pointed out.

Further, an extension to relaxing the exact mapping in favor of length tubes showed promise and was presented shortly.

4 Motion Control

The motion planning in Chapter 3 aims at providing safe and comfortably drivable, smooth trajectories since one of the two goals of this work is to achieve smoother driving. These trajectories shall guarantee spatial as well as temporal continuity, in order to allow for tight control which leads to precise trajectory following, which is the other goal: achieve increased driving precision. In this chapter, the motion-control part is introduced.

The introduced algorithms are separated into a vehicle-unspecific *high-level* part and a vehicle-specific *low-level* part. The former runs on the main computer, or high-level system, the latter on a real-time capable low-level system. This separation allows re-using the software on different vehicles. For instance, the TULF (Technologieträger Unbemanntes Landfahrzeug) ran the same high-level control on a completely different low-level platform (see Rheinmetall Defence et al. [2014]) than the Institute for Autonomous Systems Technology (TAS) vehicles, Munich Cognitive Autonomous Robot Car 3rd Generation (MuCAR-3) and MuCAR-4 (see Section 2.2.1).

The presented methods and algorithms are tailored to improve the performance in actual real-world scenarios. It is stressed that each part of the control chain is crucial, as any weak link harms the overall performance. Hence, also some mundane topics, such as practical system identification, are described — though only shortly.

The focus lies on the high-level algorithmic part, which offers more room for novelty. There, according to the proposed system architecture, primarily trajectory-following control is applied. However, for special events (e. g., emergency braking) or special scenarios (e. g., leader-follower), additional modes proved beneficial, which will be detailed later.

This chapter is structured as follows:
Initially, a very short overview of the state of the art is given in Section 4.1. Then, basic low-level controller are briefly introduced in Section 4.2. Multiple high-level controller are presented in Section 4.3: First, general topics like localization and delay compensation are handled in Sections 4.3.1 and 4.3.2, respectively. In Section 4.3.3, the basic trajectory following is explained. Thereafter, special considerations for convoying and precise positioning are presented in Sections 4.3.4 and 4.3.5. Finally, the results are summarized in Section 4.4.

4.1 State of the Art

The borders between motion planning and motion control are fluid. It depends on the architecture whether, for instance, measurements are directly fed through to controllers (as is done in classic Adaptive Cruise Control (ACC)), or if trajectories are planned to be precisely followed and then a controller is applied to guarantee this. In any case, below the high-level motion planning and control, there are always actuator level controllers, e. g., in the power train, that are out of scope for this work.

The case is especially intricate when using Model Predictive Control (MPC). Here, there are numerous works (see Section 3.1) that utilize this control framework to generate trajectories. There are combined planning and control approaches (e. g., see Gotte et al. [2015]).

However, due to most motion planners running at 10 Hz or even slower, it is necessary to have an underlying stabilizing layer, typically using simple PID (Proportional Integral Differential), LQR (Linear Quadratic Regulator) or flatness-based (see Fuchshumer et al. [2005], Menhour et al. [2014]) controller. However, there are also real-time capable MPC variants (see Borrelli et al. [2005], Falcone et al. [2007]) where the planned input is directly utilized.

The open topics in autonomous driving control are primarily in the extreme dynamic range. In Kolter et al. [2010], for instance, a mixed open- and closed-loop control was used to successfully drift a vehicle laterally into a parking slot.

Like in the motion-planning domain, there are also efforts in the control domain to incorporate learning methods. An important part in control, especially MPC, is the identification of the plant model. For this, learning was applied in Williams et al. [2017], where a model car is taught to race a dirt road in the extreme dynamics range. Rosolia and Borrelli [2019] went one step further: There, learning is not performed offline (i. e., on recorded data) but a "Learning MPC" is used that improves online with each lap (until convergence).

4.2 Low-Level Control

The so-called low-level part of the control is designed to provide a generic interface for the high-level parts of the control system. Its goal is to abstract the specifics of the vehicle actuation and low-level communication such that the high-level control can be agnostic to whether, e. g., the accelerator pedal is pressed by a linear drive that gets its data via CAN or if the platform provides a Flexray interface or if the pedal is emulated via an electric current. Ideally, the low-level part can even communicate its capabilities (e. g., acceleration, deceleration, turning rate, speed) and vehicle-specific parameters (e. g., dimensions, weight). In practice, however, those were handled by vehicle-specific parameter files.

Note that the low-level system (and control) also incorporates numerous safety and convenience features. For instance, checking whether there is a safety driver in place, checking signals for validity and, very importantly, the emergency-brake handling (from either internal or external panic buttons). But, since neither the safety concept nor the Human Machine Interface (HMI) is the focus of this work, those important parts are not treated here, only the control part.

Low-level trajectory control requires the low-level system to localize itself w. r. t. the trajectory both spatially and temporally. A sufficient localization in the low-level part is not given on all platforms. Since the approach shall be modular, this was consequently no viable option.

Instead, a very simple interface of reference values is used:

- δ_{desired} the desired steering angle (assuming bicycle model, see Appendix A.10)
- $\dot{\delta}_{\text{desired}}$ the desired steering rate (added for STS project, ported for MuCAR-4)
- v_{desired} the desired speed
- a_{desired} the desired acceleration (added for STS project, ported for MuCAR-4)
- d_{stop} a stopping distance
- (optionally, there is a direct interface to the brake and accelerator pedal for controlling the vehicle directly with a gamepad)

which, together with some bits[1] for controlling the lights, horn, etc., are communicated at 100 Hz.

The low-level control implementation is platform specific. Since all constraints are considered with a generous margin by the high-level control system and since the dynamics of the lateral and longitudinal actuation differ considerably, lateral and longitudinal control stayed separated on this level. The design philosophy is to keep it as simple and robust as possible. Hence, as much feedforward as possible and as few states (integrators) as possible were used.

The controllers are mainly designed in a two-degree-of-freedom (2DoF) manner, where, to put it very simply, the inverse dynamics of the plant is used as feed forward and only the model error needs to be corrected in a classical control manner (see Appendix A.8).

[1] bit as in byte, i. e., binary on/off flags

4.2.1 Lateral Low-Level Control

In autonomous driving, lateral control ensures that the ego vehicle exactly follows a given trajectory by controlling the steering wheel (or any abstraction from that, e. g., the steering-wheel torque). The low-level part ensures that the steering-angle trajectory is followed. Keeping the vehicle on the desired track is handled by the high-level part, where the necessary data is available. The reference steering-angle trajectory provides the angle δ, i. e., the virtual front-wheel angle of a kinematic bicycle model (see Appendix A.10), over time.

This virtual angle δ has to be mapped to something that can be measured and controlled at the low-level scope. This is not trivial, since the platforms used in this work differ vastly in their interfaces. For instance, MuCAR-3 had a chain drive directly turning the steering wheel and an extra incremental encoder to provide a more exact steering-wheel angle measurement. Other vehicles had direct or retro-fitted access to a steering-wheel angle, but the relation to δ needed to be determined.

In order to identify the relationship between what can be measured (steering-wheel angle) and the planning-space representation (δ), high-precision Real time kinematic (RTK)-Global Navigation Satellite System (GNSS) inertial navigation system (INS) is utilized. The identification procedure (see Figure 4.1) is:

1. drive circles with a fixed steering-wheel angle at a low velocity (such that slip is negligible allowing the use of the kinematic bicycle model).
2. fit a circle into the recorded path, extract its radius r.
3. correlate the controlled steering-wheel angle with the steering angle δ

This is to be done for a number of angles for left and right as most cars' steering mechanics have a zero offset and are not completely linear, especially for larger angles. Note that sometimes the zero-offset differs after each vehicle boot up and hence needs to be estimated each time. The relation between steering-wheel and steering angle is fitted and results in either a simple formula or a lookup table.

The identified map is used as feedforward. The rest is vehicle-specific. MuCAR-3, for instance, derived a desired steering rate from measured and desired steering-wheel angle which translated directly into a current fed to the steering-wheel chain drive. If a vehicle provides a steering-angle control interface, that can be used directly.

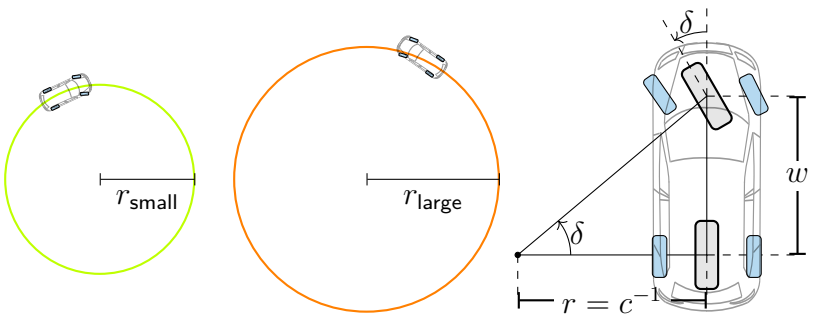


Figure 4.1:
Identification of relationship between measurable steering-wheel angle and model δ.

4.2.2 Longitudinal Control

In autonomous driving, longitudinal control ensures that the ego vehicle exactly follows a given trajectory by controlling the power train, i. e., accelerator and brake pedal (or any abstraction of it, e. g., the vehicle's acceleration). The low-level part, here, is composed of several building blocks (see Figure 4.2 for an overview and Figure A.4 for an implementation example):

Target Acceleration Generation Given the high-level command and control mode (either speed, speed and acceleration, or distance) different simple controllers are used to deduce a target acceleration.

Feedforward Given the target acceleration and current measured vehicle state, the desired torque for the power train is deduced. Important factors are: the driving resistance (from, e. g., wind and road friction, see Appendix A.9 for more details) and the creep torque, i. e., torque applied by idling motors, even when the accelerator pedal is not pressed. Note that this behavior is also emulated by electric cars!

All used test vehicles have a fixed mapping from the driver input (percent accelerator pedal is pressed) to the power-train output (torque). This is not a static mapping, but dependent on the current vehicle state: gear and speed. Thankfully, after some time spent on reverse-engineering these mappings, all manufacturers provided the respective originals. Due to the state-dependency, it is necessary to implement a two-step inverse table lookup: first interpolating in the speed range, then in the resulting (here: desired) torque to determine the correct accelerator-pedal-percent input.

Note that the pedal map includes negative torque values. This is also the case for electric cars that thus emulate the motor brake or, often drive-mode-dependently, recuperate. However, the negative torque by the motor often does not suffice to generate the desired slowdown. So, if the desired torque is lower than what can be achieved with zero accelerator pedal, the difference between zero-accelerator-pedal torque and desired torque value is passed to a brake-pressure controller.

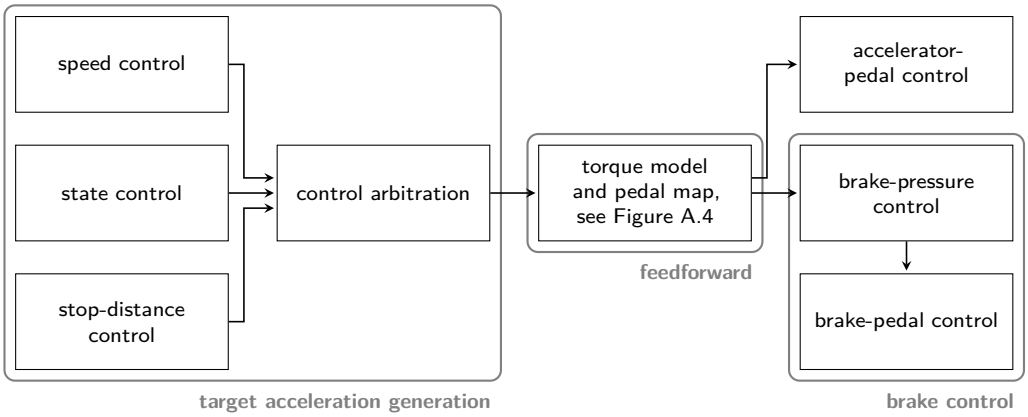


Figure 4.2: Architecture of the low-level longitudinal control

Accelerator control The accelerator-pedal control is trivial, since the pedal is not pressed physically. Instead, a custom circuit board that maps low-level system output to what the accelerator pedal would have produced is utilized. Its output is non-trivial (e. g., two voltages with different characteristics), but that complexity was handled by custom hardware. Note that the custom circuit board would also always allow the driver to overrule the system's command by physically pressing the pedal.

Brake control The brake-pedal control is more involved. Dependent on the test vehicle (and its actuation) the desired brake torque has to be mapped to something measurable and something controllable. For the sake of brevity, only one of the platforms is described here:

The brake pedal of Munich Cognitive Autonomous Robot Car 4$^{\text{th}}$ Generation (MuCAR-4) is pulled by a linear drive. This linear drive's length is controlled via a voltage generated by the low-level system. On the CAN bus, the break-pedal percent, i. e., percent of maximum way the pedal can be pressed, is available — the mapping between voltage and percent is identified once and the linear drive's dynamic is neglected. The brake pressure in bar inside the brake system and the brake torque counteracting the motor torque are also available on the CAN bus. The mapping between measured brake pressure in bar and brake torque in Nm was identified[2]. Since the value for the pressure is more frequently updated and seemed less filtered, the pressure is what is controlled.

Control is carried out by a 2DoF controller. Its reference is generated using the mapping between brake torque and brake pressure. Its feedforward is the mapping from brake-pedal percent to break pressure. As feedback part, a full PID controller with saturation and anti-windup is in place which was empirically tuned.

Special Case: High-precision stopping Usually, stopping a car 10 cm earlier or later is not an issue. Thus, it is sufficient for the general stop-distance controller to output a desired deceleration and execute the described control chain. However, for the precise-positioning use case (see Section 4.3.5), sub-centimeter precision is required.

In order to reach this level of precision, the vehicle operates at the border between stiction and friction. This is non-trivial to model and to handle correctly. A simple solution able to achieve this task is to feed through the distance to the stop point (measured by the high-level system[3]) and directly control the brake percent using a slow integrator. This way, the exact point where the vehicle moves with minuscule speeds (i. e., where the aforementioned creep torque almost exactly balances with the brake torque) can be found and utilized.

[2]Here, as in all test vehicles, the only available measurement data is what the vehicle bus offers. The models identified were based on recorded traces of responses to stepped and sinusoidal signals, simply by least-square fitting low-order polynomials to the measured input-output pairs.

[3]Note that the dynamic values (accelerations, velocities) from the inertial measurement unit (IMU) are not usable here, as will be detailed later.

4.3 High-Level Control

The so-called high-level part of the control is designed to generate an output for the low-level part of the control (see Section 4.2). It runs on the comparatively powerful high-level system, where it shares resources with the rest of the autonomous-driving software stack. Its task is to follow a trajectory (see Definition 3.3) as precisely and smoothly as possible.

The high-level control is the middle part of a cascade of three (see Figure 4.3):

- motion planning,
- high-level motion control, and
- low-level motion control.

The high-level motion control runs in one of three different modes:

- trajectory following: this is the usual mode
- convoying: for both the leader–follower and the platooning case
- positioning: for high-precision maneuvering

each consuming different information but providing the same output for low-level control.

This section is structured as follows:
First, some basics are given: The trajectory *localization* in Section 4.3.1 and the delay handling in Section 4.3.2. Then, the main trajectory-following control is introduced in Section 4.3.3. Thereafter, two special cases are presented: In Section 4.3.4, the convoying controller, which is used for both planners introduced in Sections 3.5.2 and 3.6. Last but not least, in Section 4.3.5, a contribution to accurate and precise positioning is presented.

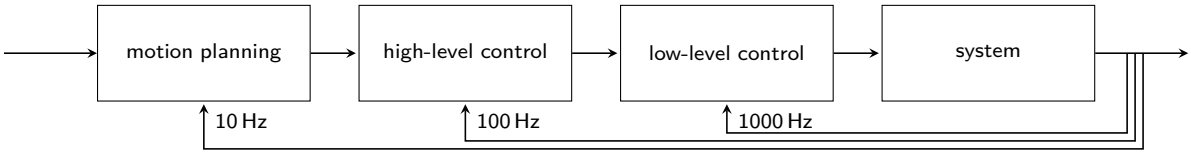


Figure 4.3:
Motion planning, high-level motion control and low-level motion control form a triple control-loop cascade. Each loop is separated by one order of magnitude in execution frequency: 10 Hz, 100 Hz and 1000 Hz.

4.3.1 Localization

When working with trajectories, the easiest way to localize the ego vehicle on a given trajectory is by interpolating the trajectory to the current system-clock time. Here, this is called the *temporal* localization.

Definition 4.1 (temporal localization)**.** Let the point on a given trajectory (Definition 3.3) at system-clock time t be denoted $\mathbf{l}(t)$.

Since obstacle avoidance is solely handled in the trajectory-generation modules, using the temporal localization as reference is not necessarily safe (see Figure 4.4). Hence, another localization is required. Here, this is called it the *spatial* localization.

Definition 4.2 (spatial localization)**.** Let the point on a given trajectory (Definition 3.3) at running length l be denoted $\mathbf{l}(l)$, if it is geometrically the closest point to the position of the ego-vehicle reference point. Note that further continuity constraints may be necessary in order to resolve ambiguities.

Using the spatial localization as reference is also not necessarily safe, since dynamic objects change their position over time (recall Figure 3.23). Consequently, it is necessary to track both the temporal and the spatial localization as precisely as possible, ideally resulting in both coinciding.

While deducing the temporal localization is trivial (if one disregards delays), the spatial localization $\mathbf{l}(l)$ requires some calculus: Let the tangent and normal of the trajectory at $\mathbf{l}(l)$ be denoted $\mathbf{m}_\mathbf{l}$ and $\mathbf{n}_\mathbf{l}$, respectively. Further, let $\mathbf{r}$ be the reference point of the vehicle for motion planning and control. Assume one can interpolate the trajectory using the running length l[4]. For a visualization, see Figure 4.5, where the rear-axle center is used as $\boldsymbol{r}$.

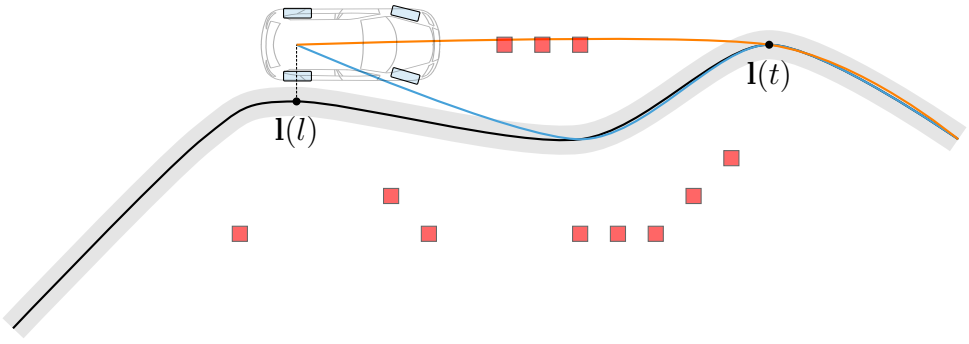


Figure 4.4:
Demonstration of the necessity of a spatial localization. Purely using the temporal localization $\mathbf{l}(t)$ as reference leads to a collision with an obstacle cell (marked red). Tracking the path, i. e., following the spatial localization $\mathbf{l}(l)$ is safe. Note that the errors in this diagram are exaggerated for illustrative purpose.

[4]This is given for a parametric curve, such as a clothoid.

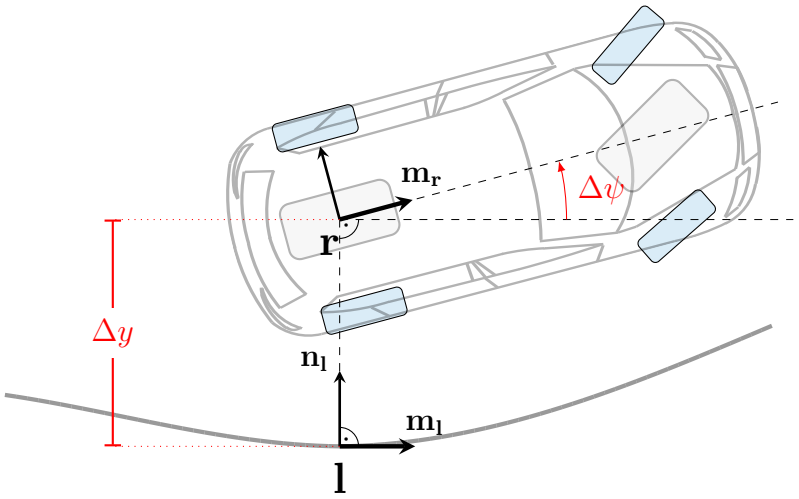


Figure 4.5:
Spatial localization $\mathbf{l}$ of the vehicle's control reference point $\mathbf{r}$. Normals and tangents are denoted $\mathbf{n}_{(\cdot)}$ and $\mathbf{m}_{(\cdot)}$, respectively. The lateral displacement Δy and the yaw orientation difference $\Delta\psi$ form the control error, and are highlighted in red. For reading convenience, the parameter l is omitted.

The spatial localization can then be deduced by finding

$$(\mathbf{r} - \mathbf{l}(l))^{\mathsf{T}} \cdot \mathbf{m}_{\mathbf{l}(l)} \stackrel{!}{=} 0\,. \tag{4.1}$$

Algorithm 748 (see Alefeld et al. [1995]) is used to find this root. Since the algorithm only supports monotonous functions, the segments of the trajectory are iteratively split until the resulting functions are monotonous.

As mentioned in Definition 4.2, it is important to impose further continuity constraints in addition to the geometric relation Equation (4.1) in order to obtain the correct localization. There are some of the edge cases that make this necessary:

If a trajectory intersects itself or has exactly parallel parts, there exist multiple 'correct' solutions. For instance, searching always from the beginning of the trajectory can lead to infinite driving loops. These cases are resolved by searching for a localization only from the last non-interpolated trajectory point[5] into the future.

Another common issue is the numeric stability of the roots. Sometimes, no exact zero can be found, and thus another minimum —e. g., far ahead on the trajectory— would be selected as better match. These cases are resolved by bounding the difference in consecutive localization's length values into a reasonable speed-dependent interval. A band-aid for generating a reasonable spatial localization in these sporadic numeric events is to simply predict the localization forward by the time between the last update and the current query. While this is not safe to do for multiple cycles in a row, it resolves the issue in all practically relevant cases since the numerical issue is usually very local.

[5]Non-interpolated points are the segment start and end points.

4.3.2 Delay Compensation

A major challenge when working with real-world systems is the handling of timestamps and delays. This becomes apparent only when closing the loop with a physical system, i. e., when a connection between *model time* and *real time* is established.

Everything that happens in software happens in so-called *model time*. Working on recorded sensor data, which is common practice in the perception domain, purely considers model time. Especially when single sensors are considered, only the time differences between data points hold any meaning. The age (or transport delay in control terms) from event to data has no influence at all. Working in simulation, there is an interaction between what is virtually perceived and the system's reaction. However, the model time is always perfectly known and processes can trivially be kept in sync. Even if processes get triggered or executed slower, e. g., due to processor load, the model-time timestamps can be utilized to never suffer a break in continuity. This means, any evaluation only has meaning in model time and only gives a hint on real-world performance.

In *real time*, things happen independently of model time. The quality of perception, fusion, scene understanding, planning and control can only really be evaluated when the chain is closed. Hence, issues along the chain often get apparent only in its last part: motion control. Thus, the handling of delay is part of the control chapter in this work.[6]

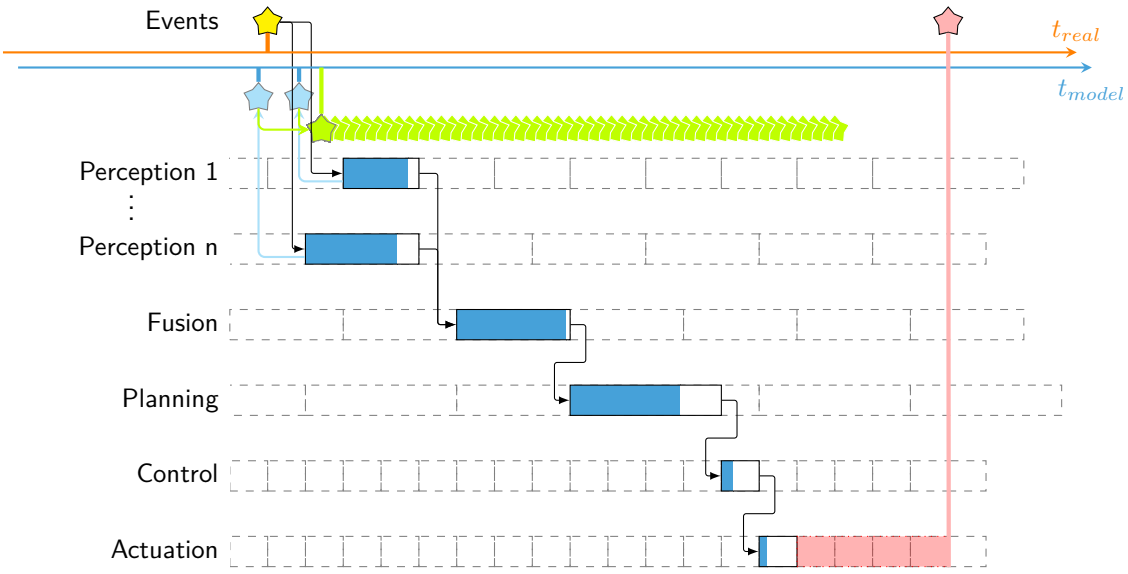


Figure 4.6: Illustration of the timeline of the introductory example.

[6]Note that the following example may be seen as being trivial. Nevertheless, the author's experience from his time at TAS and afterwards, in major industry projects, strongly suggests that it is necessary to discuss this topic for the complete event chain in the given detail. Furthermore, recent requests from the time after leaving TAS regarding issues with delay handling underline that this is an undervalued topic.

Introductory Example In the following, the timestamping complexity is illustrated with a simple example (see Figure 4.6 for a graphical representation):

1. In the real world, something happens (), e. g., a traffic light switches colors.
2. This information is perceived by different sensors, e. g., a camera. The sensing itself takes some time, e. g., exposure time plus on-chip preprocessing. The information then is processed, e. g., from an array of a specific pattern of colored pixels to some information that can be reasoned upon. The time of the event is estimated, and the event data pre-dated with a timestamp () before the current model time. Note that there is a small estimation error.
3. All available perception data, together with their history is fused for a coherent environment representation. Its timestamp is the last estimated perception event time, in model time (). Note that the time elapsed between event and fusion result is typically already a three-digit amount of milliseconds.
4. A prediction of the environment is carried out ( ). This results in a future ego localization within the environment representation by predicting ego/other agents on their respective planned/predicted trajectories.
5. Given this predicted world model, the planning module evaluates the best ego action. Typically, this results in a model-time-stamped trajectory, but, in a reactive architecture, could also directly result in a control error.
6. The controller acts according to the planning output and calculates the best actuator output, which is sent to the actuation.
7. The actuation then acts and finally, a force is applied on the actual physical ego system (). Note that the low-level processing time is usually very small, but the transport layer between control and actuation may introduce delays. A significant delay may happen between start of actuation and measurable effect due to inherent system dynamics, e. g., due to brake-pressure build up.
8. This is the point in time when the effect of the ego decision can be measured. Note that until the effect is sensed and fused, i. e., until it is possible to decide to take corrective action, steps 1.–3. have to be repeated. Naturally, the action is carried out only after the described delays.

Note that there may be a number of additional effects:

- If a discrete activation scheme is used, i. e., applications are triggered at a certain fixed time interval, there is always —for each component— an additional delay from finishing the calculations until the next time slot where the results are received (blue progress bars visualize actually utilized computation time). Note that even in a data-triggered architecture, usually at least the control module (or some supervisor) is time-triggered in order to guarantee safety/stability in case one of the links in the chain stopped sending data for a certain amount of time.
- Measurements may be out of sequence, meaning that older information arrives later at the fusion module than newer information (from other sensors). This effect needs to be handled in the fusion and can only be adequately handled if all data is consistently timestamped. If the data has a wrong timestamp, this will affect the overall estimation, and thus system, performance negatively.

Smith-Predictor Thinking Given the example above, it is clear that erroneous timestamps or the lack of prediction is devastating to the overall performance of the whole software stack. Hence, a classic method from control theory to handle delays is revisited here: the so-called *Smith Predictor* (see Smith [1957]).

Its main idea is: virtually control the system in a predicted future, created using a model of the system. Only the error of that prediction is controlled under the influence of delay. For more details, see the overview in Appendix A.7. This method is completely in line with the proactive direction proposed for autonomous-driving motion planning and control in this work.

There are at least two physically different control chains when automating vehicles: The lateral chain, controlled by the steering wheel, and the longitudinal chain, controlled by the power train.[7] In the former direction, the dynamics of the system are an order of magnitude faster than in the latter. Note further, that if there are different actuators within the same direction, e. g., for the accelerator and brake pedal, there are also different dynamics.

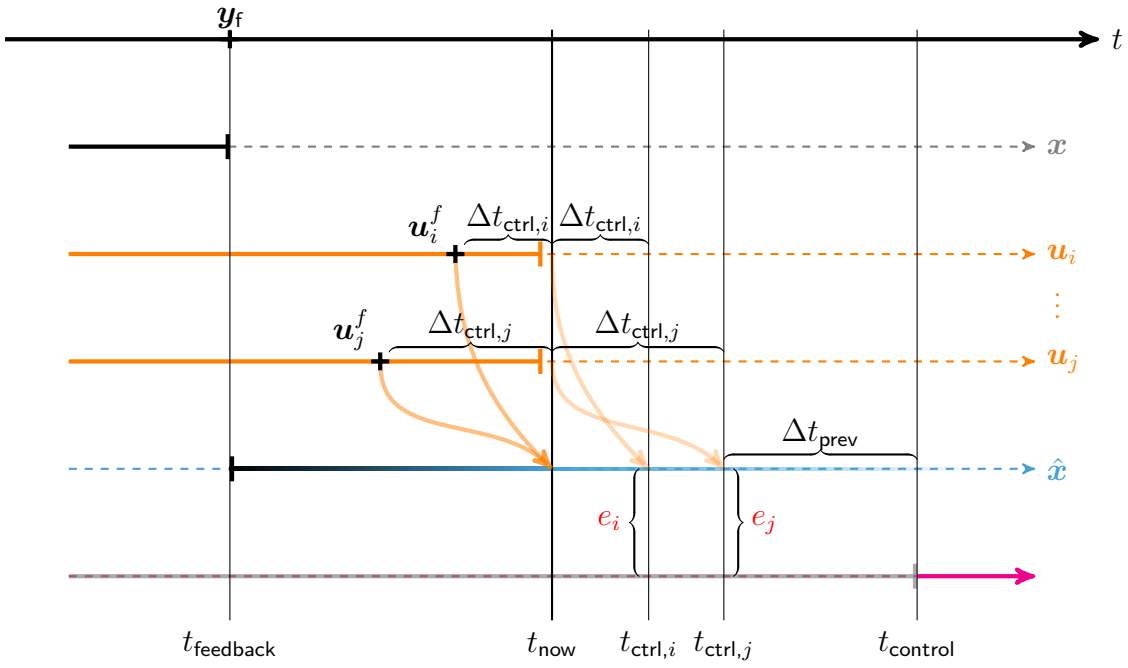


Figure 4.7:
Cause and effect in the later parts of the control chain. The measurements from t_{feedback} have a certain age when they arrive at the planning module at t_{now}. Further, each actuation chain $i, \ldots, j$ has a certain delay $\Delta t_{\mathrm{ctrl},i\ldots j}$ until an action takes effect. This means at t_{now} each command that is currently experienced has a certain age and also, that one cannot change anything until $t_{\mathrm{ctrl},i\ldots j}$. If, additionally, a preview time Δt_{prev} for low-level control is admitted, t_{control} is the control point in time on the predicted state, i. e., the earliest point at which the future ego trajectory may be changed.

[7] This is a simplification since, e. g., single-wheel braking is a routine measure in lateral stabilization. Recall that emergency maneuvers like collision avoidance are not in scope of this work. For comfort driving, it is a viable simplification.

The "Control Point In Time" For a visual overview of the complex relations described below, see Figure 4.7.

For each of the delayed control chains $i \in \{1, \dots, n\}$, it is known which command has been issued when, namely $u_i(t_{\text{cmd},i})$. Further, an estimate on delay $\Delta t_{\text{ctrl},i}$ between issuing the command and its arrival at the controller can be identified. The environment representation is measured at a point in the past t_{feedback} and —given a system model and the commands issued in the past— the environment can be predicted to a point in the near future. So, for each control chain, the earliest point in time when the ego behavior can be influenced is $t_{\text{now}} + \Delta t_{\text{ctrl},i}$.

Note that the only error that can actually be measured is at t_{feedback}. Everything further in the future is a prediction — even if it is in the past, seen from t_{now}. Moreover, the predicted error at each point in time is recorded and thus provide information about the modeling error.

The control error e_i at t_{now} is given by:

$$e_i(t_{\text{now}}) = e^*_{\text{now},i} - (e_{\text{feedback},i} - \hat{e}_{\text{feedback},i}) \,,$$

where

$e_{\text{feedback},i}$	denotes the currently measured error
$\hat{e}_{\text{feedback},i}$	denotes the error that was predicted at $t_{\text{feedback}} - \Delta t_{\text{ctrl},i}$
$e^*_{\text{now},i}$	denotes the error predicted at $t_{\text{now}} + \Delta t_{\text{ctrl},i}$.

This directly implements the Smith Predictor logic.

Especially lateral control often works on a preview point ahead in space or time (see Peng and Tomizuka [1991]), and not the current state[8]. Thus, an additional preview time Δt_{prev} needs to be accounted for/predicted to, when applying such control laws.

This results in the *control point in time* t_{ctrl}. For each control chain, this is a different point. Yet, only few planning algorithms allow planning separately for different actuators. Hence, typically the slowest chain dominates, i. e., the point farthest in the future is taken. This is the earliest point from which the motion planning may adapt the future trajectory (see Section 3.3: "Continuous Re-planning").

Note that these in-detail considerations were necessary to control the TULF, as a suboptimal low-level configuration led to a delay Δt_{ctrl} of over 0.5 s. Once it was implemented into the software stack, the smith predictor was used on all test vehicles. For the tuning, the measured delays were used and empirically tuned in a series of real-world tests.

[8] Note that while the current state is classically measurable, in practice this is not possible.

4.3.3 Trajectory-Following Control

Since the proposed motion planner provides high-quality trajectories which respect dynamic as well as kinematic constraints, the actual control can be comparatively simple. Hence, the trajectory-following control consists of a lateral path-following controller and a longitudinal spatiotemporal speed-profile tracker:

Lateral Path-Following Control For low speeds, the kinematic bicycle model (see Appendix A.10) is used. Together with the reference path, it forms the system model. Their joint model lends itself to using a flatness-based controller (see Zeitz [2010]) as it is *exactly input-output linearizable*, i. e., its nonlinear dynamic can be described as a linear system without introducing an error.

Let the vehicle reference point $\mathbf{r}$ and its spatial localization $\mathbf{l}$ be the origin of Frenet frames. Their current velocity vectors are the tangents $\mathbf{m_r}$ and $\mathbf{m_l}$, respectively. The relation between curvature c and their change w. r. t. path length l is:

$$\frac{\partial \mathbf{m}}{\partial l} = \mathring{\mathbf{m}} = c\mathbf{n} \qquad \text{and} \qquad \frac{\partial \mathbf{n}}{\partial l} = -c\mathbf{m} \qquad \text{(see Goldman [2005])}\,. \tag{4.2}$$

If the reference point moves, the localization moves such that Equation (4.1) remains zero. Its derivation via $\frac{\partial(\cdot)}{\partial t} = \frac{\partial(\cdot)}{\partial l}\frac{\mathrm{d}l}{\mathrm{d}t}$ yields:

$$\frac{\partial}{\partial t}\left((\mathbf{r}-\mathbf{l})^\mathsf{T}\cdot\mathbf{m}\right) \stackrel{!}{=} 0 \tag{4.3}$$

$$\Rightarrow (\dot{\mathbf{r}}-\dot{\mathbf{l}})\cdot\mathbf{m_l} + (\mathbf{r}-\mathbf{l})^\mathsf{T}\cdot\dot{\mathbf{m}}_\mathbf{l} \stackrel{!}{=} 0 \tag{4.4}$$

$$\Rightarrow (\mathbf{m_r}\dot{l}_\mathbf{r} - \mathbf{m_l}\dot{l}_\mathbf{l})^\mathsf{T}\mathbf{m_l} + d\mathbf{n_l}^\mathsf{T}c_\mathbf{l}\mathbf{n_l}\dot{l}_\mathbf{l} \stackrel{!}{=} 0 \tag{4.5}$$

$$\Rightarrow \mathbf{m_r}^\mathsf{T}\mathbf{m_l}\dot{l}_\mathbf{r} - \mathbf{m_l}^\mathsf{T}\mathbf{m_l}\dot{l}_\mathbf{l} + dc_\mathbf{l}\dot{l}_\mathbf{l} \stackrel{!}{=} 0 \tag{4.6}$$

$$\Rightarrow \dot{l}_\mathbf{l} = \frac{\mathbf{m_r}^\mathsf{T}\mathbf{m_l}}{1-dc_\mathbf{l}}\dot{l}_\mathbf{r} \tag{4.7}$$

$$\Rightarrow \dot{l}_\mathbf{l} = \frac{\cos(\psi)}{1-dc_\mathbf{l}}\dot{l}_\mathbf{r}\,, \tag{4.8}$$

and hence a relationship between the change of the length coordinates of the reference point and its localization over time, i. e., the speeds along their respective trajectories. Using Equation (4.8), the error dynamics of the rear-axle↔trajectory can be described system as follows: Deriving the yaw error $\psi = \psi_\mathbf{r} - \psi_\mathbf{l}$ gives:

$$\dot{\psi} = (\dot{\psi_\mathbf{r}} - \dot{\psi_\mathbf{l}}) \tag{4.9}$$

$$= \mathring{\psi}_\mathbf{r}\dot{l}_\mathbf{r} - \mathring{\psi}_\mathbf{l}\dot{l}_\mathbf{l} \tag{4.10}$$

$$= c_\mathbf{r}\dot{l}_\mathbf{r} - c_\mathbf{l}\frac{\cos(\psi)}{1-dc_\mathbf{l}}\dot{l}_\mathbf{r} \tag{4.11}$$

$$= \left(c_\mathbf{r} - c_\mathbf{l}\frac{\cos(\psi)}{1-dc_\mathbf{l}}\right)\dot{l}_\mathbf{r}\,. \tag{4.12}$$

Deriving the displacement Δy of the reference $\mathbf{r}$ to its localization $\mathbf{l}$ gives:

$$\frac{\partial}{\partial t}(d\mathbf{n_l}) = (\dot{\mathbf{r}} - \dot{\mathbf{l}}) \tag{4.13}$$

$$\Leftrightarrow \dot{d}\mathbf{n_l} + d(-c_\mathbf{l}\mathbf{m_l}\dot{l}_\mathbf{l}) = \mathbf{m_r}\dot{l}_\mathbf{r} - \mathbf{m_l}\dot{l}_\mathbf{l} \tag{4.14}$$

$$\Leftrightarrow \dot{d}\mathbf{n_l}^\mathsf{T}\mathbf{n_l} - c_\mathbf{l}d\dot{l}_\mathbf{l}\mathbf{n_l}^\mathsf{T}\mathbf{m_l} = \mathbf{n_l}^\mathsf{T}\mathbf{m_r}\dot{l}_\mathbf{r} - \mathbf{n_l}^\mathsf{T}\mathbf{m_l}\dot{l}_\mathbf{l} \tag{4.15}$$

$$\Leftrightarrow \dot{d} = \mathbf{n_l}^\mathsf{T}\mathbf{m_r}\dot{l}_\mathbf{r} \tag{4.16}$$

$$\Rightarrow \dot{d} = \sin(\psi)\,\dot{l}_\mathbf{r}\,. \tag{4.17}$$

The resulting system dynamics are

$$\dot{\mathbf{x}} = \begin{bmatrix}\dot{\Delta y}\\ \dot{\psi}\end{bmatrix} = \begin{bmatrix}\sin(\psi)\\ c_\mathbf{r} - c_\mathbf{l}\frac{\cos(\psi)}{1-dc_\mathbf{l}}\end{bmatrix}\dot{l}_\mathbf{r} = \begin{bmatrix}\sin(\psi)\\ -c_\mathbf{l}\frac{\cos(\psi)}{1-dc_\mathbf{l}}\end{bmatrix}\dot{l}_\mathbf{r} + \begin{bmatrix}0\\ \dot{l}_\mathbf{r}\end{bmatrix}c_\mathbf{r} = f(\mathbf{x}) + g(\mathbf{x})u\,, \tag{4.18}$$

where the curvature at the vehicle's reference point $c_\mathbf{r}$ is chosen as input u. The displacement Δy is chosen as output $y = h(\mathbf{x})$.

Using the Lie derivatives (see Appendix A.4) of Equation (4.18), an exact input-output linearization (see [Khalil, 1996, chapter 13.2]) is performed using the diffeomorphism

$$\mathbf{z} = \begin{bmatrix}z_1\\ z_2\end{bmatrix} = \begin{bmatrix}h(\mathbf{x})\\ L_f h(\mathbf{x})\end{bmatrix} = \begin{bmatrix}\Delta y\\ \sin(\psi)\,\dot{l}_\mathbf{r}\end{bmatrix} =: \Phi(\mathbf{x})\,. \tag{4.19}$$

Omitting the function's parameter and substituting $\dot{l}_\mathbf{r}$ with v for reading convenience, the system equations in flat coordinates are

$$\dot{\mathbf{z}} = \begin{bmatrix}z_2\\ L_f^2 h\end{bmatrix} + \begin{bmatrix}0\\ L_g L_f h\end{bmatrix}u \tag{4.20}$$

$$= \begin{bmatrix}z_2\\ -c_\mathbf{l}\frac{\cos^2(\psi)}{1-c_\mathbf{l}\Delta y}v^2\end{bmatrix} + \begin{bmatrix}0\\ \cos(\psi)\,v^2\end{bmatrix}u\,. \tag{4.21}$$

In order to eliminate the nonlinearities,

$$u = \frac{L_f^2 h}{L_g L_f h} + \frac{\nu}{L_g L_f h} = \frac{-c_\mathbf{l}\cos(\psi)}{1 - c_\mathbf{l}d} + \frac{\nu}{\cos(\psi)\,v^2} \tag{4.22}$$

can be used to reduce the input-output mapping to a chain of integrators where the dynamics of the last integrator can freely be set by choosing $\nu(\mathbf{z})$.

For intuitively tunable parameters, the desired dynamics is modelled as the well-known mass-damper-spring system

$$\ddot{z} + 2\frac{\zeta}{T}\dot{z} + \frac{1}{T^2}z = 0\,, \tag{4.23}$$

where ζ is the dampening constant (underdamped for $\zeta < 1$, overdamped for $\zeta > 1$) and T is a time constant. Accordingly, $\nu(\mathbf{z})$ is chosen as

$$\nu(\mathbf{z}) = -\frac{1}{T^2}z_1 - 2\frac{\zeta}{T}z_2\,. \tag{4.24}$$

with a dampening constant ζ of 0.8 and a length constant L of 3.5 m, where $L \cdot T = v$.

Using the well-known kinematic bicycle model Equation (A.13), the resulting system input u —i. e., the curvature at $\mathbf{r}$— is mapped to the steering angle

$$\delta = \arctan(wu) \ . \tag{4.25}$$

Longitudinal Spatiotemporal Speed-Profile Tracker For tracking the desired position in time a simplistic 2DoF approach is used. Since the speed profiles are very smooth, the speed at the current temporal localization $\mathbf{l}(t)$ is used as feed forward. The error is then state controlled in both time and space resulting in the overall control law

$$v_{\text{des}} = \mathbf{l}_{[v]}(t) + k_t \left(\mathbf{l}_{[t]}(t) - \mathbf{l}_{[t]}(l) \right) + k_l \left(\mathbf{l}_{[l]}(t) - \mathbf{l}_{[l]}(l) \right) \ , \tag{4.26}$$

where the tuning factors k_t and k_l are both set to 0.3.

4.3.4 Convoy Control

Convoying is a major application for autonomous driving at TAS. Hence, from a control point of view, special measures are taken in order to achieve the best possible performance.

While the trajectory generated by the proposed motion planning is *optimal* (e. g., smoothness w. r. t. jerk and/or acceleration minimization) it may lack reactiveness, especially when the convoy lead vehicle has to react harshly to the environment. A more direct feedback is introduced here. The resulting more decisive reaction by the ego vehicle induces confidence in both the safety driver and the passenger. Further, it actually benefits safety, as more non-nominal cases can be handled autonomously. Note, though, that this still does not make the system a true SAE Level 4 system (see Appendix A.1).

Note that parts of this subsection were pre-published in Fassbender et al. [2017b] and Heinrich and Wuensche [2017].

Leader–Follower Recall that in the leader–follower framework (see Section 3.5.2), a trajectory is generated incorporating the kinematic and dynamic constraints of the ego vehicle utilizing constraint optimization. Distance keeping is only a soft constraint, and only one of the seven terms penalized in the objective function.

This is a conscious design decision. If a certain minimal distance is guaranteed under comfort limits, the optimization becomes infeasible or may show erratic runtime behavior close to borderline cases. Therefore, the safety burden is shifted completely to the control layer, allowing the optimization to generate the best possible driving experience for the nominal use case — which covers >99% of convoy driving.

As control law, a classic ACC (see Siedersberger [2003]) is used, with:

$$d_{\text{des}} = d_{\text{min}} + t_{\text{gap}} v_{\text{ego}} \tag{4.27}$$

$$v_{\text{acc}} = v_{\text{lead}} + k_v(v_{\text{lead}} - v_{\text{ego}}) + k_d(d_{\text{act}} - d_{\text{des}})\,, \tag{4.28}$$

where

v_{acc}	Adaptive Cruise Control (ACC) reference speed
v_{lead}	lead-vehicle speed
v_{ego}	current ego-vehicle speed
d_{des}	desired distance between lead and ego
d_{act}	actual distance between lead and ego
d_{min}	minimal distance between lead and ego (parameter)
t_{gap}	time gap between lead and ego (parameter)

and k_d and k_v are tuning factors.

Note that exactly the same rule as in motion control is also used in motion planning. The only difference is a more aggressive parameterization in the planning. There, slightly smaller values are used for d_{min} and t_{gap}. This is necessary since the ACC control purely works in the longitudinal domain and does not consider lateral accelerations. In the optimization, both are considered, jointly optimized and constraints are imposed, e. g., on the lateral acceleration. Given that the planner will always try to drive closer to the lead vehicle, it can be safely used as an *upper bound* for the control.

Thus, the commanded desired speed v_{cmd} is

$$v_{\text{cmd}} = \min\Big(v_{\text{traj}}(t_{\text{ctrl}}), v_{\text{acc}}(t_{\text{ctrl}})\Big)\,,$$

where the trajectory is evaluated at t_{ctrl}, the control point in time (see Section 4.3.2).

A further performance improvement comes naturally from the control cascade (see Figure 4.3) where the motion control runs significantly faster than the planning and thus, more recent information can be used. A further increase in reactiveness can be achieved by directly using sensor measurements (like a Radar speed and distance), which eliminates the delay by fusion but results in a far noisier input signal. Nevertheless, this noise would need to be compensated by less direct control, which directly contradicts the proposed design pattern of smooth reference values and tight controls.

Platooning In the proposed platooning framework (see Section 3.6), the speed profile is a globally optimal solution for the whole convoy. It is the control reference for each individual platoon member in the form of a delta to Equations (4.27) and (4.28). Hence, the ACC is again followed at the higher control frequency, but offset by the reference trajectory.

4.3.5 Positioning Control

A further contribution of this work is the ability to autonomously position a normal-sized vehicle within a margin of ± 5 cm, ± 1 cm and ± 1 degree, laterally, longitudinally and in heading, respectively. Here, *positioning* means to physically place the vehicle at a desired position rather than determining its global position as it is sometimes used in robotics/autonomous driving (compare: GPS). Note that for notational convenience in this section, the term *precise* is used for precise and accurate in one.[9] This subsection was partly pre-published in Heinrich et al. [2016, 2018c].

Precise positioning is a valuable ability for automated valet parking (e. g., see Furgale et al. [2013], Bosch [2020]) or automated charging of electric vehicles (e. g., see Petrov et al. [2012]). In both cases, increased precision directly benefits the use case by either saving space or reducing requirements on other parts of the system, for instance, the charging-plug mechanism. Here, it was developed as an enabler for autonomous parcel delivery (see Heinrich et al. [2018c]).

Parcel delivery is an outdoor activity. Hence, the positioning needs to work day or night, sun or rain, and on a regular parcel-delivery car. The proposed solution was extensively tested on an electric vehicle provided by StreetScooter GmbH (StS) (see Figure 4.8). The operative design domain sets this apart from other publications, e. g., Andreasson et al. [2015], where similar results were achieved in a factory environment and with a small test vehicle and external localization.

Figure 4.8:
Precise positioning use case: Parcel delivery to an automated box with a vehicle-mounted robotic arm. Because the arm is not able to rotate, but can only extend laterally, the whole vehicle has to be positioned such that the extended arm hits the parcel box exactly. For more details, see Heinrich et al. [2018c], where this picture is taken from.

[9] Reminder: In terms of a Gaussian, *preciseness* describes the magnitude of the standard deviation and *accuracy* the trueness of the mean.

Challenges In order to achieve the required high precision, a plethora of challenges has to be overcome. This is due to the resulting error being the sum of all the errors in the control chain from sensor input to actuator output (see Figure 2.1).

A first realization was that the actual precision of often assumed *ground truth* RTK-INS systems is lower than the requirement, at least at low-speeds. At normal speeds, wheel-pulse sensors stabilize the egomotion estimation and the noise of the accelerometers/gyros can clearly be separated from true measurements, e. g., due to cornering, which results in an overall sub-centimeter precision. At low speeds, however, wheel pulses are not viable as they are only available every few centimeters and the small accelerations during positioning, especially back-and-forth, are virtually indistinguishable from sensor noise.

Another impediment was that the test vehicle was retrofitted with a commercially available drive-by-wire system. The system is designed to make cars drivable by disabled people and to be compliant with German traffic laws (regarding redundancies etc.) and not for high-precision maneuvering. It features a steering-wheel actuator, mounted at the steering column, and a single actuator for pressing either the accelerator or the brake pedal.

Given these insufficiently precise sensors and sub-par actuators, state-of-the-art positioning had to be achieved. How it was done is shown in the following paragraphs, where the control chain is considered from the back to front.

Actuation The insufficiencies in the actuation needed to be overcome. The design choice of the drive-by-wire system to use a single pedal actuator is presumably intended to prevent misuse (pressing both pedals at the same time). However, it introduces travel time between both pedals and, what is more, overshot when alternating between the pedals.

Parts of this impediment was solved by adding a custom electronic accelerator circuit board. This was possible since the accelerator pedal interfaces with the motor-control unit via analog voltage. The drive-by-wire kit was hence only used for braking.

As described in Section 4.2.2, a very down-to-earth approach was utilized —namely good feed forward with an added slow integrator— to practically solve a theoretically very hard problem: moving the vehicle at the border between stiction and friction. Nevertheless, it proved to be sufficient, even at light inclines and on different road-surface types. Note that if the planned trajectory is overshot, the low-level control can switch to reverse in order to improve the final positioning precision

Ego-Pose Estimate As described earlier and will be shown in the results paragraph below, the precision and accuracy of the Egomotion Estimate (EME) based on an INS utilizing differential RTK-GNSS is insufficient.[10] This issue is overcome by using the INS pose only as initialization for a vision-based localization. That localization was developed exclusively for this use case and yields a very precise relative-pose estimate (for more details see Jaspers et al. [2016]).

Note that the ego goal position is defined relative to the delivery box. Hence, the trajectory is fixed w. r. t. the box. The position estimate continuously updates the relative ego position, which is localized on the fixed trajectory. This localization is then passed on to the high-level trajectory control. I. e., no re-planning is done. Instead, the control part is used to compensate the ego-pose estimate jitter.

For a graphical representation of the involved chain of homogeneous transformations, see Figure 4.9. Note that the mapping is done via a local reference frame. This is not strictly necessary, but since it is used in the rest of the motion-planning framework, it is also utilized for the positioning use case.

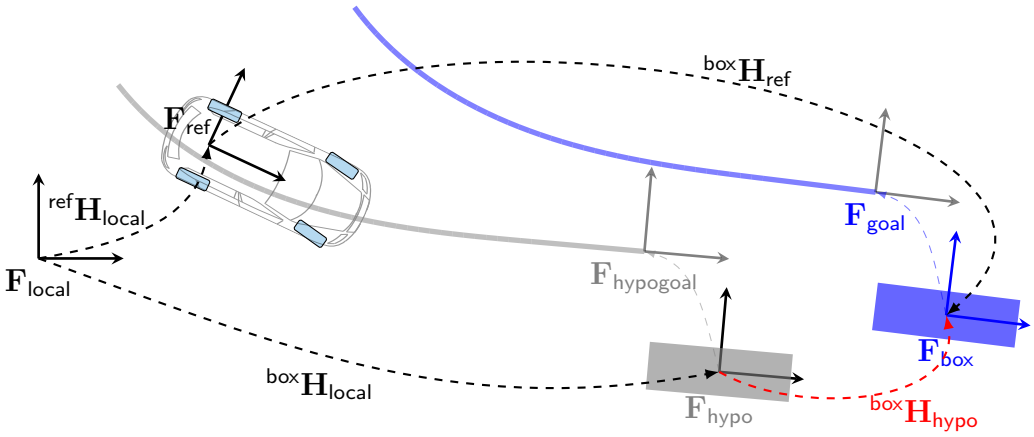


Figure 4.9:
$\mathbf{F}_{\text{ref}}$ is the reference/sensor frame, $\mathbf{F}_{\text{local}}$ the local frame. $\mathbf{F}_{\text{hypo}}$ and $\mathbf{F}_{\text{box}}$ are the frames of the initial delivery-box hypothesis and the currently tracked one, respectively. Their corresponding actual rear-axle goal position frames are $\mathbf{F}_{\text{hypogoal}}$ and $\mathbf{F}_{\text{goal}}$. The initially planned path is drawn in gray, the transformed/corrected in blue. For transforming the once-planned trajectory, the homogeneous transform ${}^{\text{box}}\mathbf{H}_{\text{hypo}}$ needs to be calculated. Note that the relative pose of the goal to the box is the same as from the hypothetical goal to the box hypothesis. Note further that proportions are not to scale. Taken and modified from Heinrich et al. [2018c].

[10] For notational convenience, only 'INS' is used for the described EME system in this section.

Motion Planning The generated trajectories need to be sufficiently smooth and very reproducible such that the controller can be tuned accordingly. This was solved by adding a custom expansion to the motion-planning framework. These expansions always end with a straight part of a predefined minimum length and have a pair of gentle (opening+closing) clothoids leading into the straight.

The motion-planning framework finds a point from where such an expansion is viable. Experiments (see Figure 4.10) show that the proposed expansions yield a high success rate and is computationally very feasible. For more details, see Heinrich et al. [2016].

Results In Figure 4.11, there are four successful approaches to the different delivery boxes. They are all positioned outdoors under open-sky conditions. A GNSS reference station is within 100 m of each. All plots are normalized such that the estimated end goal position is at $(0, 0)$.

It can be seen clearly that the INS solution has issues when nearly standing still. These problems initially occur mainly in the lateral direction; later the position performs a random walk in a 2 cm radius. The initial lateral stability is due to an underlying Kalman filter which uses an appropriate non-holonomic process model. Also, the radius is contained since the INS is fed the wheel-pulse counter.

Two of the approaches were performed at night (orange (c) and cyan (d)), but recorded with a more recent version of the tracking software and thus yield even better results. In the blue plot (b), the vehicle overshot its goal position by more than 1 cm, hence it corrected by driving a short distance reversely. This is the case where the INS performs worst. In all cases, the tracking outperforms the INS.

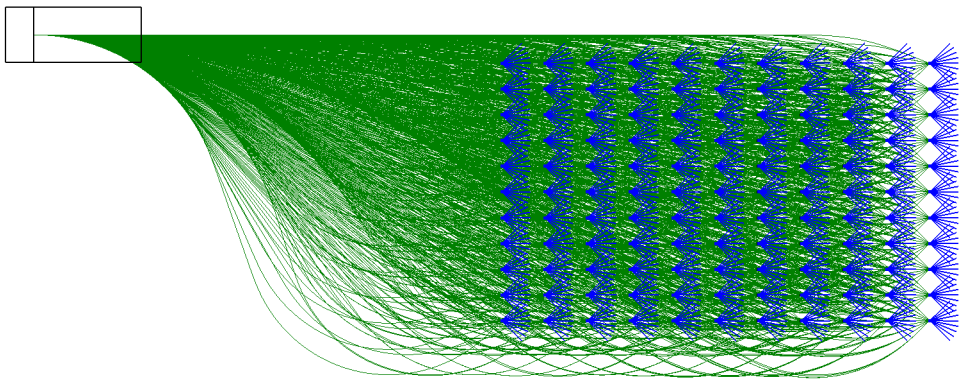


Figure 4.10:
Paths generated by the custom positioning expansion. In blue, there is a grid of target positions with nine headings each in the range $[-\frac{\pi}{4}, \frac{\pi}{4}]$. The querying point is the middle of the rear axle of the ego vehicle, shown as a black bounding box. The proposed method takes 82.56 µs per path and has a success rate of 77.48 % (4219 of 5445 cases). Comparing this to a state-of-the-art method (e. g., used in Gu et al. [2015], Heinrich et al. [2015]), which takes 192.5 µs and has a success rate of 1.03 % (56 of 5445 cases), brings the results into perspective. Taken and modified from Heinrich et al. [2016].

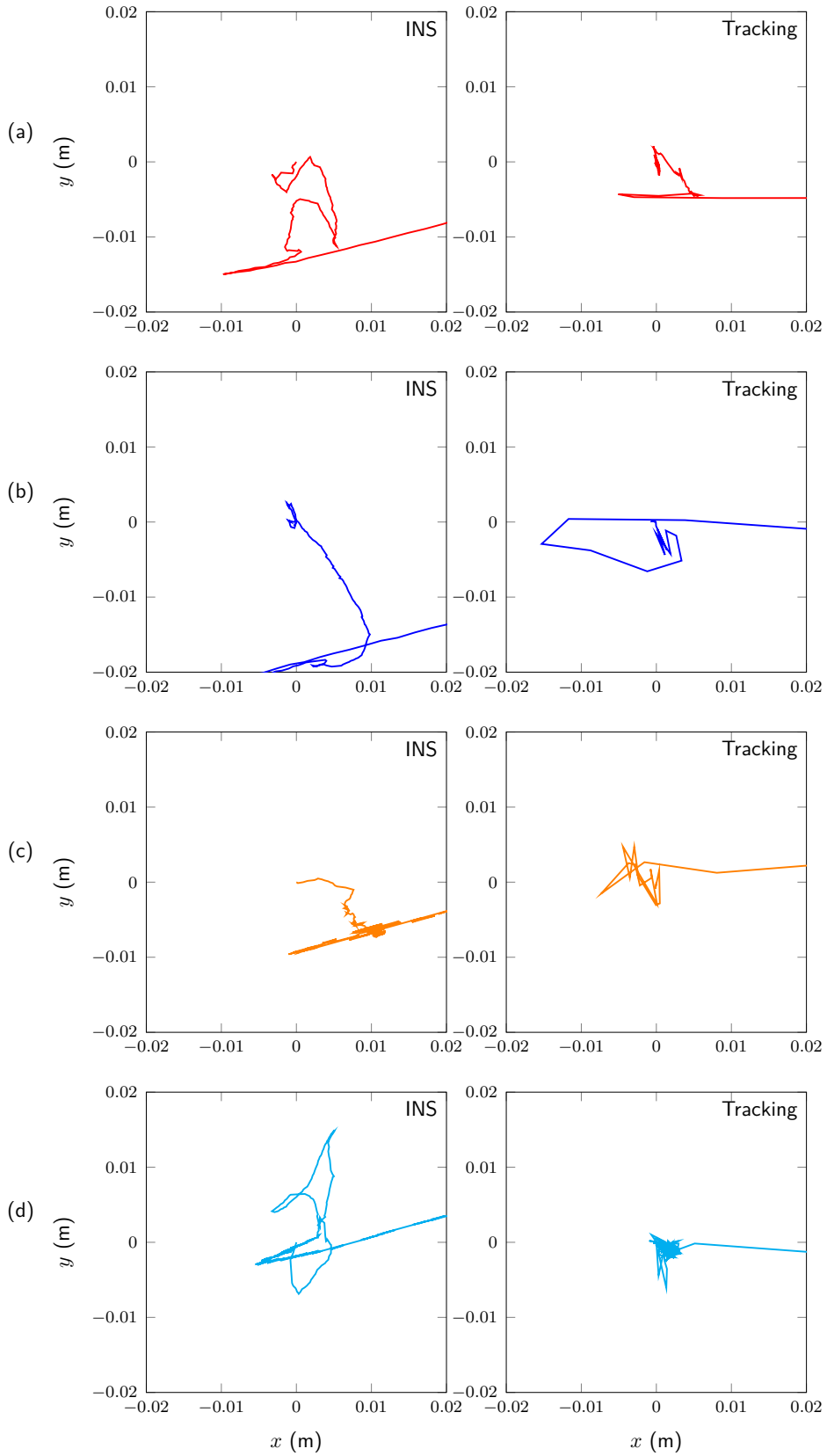


Figure 4.11:
Comparison in position stability between INS and object-relative tracking of four successful approaches and deliveries. The ego vehicle always approaches from positive x and ends up at the axis origin. The left and the right plot show the same approach, once perceived by INS, once by the proposed tracking approach.
Plots were taken and modified from Heinrich et al. [2018c].

4.4 Conclusion

In this chapter, a motion-control framework is proposed. While not as many novel methods were introduced here as in the motion-planning chapter, it is important to stress the significance of every link in the control chain. The low-level control and trajectory-following parts work reliably, but other parts are more noteworthy: Namely, the explicit handling of delay, the special considerations for convoying (and platooning) and the unprecedented positioning precision. All of these are steps on the road to smoother and more precise driving.

Delay Compensation A topic that is seldom stressed in the autonomous-driving literature is the correct handling of delays. Maybe this is due to not too many institutes actually running the full control chain on their test vehicles —if they possess any— or that the topic is just too tedious. Here, an overview was given of how a Smith-Predictor scheme can be utilized and extend for multiple control chains acting on one vehicle.

The main idea is to utilize a system model to forward simulate the vehicle's behavior and issue control commands in the 'now' that will benefit the system once they arrive. Similarly, delays in the measurement are considered. If the model was perfect, the measured error would tend to zero — apart from disturbance, that can only be measured under delay, naturally. Since no model is perfect, however, the difference between model and reality is measured and controlled under delay ordinarily.

Note that the in-depth consideration of the system delays helps in the overall system performance. Knowing from which point in time the future trajectory of the ego vehicle may be altered gives an advantage in the interplay with motion planning. Namely, the guarantee that there are no planned changes that cannot be reacted to and that would introduce control errors which need to be stabilized under delay.

Convoy Control The methods for convoy control that were presented can be seen as another variation of the path-velocity idiom. Two use cases were presented: a single leader–follower system, where the ego vehicle shall follow a lead agent as precisely as possible, and the platooning case, where any number of followers are attached to one lead vehicle. Both are useful, especially in very dangerous environments. The focus of the former is performance, the focus of the latter is stability.

For the leader–follower case, a trajectory —that is generated via a constrained optimization— is combined with classic ACC. The optimization's goal is to provide a smooth reference that takes kinematic and dynamic constraints into account. However, it does not guarantee longitudinal stability in safety-critical scenarios. This is due to the possibility of hard constraints rendering the problem infeasible. Further, the 10 Hz execution frequency of the optimization can be seen as an impediment in critical scenarios. The ACC only works in the longitudinal direction, but there it is an established method that easily runs with 100 Hz. The contribution here is the fusion of both methods. By parameterizing the ACC parameters —i. e., speed and distance keeping— of the optimal trajectory to be more aggressive, its desired speed

can be used as an upper bound for the control part. Thus, the strengths of both approaches are combined: the optimality and constrainedness of the trajectory and the reactiveness and safety of the ACC.

In the platooning case, a globally optimal solution is generated locally. This plan is to be adhered to as exactly as possible in order to not destabilize the longitudinal platoon dynamics. The solution describes a deviation from the nominally desired ACC values for distance and speed difference. For instance, it can make sense —globally— for one vehicle to drive closer than usual to its predecessor if then all followers can still brake within their comfort limits. Similarly to the leader–follower case, ACC is employed as safety measure, but here it is tuned more aggressively than the reference trajectory. Hence, in contrast to the leader–follower case, the trajectory is the reference to be followed, and the ACC provides the upper bound.

Positioning Control Placing a vehicle very precisely (and accurately) at a certain position is an ability that benefits many use cases, e. g., automated valet parking or automated charging. The method introduced here was developed for fully automated parcel delivery, where the need for very high —i. e., sub-centimeter and sub-degree— precision arises from the limited physical abilities of the delivery mechanism. To the best of this author's knowledge, this level of precision is unprecedented under real-world outdoor conditions.

The secret to achieving this precision is that there may not be any weak link in the whole control chain. It starts from a perception that is able to give a very good estimate on the relative position of a known (and marked) object (see Jaspers et al. [2016]). Then, the motion planning reliably provides reproducible approach trajectories, which are only planned once. The trajectory is then not fixed to the local reference coordinate frame, as usual, but instead fixed to a coordinate frame relative to the detected goal pose. Since that is not a dead-reckoning frame, the trajectory (and hence localization) may move under the ego vehicle. This is a calculated risk which, given the gentle approach trajectory and the high quality of the goal-pose tracking, introduces an error which the control part of the system is able to handle well.

In order to achieve the last order of magnitude in precision, the actuation was modified: the hardware interface to accelerator and brake (provided by a commercial drive-by-wire kit) was split to only mechanically handle the brake and an electronic accelerator interface was added. Then, after careful system identification, feedforwarding etc., a direct feedback from the estimated longitudinal error to a very slow integrator part tipped the scale. In the end, the test vehicle (a normal-sized electric car) is able to position even on inclines, operating the brake pedal such that exactly the point where the vehicle begins to creep at a glacial pace is found and utilized.

5 Discussion

The goal of this work is to improve both the smoothness and the precision of the autonomous-driving performance. The focus is on planning- and control-related aspects, i. e., the parts that actually make the vehicle drive and hence interact with its environment. As a result of the explicitly stated goal to improve the *overall* performance (not isolated algorithms in benchmarks or specific test cases), the spread of topics addressed in this work is necessarily comparatively wide.

The proposed approaches were extensively tested under real-world conditions on the test vehicles Munich Cognitive Autonomous Robot Car 3rd Generation (MuCAR-3) and MuCAR-4 (see Figure 1.1) of the Institute for Autonomous Systems Technology (TAS) as well as on vehicles from cooperation partners in the industry. Test were conducted primarily in unstructured environments, i. e., no highway scenarios with clear structures such as line markings and foreseeable road geometries, but rather tiny curved roads, gravel roads or even off-road scenarios. As a consequence, more cautious low-speed driving is performed rather than high-speed highway applications. This is a conscious decision, since the highway sector is in large parts researched and results are already brought to series development by the automotive industry, while unstructured environments still offer many unexplored scenarios.

A main lesson that should be learned from this work is a very old one:

> "A chain is no stronger than its weakest link"

(freely after Reid [1786]).

Therefore, the work started with an inspection and re-design of the TAS software architecture to clearly identify the links and their interaction. Then, real-world problems in the motion-planning domain were solved with a general trajectory-following approach, but also special considerations for different scenarios, such as convoying or positioning were presented. The work continues at the next link, motion control, and makes sure that the solutions fit together exactly.

In this last chapter, the work is concluded by re-visiting and discussing the contributions, see Section 5.1. Then, a short outlook is given in Section 5.2, where also possible future work is laid out.

5.1 Contributions

In order to recapitulate the contributions towards smoother and more precise autonomous driving in this work, not every already described novelty is listed here. Instead, the overarching story behind the major steps in the architectural, motion-planning and motion-control domain, respectively, is presented.

Architecture When this work started, the software architecture at TAS was reactive in nature. To-date, this is often still the case in the automotive industry's software stacks, at least in series production. TAS's software stack —which had been state-of-the-art, especially in the video perception domain— needed an overhaul to not fall behind w. r. t. to cooperate research. While component-wise (e. g., in the video perception domain) there is not much a small institute can do to outpace the industry, at least from an architectural point of view, a solid foundation was proposed.[1]

This work brings more system thinking to the institute's architecture. Further, the institute-internal software framework was significantly improved in the process. While this has no immediate gain for the scientific community, this work was intended to also facilitate the next generation of TAS researchers to build their work on a more solid ground and hence made contributing easier.

Driving in Unstructured Terrain While it is much easier to "show improvements" in simulation or under specific simplifications like perfect sensing or perfect control, the proposed approaches in this work were —in accordance with TAS best practice— driven and tested in the real world with test vehicles, e. g., in the woods or on mountain trails. Few institutes drive in unstructured terrain, and —apart from some military applications— this is also not extensively explored by the major autonomous driving companies. Hence, the presented motion-planning and -control framework provides novel insights and is hopefully a valuable contribution in this sector.

Real-world testing teaches valuable lessons w. r. t. to actual sensing capabilities, the subtleties of actuating a three- (or much more) ton vehicle and, last but not least, the importance of the handling of time in a distributed system. All of which are embedded (or were planned to be embedded) into the TAS autonomous-driving software stack. The latter is the joint work of almost a dozen PhD students.

This work's contribution is the tightening of the control chain and the introduction of more system thinking —rather than component thinking— into the software stack. While there was no time to implement all the proposed architectural changes, considerable improvements were presented in the motion-planning and -control domain, as well as the exploratory driving. During the course of this work, the high-level motion-control software was written from scratch, the low-level software for the older test vehicles was maintained and enhanced, and —incorporating the

[1] Presently being Technical Lead/Architect at the world's largest car supplier in a huge project with one of the world's largest car manufacturer, the author can still second this sentence in 2023.

learnings of this— a new low-level software for the project test vehicles was written from scratch.

Driving in Unstructured Terrain...in a Convoy The platooning use case is as old as driving automation (for surveys, see Alfraheed et al. [2011], Bergenhem et al. [2012]). However, usually only the highway use case is considered, and often only the longitudinal component is solved. The latter is possible if lateral component can be decoupled by, e. g., using a lane-keeping assistant — and as long as use cases such as evasion of partly blocked lanes is not considered. Hence, this work on jointly considering lateral and longitudinal convoy motion-planning and -control, that also works in unstructured environments without any infrastructure, provides novel insights and is hopefully a valuable contribution in this sector.

In the convoy case, in contrast to the free driving in unstructured environments, there is a lead vehicle to follow. This is both a blessing and a curse since a highly dynamic target needs to be tracked accurately. This part is described in Manz [2013] and continued in Fries [2019]. Once the target vehicle track is available, however, it is also a valuable virtual sensor since its trajectory provides useful information. For instance, the single convoy was often driven through high grass or under other adverse conditions that make it very hard for the ego vehicle to judge where driving is safe — this information comes for free with the target's trajectory, assuming dynamic objects are detected locally. Another example is partial blockage of a lane in the platooning case. Again, assuming that it will evade, the lead vehicle acts as sensor, making up for it occluding the ego's field of view to the front.

The contribution of this work is the best-of-both-worlds architecture for both convoying and platooning as well as the complete platooning framework, including control and communication.[2]

Positioning a Vehicle The special use case of positioning a normal-sized vehicle with sub-centimeter precision was part of a project with the Streetscooter GmbH. While the imposed restrictions due to the prototypical robotic delivery arm are admittedly a bit academic, it posed a unique challenge. The ability to achieve the required level of precision, however, is also applicable to more common use cases, e. g., automated valet parking, and hence provides novel insights and is hopefully a valuable contribution in this sector.

In order to achieve the desired precision on a pre-series test vehicle with an aftermarket drive-by-wire system, the complete control chain had to be scrutinized. Weak links were found, sometimes where they were not expected —e. g., the often-assumed ground-truth inertial navigation system (INS) with differential Real time kinematic (RTK)-Global Navigation Satellite System (GNSS) egomotion estimation— and purged. Moreover, issues which were assumed to be nearly unsurmountable —like controlling the vehicle with sub-par actuation on the border between stiction and friction— were found to be solvable with very down-to-earth methods.

[2]Note that the zero-overhead networking protocol library was developed by Dennis Faßbender who contributed this while working at TAS but did not publish this result at the time of writing.

The contribution in this work —in addition to the already mentioned control software for the low- and high-level systems (including system identification for the new test vehicles)— is the design of the positioning in box-relative coordinates as well as the approach trajectories (both path and speed profile) as an extension of the TAS motion-planning framework.

5.2 Outlook

After this work was recapitulated, an outlook on what is ahead on the road towards smoother and more precise autonomous driving follows.

There are proposed changes from when the system architecture was scrutinized that were not implemented yet. Namely, the usage of one fusion module for all sensors, which internally is split into one object-based part and a part for the unclassified information.

The former provides the most promising fused hypotheses to all perception modules, which use them utilizing the best sensor-domain knowledge. This split facilitates the development of different perception modules that still contribute to one joined environment representation, which in turn makes it easier to develop all further scene understanding, planning and control parts.

The latter gathers information that is not yet certain enough to be classified as some kind of object —or never will be, e. g., vegetation— such that the known uncertainty can be reasoned upon and reacted to accordingly. Note that this incorporates also visibility information, i. e., the lack of measurements in certain regions.

Given the resulting environment representation, a tactical planning layer could be used for decision-making, e. g., to decide whether to also use the opposing lane to evade some blockage. A first implementation was successfully tested where a high-definition map was used and traffic rules (also based on Vehicle-to-Infrastructure (V2I)) were imposed by modifying the environment to give the motion-planning framework more or less room to explore (see Heinrich et al. [2018c]). This kind of lightweight behavior planning allows to re-use the same motion planning in different scenarios while separating concerns: what should be done from how exactly it should be done. Doing both in one mighty state machine was a popular approach back in the Urban Challenge (see Buehler et al. [2009]), but is not considered expedient in this work.

Quite some time was spent to cope with actuation and the vehicle interfaces during this work. In the future, hopefully, test vehicles natively provide interfaces for speed/acceleration and steering angle/rate (or even spatiotemporal trajectories) that offer high tracking quality. This is, sadly, out of the hand of institutes, but it stands to reason that the vehicle manufacturers would benefit greatly from institutes not having to worry about low-level actuation, as this would free a lot of valuable resources that could be used for actual innovation instead.

Room for innovation exists in basically all topics mentioned in this work. Some areas, such as platooning, are hard to handle practically for an institute, as the acquisition and maintenance of the necessary assets (vehicles with sensors, hardware

and communication) is prohibitive. It stands to reason that the best scientific return of investment would be to pursue scene understanding and behavior planning. There are plenty of datasets to learn from (offline) and real-world tests can be conducted with only one closed-loop vehicle.

Obviously, it would help a lot if there was a potent simulation framework, able to simulate realistic variations of scenes with the according sensor output that react realistically on the ego decisions. If such a framework was available, one could finally benchmark different behavior algorithms, which would greatly benefit the objectivity of evaluations and thus the scientific competition. Alas, there is none — yet.

In other disciplines, like object detection or reinforcement learning, there are such benchmark suits (e. g., see Geiger et al. [2012b], Brockman et al. [2016]). There was an attempt at scenario standardization of road scenarios in Althoff et al. [2017]. But there was no implementation behind it, so its practical benefit is questionable since implementation differences negate objective evaluation. The closest thing to the desired framework at the time of writing —here, again for on-road scenarios— is Bernhard et al. [2020]. Yet, it currently has 0 citations (according to IEEE) and 14 according to Google Scholar, so apparently there is still work to do for a widely accepted benchmarking framework.

A real challenge is what happens between receiving sensor information and providing one consistent scene, such that some behavioral layer can act on it. This means that important information needs to be extracted —to not overburden the decision layer—, but no risk may be ignored. There needs to be temporal reasoning, visibility reasoning, cooperative reasoning, reasoning how to handle unexpected things —i. e., things not considered directly in the algorithm and/or that were not in the training data—, and many more situation-interpretation topics. How this will be solved is still an open question. Which is great news, as it means that autonomous driving remains an interesting field of research for the next generations of PhD students!

A Appendix

A.1 The SAE Levels of Automated Driving

SUMMARY OF SAE INTERNATIONAL'S LEVELS OF DRIVING AUTOMATION FOR ON-ROAD VEHICLES

Issued January 2014, **SAE international's J3016** provides a common taxonomy and definitions for automated driving in order to simplify communication and facilitate collaboration within technical and policy domains. It defines more than a **dozen key terms**, including those italicized below, and provides **full descriptions and examples** for each level.

The report's **six levels of driving automation** span from *no automation* to *full automation*. A **key distinction** is between level 2, where the *human driver* performs part of the *dynamic driving task*, and level 3, where the *automated driving system* performs the entire *dynamic driving task*.

These levels are **descriptive** rather than normative and **technical** rather than legal. They imply **no particular order** of market introduction. Elements indicate **minimum** rather than maximum system capabilities for each level. A particular vehicle may have multiple driving automation features such that it could operate at **different levels** depending upon the feature(s) that are engaged.

System refers to the driver assistance system, combination of driver assistance systems, or *automated driving system*. **Excluded** are **warning and momentary intervention systems**, which do not automate any part of the *dynamic driving task* on a sustained basis and therefore do not change the *human driver's* role in performing the *dynamic driving* task.

SAE level	Name	Narrative Definition	Execution of Steering and Acceleration/ Deceleration	*Monitoring* of Driving Environment	Fallback Performance of *Dynamic Driving Task*	System Capability *(Driving Modes)*
***Human driver* monitors the driving environment**						
0	**No Automation**	the full-time performance by the *human driver* of all aspects of the *dynamic driving task*, even when enhanced by warning or intervention systems	Human driver	Human driver	Human driver	n/a
1	**Driver Assistance**	the *driving mode*-specific execution by a driver assistance system of either steering or acceleration/deceleration using information about the driving environment and with the expectation that the *human driver* perform all remaining aspects of the *dynamic driving task*	Human driver and system	Human driver	Human driver	Some driving modes
2	**Partial Automation**	the *driving mode*-specific execution by one or more driver assistance systems of both steering and acceleration/ deceleration using information about the driving environment and with the expectation that the *human driver* perform all remaining aspects of the *dynamic driving task*	**System**	Human driver	Human driver	Some driving modes
***Automated driving system* ("system") monitors the driving environment**						
3	**Conditional Automation**	the *driving mode*-specific performance by an *automated driving system* of all aspects of the dynamic driving task with the expectation that the *human driver* will respond appropriately to a *request to intervene*	System	**System**	Human driver	Some driving modes
4	**High Automation**	the *driving mode*-specific performance by an automated driving system of all aspects of the *dynamic driving task*, even if a *human driver* does not respond appropriately to a *request to intervene*	System	System	**System**	Some driving modes
5	**Full Automation**	the full-time performance by an *automated driving system* of all aspects of the *dynamic driving task* under all roadway and environmental conditions that can be managed by a *human driver*	System	System	System	**All driving modes**

Key definitions in J3016 include (among others):

Dynamic driving task includes the operational (steering, braking, accelerating, monitoring the vehicle and roadway) and tactical (responding to events, determining when to change lanes, turn, use signals, etc.) aspects of the driving task, but not the strategic (determining destinations and waypoints) aspect of the driving task.

Driving mode is a type of driving scenario with characteristic *dynamic driving task* requirements (e.g., expressway merging, high speed cruising, low speed traffic jam, closed-campus operations, etc.).

Request to intervene is notification by the *automated driving system* to a *human driver* that s/he should promptly begin or resume performance of the *dynamic driving task*.

Figure A.1: taken from SAE International [2014]

A.2 The Homogeneous Transformation

Homogeneous coordinates stem from projective geometry which goes back to Möbius [1827]. Today, they are often used in robotics where points are regularly transformed between coordinate frames. To represent a point in homogeneous coordinates, simply extend it by a scaling factor —throughout this work, always 1—, for instance:

$$\mathbf{p} = \begin{bmatrix} x \\ y \end{bmatrix} \in \mathbb{R}^2 \quad \xrightarrow{\text{homogenize}} \quad \underline{\mathbf{p}} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \in \mathbb{R}^3 .$$

These homogeneous points can then be transformed using so-called Homogeneous transformation matrixs (HTMs). A transformation from a coordinate frame *from* to a coordinate frame *to* is given by

$${}^{\text{to}}\mathbf{H}_{\text{from}} = \begin{bmatrix} R & \boldsymbol{t} \\ 0 & 1 \end{bmatrix} ,$$

where R is a rotation matrix and $\boldsymbol{t}$ is a translation vector.

A HTM can be decomposed into $H = R_x R_y R_z T$, where

$$T = \begin{bmatrix} 1 & 0 & 0 & -x \\ 0 & 1 & 0 & -y \\ 0 & 0 & 1 & -z \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad R_z = \begin{bmatrix} \cos(\psi) & \sin(\psi) & 0 & 0 \\ -\sin(\psi) & \cos(\psi) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$R_y = \begin{bmatrix} \cos(\theta) & 0 & -\sin(\theta) & 0 \\ 0 & 1 & 0 & 0 \\ \sin(\theta) & 0 & \cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad R_x = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\phi) & \sin(\phi) & 0 \\ 0 & -\sin(\phi) & \cos(\phi) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} ,$$

x, y and z are spatial transitions in the respective coordinates and ϕ, θ, ψ are roll, pitch and yaw angles, respectively. Note that the definition of the signs and/or the order of rotations varies in the literature. Note further that HTMs can be chained. For a simple example, see Figure A.2, where it is used that ${}^{\text{b}}\mathbf{H}_{\text{a}}^{-1} = {}^{\text{a}}\mathbf{H}_{\text{b}}$.

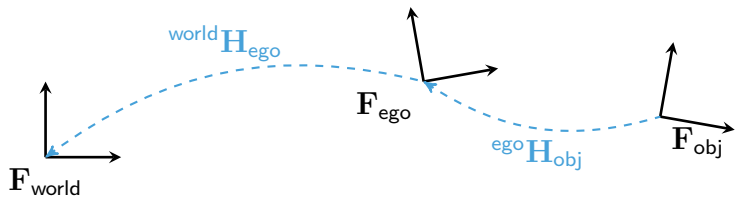


Figure A.2:
A typical example of a two-dimensional transformation chain. An object is perceived by the ego vehicle whose position is given in the world frame. Features in the object frame $\mathbf{F}_{\text{obj}}$ can be translated to the world frame $\mathbf{F}_{\text{world}}$ using the transform ${}^{\text{world}}\mathbf{H}_{\text{obj}} = {}^{\text{world}}\mathbf{H}_{\text{ego}}\,{}^{\text{ego}}\mathbf{H}_{\text{obj}}$.
The other way round, other objects in world frame can be related to object frame using ${}^{\text{obj}}\mathbf{H}_{\text{world}} = {}^{\text{world}}\mathbf{H}_{\text{obj}}^{-1} = \left({}^{\text{world}}\mathbf{H}_{\text{ego}}\,{}^{\text{ego}}\mathbf{H}_{\text{obj}}\right)^{-1} = {}^{\text{ego}}\mathbf{H}_{\text{obj}}^{-1}\,{}^{\text{world}}\mathbf{H}_{\text{ego}}^{-1} = {}^{\text{obj}}\mathbf{H}_{\text{ego}}\,{}^{\text{ego}}\mathbf{H}_{\text{world}}$.

A.3 The Clothoid

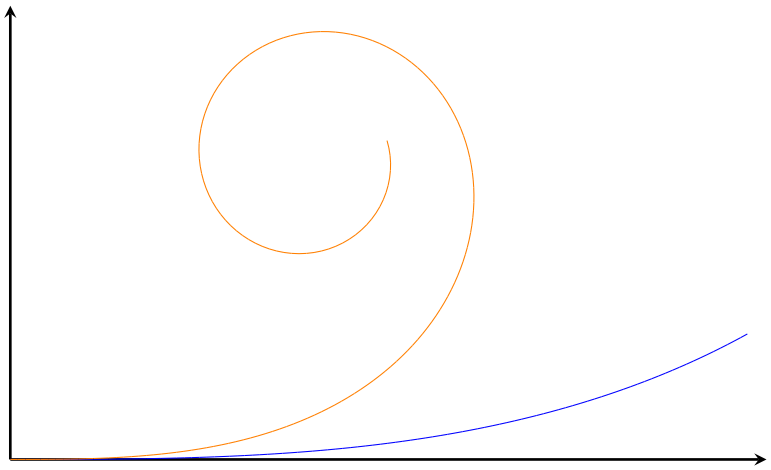


Figure A.3: Clothoids

Clothoids are geometric curves whose curvature c changes linearly over the curve's length l. The curvature of a curve is defined as the reciprocal to the radius r of the momentary osculating circle. Thus,

$$l \cdot r = \frac{l}{c} = \text{const.} \triangleq A^2, \tag{A.1}$$

where A is the so-called clothoid parameter, denotes the 'natural clothoid equation' (Kaper et al. [1954]). The heading angle θ is given by

$$\theta \triangleq \frac{c \cdot l}{2} \tag{A.2}$$

Its x–y representation can be written using Fresnel integrals as

$$x(l) = A \int_0^l \cos\left(s^2\right) \mathrm{d}s \tag{A.3}$$

$$y(l) = A \int_0^l \sin\left(s^2\right) \mathrm{d}s\,. \tag{A.4}$$

where s is the normalized angle $\frac{2\theta}{\pi}$. Given how often clothoids are evaluated in the proposed framework, the exact formulas cannot be applied for lack of computational resources. However, the transformation from parametric curve to Cartesian coordinates can be approximated efficiently, e. g., using Mielenz [2000].

A.4 The Lie Derivative

Let the Lie derivative of the function $h(\mathbf{x})$ w. r. t. the vector field $f(\mathbf{x})$ be denoted as follows (taken from Khalil [1996]):

$$\begin{aligned} L_f h(\mathbf{x}) &\triangleq \frac{\partial h}{\partial \mathbf{x}} f(\mathbf{x}) \\ L_f^0 h(\mathbf{x}) &\triangleq h(\mathbf{x}) \\ L_f^k h(\mathbf{x}) &\triangleq \frac{\partial L_f^{k-1} h}{\partial \mathbf{x}} f(\mathbf{x}) \\ L_g L_f h(\mathbf{x}) &\triangleq \frac{\partial L_f h}{\partial x} g(x) \end{aligned} \tag{A.5}$$

A.5 The Low-Level Modelling

The low-level system was implemented as a dSpace (Micro-)Autobox. It was programmed model-based via Matlab/Simulink. An example of such a model is given in Figure A.4.

A.6 The Signal Flow Charts

For a better overview of the hardware components and the signal flow through the system, a visualization can be found in Figures A.5 and A.6. Signals are either conveyed analogously (arrows with D/A or A/D), via Ethernet (TCP or UDP) or via CAN.

Both vehicle platforms feature different sensor sets and actuation interfaces. The main difference in architecture, however, is the switch from two low-level systems in MuCAR-3 to one more modern one in MuCAR-4. The interface between the high-level and the low-level system changed from CAN (Controller Area Network) to UDP which means that the high-level system is now decoupled from the vehicle bus system. This was a further step to separating the platform-specific low-level from the generic high-level.

The blocks inside the computer frames are only a subset of the software modules that are actually deployed. Additionally, the middle-ware (KogniMobil Real Time Data Base (KogMo-RTDB)) is shown which handles all inter-process communication on the main computer. Note that during the time of this work, the modules were changed multiple times. Hence, a demonstrative example set was chosen for illustrative purpose.

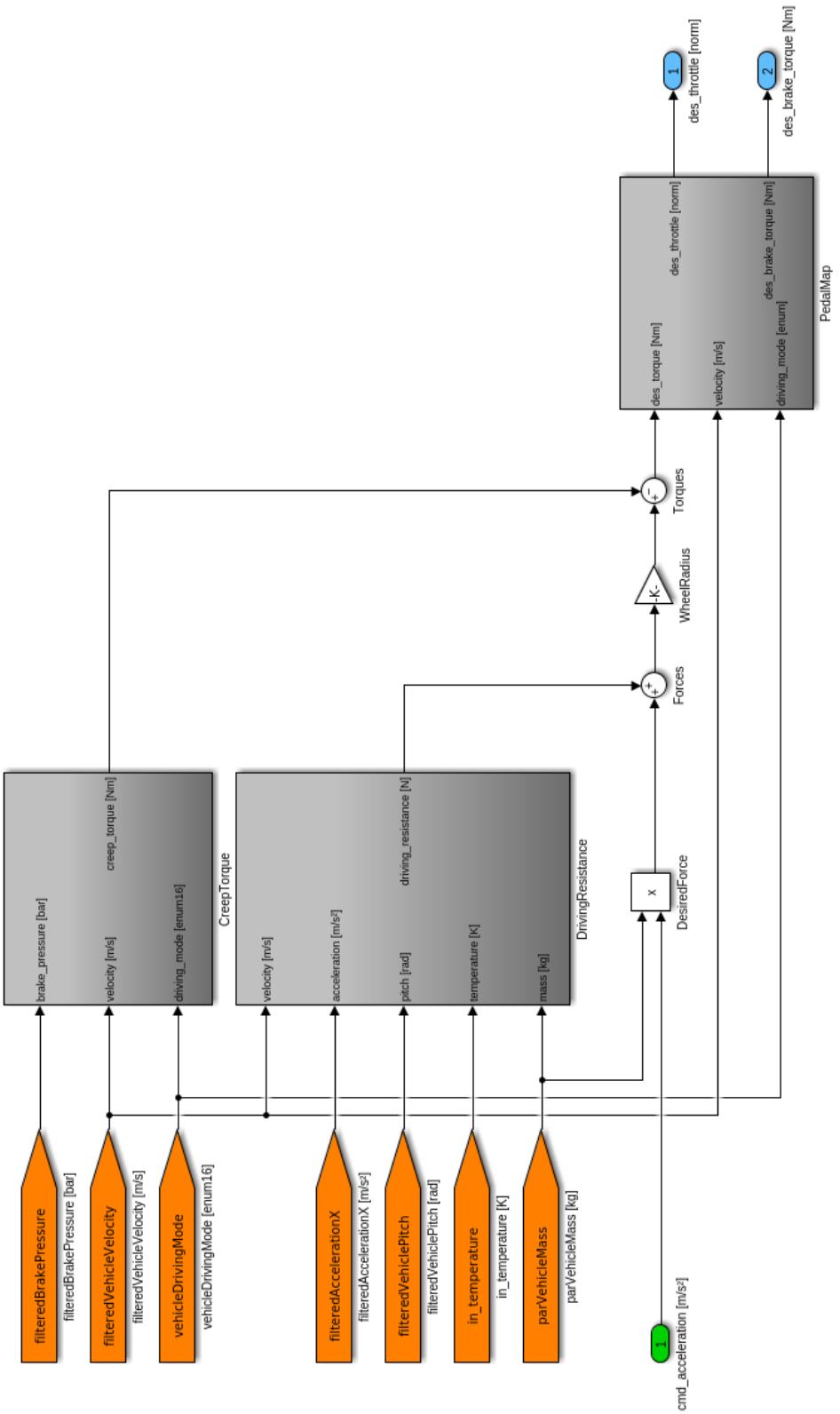


Figure A.4:
Exemplary model-based implementation for the low-level system in Simulink, here: the torque model.

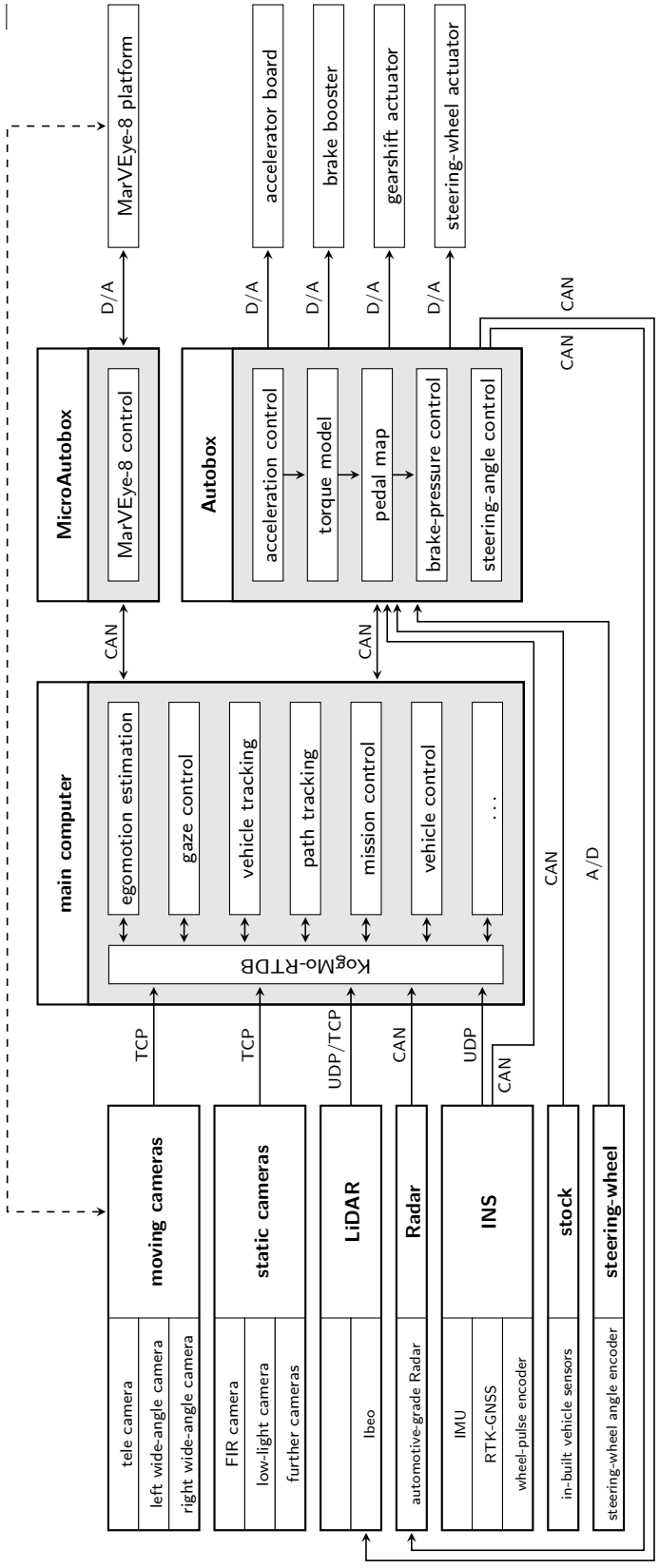


Figure A.5:
Munich Cognitive Autonomous Robot Car 3rd Generation (MuCAR-3) hardware signal flow

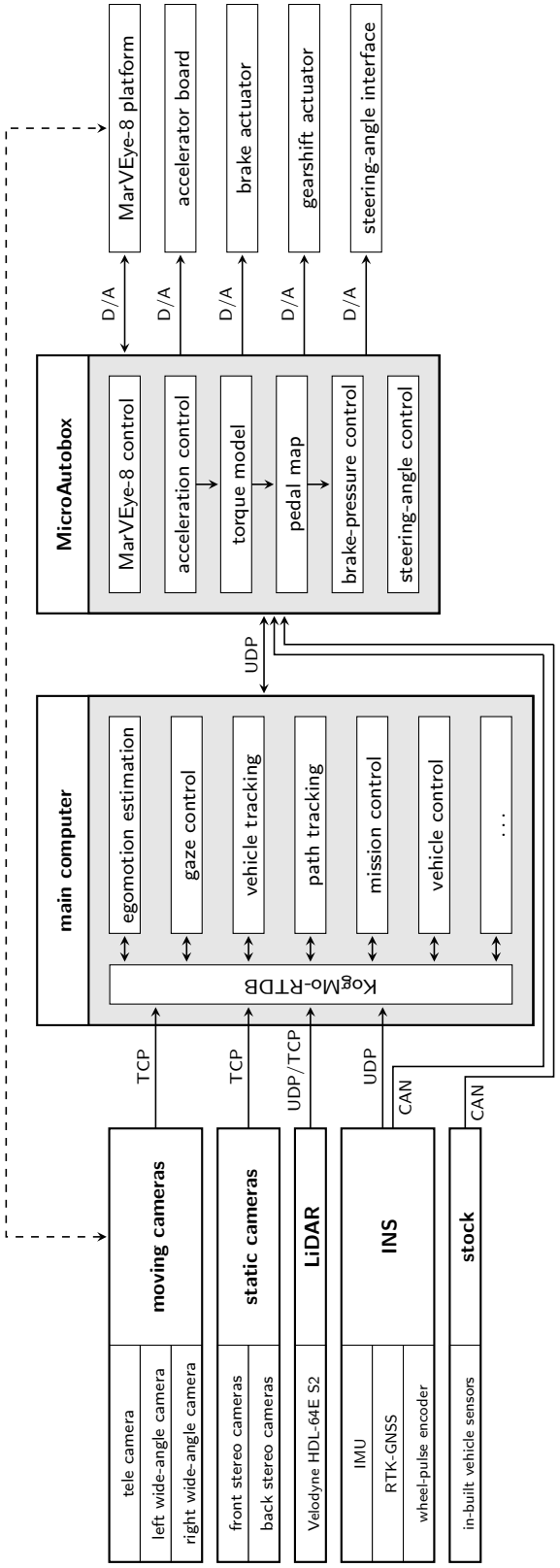


Figure A.6:
Munich Cognitive Autonomous Robot Car 4th Generation (MuCAR-4) hardware signal flow

A.7 The Smith Predictor

The so-called *Smith Predictor* (see Smith [1957, 1959]) was introduced in the context of process control of chemical reactions. However, it has been shown that it is applicable to systems where the time constants and delays are orders of magnitudes smaller (see, e. g., Heinrich [2011] or even in kHz: Schmirgel et al. [2006]). Regardless of ongoing discussions on its merit (see Skogestad [2018]), it was found to be quite beneficial in this work.

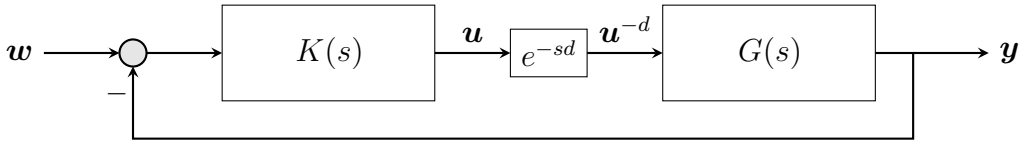


Figure A.7: classic control loop with added delay d

If a delay block e^{-sd} is added to a classic control loop, it becomes Figure A.7, which has the transfer function

$$\boldsymbol{y} = G(s)e^{-sd}K(s)(\boldsymbol{w} - \boldsymbol{y}) \quad \Leftrightarrow \quad \boldsymbol{y} = \frac{G(s)K(s)}{e^{sd} + G(s)K(s)}\boldsymbol{w}\,, \tag{A.6}$$

i. e., a delayed closed-loop system has an infinite amount of poles since the characteristic equation is non-algebraic. In order to mitigate this, consider Figure A.8, where two parts are added: a model-based prediction of the behavior of the delayed plant and a simulation loop, that models an undelayed response for optimal performance. The transfer function

$$\boldsymbol{y} = e^{-sd}\frac{G(s)K(s)}{1 + \hat{G}(s)K(s) - \underbrace{(K(s)\hat{G}(s)e^{-s\hat{d}} - G(s)e^{-sd}K(s))}_{\to 0}}\boldsymbol{w} \tag{A.7}$$

exhibits only transport delay on the output, but no change in the dynamics. Note that the plain transport delay can be further mitigated if the reference is known in advance for at least the delay d.

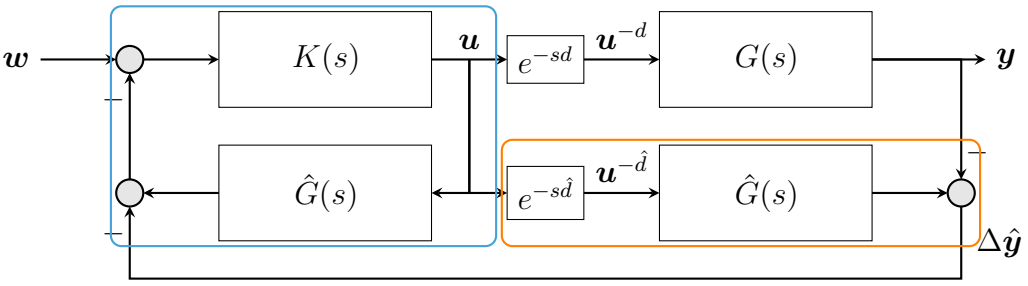


Figure A.8:
The classic smith predictor: the orange part simulates the delayed plant such that only the model error $\Delta\hat{y}$ is controlled with delay. The nominal control happens in the blue simulated no-delay loop.

A.8 The Two-Degrees-of-Freedom Controller

The controller with two degrees of freedom allows separating between

- feedforward: the reaction to the reference trajectory (evolution of the reference signal over time) and
- feedback: the reaction to errors/offsets for stabilizing the system

This can easily be shown by writing down the transfer function of Figure A.9:

$$\begin{aligned} \boldsymbol{y} &= \boldsymbol{d} + G(s)[V(s)\boldsymbol{w} + K(s)(\boldsymbol{w} - \boldsymbol{y})] \\ &= \frac{1}{1+GK}\boldsymbol{d} + \frac{GV+GK}{1+GK}\boldsymbol{w} && \text{with } V(s) = G^{-1}(s) \\ &= \frac{1}{1+GK}\boldsymbol{d} + \boldsymbol{w} && \text{with } K(s) = \text{problem-specific}\,, \end{aligned} \tag{A.8}$$

where it can be extracted that one

a) needs to have a certain knowledge about the plant transfer function $G(s)$ and
b) that its inverse (or the approximation of it) needs to be causal, in order to be implementable.

Given Equation (A.8), it can be clearly seen that the feedforward $V(s)$ can be used to influence the trajectory-following behavior and the feedback $K(s)$ to stabilize the system as usual. Having those two separated (i. e., independently tunable) parts gives this controller its name.

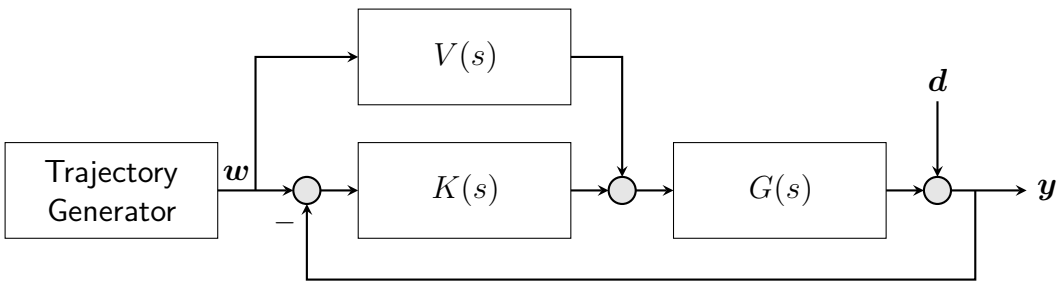


Figure A.9: The two-degrees-of-freedom controller.

A.9 The Longitudinal Vehicle-Dynamics Model

Consider the following plant model for longitudinal vehicle dynamics:

$$F_{\text{engine}} = ma_{\text{ref}} - F_{\text{drag}} - F_{\text{roll}} - F_{\text{grade}}\,, \tag{A.9}$$

with

$$F_{\text{drag}} = -\sigma\frac{pc_wAv^2}{2rT} \tag{A.10}$$

$$F_{\text{roll}} = -\sigma f_r mg\cos(\theta) \tag{A.11}$$

$$F_{\text{grade}} = mg\sin(-\theta), \tag{A.12}$$

where

- m is the mass of the vehicle,
- a_{ref} is the controller's output,
- σ is the signum —i. e., one of $\{-1, 0, 1\}$— of the current direction of movement
- c_w its drag coefficient and
- A its transverse frame.

Measurements are

- T the air temperature and
- θ the pitch angle.

Assumed constants are

- g the gravitational constant
- r the specific accelerator constant and
- p the absolute pressure (at least for one mission this should not change radically).
- f_r the wheels' friction constant is identified experimentally.

The required force F_{engine} is translated into desired torque

$$\tau_{\text{des}} = F_{\text{engine}}r_{\text{wheel}} - \tau_{\text{creep}}\,,$$

where

- r_{wheel} is the wheel radius and
- τ_{creep} is the artificial creep torque of the test vehicle.

The creep torque simulates a combustion-engine-like behavior, i. e., when the driver releases the brake, the car slowly starts moving without the need to push the accelerator pedal. The applied τ_{creep} is given as a look-up table from current velocity and brake pressure. Thus, since the amount of applied creep torque is known, it can be compensated for.

A.10 The Kinematic Bicycle Model

In the kinematic bicycle model, each vehicle axle is condensed to a single virtual wheel in its center, resulting in two wheels — like a bicycle. Further, the assumption is that there is no slip, which allows for a purely geometrical, i. e., kinematic, representation of the vehicle's motion. A graphic representation is shown in Figure A.10.

The steering angle δ of the virtual front wheel is the mean of the steering angles of the real front wheels. With δ and the assumption of perfect traction, the geometrical relation

$$\tan(\delta) = \frac{w}{r} = wc \tag{A.13}$$

follows, where $w > 0$ is the vehicle's wheelbase and r is the current radius of the virtual rear wheel's path.

The reference point for the motion model is the virtual rear wheel. There, when the vehicle moves with a velocity v, the following holds:

$$\dot{x} = v\cos(\psi) \tag{A.14}$$
$$\dot{y} = v\sin(\psi) \tag{A.15}$$
$$\dot{\psi} = vc \tag{A.16}$$
$$\dot{c} = \frac{1}{w\cos^2(\delta)}\dot{\delta} = \frac{c^2w^2+1}{w}\dot{\delta}\,. \tag{A.17}$$

Since for normal vehicles $|\delta| \ll \frac{\pi}{2}$, there are no discontinuities in this model.

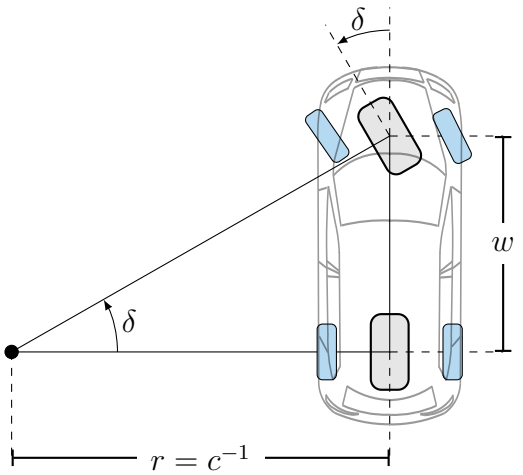


Figure A.10:
Graphical representation of the kinematic bicycle model. The blue real vehicle wheels are condensed into gray virtual wheels. There is a geometric relationship between the steering angle δ and the driven rear-wheel radius r, given the wheelbase w.

Bibliography

Alefeld, G. E., Potra, F. A., and Shi, Y. (1995). Algorithm 748: Enclosing Zeros of Continuous Functions. *ACM Transactions on Mathematical Software*, 21(3):327–344.

Alfraheed, M., Dröge, A., Klingender, M., Schilberg, D., and Jeschke, S. (2011). Longitudinal and Lateral Control in Automated Highway Systems: Their Past, Present and Future. In *Proc. Int. Conf. Intelligent Robotics and Applicat. (ICIRA)*, Aachen, Germany.

Althoff, M., Koschi, M., and Manzinger, S. (2017). CommonRoad: Composable benchmarks for motion planning on roads. In *Proc. IEEE Intelligent Vehicles Symp. (IV)*, pages 719–726. IEEE.

Andreasson, H., Saarinen, J., Cirillo, M., Stoyanov, T., and Lilienthal, A. J. (2015). Fast, Continuous State Path Smoothing to Improve Navigation Accuracy. In *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*, pages 662–669, Seattle, WA, USA.

Bacha, A., Bauman, C., Faruque, R., Fleming, M., Terwelp, C., Reinholtz, C., Hong, D., Wicks, A., Alberi, T., Anderson, D., et al. (2008). Odin: Team VictorTango's Entry in the DARPA Urban Challenge. *Journal of field Robotics*, 25(8):467–492.

Baker, C. R. and Dolan, J. M. (2008). Traffic Interaction in the Urban Challenge: Putting Boss on its Best Behavior. In *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Syst. (IROS)*, pages 1752–1758. IEEE.

Bayerl, S. F. X., Luettel, T., and Wuensche, H.-J. (2015). Following Dirt Roads at Night Time: Sensors and Features for Lane Recognition and Tracking. In *Proceedings of 7th Workshop On Planning, Perception and Navigation for Intelligent Vehicles (PPNIV), IEEE/RSJ International Conference on Intelligent Robots and Systems*, Hamburg, Germany.

Bayerl, S. F. X. and Wuensche, H.-J. (2014). Detection and Tracking of Rural Crossroads Combining Vision and LiDAR Measurements. In *Proc. IEEE Intelligent Transportation Syst. Conf. (ITSC)*, Qingdao, China.

Bellman, R. (1957). Dynamic Programming. *Princeton, USA: Princeton University Press*, 1(2):3.

Bergenhem, C., Pettersson, H., Coelingh, E., Englund, C., Shladover, S., and Tsugawa, S. (2012). Overview of Platooning Systems. In *Proceedings of the 19th ITS World Congress*, Vienna, Austria.

Bernhard, J., Esterle, K., Hart, P., and Kessler, T. (2020). BARK: Open behavior benchmarking in multi-agent environments. In *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Syst. (IROS)*, pages 6201–6208. IEEE.

Bertoncello, M. and Wee, D. (2015). Ten Ways Autonomous Driving Could Redefine the Automotive World. 6.

Borrelli, F., Falcone, P., Keviczky, T., Asgari, J., and Hrovat, D. (2005). MPC-Based Approach to Active Steering for Autonomous Vehicle Systems. *International journal of vehicle autonomous systems*, 3(2-4):265–291.

Bosch (2020). Vollautomatisiertes und fahrerloses Parken kommt an den Flughafen Stuttgart. URL: https://www.bosch-presse.de/pressportal/de/de/apcoa-bosch-und-mercedes-benz-arbeiten-am-weltweit-ersten-serieneinsatz-von-automated-valet-parking-am-flughafen-stuttgart-219648.html.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. (2016). OpenAI Gym.

Brooks, R. (1986). A Robust Layered Control System for a Mobile Robot. *IEEE journal on robotics and automation*, 2(1):14–23.

Buehler, M., Iagnemma, K., and Singh, S. (2007). *The 2005 DARPA Grand Challenge: the Great Robot Race*, volume 36. Springer.

Buehler, M., Iagnemma, K., and Singh, S. (2009). *The DARPA Urban Challenge: Autonomous Vehicles in City Traffic*, volume 56. Springer.

Burget, F., Bennewitz, M., and Burgard, W. (2016). BI²RRT*: An Efficient Sampling-Based Path Planning Framework for Task-Constrained Mobile Manipulation. In *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Syst. (IROS)*, pages 3714–3721. IEEE.

Dickmanns, E. D. (2007). *Dynamic Vision for Perception and Control of Motion*. Springer-Verlag, London, 1 edition.

Dickmanns, E. D. and Graefe, V. (1988). Dynamic Monocular Machine Vision. *Machine vision and applications*, 1(4):223–240.

Dickmanns, E. D., Mysliwetz, B., and Christians, T. (1990). An Integrated Spatio-Temporal Approach to Automatic Visual Guidance of Autonomous Vehicles. *IEEE Trans. Syst., Man, Cybern.*, 20(6):1273–1284.

Dickmanns, E. D. and Zapp, A. (1987). Autonomous High Speed Road Vehicle Guidance by Computer Vision. In *Proc. 10th IFAC World Congress*, volume 4 (preprint), pages 232–237.

Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271.

Dolgov, D., Thrun, S., Montemerlo, M., and Diebel, J. (2008). Practical Search Techniques in Path Planning for Autonomous Driving. In *Proceedings of the First International Symposium on Search Techniques in Artificial Intelligence and Robotics (STAIR-08)*, Chicago, USA. AAAI.

Dolgov, D., Thrun, S., Montemerlo, M., and Diebel, J. (2010). Path Planning for Autonomous Vehicles in Unknown Semi-structured Environments. *Int. J. Robotics Research*, 29(5):485–501.

Domahidi, A., Zgraggen, A. U., Zeilinger, M. N., Morari, M., and Jones, C. N. (2012). Efficient Interior Point Methods for Multistage Problems Arising in Receding Horizon Control. In *Proc. IEEE Conf. Decision and Control (CDC)*, pages 668–674. IEEE.

Duarte, F. and Ratti, C. (2018). The Impact of Autonomous Vehicles on Cities: A Review. *Journal of Urban Technology*, 25(4):3–18.

Ebert, F., Berthold, P., Burger, P., Engler, T., Frericks, A., Heinrich, B. C., Kallwies, J., Kusenbach, M., Luettel, T., Metzger, K. A., Michaelis, M., Naujoks, B., Sticht, A. F., and Wuensche, H.-J. (2018). ELROB 2018 – Convoy and Mule of Team MuCAR. In *Proceedings of 21th International Symposium on Measurement and Control in Robotics (ISMCR)*, Mons, Belgium.

Engler, T., Ebert, F., and Wuensche, H.-J. (2018). Semantic Grid Mapping based on Surface Classification with Supervised Learning. In *Proceedings of 21th International Symposium on Measurement and Control in Robotics (ISMCR)*, Mons, Belgium.

Falcone, P., Borrelli, F., Asgari, J., Tseng, H. E., and Hrovat, D. (2007). Predictive Active Steering Control for Autonomous Vehicle Systems. *IEEE Transactions on control systems technology*, 15(3):566–580.

Fassbender, D., Burger, P., and Luettel, T. (2016a). BA1909 BAAINBw: Konvoi- und Landmarken-Navigation – Zwischenbericht E E810 CF159. Technical Report UniBwM / LRT / TAS / TR 2016:3, Universität der Bundeswehr München, Neubiberg.

Fassbender, D., Burger, P., and Luettel, T. (2017a). BA1909 BAAINBw: Konvoi- und Landmarken-Navigation – Zwischenbericht E E810 CF159. Technical Report UniBwM / LRT / TAS / TR 2017:4, Universität der Bundeswehr München, Neubiberg.

Fassbender, D., Fries, C., Heinrich, B. C., Himmelsbach, M., and Luettel, T. (2015). BA1909 BAAINBw – Zwischenbericht E E810 CF159 Konvoi- und Landmarken-Navigation. Technical Report UniBwM / LRT / TAS / TR 2015:6, Universität der Bundeswehr München, Neubiberg.

Fassbender, D., Fries, C., Luettel, T., and Wuensche, H.-J. (2014a). TULF - Automated Convoy Driving. In *2013 Annual Military Scientific Research Report*. Bundesministerium der Verteidigung.

Fassbender, D., Heinrich, B. C., Luettel, T., and Wuensche, H.-J. (2017b). An Optimization Approach to Trajectory Generation for Autonomous Vehicle Following. In *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Syst. (IROS)*, Vancouver, BC, Canada.

Fassbender, D., Heinrich, B. C., and Wuensche, H.-J. (2016b). Motion Planning for Autonomous Vehicles in Highly Constrained Urban Environments. In *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Syst. (IROS)*, Daejeon, South Korea.

Fassbender, D., Mueller, A., and Wuensche, H.-J. (2014b). Trajectory Planning for Car-Like Robots in Unknown, Unstructured Environments. In *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Syst. (IROS)*, Chicago, IL, USA.

Fennel, H., Bunzel, S., Heinecke, H., Bielefeld, J., Fürst, S., Schnelle, K.-P., Grote, W., Maldener, N., Weber, T., Wohlgemuth, F., et al. (2006). Achievements and Exploitation of the AUTOSAR Development Partnership. *SAE Convergence Congress*, 2006:10.

Fletcher, L., Teller, S., Olson, E., Moore, D., Kuwata, Y., How, J., Leonard, J., Miller, I., Campbell, M., Huttenlocher, D., et al. (2009). The MIT–Cornell Collision and Why it Happened. In *The DARPA Urban Challenge*, pages 509–548. Springer.

Fraichard, T. and Scheuer, A. (2004). From Reeds and Shepp's to Continuous-Curvature Paths. *IEEE Transactions on Robotics*, 20(6):1025–1035.

Frenet, F. (1852). Sur Les Courbes à Double Courbure. *Journal de Mathématiques Pures et Appliquées*, pages 437–447.

Fries, C. (2019). *Modellbasierte Fahrzeugkerkennung eines Fahrerassistenzsystems zum autonomen Folgen im Konvoi*. Dissertation, Universität der Bundeswehr München, Fakultät für Luft- und Raumfahrttechnik, Neubiberg.

Fries, C., Heinrich, B. C., Himmelsbach, M., Luettel, T., and Mueller, A. (2012). BA1909 BAAINBw – Zwischenbericht E E810 CF159 Konvoi- und Landmarken-Navigation. Technical Report UniBwM / LRT / TAS / TR 2012:3, Universität der Bundeswehr München, Neubiberg.

Fries, C., Heinrich, B. C., Himmelsbach, M., Luettel, T., and Mueller, A. (2013a). BA1909 BAAINBw – Zwischenbericht E E810 CF159 Konvoi- und Landmarken-Navigation. Technical Report UniBwM / LRT / TAS / TR 2013:7, Universität der Bundeswehr München, Neubiberg.

Fries, C., Heinrich, B. C., Himmelsbach, M., Luettel, T., and Mueller, A. (2013b). BA1909 BAAINBw – Zwischenbericht E E810 CF159 Konvoi- und Landmarken-Navigation. Technical Report UniBwM / LRT / TAS / TR 2013:11, Universität der Bundeswehr München, Neubiberg.

Fries, C., Heinrich, B. C., Himmelsbach, M., Luettel, T., and Mueller, A. (2015). BA1909 BAAINBw – Zwischenbericht E E810 CF159 Konvoi- und Landmarken-Navigation. Technical Report UniBwM / LRT / TAS / TR 2015:3, Universität der Bundeswehr München, Neubiberg.

Fries, C. and Wuensche, H.-J. (2015). Autonomous Convoy Driving by Night: The Vehicle Tracking System. In *Proc. IEEE Int. Conf. Technologies for Practical Robot Applicat. (TePRA)*, Boston, MA, USA.

Fries, C. and Wuensche, H.-J. (2016). Real-time Unsupervised Feature Model Generation for a Vehicle Following System. In *Proc. IEEE Intelligent Transportation Syst. Conf. (ITSC)*, Rio de Janeiro, Brazil.

Fuchshumer, S., Schlacher, K., and Rittenschober, T. (2005). Nonlinear Vehicle Dynamics Control – A Flatness-Based Approach. In *Proc. IEEE Conf. Decision and Control (CDC)*, pages 6492–6497. IEEE.

Furgale, P., Schwesinger, U., Rufli, M., Derendarz, W., Grimmett, H., Mühlfellner, P., Wonneberger, S., Li, B., Schmidt, B., Nguyen, T. N., Cardarelli, E., Cattani, S., Brüning, S., Horstmann, S., Stellmacher, M., Rottmann, S., Mielenz, H., Köser, K., Timpner, J., Beermann, M., Häne, C., Heng, L., Lee, G. H., Fraundorfer, F., Iser, R., Triebel, R., Posner, I., Newman, P., Wolf, L., Pollefeys, M., Brosig, S., Effertz, J., Pradalier, C., and Siegwart, R. (2013). Toward Automated Driving in Cities using Close-to-Market Sensors - An Overview of the V-Charge Project. In *Proc. IEEE Intelligent Vehicles Symp. (IV)*, pages 809–816, Gold Coast, QLD, Australia.

Gammell, J. D., Barfoot, T. D., and Srinivasa, S. S. (2020). Batch Informed Trees (BIT*): Informed Asymptotically Optimal Anytime Search. *The International Journal of Robotics Research*, 39(5):543–567.

Gammell, J. D., Srinivasa, S. S., and Barfoot, T. D. (2014). Informed RRT*: Optimal Sampling-Based Path Planning Focused Via Direct Sampling of an Admissible Ellipsoidal Heuristic. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2997–3004. IEEE.

Gammell, J. D., Srinivasa, S. S., and Barfoot, T. D. (2015). Batch informed trees (BIT*): Sampling-Based Optimal Planning via the Heuristically Guided Search of Implicit Random Geometric Graphs. In *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*, pages 3067–3074. IEEE.

Gat, E., Bonnasso, R. P., Murphy, R., et al. (1998). On Three-Layer Architectures. *Artificial intelligence and mobile robots*, 195:210.

Geiger, A., Lauer, M., Moosmann, F., Ranft, B., Rapp, H., Stiller, C., and Ziegler, J. (2012a). Team AnnieWAY's Entry to the 2011 Grand Cooperative Driving Challenge. *IEEE Trans. Intell. Transp. Syst.*, 13(3):1008–1017.

Geiger, A., Lenz, P., and Urtasun, R. (2012b). Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite. In *Conference on Computer Vision and Pattern Recognition (CVPR)*.

Goebl, M., Althoff, M., Buss, M., Färber, G., Hecker, F., Heissing, B., Kraus, S., Nagel, R., Leon, F. P., Rattei, F., Russ, M., Schweitzer, M., Thuy, M., Wang, C., and Wünsche, H.-J. (2008). Design and Capabilities of the Munich Cognitive Automobile. In *Proc. IEEE Intelligent Vehicles Symp. (IV)*. IEEE Press.

Goldman, R. (2005). Curvature Formulas for Implicit Curves and Surfaces. *Computer Aided Geometric Design*, 22(7):632–658.

Gotte, C., Keller, M., Hass, C., Glander, K.-H., Seewald, A., and Bertram, T. (2015). A model predictive combined planning and control approach for guidance of automated vehicles. In *2015 IEEE International Conference on Vehicular Electronics and Safety (ICVES)*, pages 69–74.

Grigorescu, S., Trasnea, B., Cocias, T., and Macesanu, G. (2019). A Survey of Deep Learning Techniques for Autonomous Driving. *Journal of Field Robotics*, 37(3):362–386.

Grüne, L. and Pannek, J. (2017). *Nonlinear Model Predictive Control*. Springer International Publishing, Switzerland, 2 edition. YANE software library url `www.nonlinearmpc.com`.

Gu, T., Atwood, J., Dong, C., Dolan, J. M., and Lee, J.-W. (2015). Tunable and stable real-time trajectory planning for urban autonomous driving. In *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Syst. (IROS)*, pages 250–256, Hamburg, Germany.

Gutjahr, B., Gröll, L., and Werling, M. (2017). Lateral Vehicle Trajectory Optimization Using Constrained Linear Time-Varying MPC. *IEEE Trans. Intell. Transp. Syst.*, 18(6):1586–1595.

Hart, P. E., Nilsson, N. J., and Raphael, B. (1968). A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107.

Heinrich, B. C. (2011). Delay-Stability of Power Control in Wireless Networks. Master's thesis.

Heinrich, B. C. (2015). Multikonvoi – Abschlussbericht Phase I 2012-2015. Technical Report UniBwM / LRT / TAS / TR 2015:16, Universität der Bundeswehr München, Neubiberg.

Heinrich, B. C., Fassbender, D., and Wuensche, H.-J. (2016). Precise Object-Relative Positioning for Car-Like Robots. In *Proc. IEEE Intelligent Transportation Syst. Conf. (ITSC)*, pages 1720–1726, Rio de Janeiro, Brazil.

Heinrich, B. C., Fassbender, D., and Wuensche, H.-J. (2018a). Faster Collision Checks for Car-like Robot Motion Planning. In *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Syst. (IROS)*, Madrid, Spain.

Heinrich, B. C., Frericks, A., and Wuensche, H.-J. (2018b). Spatiotemporally Consistent Smooth Speed Profiles for Autonomous Driving. In *Proc. IEEE Intelligent Transportation Syst. Conf. (ITSC)*, Maui, Hawaii, USA.

Heinrich, B. C., Luettel, T., Fassbender, D., Burger, P., Ebert, F., Himmelsbach, M., Jaspers, H., Kusenbach, M., Mueller, G. R., Naujoks, B., Orth, F., Schmitt, F., and Wuensche, H.-J. (2018c). Prototyping an Autonomous Delivery Vehicle. *at – Automatisierungstechnik*, 66(2):160–182.

Heinrich, B. C., Luettel, T., and Wuensche, H.-J. (2017). A New Control Architecture for MuCAR. In *Proc. IEEE Intelligent Vehicles Symp. (IV)*, Redondo Beach, CA, USA.

Heinrich, B. C. and Wuensche, H.-J. (2017). Autonomous Cooperative Multi-Convoy Control for Off-Road Scenarios. In *8. VDI/VDE Fachtagung "AUTOREG 2017 - Automatisiertes Fahren und vernetzte Mobilität"*, volume 2292 of *VDI-Berichte*, pages 365–376, Berlin, Germany. VDI-Verlag.

Heinrich, S., Zoufahl, A., and Rojas, R. (2015). Real-time trajectory optimization under motion uncertainty using a GPU. In *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Syst. (IROS)*, pages 3572–3577, Hamburg, Germany.

Himmelsbach, M., Luettel, T., Hecker, F., von Hundelshausen, F., and Wuensche, H.-J. (2011). Autonomous Off-Road Navigation for MuCAR-3 – Improving the Tentacles Approach: Integral Structures for Sensing and Motion. *KI – Künstliche Intelligenz*, 25(2):145–149. Special Issue on Off-Road-Robotics.

Himmelsbach, M., von Hundelshausen, F., and Wuensche, H.-J. (2010). Fast Segmentation of 3D Point Clouds for Ground Vehicles. In *Proc. IEEE Intelligent Vehicles Symp. (IV)*, San Diego, CA, USA.

Himmelsbach, M. and Wuensche, H.-J. (2012). Tracking and Classification of Arbitrary Objects with Bottom-Up/Top-Down Detection. In *Proc. IEEE Intelligent Vehicles Symp. (IV)*, pages 577–582, Alcalá de Henares, Spain.

Hörl, S., Ciari, F., and Axhausen, K. W. (2016). Recent Perspectives on the Impact of Autonomous Vehicles. *Arbeitsberichte Verkehrs-und Raumplanung*, 1216.

Jaspers, H. (2021). *Towards Autonomous Driving with Visual Landmarks – Camera-based Mapping and Localization in Unknown Terrain*. Dissertation, Universität der Bundeswehr München, Fakultät für Luft- und Raumfahrttechnik, Neubiberg.

Jaspers, H., Himmelsbach, M., and Wuensche, H.-J. (2017). Multi-modal Local Terrain Maps from Vision and LiDAR. In *Proc. IEEE Intelligent Vehicles Symp. (IV)*, Redondo Beach, CA, USA.

Jaspers, H., Mueller, G. R., and Wuensche, H.-J. (2016). High Accuracy Model-Based Object Pose Estimation for Autonomous Recharging Applications. In *Proc. IEEE Winter Conf. Applicat. Comput. Vision (WACV)*, Lake Placid, NY, USA.

Jordan, M. and Perez, A. (2013). Optimal Bidirectional Rapidly-Exploring Random Trees.

Kallwies, J., Engler, T., Forkel, B., and Wuensche, H.-J. (2020). Triple-SGM: Stereo Processing using Semi-Global Matching with Cost Fusion. In *Proc. IEEE Winter Conf. Applicat. Comput. Vision (WACV)*, Snowmass Village, CO, USA.

Kallwies, J. and Wuensche, H.-J. (2018). Effective Combination of Vertical and Horizontal Stereo Vision. In *Proc. IEEE Winter Conf. Applicat. Comput. Vision (WACV)*, Lake Tahoe, NV/CA, USA.

Kammel, S., Ziegler, J., Pitzer, B., Werling, M., Gindele, T., Jagzent, D., Schröder, J., Thuy, M., Goebl, M., Hundelshausen, F. v., Pink, O., Frese, C., and Stiller, C. (2008). Team AnnieWAY's autonomous system for the 2007 DARPA Urban Challenge. *Journal of Field Robotics*, 25(9):615–639.

Kant, K. and Zucker, S. W. (1986). Toward Efficient Trajectory Planning: The Path-Velocity Decomposition. *Int. J. Robotics Research*, 5(3):72–89.

Kaper, H., Schürba, W., and Lorenz, H. (1954). *Die Klotoide als Trassierungselement*. Dümmler, Bonn, 5 edition.

Karaman, S. and Frazzoli, E. (2011). Sampling-Based Algorithms for Optimal Motion Planning. *The international journal of robotics research*, 30(7):846–894.

Kelly, A. and Nagy, B. (2003). Reactive Nonholonomic Trajectory Generation via Parametric Optimal Control. *Int. J. Robotics Research*, 22(7-8):583–601.

Kesting, A., Treiber, M., Schönhof, M., Kranke, F., and Helbing, D. (2007). Jam-Avoiding Adaptive Cruise Control (ACC) and its Impact on Traffic Dynamics. In *Traffic and Granular Flow'05*, pages 633–643. Springer.

Khalil, H. K. (1996). *Nonlinear Systems*. Prentice Hall, Upper Saddle River, NJ, USA, 3 edition.

Klaudt, S., Zlocki, A., and Eckstein, L. (2017). A-priori Map Information and Path Planning for Automated Valet-Parking. In *Proc. IEEE Intelligent Vehicles Symp. (IV)*, pages 1770–1775.

Koenig, S., Likhachev, M., and Furcy, D. (2004). Lifelong Planning A*. *Artificial Intelligence*, 155(1-2):93–146.

Kolter, J. Z., Plagemann, C., Jackson, D. T., Ng, A. Y., and Thrun, S. (2010). A Probabilistic Approach to Mixed Open-Loop and Closed-Loop Control, With Application to Extreme Autonomous Driving. In *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*, pages 839–845.

Kuffner, J. J. and LaValle, S. M. (2000). RRT-Connect: An Efficient Approach to Single-Query Path Planning. In *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*, volume 2, pages 995–1001. IEEE.

Kunz, F. and Dietmayer, K. (2016). Hybrid Discrete-Parametric Optimization for Trajectory Planning in On-Road Driving Scenarios. In *Proc. IEEE Intelligent Transportation Syst. Conf. (ITSC)*, pages 802–807, Rio de Janeiro, Brazil.

Kunz, F., Nuss, D., Wiest, J., Deusch, H., Reuter, S., Gritschneder, F., Scheel, A., Stübler, M., Bach, M., Hatzelmann, P., Wild, C., and Dietmayer, K. (2015). Autonomous Driving at Ulm University: A Modular, Robust, and Sensor-Independent Fusion Approach. In *Proc. IEEE Intelligent Vehicles Symp. (IV)*, pages 666–673.

LaValle, S. M. (1998). Rapidly-Exploring Random Trees: A New Tool for Path Planning.

Likhachev, M. and Ferguson, D. (2009). Planning Long Dynamically Feasible Maneuvers for Autonomous Vehicles. *Int. J. Robotics Research*, 28(8):933–945.

Luettel, T., Himmelsbach, M., Manz, M., Mueller, A., von Hundelshausen, F., and Wuensche, H.-J. (2011). Combining Multiple Robot Behaviors for Complex Off-Road Missions. In *Proc. IEEE Intelligent Transportation Syst. Conf. (ITSC)*, Washington, DC, USA.

Lüttel, T. (2008). Report on CoTeSys Project 118: MuCAR-3: Infrastructure and Support for Munich's Cognitive Autonomous Robot Car. Technical Report UniBwM / LRT / TAS / TR 2008:3, Universität der Bundeswehr München, Neubiberg.

Manz, M. (2013). *Modellbasierte visuelle Wahrnehmung zur autonomen Fahrzeugführung*. Dissertation, Universität der Bundeswehr München, Fakultät für Luft- und Raumfahrttechnik, Neubiberg.

Manz, M., Himmelsbach, M., Luettel, T., and Wuensche, H.-J. (2009). Fusing LIDAR and Vision for Autonomous Dirt Road Following – Incorporating a visual feature into the tentacles approach. In Dillmann, R., Beyerer, J., Stiller, C., Zöllner, J. M., and Gindele, T., Editor, *Autonome Mobile Systeme (AMS)*, Informatik aktuell, pages 17–24, Berlin, Heidelberg. Springer.

Manz, M., Himmelsbach, M., Luettel, T., and Wuensche, H.-J. (2011). Detection and Tracking of Road Networks in Rural Terrain By Fusing Vision and LIDAR. In *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Syst. (IROS)*, San Francisco, CA, USA.

Marsden, G., McDonald, M., and Brackstone, M. (2001). Towards an Understanding of Adaptive Cruise Control. *Transportation Research Part C: Emerging Technologies*, 9(1):33–51.

Maurer, M., Gerdes, J. C., Lenz, B., and Winner, H. (2016). *Autonomous Driving: Technical, Legal and Social Aspects*. Springer Nature.

Mayne, D. Q., Kerrigan, E. C., Van Wyk, E., and Falugi, P. (2011). Tube-Based Robust Nonlinear Model Predictive Control. *International Journal of Robust and Nonlinear Control*, 21(11):1341–1353.

Mayne, D. Q., Rawlings, J. B., Rao, C. V., and Scokaert, P. O. (2000). Constrained Model Predictive Control: Stability and Optimality. *Automatica*, 36(6):789–814.

Menhour, L., d'Andréa Novel, B., Fliess, M., and Mounier, H. (2014). Coupled Nonlinear Vehicle Control: Flatness-Based Setting With Algebraic Estimation Techniques. *Control Engineering Practice*, 22:135–146.

Michaelis, M., Berthold, P., Luettel, T., Meissner, D., and Wuensche, H.-J. (2019). A Merging Strategy for Gaussian Process Extended Target Estimates in Multi-Sensor Applications. In *Proc. IEEE Intelligent Vehicles Symp. (IV)*, Paris, France. accepted for publication.

Mielenz, K. D. (2000). Computation of fresnel integrals. ii. *Journal of research of the National Institute of Standards and Technology*, 105(4):589.

Montemerlo, M., Becker, J., Bhat, S., Dahlkamp, H., Dolgov, D., Ettinger, S., Haehnel, D., Hilden, T., Hoffmann, G., Huhnke, B., et al. (2008). Junior: The Stanford Entry in the Urban Challenge. *Journal of Field Robotics*, 25(9):569–597.

Möbius, A. F. (1827). *Der barycentrische Calcul ein neues Hülfsmittel zur analytischen Behandlung der Geometrie dargestellt und insbesondere auf die Bildung neuer Classen von Aufgaben und die Entwickelung mehrerer Eigenschaften der Kegelschnitte angewendet von August Ferdinand Mobius Professor der Astronomie zu Leipzig*. Verlag von Johann Ambrosius Barth.

Peng, H. and Tomizuka, M. (1991). Preview Control for Vehicle Lateral Guidance in Highway Automation. In *1991 American Control Conference*, pages 3090–3095.

Petrov, P., Boussard, C., Ammoun, S., and Nashashibi, F. (2012). A Hybrid Control for Automatic Docking of Electric Vehicles for Recharging. In *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*, pages 2966–2971, St. Paul, MN, USA.

Pucher, F. (2018). Trajectory Planning in the Frenet Space. URL: `https://fjp.at/posts/optimal-frenet/`.

Qu, Z., Wang, J., and Plaisted, C. E. (2004). A New Analytical Solution to Mobile Robot Trajectory Generation in the Presence of Moving Obstacles. *IEEE Trans. Robot.*, 20(6):978–993.

Quinlan, S. and Khatib, O. (1993). Elastic bands: connecting path planning and control. In *[1993] Proceedings IEEE International Conference on Robotics and Automation*, pages 802–807 vol.2.

Rakovic, S. V., Kouvaritakis, B., Cannon, M., Panos, C., and Findeisen, R. (2012). Parameterized Tube Model Predictive Control. *IEEE Transactions on Automatic Control*, 57(11):2746–2761.

Redmon, J., Divvala, S., Girshick, R., and Farhadi, A. (2016). You Only Look Once: Unified, Real-Time Object Detection. In *Proc. IEEE Conf. Comput. Vision and Pattern Recognition (CVPR)*, pages 779–788.

Reeds, J. and Shepp, L. (1990). Optimal paths for a car that goes both forwards and backwards. *Pacific journal of mathematics*, 145(2):367–393.

Reichart, G. and Asmus, R. (2021). Progress on the AUTOSAR Adaptive Platform for Intelligent Vehicles. In Bertram, T., Editor, *Automatisiertes Fahren 2020*, pages 67–75. Springer, Wiesbaden.

Reid, T. (1786). *Essays On The Intellectual Powers Of Man*.

Reuter, S., Vo, B.-T., Vo, B.-N., and Dietmayer, K. (2014). The Labeled Multi-Bernoulli Filter. *IEEE Transactions on Signal Processing*, 62(12):3246–3260.

Rheinmetall Defence, Diehl BGT Defence, and Universität der Bundeswehr München (2014). Projektabschlussbericht – Technologieträger Unbemanntes Landfahrzeug (TULF). Technical Report STU 9999376-000000.000.000, Rheinmetall Defence.

Rosolia, U. and Borrelli, F. (2019). Learning How to Autonomously Race a Car: A Predictive Control Approach. *IEEE Transactions on Control Systems Technology*, 28(6):2713–2719.

Rösmann, C., Feiten, W., Wösch, T., Hoffmann, F., and Bertram, T. (2013). Efficient trajectory optimization using a sparse model. In *2013 European Conference on Mobile Robots*, pages 138–143.

Rösmann, C., Hoffmann, F., and Bertram, T. (2015). Timed-elastic-bands for time-optimal point-to-point nonlinear model predictive control. In *2015 European Control Conference (ECC)*, pages 3352–3357.

Rösmann, C., Hoffmann, F., and Bertram, T. (2017). Kinodynamic trajectory optimization and control for car-like robots. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5681–5686.

SAE International (2014). Summary of SAE International's Levels of Driving Automation for On-Road Vehicles. URL: https://web.archive.org/web/20180701034327/https://cdn.oemoffhighway.com/files/base/acbm/ooh/document/2016/03/automated_driving.pdf.

Schittkowski, K. (2010). NLPQLP: A Fortran Implementation of a Sequential Quadratic Programming Algorithm with Distributed and Non-Monotone Line Search. Technical report, University of Bayreuth.

Schmirgel, H., Krah, J. O., and Berger, R. (2006). Delay Time Compensation in the current control loop of servo Drives–higher Bandwidth at no Trade-off. In *Proc. of the International Exhibition and Conference for Power Electronics, Intelligent Motion, Renewable Energy and Energy Management (PCIM Europe)*, pages 541–546.

Siedersberger, K.-H. (2003). *Komponenten zur automatischen Fahrzeugführung in sehenden (semi-)autonomen Fahrzeugen*. PhD thesis, Universität der Bundeswehr München.

Skogestad, C. G. S. (2018). Should we forget the Smith Predictor? *IFAC-PapersOnLine*, 51(4):769–774.

Smith, O. J. (1957). Closer Control of Loops with Dead Time. *Chemistry Engineering Progress*, 53(5):217–219.

Smith, O. J. (1959). A controller to overcome dead time. *ISA J.*, 6:28–33.

Snider, J. M. (2009). Automatic Steering Methods for Autonomous Automobile Path Tracking. Technical Report CMU-RI-TR-09-08, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, USA.

Steyer, S., Lenk, C., Kellner, D., Tanzmeister, G., and Wollherr, D. (2020). Grid-Based Object Tracking With Nonlinear Dynamic State and Shape Estimation. *IEEE Transactions on Intelligent Transportation Systems*, 21(7):2874–2893.

Steyer, S., Tanzmeister, G., and Wollherr, D. (2017). Object Tracking Based on Evidential Dynamic Occupancy Grids in Urban Environments. In *Proc. IEEE Intelligent Vehicles Symp. (IV)*, pages 1064–1070.

Takahashi, A., Hongo, T., Ninomiya, Y., and Sugimoto, G. (1989). Local Path Planning And Motion Control For AGV In Positioning. In *Proceedings. IEEE/RSJ International Workshop on Intelligent Robots and Systems '. (IROS '89) 'The Autonomous Mobile Robots and Its Applications*, pages 392–397.

Tanzmeister, G., Friedl, M., Wollherr, D., and Buss, M. (2014a). Efficient Evaluation of Collisions and Costs on Grid Maps for Autonomous Vehicle Motion Planning. *IEEE Trans. Intell. Transp. Syst.*, 15(5):2249–2260.

Tanzmeister, G., Thomas, J., Wollherr, D., and Buss, M. (2014b). Grid-Based Mapping and Tracking in Dynamic Environments Using a Uniform Evidential Environment Representation. In *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*, pages 6090–6095.

Thorpe, C., Herbert, M., Kanade, T., and Shafter, S. (1991). Toward Autonomous Driving: The CMU Navlab. II. Architecture and Systems. *IEEE expert*, 6(4):44–52.

Thrun, S., Montemerlo, M., Dahlkamp, H., Stavens, D., Aron, A., Diebel, J., Fong, P., Gale, J., Halpenny, M., Hoffmann, G., et al. (2006). Stanley: The Robot that Won the DARPA Grand Challenge. *J. Field Robotics*, 23(9):661–692.

Trommer, S., Kolarova, V., Fraedrich, E., Kröger, L., Kickhöfer, B., Kuhnimhof, T., Lenz, B., and Phleps, P. (2016). Autonomous Driving - The Impact of Vehicle Automation on Mobility Behaviour.

Turri, V., Carvalho, A., Tseng, H. E., Johansson, K. H., and Borrelli, F. (2013). Linear Model Predictive Control for Lane Keeping and Obstacle Avoidance on Low Curvature Roads. In *Proc. IEEE Intelligent Transportation Syst. Conf. (ITSC)*, pages 378–383. IEEE.

Unterholzner, A. (2016). *Sensor Orientation Selection and Adaptive Control of an Actuated Sensor Platform for Autonomous Vehicles*. Dissertation, Universität der Bundeswehr München, Fakultät für Luft- und Raumfahrttechnik, Neubiberg.

Urmson, C., Anhalt, J., Bagnell, D., Baker, C., Bittner, R., Clark, M., Dolan, J., Duggins, D., Galatali, T., Geyer, C., et al. (2008). Autonomous Driving in Urban Environments: Boss and the Urban Challenge. *Journal of Field Robotics*, 25(8):425–466.

Urmson, C. and Simmons, R. (2003). Approaches for Heuristically Biasing RRT Growth. In *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Syst. (IROS)*, volume 2, pages 1178–1183. IEEE.

Vo, B.-N. and Ma, W.-K. (2006). The Gaussian Mixture Probability Hypothesis Density Filter. *IEEE Transactions on signal processing*, 54(11):4091–4104.

von Hundelshausen, F., Himmelsbach, M., Hecker, F., Mueller, A., and Wuensche, H.-J. (2008). Driving with Tentacles: Integral Structures of Sensing and Motion. *J. Field Robotics*, 25(9):640–673.

Walton, D. J. and Meek, D. S. (2005). Technical Section: A Controlled Clothoid Spline. *Comput. Graph.*, 29(3):353–363.

Werling, M., Ziegler, J., Kammel, S., and Thrun, S. (2010). Optimal trajectory generation for dynamic street scenarios in a Frenet Frame. In *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*, pages 987–993.

Williams, G., Wagener, N., Goldfain, B., Drews, P., Rehg, J. M., Boots, B., and Theodorou, E. A. (2017). Information Theoretic MPC for Model-Based Reinforcement Learning. In *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*, pages 1714–1721.

Yu, M.-Y., Vasudevan, R., and Johnson-Roberson, M. (2019). Occlusion-Aware Risk Assessment for Autonomous Driving in Urban Environments. *IEEE Robotics and Automation Letters*, 4(2):2235–2241.

Yuan, T., Nuss, D. S., Krehl, G., Maile, M., and Gern, A. (2015). Fundamental Properties of Dynamic Occupancy Grid Systems for Vehicle Environment Perception. In *2015 IEEE 6th International Workshop on Computational Advances in Multi-Sensor Adaptive Processing (CAMSAP)*, pages 153–156.

Zanelli, A., Domahidi, A., Jerez, J., and Morari, M. (2020). FORCES NLP: an efficient implementation of interior-point methods for multistage nonlinear nonconvex programs. *International Journal of Control*, 93(1):13–29.

Zeitz, M. (2010). Differential Flatness: A Useful Method also for Linear SISO Systems (Differenzielle Flachheit: Eine nützliche Methodik auch für lineare SISO-Systeme). *at - Automatisierungstechnik*, 58(1):5–14.

Ziegler, J., Bender, P., Dang, T., and Stiller, C. (2014a). Trajectory Planning for Bertha – a Local, Continuous Method. In *Proc. IEEE Intelligent Vehicles Symp. (IV)*, pages 450–457, Dearborn, MI, USA.

Ziegler, J., Bender, P., Schreiber, M., Lategahn, H., Strauss, T., Stiller, C., Dang, T., Franke, U., Appenrodt, N., Keller, C. G., et al. (2014b). Making Bertha Drive – An Autonomous Journey on a Historic Route. *IEEE Intelligent Transportation Systems Magazine*, 6(2):8–20.

Ziegler, J. and Stiller, C. (2010). Fast Collision Checking for Intelligent Vehicle Motion Planning. In *Proc. IEEE Intelligent Vehicles Symp. (IV)*, pages 518–522, San Diego, CA, USA.

Ziegler, J., Werling, M., and Schroder, J. (2008). Navigating Car-Like Robots in Unstructured Environments Using an Obstacle Sensitive Cost Function. In *Proc. IEEE Intelligent Vehicles Symp. (IV)*, pages 787–791. IEEE.

List of Symbols

Abbreviations

ACC	Adaptive Cruise Control
AEB	Automatic Emergency Brake
AUTOSAR	AUTomotive Open System ARchitecture
CAN	Controller Area Network
CCC	Cooperative Cruise Control
CoTeSys	Cognition for Technical Systems
DARPA	Defense Advanced Research Projects Agency
DoF	Degrees of Freedom
EKF	Extended Kalman Filter
ELROB	European Land Robot Trial
EME	Egomotion Estimate
FIR	Far-InfraRed
GNSS	Global Navigation Satellite System
GPS	Global Positioning System
GUI	Graphical User Interface
HMI	Human Machine Interface
HSV	Hue saturation value
HTM	Homogeneous transformation matrix
IMU	inertial measurement unit
INS	inertial navigation system
KF	Kalman Filter
KogMo-RTDB	KogniMobil Real Time Data Base
LiDAR	Light Detection And Ranging
LQR	Linear Quadratic Regulator
MarVEye-8	Multifocal active / reactive Vehicle Eye 8^{th} Generation
MPC	Model Predictive Control
MuCAR	Munich Cognitive Autonomous Robot Car
MuCAR-3	Munich Cognitive Autonomous Robot Car 3^{rd} Generation
MuCAR-4	Munich Cognitive Autonomous Robot Car 4^{th} Generation
PID	Proportional Integral Differential
PKW	Personenkraftwagen
PVD	Path-Velocity Decomposition
Radar	Radio Detection And Ranging
ROS	Robot Operating System
RRT	Rapidly-Exploring Random Tree
RTK	Real time kinematic
SLAM	Simultaneous Localization and Mapping
SQP	Sequential Quadratic Programming
StS	StreetScooter GmbH
TAS	Institute for Autonomous Systems Technology
TULF	Technologieträger Unbemanntes Landfahrzeug
UKF	Unscented Kalman filter
UTM	Universal Transverse Mercator

V2I	Vehicle-to-Infrastructure
V2V	Vehicle-to-Vehicle
VaMoRs	test vehicle for autonomous mobility and computer vision, *german: Versuchsfahrzeug für autonome Mobilität und Rechnersehen*
VaMP	VaMoRs-PKW
WLAN	Wireless Local Area Network

Conventions

Basic Notations

$\overset{\circ}{(\cdot)}$	derivative with respect to runlength $\left(\frac{\partial(\cdot)}{\partial l}\right)$
$\dot{(\cdot)}$	derivative with respect to time $\left(\frac{\partial(\cdot)}{\partial t}\right)$
$(\cdot)_{[i]}$	ith component
$(\cdot)^{\mathsf{T}}$	Transposed
$\mathbf{M}$	Matrix
s	Scalar
$\mathbf{v}$	Vector

Coordinates, Transformations

$\mathbf{F}$	Cartesian coordinate system
$\mathbf{H}$	Homogeneous transformation matrix

Mathematical Definitions

$\lvert\cdot\rvert$	Absolute value
$\mathbb{N}$	Set of natural numbers
$\mathbb{R}$	Set of real numbers
$\mathbf{p}$	Point in 3D
RMSE	Root Mean Square Error

State Space

$\mathbf{x}$	State vector
$\hat{(\cdot)}$	Estimated values

Time

Δt	Duration (difference between two points in time: $\Delta t = t_1 - t_0$)
t	Point in time

Variables

Variable	unit/dim	description
$\mathbf{A}$	$\in \mathbb{R}^{n \times n}$	matrix, system dynamics matrix or constraints (dependent on context)
A	(m^2)	in collision checking: area
ΔA	(m^2)	in collision checking: discretized area
a	$(\frac{\mathsf{m}}{\mathsf{s}^2})$	acceleration
a_{lat}	$(\frac{\mathsf{m}}{\mathsf{s}^2})$	lateral acceleration (in y direction)
a_{lon}	$(\frac{\mathsf{m}}{\mathsf{s}^2})$	longitudinal acceleration (in x direction)
Δa	$(\frac{\mathsf{m}}{\mathsf{s}^2})$	difference in acceleration, can also be delta to desired
$\mathbf{B}$	$\in \mathbb{R}^{m \times 1}$	system input matrix
B	$\in \mathbb{R}^{\eta n \times \lceil \nu h \rceil}$	in CCC: state buffer
b	(m)	in collision checking: half-width of an object
$\mathbf{b}$	$\in \mathbb{R}^{n \times 1}$	vector of bounds/constraints (e. g., upper: $\overline{\mathbf{b}}$, lower: $\underline{\mathbf{b}}$)
C	$(\cdot)$	coordinate frame: vehicle center of rotation
D	(m)	sampling distance
c	$(\frac{1}{\mathsf{m}})$	curvature
c_{max}	$(\frac{1}{\mathsf{m}})$	curvature maximum
$\mathring{c}$	$(\frac{1}{\mathsf{m}^2})$	change of curvature w. r. t. run length
d	(m)	distance
d_{min}	(m)	distance minimum
e	$(\cdot)$	error (desired value − current value), dimensions/unit appropriately
$f()$	$(\cdot)$	control/state-update law $\mathbf{x}^{\mathsf{k}+1} = f(\mathbf{x}^{\mathsf{k}})$
$\mathbf{f}$	(m,m)	in collision checking: disc center front
$\mathbf{H}$	$\in \mathbb{R}^{n \times n}$	Hessian matrix
h	$\in \mathbb{N}$	length of horizon (number of timesteps predicted)
i	$\in \mathbb{N}$	index
i_{max}	$\in \mathbb{N}$	index maximum
J	$(\cdot)$	cost term to be minimized
j	$\in \mathbb{N}$	another index, if multiple indices are needed
j	$(\frac{\mathsf{m}}{\mathsf{s}^3})$	jerk, third derivative of place in time
$\mathbf{K}$	$\in \mathbb{R}^{1 \times m}$	system control/feedback matrix
$\mathcal{K}$	$(\cdot)$	set of all key indices in two-step trajectory generation
k	$\in \mathbb{N}$	counting variable, often for timesteps
$k_{(\cdot)}$	$(\cdot)$	weighting factor for $(\cdot)$, unit accordingly
$\mathbf{l}$	(m,m)	localization on trajectory, can be function of l or t
l	(m)	length (coordinate along a path or of an object)
l_f	(m)	in collision checking: length from rear-axle center to front of vehicle
l_r	(m)	in collision checking: length from rear-axle center to rear of vehicle
Δl	(m)	difference in length coordinate
M	$\in \mathbb{N}$	degree of polynomial
$\mathbf{m}_{(\cdot)}$	(m,m)	tangent at $(\cdot)$
$\mathbf{n}_{(\cdot)}$	(m,m)	normal at $(\cdot)$
N	$\in \mathbb{N}$	maximum number of things, often upper bounding on of n, i, j, k
n	$\in \mathbb{N}$	number of things, often number of states
O	(m^2)	in collision checking: oversampling area
ΔO	(m^2)	in collision checking: oversampling area, discretized
$\mathbf{P}$	$(\cdot)$	in CCC: driven path

$\mathbf{p}$	$\in \mathbb{R}^n$	point
p	$(\cdot)$	polynomial coefficient
Q	$\in \mathbb{R}^{n \times q_{\text{max}}}$	in CCC: time-sorted update queue
$\mathbf{q}$	$(\cdot)$	in CCC: time-sorted update queue entry
q_{max}	$\in \mathbb{N}$	in CCC: maximum number of time-sorted update queue entries
$\mathbf{r}$	(m,m)	in collision checking: disc center rear
$\mathbf{r}$	(m,m)	in localization: vehicle rear-axle center point
$\mathbf{r}$	$\in \mathbb{R}^{n \times N}$	reference line or trajectory (e. g., for speed profile generation)
r	(m)	radius
r_c	(m)	in collision checking: radius, central-circle
r_i	(m)	in collision checking: radius, inner
r_m	(m)	in collision checking: radius, momentarily driven by vehicle
r_o	(m)	in collision checking: radius, outer
r_p	(m)	in collision checking: radius, of frontal collision disc
r_r	(m)	in collision checking: radius, of rear collision disc
s	(m)	in collision checking: tuning variable for collision-disc placement
T	(s)	time constant
$\mathrm{T}(\cdot)$	$(\cdot)$	transition function used to interpolate between points
t	(s)	time
t_{gap}	(s)	in ACC: constant-time gap
Δt	(s)	duration, difference in time, can also be delta to desired
U	(m^2)	in collision checking: undersampling area
ΔU	(m^2)	in collision checking: undersampling area, discretized
$\boldsymbol{u}$	$\in \mathbb{R}^m$	vector of inputs
v	($\frac{\text{m}}{\text{s}}$)	speed
$\bar{v}^{\{\leftarrow,\rightarrow\}}$	($\frac{\text{m}}{\text{s}}$)	maximal reachable speed from {left, right}
$\tilde{v}$	($\frac{\text{m}}{\text{s}}$)	speed a constant acceleration profile would have
Δv	($\frac{\text{m}}{\text{s}}$)	difference in speed, can also be delta to desired
w	(m)	vehicle's wheelbase
w	(m)	in collision checking: width of an object
$\mathbf{x}$	$\in \mathbb{R}^n$	vector of states
x	(m)	x coordinate
y	(m)	y coordinate
$\boldsymbol{u}$	$\in \mathbb{R}^m$	vector of inputs
Δy	(m)	displacement (lateral deviation w. r. t. localization)
$\boldsymbol{z}$	$\in \mathbb{R}^n$	vector of flat coordinates, see section 4.3.3
α	(rad)	in collision checking: angle used in collision-disc placement
δ	(rad)	steering angle of virtual front wheel (one-track model)
$\dot{\delta}$	(rad)	steering rate of virtual front wheel (one-track model)
ε	(m,rad)	in CCC: lateral control error
η	$(\cdot)$	number of followers in a convoy
ν	$(\cdot)$	in CCC: tuning parameter $> 1 \in \mathbb{R}$ for state buffer size
ξ	$\in \mathbb{R}^{3\eta}$	in CCC: longitudinal state vector
τ	(s)	elapsed time on a segment (in two-step trajectory generation)
τ	(s)	in CCC: point in time (running variable)
Φ	$(\cdot)$	diffeomorphism, see section 4.3.3
ψ	(rad)	yaw angle