universität freiburg

Optimization-Based Motion Planning and Obstacle Avoidance for Autonomous Driving and Racing

Rudolf Reiter

University of Freiburg, Faculty of Engineering
Department of Microsystems Engineering
Systems Control and Optimization Laboratory

November 12, 2024

Optimization-Based Motion Planning and Obstacle Avoidance for Autonomous Driving and Racing

Rudolf REITER

Dean: Prof. Dr. Frank Balle

Examination committee:

First reviewer: Prof. Dr. Moritz Diehl
Second reviewer: Prof. Dr. Melanie Zeilinger
Observer: Prof. Dr. Joschka Bödecker
Chair of committee: Prof. Dr. Stefan J. Rupitsch

Dissertation zur Erlangung des Doktorgrades der Technischen Fakultät der Albert-Ludwigs-Universität Freiburg im Breisgau

November 12, 2024

© 2024 University of Freiburg – Faculty of Engineering
Self-published, Rudolf Reiter, Georges-Köhler-Allee 102, DE-79110 Freiburg i. Br. (Germany)

Alle Rechte vorbehalten. Alle Inhalte dieses Werkes, insbesondere Texte, Fotografien und Grafiken, sind urheberrechtlich geschützt. Das Urheberrecht liegt, soweit nicht ausdrücklich anders gekennzeichnet, bei der Albert-Ludwigs-Universität Freiburg.

All rights reserved. No part of the publication may be reproduced in any form by print, photoprint, microfilm, electronic or any other means without written permission from Albert-Ludwigs-Universität Freiburg.

Acknowledgments

The journey of writing this thesis was an exciting and dense period of my life. I met great people, immersed in interesting scientific fields, and visited stunning research institutions and networking events worldwide.

My great thanks and admiration go to Moritz Diehl, who sparked my interest in optimization, taught me the fundamentals constituting the base of this thesis, set the ground for my vast network of scientists, and illuminated viewpoints of complex topics far beyond research. Truly, Moritz Diehl is one of the most influential people I have met. Not only does his profound technical analysis of any topic he encountered impress me, but also the way he handles social interaction, integration, and conflict resolution. Beyond his highest-quality scientific supervision, he connects people of all fields to propel their careers and strengthen the community. Moreover, he ignited inspiring discussions with our diverse international group about challenging topics of our time.

I thank my co-supervisor, Alberto Bemporad, for the prosperous and insightful discussions, with never-ending new ideas and facets of research problems. Moreover, I am grateful to Stefano Di Cariano and Rien Quirynen for their supervision and engagement in making my research visit at Mitsubishi Electric Research Laboratories an excellent experience. Melanie Zeilinger enabled me to visit her highly skilled team at the ETH in Zürich, which involved many interactions about the latest research directions, which I highly appreciate.

At the University of Freiburg, the supervision of my senior research fellow, Armin Nurkanović, and the collaboration with my favorite reinforcement learning colleague, Jasper Hoffmann, substantially improved this research. Several further colleagues gave highly constructive feedback to this thesis, including SYSCOP fellows Andrea Ghezzi, Florian Messerer, and Jochem De Schutter, senior researchers Johannes Köhler from ETH Zürich, Jon Arrizabalaga from the Technical University of Munich, and Karin Festl from the Virtual Vehicle Research Center.

My home institution, the SYSCOP laboratory in Freiburg, was a warm and welcoming place, balancing fun coffee chats with serious discussions about research and politics. The diverse backgrounds of people made it very rich, and our visitors always brought new points of view with them. I thank the SYSCOP family, including, besides already mentioned colleagues, Katrin Baumgärtner, Adrian Bürger, Leonardo Cecchin, Christian Dietz, Shiying Dong, Lilli Frison, Gianluca Frison, Jonathan Frey, Yunfan Gao, Arne Groß, Jakob Harzer, Hannes Homburger, David Kiessling, Mikhail Katliar, Rachel Leuthold, Anton Pozharskiy, Per Rutquist, Tommaso Sartor, Leo Simpson, Wim Van Roy and Wenliang Zhang.

Pivotal to this thesis was the Marie-Sklodowska Curie Innovative Training Network program ELO-X, a stunning scientific program of the European Union. It involved an utterly successful network buildup among 23 doctoral students, 14 professional supervisors, and six advisors. At the numerous events, we grew together not only as a group of colleagues but also as friends. Particularly, I want to mention the fellows Ramin Abbasi, Flavia Acerbo, Jean Pierre Allamaa, Katrin Baumgärtner, Lahcen El Bourkhissi, Leonardo Cecchin, Andrea Ghezzi, Yunfan Gao, Jasper Hoffmann, Nicolas Kessler, Kristoffer Loewenstein, Amon Lahr, Florian Messerer, Leo Simpson, Wim Van Roy, Johannes Waibel, Renzi Wang, Jing Xie, Shaohui Yang, Shuhao Zhang, Yuan Zhang. Moreover, the scientific advisors Daniele Bernardini, Joschka Bödecker, Antonella Ferrara, Lorenzo Fagiano, Boris Houska, Colin Jones, Silvia Mastellone, Ion Necoara, Toshiyuki Ohtsuka, Panos Patrinos, Riccardo Scattolini, Jan Swevers, Son Tong, Adrian Trachte and Melanie Zeilinger contributed to the success of the program and, therefore to the outcome of this thesis. Moritz Diehl initiated and excellently guided this program and made Freiburg the scientific hub of this community.

The idea of this thesis originated in the Virtual Vehicle Research Center in Austria with the fun group of friends Zlatan Ajanović, Christian Doppler, Karin Festl, Hannes Holzer, Christoph Pilz, and the introduction to the scientific world by Johannes Rumetshofer and Michael Stolz. My initial time in Graz involved participation in race car competitions, which was very exciting. I particularly thank the Autonomous Racing Graz colleagues and initiators Martin Kirchengast, Tobias Renzler, Markus Schratter, Michael Stolz, Daniel Watzenig, and Jasmina Zubača for their hard work in pushing our team toward top ranks in the autonomous racing competitions and teaching me a lot about robotic software development.

Numerous discussions along the way helped me grasp difficult scientific topics. In addition to the already mentioned colleagues, I highly enjoyed digging into complex problems with Johannes Köhler, Shamil Mamedov, Robert McAllister, Angel Romero, and Jelena Trisovic.

Last but not least, I thank my parents, Susanne and Georg, for their encouraging words and warm welcomes at their home, as well as my brother Gerald, his wife Johanna, and their wonderful toddlers Klara, Lea, Paul, and Tobias, for brightening up our lives. My mother and father-in-law, Evelyn and Andreas, also deserve a huge thanks for supporting me with kind words and delicious food.

This work would not have been possible without Eli. My heartfelt gratitude goes to the person who not only moved with me to Germany into a city completely unknown to us but also stood behind me every second, supporting and encouraging me in every situation. She managed to endure me in difficult times and celebrated the good times with me. She always found the right words to push me forward and the wrong words when proofreading my drafts.

Freiburg im Breisgau *Rudolf Reiter*
December 2024

Funding

This research was mainly funded by the European Union's Horizon 2020 research and innovation programme ELO-X (Embedded Learning and Optimization for the Next Generation of Industrial Systems) under the Marie Sklodowska-Curie grant agreement No. 953348.
Moreover, this research was supported by DFG via projects 424107692, 504452366 (SPP 2364), and 525018088, by BMWK via 03EI4057A and 03EN3054B, and from the ECSEL Joint Undertaking (JU) under grant agreement No 826653. The JU receives support from the European Union's Horizon 2020 research and innovation program and Germany, Austria, Netherlands, France, Finland, Italy, Belgium, Czech Republic, Spain, Cyprus, Lithuania, and Turkey. This research was also funded by the program "IKT der Zukunft" of the Austrian Federal Ministry for Climate Action (BMK). Two publications were written at Virtual Vehicle Research GmbH in Graz and partially funded by the COMET K2 Competence Centers for Excellent Technologies from the Austrian Federal Ministry for Climate Action (BMK), the Austrian Federal Ministry for Labour and Economy (BMAW), the Province of Styria (Dept. 12) and the Styrian Business Promotion Agency (SFG).

Abstract

Planning feasible trajectories and considering the nonconvex problem of obstacle avoidance pose significant challenges in autonomous driving. The complexity is, among other sources, due to the high-dimensional planning space, combinatorial choices that scale the problem difficulty exponentially, hard real-time requirements, major nonconvexities, and the difficult motion prediction of other vehicles.

This thesis reports on motion planning and obstacle avoidance for autonomous driving. The proposed algorithms address the abovementioned difficulties by utilizing optimization-based methods. Particularly, this thesis proposes to use nonlinear and combinatorial optimization techniques, possibly improved algorithmically by machine learning techniques. The thesis pivots around three main fields within this context: (i) the vehicle model for on-road driving as part of an optimization solver, (ii) strategic decision-making and planning in a highly nonconvex space, and (iii) interactive planning in competitive scenarios. Optimization-based techniques provide the advantages of utilizing model knowledge, providing safety guarantees (or at least safety certifications), and separating the model identification, problem formulation, and solution algorithms.

The vehicle models for on-road driving are based on road-aligned coordinates. An a priori computation of the road-aligned coordinate transformation curve is proposed to allow numerical optimization algorithms, particularly sequential quadratic programming, to solve the problem efficiently. Due to the numerically favorable properties of collision avoidance in the Cartesian coordinate frame, a novel lifted formulation in both the transformed and the Cartesian configuration states expands the state space. By barely increasing the computation time, the obstacles can be tightly and safely over-approximated within the lifted formulation, as shown in simulations.

A major challenge of obstacle avoidance is the inherent nonconvexity, which, for example, involves the decision of overtaking left or right. Relying purely on discrete planning is burdened by the high-dimensional planning space and suffers from the curse of dimensionality. Mixed-integer optimization utilizes gradients of continuous variables and discrete optimization to find integer assignments in a combined framework. However, the computation time scales exponentially with the number of integer variables. Three motion planning algorithms based on mixed-integer optimization for specific obstacle characteristics are proposed to reduce the online computational burden. For a static environment, a spatial reformulation allows the consideration of a large number of obstacles. The performance of this algorithm was evaluated on a real-world race track on embedded hardware. For structured highway driving, a novel spatio-temporal reformulation significantly reduces the number of integer variables and allows for long-term planning. A final contribution proposes a generic way of learning to predict integer variable assignments by machine learning, enabling real-time planning with high closed-loop performance in the simulated scenarios.

Besides the challenge of nonconvexity, planning problems stemming from autonomous racing competitions may also comprise the task of obtaining interactive competitive behavior. For this third major field, this thesis contributes with real-time feasible algorithms for planning and predicting other vehicles. A hierarchical approach using reinforcement learning and model predictive control can learn interactive behavior, such as blocking other vehicles from overtaking in simulations. The optimization layer provides safety in this context.

Kurze Zusammenfassung

Die Planung von sicheren Trajektorien für selbstfahrende Fahrzeuge in Echtzeit ist komplex und höchst anspruchsvoll. Die Komplexität ist unter anderem auf den hochdimensionalen Planungsraum, kombinatorische Entscheidungen, welche die Schwierigkeit des Problems exponentiell erhöhen, Echtzeitanforderungen, Nichtkonvexitäten und die schwierige Vorhersage anderer Fahrzeuge zurückzuführen.

In dieser Doktorarbeit werden Methoden zur Trajektorienplanung und Hindernisvermeidung für das autonome Fahren vorgestellt, welche die oben genannten Schwierigkeiten durch den Einsatz von optimierungsbasierten Methoden lösen. Insbesondere werden in dieser Arbeit nichtlineare und kombinatorische Optimierungstechniken vorgeschlagen, die, unter anderem, durch maschinelle Lerntechniken algorithmisch verbessert werden. Die Arbeit konzentriert sich in diesem Zusammenhang auf drei Hauptbereiche: (i) das Fahrzeugmodell als Teil eines Optimierungslösers, (ii) strategische Entscheidungsfindung und Planung in einem hochgradig nichtkonvexen Raum und (iii) interaktive Planung bei Rennwettbewerben mit autonomen Fahrzeugen. Optimierungsbasierte Techniken bieten den Vorteil, dass sie Modellwissen in den Planungsalgorithmus integrieren, Sicherheitsgarantien oder zumindest Sicherheitszertifizierungen bieten und die Modellidentifikation, die Problemformulierung und die Lösungsalgorithmen voneinander trennen.

Es werden Fahrzeugmodelle verwendet, die in straßenbündige Koordinaten transformiert werden. Um einhergehende numerische Optimierungsalgorithmen echtzeitfähig lösen zu können, wird eine effiziente Vorausberechnung der Transformationskurve verwendet. Zudem wird eine erweiterte Zustandsraumdarstellung in zwei Koordinatensystemen entwickelt. Es werden die numerisch günstigen Eigenschaften von Formulierungen der Kollisionsvermeidung im kartesischen Koordinatensystem mit Vorteilen der straßenbündigen Transformation kombiniert. Bei kaum erhöhter Rechenzeit können die Hindernisse in der erweiterten Formulierung effizent und sicher überapproximiert werden.

Eine große Herausforderung bei der Hindernisvermeidung ist die inhärente Nichtkonvexität, die zum Beispiel die Entscheidung darüber beinhaltet, links oder rechts zu überholen. Eine Planung im rein diskreten Zustandsraum wird durch den hochdimensionalen Raum erschwert, der für zustandsdiskrete Methoden Probleme bereitet. Die gemischt-ganzzahlige Optimierung nutzt Gradienten von kontinuierlichen Variablen und Methoden der diskreten Optimierung, um Optimierungsprobleme zu lösen. Allerdings skaliert die Berechnungszeit exponentiell mit der Anzahl der ganzzahligen Variablen. Es werden drei Algorithmen zur Trajektorienplanung vorgeschlagen, die auf der gemischt-ganzzahligen Optimierung basieren, um den Online-Rechenaufwand zu reduzieren. Für eine statische Umgebung ermöglicht eine räumliche Umformulierung die Berücksichtigung einer großen Anzahl von Hindernissen. Die Effktivität dieses Algorithmus wurde auf einer realen Rennstrecke auf eingebetteter Hardware evaluiert. Für das strukturierte Fahren auf Autobahnen reduziert eine neuartige Formulierung in den Weg- und Zeit-Koordinaten die Anzahl der ganzzahligen Variablen erheblich und ermöglicht eine Planung in großer zeitlicher Distanz. In einem letzten Kapitel wird eine generische Methode zur Vorhersage von ganzzahligen Variablenzuordnungen durch maschinelles Lernen vorgeschlagen, die eine Echtzeitplanung mit hoher Regelgüte in den simulierten Szenarien ermöglicht.

Neben der Herausforderung der Nichtkonvexität können Planungsprobleme, die vor allem in autonomen Rennwettbewerben auftreten, auch die Schwierigkeiten des interaktiven Planens mit sich bringen. Diese Arbeit stellt in diesem dritten großen Bereich Echtzeit-Algorithmen zur Planung und Vorhersage der Trajektorien anderer Fahrzeuge vor. Ein hierarchischer Ansatz, der Reinforcement Learning und modelprädiktive Regelung verwendet, kann interaktives Verhalten in Simulationen erlernen. Dabei wird zum Beispiel das Blockieren von Überholmanövern anderer Rennteilnehmer erlernt. Das Optimierungsmodul und die zugehörigen Beschränkungen liefern in diesem Zusammenhang die notwendige Sicherheit.

Abbreviations

AD	autonomous driving
AI	artificial intelligence
ARG	Autonomous Racing Graz
BnB	branch-and-bound
BFGS	Broyden–Fletcher–Goldfarb–Shanno
CC	Chebychev center
CCF	Cartesian coordinate frame
CF	coordinate frame
CG	center of gravity
COCO	combinatorial offline convex online
CP	convex program
DAE	differential algebraic equation
DAgger	dataset aggregation
DM	decision making
DNN	deep neural network
DP	dynamic programming
EDS	equivariant deep set
EV	ego vehicle
FCF	Frenet coordinate frame
FF	feed forward

FONC	first-order necessary condition
FP	feasibility projector
GAIL	generative adversarial inverse reinforcement learning
HLNLP	high-level nonlinear program
IL	imitation learning
IOC	inverse optimal control
IRL	inverse reinforcement learning
IVP	initial value problem
KKT	Karush-Kuhn-Tucker
LLNLP	low-level nonlinear program
LP	linear program
LSTM	long short term memory
LSTMP	long short term motion planner
LTF	long-term motion planning formulation
MI	mixed-integer
MILP	mixed-integer linear program
MIMP	mixed-integer motion planner
MINLP	mixed-integer nonlinear programming
MIOCP	mixed-integer optimal control problems
MIP-DM	mixed-integer programming-based decision maker
MIP	mixed-integer program
MIQP	mixed-integer quadratic program
MPC	model predictive control
MPPI	model predictive path integral
NLP	nonlinear program
NMPC	nonlinear model predictive control
NN	neural network

OCP	optimal control problem
ODE	ordinary differential equation
PCC	Pearson correlation coefficient
PPO	proximal policy optimization
QP	quadratic program
REDS	recurrent equivariant deep set
RL	reinforcement learning
RNN	recurrent neural network
RTI	real-time iteration
SAC	soft actor critic
SD	simulation distribution
SLT	spatio-lateral-temporal
SQP	sequential quadratic programming
ST	spatio-temporal
STF	short-term motion planning formulation
SV	surrounding vehicle
TD	training distribution
VT	velocity-time

Notation and Symbols

In the following, the notation and symbols used within this thesis are listed in a tabular form.

Notation	
$\mathbb{R}$	set of real numbers
$\mathbb{R}^n$	set of real valued n-vectors
$\mathbb{R}^{n\times m}$	set of real valued $n \times m$-matrices
$\mathbb{R}_{\geq 0}$	set of non-negative real numbers
$\mathbb{Z}$	set of integer numbers
$\mathbb{N}$	set of natural numbers
$\mathbb{N}_{>0}$	set of strictly positive natural numbers
$\mathbb{N}_{[m:n]}$	set of natural numbers in the interval $[m, n]$, with $m < n$
$\det(A)$	determinant of matrix A
$\forall$	for all
$\lceil x \rceil$	rounding x to the smallest larger integer
$\lfloor x \rfloor$	rounding x to the largest smaller integer
$x^\top$	the transposed of vector x
$\partial/\partial x$	partial derivative w.r.t. x
$\mathrm{d}/\mathrm{d}x$	total derivative w.r.t. x
$\dot{x}$	time derivative $\mathrm{d}x/\mathrm{d}t$ of x
x'	spatial derivative $\mathrm{d}x/\mathrm{d}\sigma$ of x with spatial variable σ
$\lvert x\rvert$	absolute value of x
$\lvert\lvert x\rvert\rvert$, $\lvert\lvert x\rvert\rvert_2$	2-norm of x
$\lvert\lvert x\rvert\rvert_p$	p-norm of x

$\nabla_x f(x),\ f:\mathbb{R}^n \to \mathbb{R}$	gradient vector of $f(x)$ in $\mathbb{R}^n$
$\nabla_x^2 f(x),\ f:\mathbb{R}^n \to \mathbb{R}$	Hessian matrix of $f(x)$ in $\mathbb{R}^{n\times n}$
$\mathcal{O}\big(f(n)\big)$	"big O of $f(n)$", asymptotic notation for limiting behavior when argument n goes towards infinity
$x \in A$	x is element of the set A
$x \notin A$	x is not element of the set A
$A \subset B$	set A is strict subset of the set B
$A \subseteq B$	set A is subset of set B
$A \cup B$	union of set A and set B: $x \in A \cup B \Leftrightarrow x \in A \vee x \in B$
$A \cap B$	intersection of set A and set B: $x \in A \cap B \Leftrightarrow x \in A \wedge x \in B$
$A \setminus B$	set difference: $x \in A \setminus B \Leftrightarrow x \in A \wedge x \notin B$
$x \sim P_{\mathcal{X}}$	x has probability density $P_{\mathcal{X}}$
$x \sim P_{\mathcal{X}}(\cdot\|y)$	x has conditional probability density $P_{\mathcal{X}}$ w.r.t. y
$\mathbb{E}_{x\sim P_{\mathcal{X}}}\big[f(x)\big]$, $\mathbb{E}\big[f(x)\big\|x \sim P_{\mathcal{X}}\big]$	expectation of $f(x)$ under random variable x with probability density $P_{\mathcal{X}}$
$\mathcal{N}(\mu, \Sigma)$	normal distribution with mean μ and variance Σ
min	minimize
s.t.	subject to
arg	argument of
$a \implies b$	proposition a implies proposition b
$a \impliedby b$	proposition a is implied by proposition b
$a \Leftrightarrow b$	$a \implies b$ and $a \impliedby b$
$\neg a$	not a (proposition)
$a \wedge b$	a and b (proposition)
$a \vee b$	a or b (proposition)
$[x > y]$	proposition "x is bigger than y"
$\mathcal{F}(x)$	Frenet transformation of Cartesian states x
$\mathcal{F}^{-1}(y)$	inverse Frenet transformation of Frenet states y

Symbols	
$x \in \mathbb{R}^{n_x}$	state vector with dimension n_x
$u \in \mathbb{R}^{n_u}$	control vector with dimension n_u
$w \in \mathbb{R}^{n_w}$	noise vector with dimension n_w
$\mathcal{X} \subseteq \mathbb{R}^{n_x}$	state space
$\mathcal{U} \subseteq \mathbb{R}^{n_u}$	control space
$\mathbb{X}$	state constraints set $\mathbb{X} \subseteq \mathcal{X}$
$\mathbb{X}^{\mathrm{t}}$	terminal state constraints set $\mathbb{X}^{\mathrm{t}} \subseteq \mathbb{X}$
$\mathbb{U}$	control constraints set $\mathbb{U} \subseteq \mathcal{U}$
$f(x,u)$	continuous-time differential equation right-hand side, deterministic controlled system model $\dot{x} = f(x,u)$
$F(x,u)$	discrete-time integration function right-hand side, deterministic system model $x_{k+1} = F(x_k, u_k)$
$F(x,u,w)$	discrete-time integration function right-hand side, stochastic system model $x_{k+1} = F(x_k, u_k, w_k)$, $w \sim P_{\mathcal{W}}(\cdot \mid x_k, u_k)$
$l(x,u)$	running or stage cost
t_{Δ}	sampling or discretization time
N	horizon length
γ	discount factor
$\pi(x)$	policy or control law as a function of state x
$\pi^{\mathrm{sv}}(x)$	policy of surrounding vehicle (SV)
Π	policy function space
$\pi^{\theta}(x),\ \pi(x;\theta)$	policy parameterized by θ
J^{mpc}	terminal value function approximation of model predictive control (MPC)
J^{π}	value function under policy π
J_{γ}	value function involving a discounted running cost
$J^{\pi^\star},\ J^\star$	optimal value function
$\hat{J}^{\pi}$	approximated value function under policy π
Q^{π}	action-value function under policy π
$Q^{\pi^\star},\ Q^\star$	optimal action-value function
σ	path length

$\gamma(\sigma)$	path function $\gamma : \mathbb{R}_{\geq 0} \to \mathbb{R}^2$ parameterized by path length $\sigma \in \mathbb{R}_{\geq 0}$
Γ	path $\Gamma := \{\gamma(\sigma) \mid \sigma \in \mathbb{R}_{\geq 0}\}$
$\mathcal{T}$	unit tangent vector
$\tilde{\mathcal{N}}$	unit normal vector
$\mathcal{N}$	signed unit normal vector
$\tilde{\kappa}$	curvature
κ	signed curvature
$p_x,\ p_y$	Cartesian position in x and y coordinate
φ	heading angle
s	projected longitudinal position on path
n	signed lateral distance to path
φ^γ	tangent angle of curve $\Gamma := \{\gamma(\sigma) \mid \sigma \in \mathbb{R}_{\geq 0}\}$
α	heading angle mismatch
v	velocity
δ	steering angle
m	vehicle mass
$l_\mathrm{f},\ l_\mathrm{r}$	vehicle wheelbase from center of gravity to front (f) or rear (r)
$\mathcal{O}$	set of all points that are occupied by an obstacle

Contents

Abstract v

Kurze Zusammenfassung vii

Abbreviations ix

Notation and Symbols xiii

Contents xvii

1 Introduction 1
- 1.1 Motion Planning and Control in Autonomous Driving 3
- 1.2 Contributions and Outline 9

2 Optimal Control 19
- 2.1 Mathematical Background 20
  - 2.1.1 Optimization Problem Classes 20
  - 2.1.2 Continuous Optimization 23
  - 2.1.3 Mixed-Integer Programming 27
- 2.2 Optimal Control Problem 33
- 2.3 Optimal Control Algorithms 35
  - 2.3.1 Environment Models 36
  - 2.3.2 Implicit and Explicit Policies 39
  - 2.3.3 Derivative-Based Online Optimization 41
  - 2.3.4 Sampling-Based Online Optimization 45
  - 2.3.5 Imitation Learning 46
  - 2.3.6 Reinforcement Learning and Dynamic Programming . . 48

3 Vehicle Models for Motion Planning 57
- 3.1 Double-Track Model 58
- 3.2 Dynamic Single-Track Model 59
- 3.3 Kinematic Single-Track Model 62
- 3.4 Point-Mass Model 62

3.5 Tires and Longitudinal Dynamics 64
    3.5.1 Tire Models 64
    3.5.2 Longitudinal Forces 65
3.6 Comparison of Models 66
3.7 Coordinate Frames for Motion Planning 67
    3.7.1 Projection of Configuration States 71
    3.7.2 Contouring Control Formulation 75
    3.7.3 Projected Formulation 76
3.8 Collision Avoidance 80
    3.8.1 Obstacle Shapes and Deterministic Formulation 80
    3.8.2 Stochastic Collision Avoidance 84
    3.8.3 Game-Theoretic Collision Avoidance 85

4 Software and Hardware Environments 87
4.1 Autonomous Racing Graz Stack 87
4.2 CommonRoad Interactive Simulation Environment 90
4.3 Custom Python-Based Environment Vehiclegym 91

5 Model Formulations for Optimization-Based Motion Planning 95
5.1 Parameterization Approach of the Frenet Transformation for MPC 96
    5.1.1 Introduction 96
    5.1.2 System Model 98
    5.1.3 Newton-Type Optimization 101
    5.1.4 Singularity and Smoothness Problem 102
    5.1.5 Optimal Curvilinear Parameterization 103
    5.1.6 Simulation Results 106
    5.1.7 Conclusions 109
5.2 Frenet-Cartesian Model Representations for NMPC 110
    5.2.1 Introduction 111
    5.2.2 Vehicle Models 113
    5.2.3 Obstacle Avoidance Formulations 119
    5.2.4 NMPC Formulation 121
    5.2.5 Numerical Experiments 124
    5.2.6 Conclusions 126
5.3 Critical Discussion 128

6 Mixed-Integer Optimization for Collision Avoidance 131
6.1 Mixed-Integer Optimization-Based Planning 133
    6.1.1 Introduction 134
    6.1.2 Vehicle and Object Models 136
    6.1.3 Combinatorial Optimization 139
    6.1.4 Trajectory Optimization 145
    6.1.5 Real-World and Simulation Results 147
    6.1.6 Conclusion 150
6.2 A Long-Short-Term Mixed-Integer Formulation 151

6.2.1 Introduction 152
6.2.2 Preliminaries and Notation 156
6.2.3 General Lane Changing Problem 157
6.2.4 Short-Horizon Approximations 163
6.2.5 Long-Horizon Approximations 167
6.2.6 Long-Short-Horizon Motion Planner 172
6.2.7 Evaluation 175
6.2.8 Conclusion and Discussion 184
6.3 Learning of Mixed-Integer Optimal Control Solutions 185
6.3.1 Introduction 186
6.3.2 Problem Setup and Formulation 190
6.3.3 Expert Motion Planner 192
6.3.4 Scalable Equivariant Deep Neural Network 197
6.3.5 Soft QP Solution and Selection Method 201
6.3.6 Feasibility Projection and SQP Algorithm 202
6.3.7 Implementation Details 205
6.3.8 In-Distribution Evaluations 206
6.3.9 Closed-loop Validations with SUMO Simulator 210
6.3.10 Conclusions and Discussion 213
6.3.11 Appendix 216
6.4 Critical Discussion 219

7 Collision Avoidance for Autonomous Racing 223
7.1 Inverse Optimal Control for Trajectory Prediction 224
7.1.1 Introduction 225
7.1.2 Prediction Architecture 228
7.1.3 Prediction Algorithm 228
7.1.4 Results 236
7.1.5 Conclusions 240
7.2 Hierarchical Approach for Strategic Motion Planning 243
7.2.1 Introduction 244
7.2.2 Background and Motivation 245
7.2.3 General Method 246
7.2.4 Parameterized Model Predictive Planner 248
7.2.5 Hierarchical Learning-based Predictive Planner 252
7.2.6 Simulated Experiments 256
7.2.7 Conclusions 259
7.3 Critical Discussion 260

8 Conclusion 265

Bibliography 271

Curriculum Vitae 301

List of Publications 303

Chapter 1

Introduction

According to a study from McKinsey published in 2023 [75], autonomous driving (AD) could create revenue of up to \$300 to \$400 billion in 2035. While in 2022, only partial driving automation dominated the market with a revenue of \$30 to \$40 billion, the study estimates that highly automated driving functions will dominate the market in 2035 with a revenue of \$170 to \$230. Moreover, the study claims that, besides some setbacks, the mobility community still agrees that AD could remarkably shape the future of transportation. Also, a quarter of all car buyers are willing to pay more than \$10.000 for premium AD features. Shared autonomous vehicles could potentially even reduce energy consumption by 53% to 61%, according to an urban sustainability report [322], despite difficulties measuring the environmental impact due to unknown business models of AD industries.

Self-driving vehicles operating in real-world conditions in the US caused eleven deaths within only four months in 2022 [66]. This underscores the importance of safety in autonomous driving. One primary requirement to achieve public acceptance for AD is preventing accidents, i.e., collision avoidance is a core interest. Collision avoidance requires several sub-tasks to be solved reliably. Objects need to be recognized by computer vision systems, and their behavior needs to be predicted intertwined with the own planning of decisions. On top of the recognition of objects, reliable motion planning algorithms in an environment with other agents are crucial.

Motion planning for autonomous vehicles involves determining feasible and optimal paths while considering dynamic constraints, such as vehicle kinematics and road conditions, as well as avoiding collisions or forcing emergency behavior of other agents. Effective collision avoidance requires accurate prediction of potential collision hazards and appropriate reactions to these hazards in a short response time. The complexity of this problem is significantly increased by the need to ensure real-time performance. Traditional methods, often based on rule-based systems or heuristic approaches, struggle to cope with the difficulties inherent in real-world scenarios. This gap shows the need for advanced techniques to provide safe and adaptive solutions.

Machine learning plays a crucial role in AD technologies at the current state-of-the-art. However, machine learning techniques also lack some crucial properties related to safe motion planning [66]. Remarkably, the review [66] mentions the stochastic behavior of the ego decision-making system, limited interpretability, and intricate debugging. This thesis focuses on deterministic optimization-based algorithms that are possibly supported by machine learning. Optimization-based motion planning has emerged as a promising approach due to its ability to handle constraints and objectives systematically. By formulating motion planning as an optimization problem, we can leverage mathematical optimization techniques to find solutions that not only satisfy all constraints but also optimize specific performance criteria, such as minimizing travel time and energy consumption or maximizing passenger comfort. Moreover, compared to pure machine learning, optimization-based algorithms are interpretable, easier to debug, and their results are deterministic and reproducible. This paradigm enables a more rigorous and flexible framework for developing motion planning algorithms. Along the lines of this paradigm, this thesis shows in several variants that a combination of machine learning and optimization-based algorithms can achieve superior overall performance on the specific scenarios that were chosen for evaluation.

The following chapters delve into various optimization techniques, including but not limited to linear, quadratic, and nonlinear programming, inverse optimal control, mixed-integer programming, and optimization methods supported by machine learning predictions. They are all required to address the challenges of motion planning. The algorithms were tested on various platforms, including interactive traffic simulations and real-world autonomous racing events. Autonomous racing competitions effectively contributed to advancing AD planning and control technologies. Citing the authors in [41]: "*What aerospace engineering is to aviation, motorsport is to automotive technology*". Possibly, some parallels can be drawn between autonomous racing competitions and autonomous driving. By advancing the field of optimization-based motion planning and control, this research aims to contribute to the broader goal of realizing safe and efficient autonomous transportation systems. The insights and methodologies developed in this thesis aim to bridge the gap between theoretical advances and practical applications, adding a puzzle piece to the next generation of autonomous vehicles.

1.1 Motion Planning and Control in Autonomous Driving

Motion planning for autonomous vehicles considers the task of using vehicle and environment information to compute vehicle control signals that maintain safety and carry out a desired high-level plan. In autonomous racing, this could be finishing the race fastest or, in conventional autonomous vehicles, driving to a desired location on a map. This field of research has attracted a great deal of attention in industry and academia and requires the combined efforts of many research areas. However, the challenges remain immense, and some open problems persist. Different communities have emerged that favor specific technologies and algorithms. As machine learning significantly impacts certain technological fields, a particular stream of work aims to solve huge, or even all, parts of the autonomous driving stack by machine learning techniques. Replacing most parts of the autonomous driving stack by machine learning is called end-to-end learning [278] and suffers from the disadvantages of lacking interpretability, missing reproducibility, missing adaptability, and difficult debugging [66, 67]. End-to-end learning is not part of this thesis.

A common approach to motion planning and control in autonomous driving is a hierarchy of modules with specific responsibilities. Each module has specific inputs, outputs, and usually different sampling frequencies. The communication between the modules may be organized by a middleware like `ROS` [174]. The set of modules that work together to support the execution of the autonomous driving control is the software stack. The whole software stack can be separated into *sensing*, i.e., components for sensing the environment such as computer vision related modules or state estimation, *planning*, i.e., components that plan the ego motion according to the perceived environment, and *acting*, which refers to low-level actuation. Many approaches and open-source autonomous driving software stacks propose a similar structure, e.g., [27, 67, 138, 215, 176, 183, 199, 247, 256, 329]. In this thesis, given inputs from the *sensing* module are assumed. The focus is on the *planning* and *acting* modules, as shown in the overview in Fig. 1.1. The controlled system is also referred to as “system”, “plant” or “environment”.

The requirements on the planning stack vary based on its application. Typical applications are driving on well-marked structured highways, urban driving in densely populated areas, rural roads without markings, autonomous racing events, and unstructured environments such as parking lots [41, 165, 197]. The different scenarios differ in the safety requirements, the available data, and the performance requirements that could be measured in terms of general driving costs. Highway driving usually has lower requirements on the perception system due to well-marked roads and precise localization, such as in [108]. Challenges are relatively high velocities, increased controller requirements, and the navigation on multiple lanes and merging lanes [213]. Urban driving is

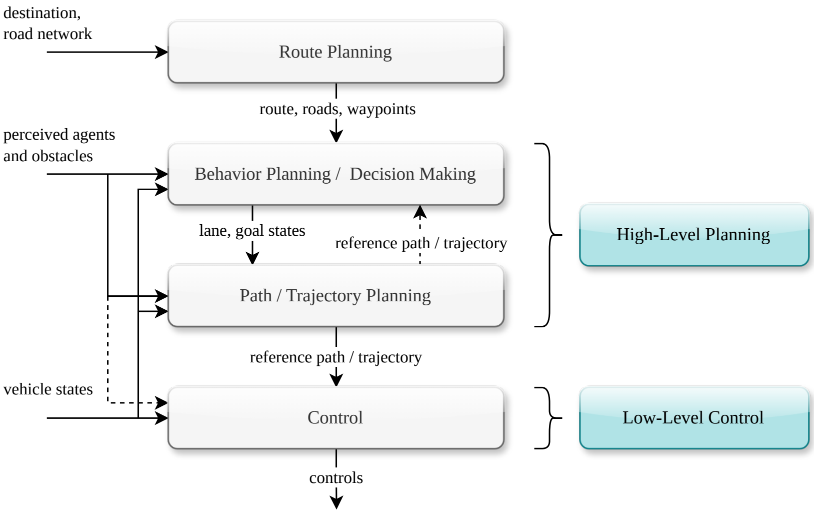

Figure 1.1: Modules of the autonomous driving stack. The blue blocks are considered within this thesis. The block of “high-level planning” is often used interchangeably with “motion planning” [67, 256].

usually less complex for vehicle control but poses considerable difficulties to the perception system and partly to the decision maker due to the abundance of different traffic participants and rare events. In urban driving, extremely precise and manually annotated maps are available [197]. Similarly, rural roads require a superior perception system and external positioning systems. Autonomous racing is a special case due to the complete absence of humans and rather arbitrary rules defined by an event organizer. The organizers may focus on competitive vehicle and obstacle interaction [233] or push the speed towards the vehicle limits [6]. Vehicles that operate in unstructured environments such as parking lots or on open land drive with lower speeds. The environments demand simpler control algorithms and has fewer requirements on the online planning time. The complexity may instead emerge from the potentially combinatorial obstacle avoidance and route-finding problem, interactions with other agents, or poor mapping. In unstructured environments, the concept of ordinary traffic rules and roads may not be applicable. This thesis focuses on algorithms for autonomous racing and highway driving.

The classical software stack for motion planning and control comprises a route planner, a behavior planner, also referred to as a decision maker, a path or trajectory planner, and a controller, see Fig. 1.1.

Route planning. At the highest level, the path through an environment is computed by the route planner. Typically, maps are available except for unstructured planning. These maps are usually given in a vast graph, requiring graph-search algorithms [25, 199]. A well-known fundamental algorithm for graph search is Dijkstra's algorithm [81], which has a high query time but low preprocessing time and a low memory requirement. In fact, the application to road networks was an essential driving force to improve Dijkstra's algorithm [25]. The exhaustive survey in [25] compares different algorithms with respect to their query time, memory footprint, and preprocessing time. Oppositely to Dijkstra's algorithm, an algorithm utilizing a lookup table [76] has a low query time but an enormous preprocessing time and memory footprint. A large number of algorithms trade off these properties in addition to further requirements such as the robustness to map or input changes.

Behavior planning and decision making. Once a route is computed, it is passed to a behavior planner, also referred to as a decision maker. This layer decides among options given a certain set of rules that apply to the current scenario. It aims to fulfill the task at hand with a low specified cost. In racing scenarios, this could be the decision of when to overtake another vehicle, when to start racing, or when to leave the track. In highway scenarios, this includes the decision to change lanes, and in urban traffic, this may be stopping in front of a red traffic light. The time horizon for the decision maker is up to ~ 1 min [67]. A simple implementation of such a layer in early papers is rule-based state machines, e.g., team "AnnieWay" [134] at the DARPA Urban Challenge [1].

However, state machines may ignore certain aspects of the planning problem. Most importantly, the behavior planner requires the perception of the current scene and a potential prediction of other agents. Predicting other agents is a core element of behavior planning and can, in general, not be performed separately, as one's own decisions influence the decisions of other traffic participants. A simple example of such an interaction would be a succeeding vehicle that will most likely brake if the preceding one brakes. When assuming constant policies, these interactions could be treated by stochastic, interactive vehicle models, such as in [224, 326]. A more general consideration of interactions among agents requires a game theoretic or, likewise, a multi-agent framework, e.g., [56, 57, 68, 160, 251, 252]. In dynamic games, agents can change their policies in order to maximize a combination of a selfish reward and a reward given to others [56]. Given the high computational burden of solving general dynamic games, the behavior planner may use a coarse approximation of the ego vehicle model, e.g., [168] for autonomous racing, and assume hierarchically lower levels to be able to track the proposed plans safely. In case a lower-level planner is unable to track a reference plan, some feedback needs to be given to the behavior planner, cf. Fig. 1.1.

Path and trajectory planning. Given a behavior specification, e.g., a goal lane and speed limits or a reference trajectory, a path/trajectory planner is used to provide a feasible path or trajectory (w.r.t. a certain vehicle model and constraints) to a vehicle controller. Note that path planning is commonly associated with geometric or kinematic planning, and trajectory planning involves the time information in addition to the position [199]. Path planners may also provide a reference velocity that implicitly defines a trajectory again when integrated. However, a consecutive low-level controller may react differently to a velocity error than to a position error. One could view the position trajectory tracking problem as a path and velocity tracking problem where the plant integrates velocity to position. Tracking the position with time information may be inevitable to avoid dynamic obstacles. Given the dependency of the behavior planner and the path/trajectory planner, some papers formulate both in a single formulation, e.g., in a mixed-integer optimization framework [213, 227]. This thesis refers to behavior and path/trajectory planning as “high-level planning”.

Remarkably, a separation between the behavior planning and path/trajectory planning layers can be motivated by separating the overall problem based on optimization problem classes. An outstanding division line in mathematical optimization is between combinatorial and continuous optimization. While polynomial-time algorithms can solve some combinatorial optimization problems, the problems occurring in autonomous driving are often NP-hard [159]. Combinatorial problem aspects include on which side to overtake an obstacle or choosing among multiple highway lanes. The combinatorial complexity, in addition to real-time requirements, motivated many publications to find trade-offs between optimality, safety, and computation time [67]. Approaches consider mixed-integer programming [181, 213], learning-based [51] or graph-search techniques [8, 241], among many others [67]. While simple interactive behavior can be included as part of a classical combinatorial optimization problem formulation, e.g., as in [227], considering general multi-agent settings requires additional game-theoretic concepts [70]. Combinatorial complexity may be hidden in the nonconvexity of optimization problem formulations. For example, obstacle avoidance can be formulated as a nonlinear, nonconvex optimization problem without combinatorial integer variables. However, solving such a problem assumes an initial guess “close” to a local or global optimum, requiring combinatorial optimization algorithms. Given an approximate solution of the combinatorial part, the trajectory or path planner can be initialized efficiently and often instead corrects a given coarse trajectory to be kinematically feasible and safe; see, for example, [229] for such an architecture. The high-dimensional state space of higher fidelity vehicle models challenges graph-search algorithms, and the trajectory or path planner may use numerical optimization-based algorithms [293].

Control. The longitudinal and the lateral dynamics of vehicles have different time constants. Therefore, the low-level controller may be split into a longitudinal and lateral control. However, this ignores coupling effects, particularly for highly dynamic maneuvers. In this thesis, the lowest level controller is assumed to output a steering angle or steering angle rate and an acceleration or braking torque. In fact, further low-level controllers are required to provide the actual actuator signals. However, these lowest-level controllers are usually not modifiable on many vehicle platforms, e.g., in the `Roborace` competition [233].

This thesis considers behavior planning, path/trajectory planning, and low-level control. Due to the intertwined formulations of the behavior and path or trajectory planner, those are referred to as “high-level planners”. Accordingly, the controller is referred to as a “low-level controller”, cf., Fig. 1.1. Both problems are formulated as optimal control problems (OCPs) that approximate the inherently stochastic real-world problem, cf., Chapter. 2. Treating stochasticity explicitly in numerical algorithms is challenging and not part of this thesis. Stochasticity is instead considered by closed-loop control with feedback on the current environment state and consecutively adapting the prediction of surrounding vehicles (SVs) and the ego-motion plan. The aim is to optimize an ego trajectory $x(t) \in \mathcal{X}$ in the state space $\mathcal{X} \subseteq \mathbb{R}^{n_x}$ by modifying controls $u(t) \in \mathcal{U}$ in the control space $\mathcal{U} \subseteq \mathbb{R}^{n_u}$ depending on the time $t \in \mathbb{R}$. The start of the trajectory is equal to an estimated state $x_0 \in \mathbb{R}^{n_x}$ at the current time $t_0 \in \mathbb{R}$. The trajectory of the deterministic vehicle dynamics is constrained to the infinite-dimensional manifold described by the ordinary differential equation $\dot{x}(t) = f(x(t), u(t))$. Additionally, the states are constrained by physical limitations using the set $\mathbb{X} \subseteq \mathcal{X}$, and the controls are limited by $u(t) \in \mathbb{U} \subseteq \mathcal{U}$. The states are further constrained by a simplified obstacle-free space $\mathbb{X}_{\text{free}}(t)$, which may be more sophisticated, than stated in this deterministic non-interactive form. For example, it can depend on the previously driven ego trajectory $\{x(\tau) \mid 0 \leq \tau \leq t\}$ or involve uncertainty about SV predictions. A discussion about collision avoidance constraints is given in Sect. 3.8. The OCP includes a cost $J\big(x(\cdot), u(\cdot)\big)$ for the planned trajectories of controls and states and can be written in a general continuous-time form as

$$\min_{x(\cdot), u(\cdot)} J\big(x(\cdot), u(\cdot)\big) \quad \text{s.t.} \begin{cases} x(t_0) = x_0, \\ \dot{x} = f(x(t), u(t)), \quad t \in [t_0, \infty), \\ x(t) \in \mathbb{X}_{\text{free}}(t) \cap \mathbb{X}(t), \quad t \in [t_0, \infty), \\ u(t) \in \mathbb{U}(t), \quad t \in [t_0, \infty). \end{cases} \tag{1.1}$$

The overall paradigm of this thesis pivots on solving a simplified form of the

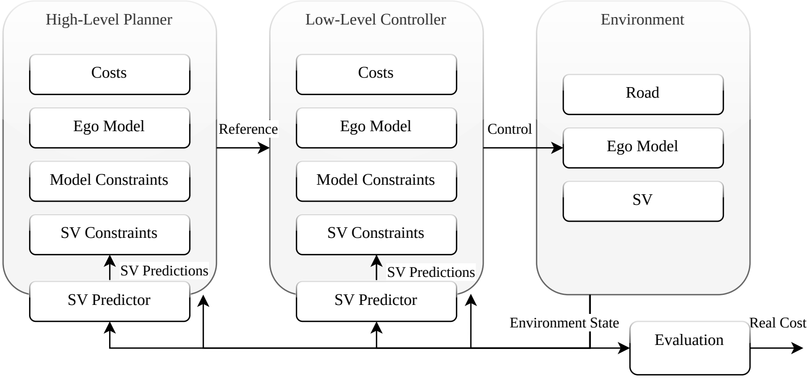

Figure 1.2: A generic architecture of optimization-based planners and controllers. This architecture is used as a template to categorize each individual contribution. In this thesis, the descriptions environment, system, and plant are used interchangeably.

OCP (1.1) as part of the high-level planner and, possibly, the low-level controller repeatedly. The time it takes to solve the optimization problem online is crucial to the overall performance. Therefore, a central goal of the proposed novel techniques is lowering the online computation time to integrate more detail into the approximation of OCP (1.1), thus increasing the performance. For the high-level planner, slightly longer computation times are accepted than for the lower-level controller.

Generic Optimization-Based Architecture. At the beginning of Sect. 1.1, the main scope of this thesis was introduced. The focus of the autonomous driving stack, as shown in Fig. 1.1, was set on high-level planning and low-level control. From the viewpoint of optimization-based algorithms, those two layers can be drawn in more detail, as shown in Fig. 1.2. This sketch illustrates the necessary modules and dependencies of the algorithms. Moreover, this sketch is used to point out the contributions proposed in this thesis. Each contribution focuses on different subsets of these modules. In order to evaluate the proposed algorithms, additional modules were part of an integrated software stack.

The optimization-based planner and controller are similarly structured, cf., Fig. 1.2. Both contain a particular type of objective, a vehicle model, model constraints, collision constraints to avoid SVs, and a prediction of these SVs. In a simplified case, the prediction of SVs may be performed by an external module. However, the separation of ego planning and prediction does not allow for interactive planning, i.e., respecting the influence of the planned ego trajectory on the prediction of the SV. It may also be integrated into the optimization problem and, therefore, be part of the planner or controller module. The high-level planner passes a reference trajectory to the low-level controller and is usually executed at a frequency of 1 to 10 Hz. The objective of the high-level planner is usually to minimize an economic cost. The low-level controller is executed at 10 to 100 Hz and outputs the control signals. The controller's objective is usually to track the reference. The environment could either be a high-fidelity simulation of the ego vehicle or an embedded hardware with possibly simulated SVs. The performance of the closed-loop system is evaluated in various randomized test scenarios.

1.2 Contributions and Outline

This thesis presents its main contributions in the form of nearly unchanged peer-reviewed publications in Chapters 5 to 7 after a detailed introduction of relevant mathematical and technological concepts, cf., Fig. 1.3.

Within the introduction Chapters 2 to 4, basic concepts are explained that appear as a common theme throughout this work. Mathematical background is given in Chapter 2. Chapter 3 introduces the main concepts for automotive vehicle modeling for optimization-based motion planning and control. Important low to medium-fidelity vehicle models are introduced in Sect. 3.1 to Sect. 3.6. These vehicle models are used either in simulation, as part of an optimization-based planner, or as an optimization-based controller. Sect. 3.7 introduces the concept of a projection of the vehicle dynamics onto a reference path. Again, this projection is considered as part of an optimization problem. In Sect. 3.8, different aspects of collision avoidance within numerical optimization algorithms are reviewed. In Chapter 4, the last chapter of the introductory part, the different software and hardware stacks used for evaluating the presented contributions are summarized.

The following summarizes the contribution of each published paper. The components involved within each publication are shown with respect to the general motion planning and control architecture in Fig. 1.2. The blue boxes indicate in which modules the contributions were made, and the green boxes show the overall components involved in the related software and hardware stack. Sect. 5.1 and 5.2 are aiming to improve the model and the constraint formulations inside the optimization problem that is solved as part of model predictive control (MPC). Sect. 6.1 to 6.3 are related to the nonconvex combinatorial planning problems due to multiple obstacles. Sect. 7.1 and 7.2 consider the prediction and interaction with SVs in competitive racing scenarios.

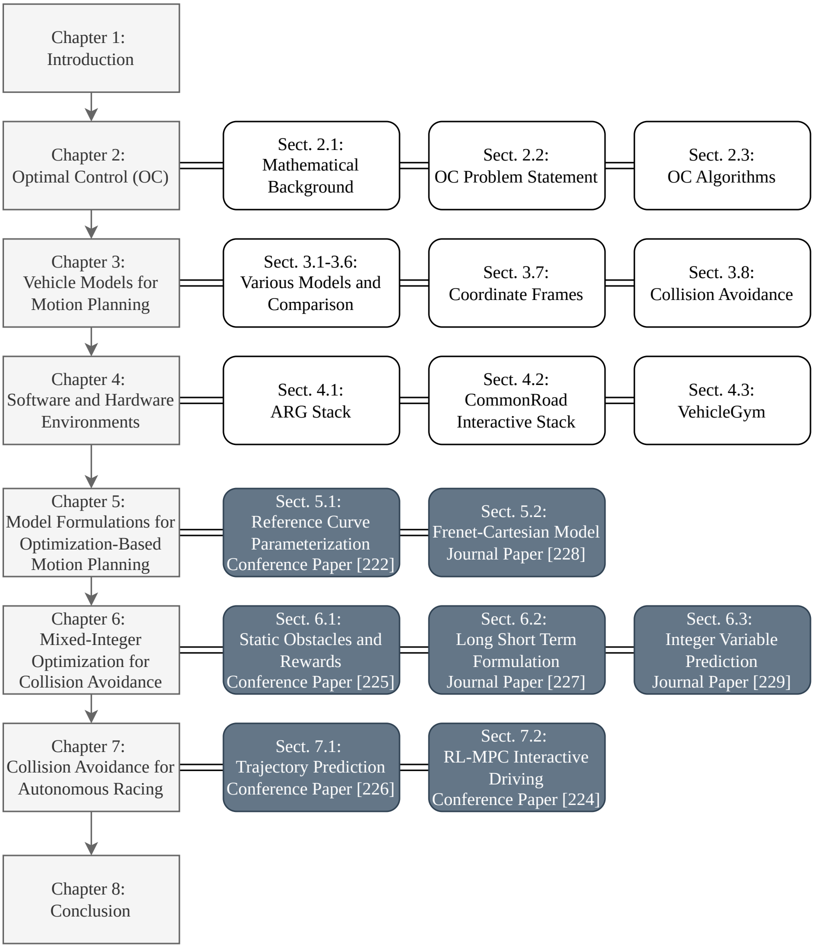

Figure 1.3: Overview of the chapters and sections of this thesis. Chapters 5 to 7 comprise the main contribution in the form of nearly unchanged publications. The original paper titles are abbreviated by a short description.

Section 5.1: Parameterization Approach of the Frenet Transformation for Model Predictive Control of Autonomous Vehicles. The contribution of this section related to the publication [222] pivots around an efficient numerical formulation of a vehicle model formulated in projected Frenet coordinates as part of an optimization-based low-level controller. The proposed algorithms solve a prior optimization problem in order to parameterize the projection curve in a numerically favorable way. The improved performance is evaluated on randomized tracks. Fig. 1.4 shows the involved components within the control and planning framework.

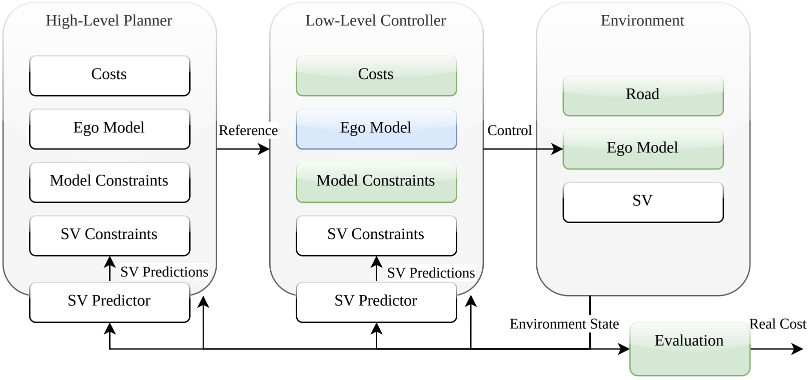

Figure 1.4: Contributions of Sect. 5.1: Parameterization Approach of the Frenet Transformation for Model Predictive Control of Autonomous Vehicles. In this work, no obstacles or SVs were simulated. The contributions are made within the blue modules. The green boxes were used as part of the overall framework in this specific setting.

Section 5.2: Frenet-Cartesian Model Representations for Automotive Obstacle Avoidance within Nonlinear Model Predictive Control. This section and the related publication [228] improve the MPC problem formulation. The vehicle model used within MPC is enhanced by expanding its state space to include multiple coordinate frames, applying constraints and costs in the most suitable frame. The results demonstrate improved overall performance in various deterministic obstacle avoidance simulations attributed to the representation of obstacle shapes in either frame. Remarkably, the computation time can be reduced compared to a Frenet coordinate frame (FCF) representation despite the increased state dimensions. Furthermore, references can be set in either coordinate frame, enabling flexible integration with planning modules utilizing a specific frame.

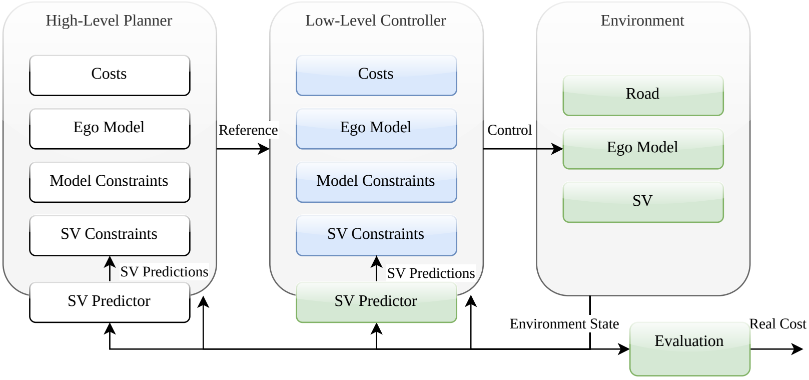

Figure 1.5: Modules related to Sect. 5.2: Frenet-Cartesian Model Representations for Automotive Obstacle Avoidance within Nonlinear MPC. Blue boxes indicate contributions; green boxes indicate involved components.

Section 6.1: Mixed-integer optimization-based planning for autonomous racing with obstacles and rewards. This section related to the publication [225] introduces a planning procedure for avoiding obstacles and collecting rewards through a combination of offline and online steps. Initially, an optimal racing line is computed based on the track geometry without obstacles, similar to the approach presented in Sect. 5.1. The online phase comprises two steps: first, a mixed-integer linear program (MILP) formulation selects a homotopy class to determine reward collection and obstacle avoidance, represented by deformed track boundaries. Second, a continuous optimization problem minimizes deviations from the racing line while considering these modified boundaries. Homotopy iterations are used to enhance the solver convergence. This novel motion planning approach effectively solves time-optimal motion planning problems with a combinatorial structure, avoiding the full state-space discretization required by graph-based algorithms.

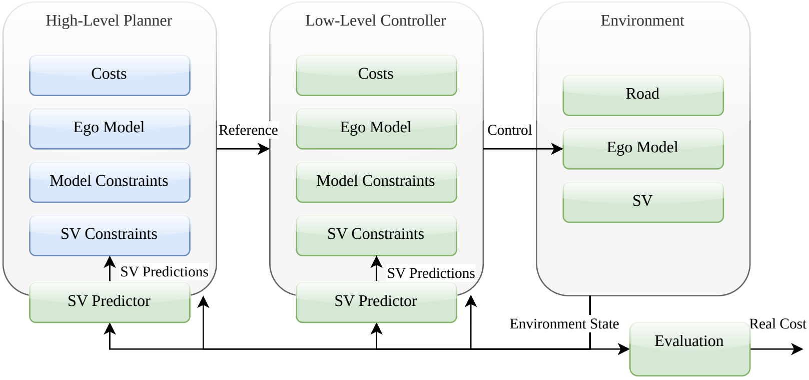

Figure 1.6: Modules related to Sect. 6.1: Mixed-integer optimization-based planning for autonomous racing with obstacles and rewards. Blue boxes indicate contributions; green boxes indicate involved components.

Section 6.2: A Long-Short-Term Mixed-Integer Formulation for Highway Lane Change Planning. This section related to [227] presents a novel algorithm for optimal lane-changing maneuvers on highways. Unlike other highway motion planning algorithms, the presented lane change motion planner effectively approximates long-term dependencies in the spatiotemporal domain with a computational burden that remains independent of the lane change position and timing. By addressing the short-term and long-term approximations as a unified problem, inconsistent decoupling is avoided. The method improves closed-loop performance by 15% compared to [213] and [8] and significantly reduces average computation time in randomized interactive simulations. Additionally, compared to [213], the algorithm reduces the number of integer variables in the optimization problem.

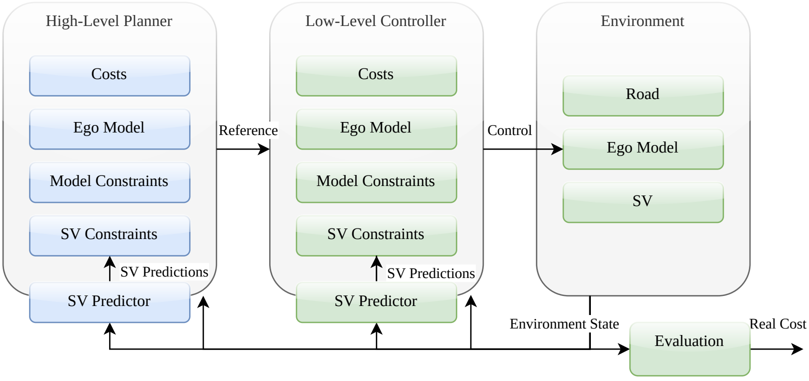

Figure 1.7: Modules related to Sect. 6.2: A Long-Short-Term Mixed-Integer Formulation for Highway Lane Change Planning. Blue boxes indicate contributions; green boxes indicate involved components.

Section 6.3: Equivariant Deep Learning of Mixed-Integer Optimal Control Solutions for Vehicle Decision Making and Motion Planning. The main contribution of this Chapter, which is related to [229], is a recurrent equivariant deep set architecture for predicting integer variables in mixed-integer quadratic programs, optimized for time series and obstacle-related binary variables in motion planning. This architecture ensures consistent collision avoidance predictions regardless of the obstacle order and supports a variable number of obstacles. The framework combines an ensemble of neural networks with a feasibility projector to enhance safe trajectory computation. Compared to state-of-the-art methods, it improves prediction accuracy, introduces permutation equivariance, and generalizes unseen data. A novel integrated planning system is presented for real-time vehicle decision-making and motion planning and validated in closed-loop simulations with interactive agents.

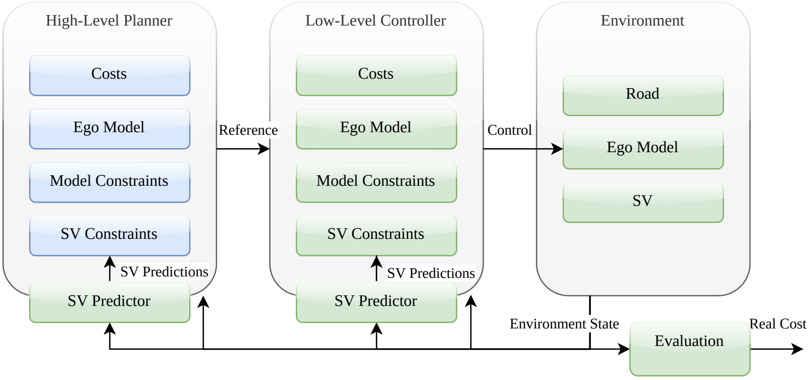

Figure 1.8: Modules related to Sect. 6.3: Equivariant Deep Learning of Mixed-Integer Optimal Control Solutions for Vehicle Decision Making and Motion Planning. Blue boxes indicate contributions; green boxes indicate involved components.

Section 7.1: An Inverse Optimal Control Approach for Trajectory Prediction of Autonomous Race Cars. In the domain of autonomous racing, this section and the related paper [226] are, to the best of the author's knowledge, among the first to employ bi-level optimization for trajectory prediction. Given the difficulty of solving bi-level problems, this work utilized the relaxation of complementarity constraints to find a stationary point of the lower-level problem. The algorithm exhibits a stronger initial performance without prior training, compared to other learning-based algorithms, and adapts quickly online to observed trajectories of other drivers.

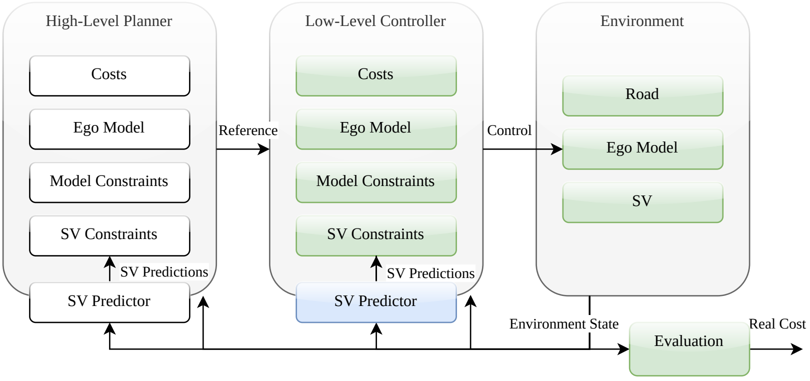

Figure 1.9: Modules related to Sect. 7.1: An Inverse Optimal Control Approach for Trajectory Prediction of Autonomous Race Cars. Blue boxes indicate contributions; green boxes indicate involved components.

Section 7.2: A Hierarchical Approach for Strategic Motion Planning in Autonomous Racing. The contribution of Sect. 7.2 and the related publication [224] involves developing and assessing an efficient and safe motion planning algorithm tailored for interactive behavior occurring in autonomous racing. The presented algorithm features a novel cost function formulation that integrates MPC and reinforcement learning (RL), providing strong prior performance, real-time feasibility, and clear interpretability.

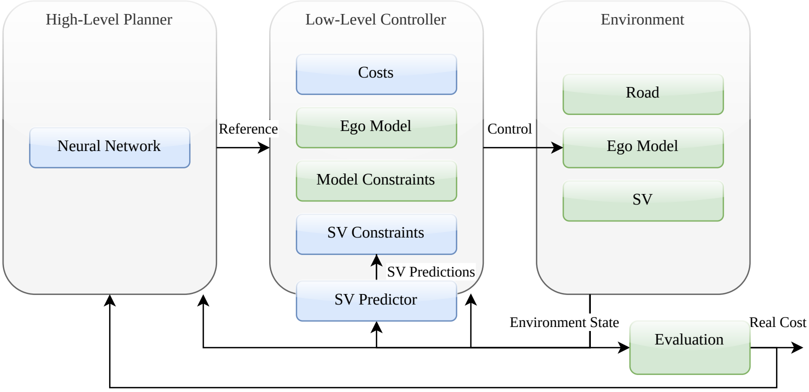

Figure 1.10: Modules related to Sect. 7.2: A Hierarchical Approach for Strategic Motion Planning in Autonomous Racing. Blue boxes indicate contributions; green boxes indicate involved components.

Chapter 2

Optimal Control

This chapter introduces the mathematical framework and the relevant numerical algorithms that serve as a basis for contributions within this thesis to solve motion planning and control problems as formulated in (1.1). First, some basic concepts of numerical optimization are repeated in Sect. 2.1. The next Sect. 2.2 introduces optimal control problems (OCPs) as means to formalize the goal of motion planning and control algorithms for autonomous driving. Finally, Sect. 2.3 introduces algorithms for solving OCPs in a closed-loop environment. The OCPs are often approximated and solved online in a feedback control system. A crucial approximation involves finding a reasonable model for the real-world environment.

Algorithms for solving the OCP online are separated into derivative-based online optimization, cf., Sect. 2.3.3, or sampling-based online optimization, cf., Sect. 2.3.4. The potentially computationally expensive online optimization may be circumvented by utilizing an offline optimization algorithm to solve a large number of problems with the distribution of parameters encountered in the real-world setting. After storing the samples of the solution map from the OCP parameters to the optimal decision variables, this mapping function can be learned, referred to as imitation learning (IL).

The rather different approach of reinforcement learning (RL) is capable of avoiding a model of the environment. It focuses on learning a policy that achieves optimal closed-loop behavior by interacting with the environment and evaluating obtained rewards after applying specific actions in particular states. Environment models may still be used in an initial learning phase to generate interactions more efficiently. Relevant concepts of IL and RL, including dynamic programming (DP), are provided in Sect. 2.3.5 and Sect. 2.3.6, respectively.

2.1 Mathematical Background

The following section introduces essential concepts of continuous and mixed-integer optimization concepts. Sect. 2.1.1 states the different classes of optimization problems. In Sect. 2.1.2, algorithms for solving the class of nonlinear programs (NLPs) are sketched on a high level along the lines of [217], and in Sect. 2.1.3 algorithms for solving mixed-integer nonlinear programmings (MINLPs) are surveyed based on [29] and [244].

2.1.1 Optimization Problem Classes

The following repeats the definition of relevant complexity classes and, thereafter, defines optimization problem classes most relevant for the developed algorithms.

Computational Complexity. Computational complexity describes how difficult it is to solve problem instances of a specific class when scaling the problem dimensions. In the following, the main concepts are illustrated along the lines of [159]. Typically, the big-O notation is used to upper-bound the computation time depending on the problem size in the limit [190]. As a further refinement, lower and upper bounds on the computational complexity are used. An upper bound can be found, for example, by any algorithm that solves the problem and whose computational complexity is known. A lower bound can be found by intricate mathematical analysis.

Optimization problem classes can be related to complexity classes which provides an insight of how difficult the problems are to solve in general. Relevant complexity classes for this thesis are explained in the following. For problems belonging to the polynomial complexity class $\mathcal{P}$, algorithms can be found that solve the problem in polynomial time, i.e., the solution time scales with $\mathcal{O}(n^k)$, where $k \in \mathbb{N}$ and n is the problem size. For problems in the complexity class $\mathcal{NP}$, algorithms can be found that verify in polynomial time if any given solution solves the problem. An *efficient* algorithm solves a problem in polynomial time. For an *intractable* problem, no efficient algorithm can be found. Problems in $\mathcal{P}$SPACE= $\mathcal{NP}$SPACE can be solved with a polynomial amount of storage space. Finally, the class of EXPTIME is used for problems that can be solved in $\mathcal{O}\left(2^{n^k}\right)$, with $k \in \mathbb{N}$. Notably, boundaries of the problem classes $\mathcal{P}$SPACE and $\mathcal{NP}$ are not yet found. It is known that $\mathcal{P} \subset$ EXPTIME. However, some mathematicians only assume that $\mathcal{P} \subset \mathcal{NP} \subset \mathcal{P}$SPACE $\subset$ EXPTIME. It could also be true that $\mathcal{P} = \mathcal{NP} = \mathcal{P}$SPACE or $\mathcal{NP} = \mathcal{P}$SPACE $=$ EXPTIME. Therefore, for a problem in $\mathcal{NP}$, finding a polynomial time algorithm may still be possible.

In the following, the concepts of “hardness” and “completeness” are repeated. Let X be one of the problem classes $\mathcal{P}, \mathcal{NP}, \mathcal{P}$SPACE or EXPTIME. If problem A is in the class X-hard, any other problem in X can be reduced to problem A in polynomial time. Remarkably, a problem that is X-hard is not required to be within the class X. However, if the problem is in class X and X-hard, it is named X-complete.

In complexity theory,“intractable” refers to problems that are extremely difficult or impossible to solve efficiently, usually because they require an impractical amount of computational time or space resources as the size of the input grows. Due to the unknown bound of $\mathcal{NP}$ and $\mathcal{P}$SPACE, it cannot be argued that a problem in $\mathcal{NP}$ is intractable. Nonetheless, problems in EXPTIME are intractable. Lower bounds of algorithms are particularly useful, because they provide insight of how much an existing algorithm could be improved.

Optimization problem classes are used to define the input format of optimization solvers since the underlying algorithms are specifically designed for certain structures. The optimization classes allow problem formulations that exhibit a particular computational complexity, which provides a complexity lower bound for any algorithms or solvers that are used to solve these problems. The most important optimization problem classes are as follows.

Linear Programs. Linear programs (LPs) are defined as

$$\min_{z \in \mathbb{R}^{n_z}} \; c^\top z \quad \text{s.t.} \quad \begin{cases} Az - b = 0, \\ Cz - d \leq 0, \end{cases} \tag{2.1}$$

where $c \in \mathbb{R}^{n_z}$, $A \in \mathbb{R}^{n_g \times n_z}$, $C \in \mathbb{R}^{n_h \times n_z}$, $b \in \mathbb{R}^{n_g}$ and $d \in \mathbb{R}^{n_h}$. The number of equality constraints are n_g and the number of inequality constraints are n_h. LPs can be solved efficiently in finite time and at most exponential in the number of variables by, for example, the Simplex algorithm [72] or even in polynomial time by the interior point (IP) algorithms [15] or by the ellipsoid method [143, 24]. Therefore, LPs are in the problem class $\mathcal{P}$.

Quadratic Programs. Quadratic programs (QPs) are defined by

$$\min_{z \in \mathbb{R}^{n_z}} \; c^\top z + \frac{1}{2} z^\top B z \quad \text{s.t.} \quad \begin{cases} Az - b = 0, \\ Cz - d \leq 0, \end{cases} \tag{2.2}$$

where $B \in \mathbb{R}^{n_z \times n_z}$. QPs are further classified into convex QPs, where the matrix B is positive semi-definite, or even strictly convex QPs, where B is positive definite. Convex QPs can be solved efficiently in finite polynomial time by, e.g., the IP algorithm [154]. Nonconvex QPs are hard to solve, and in the problem class $\mathcal{NP}$-hard [200]. Established solvers for usually convex

QPs are, for instance, the commercial solvers of `MOSEK` [184] and `Gurobi` [114] or the open-source solvers `OSQP` [268], `qpOASES` [94] and `QPALM` [121]. Some solvers are explicitly designed to exploit the structure of the model predictive control (MPC) optimization problem, such as the open-source solver `HPIPM` [97].

Convex Programs. A convex optimization program is written as

$$\min_{z\in\mathbb{R}^{n_z}} f_{\mathrm{c}}(z) \quad \text{s.t.} \quad \begin{cases} Az-b=0, \\ h_{\mathrm{c},i}(z)\le 0, \quad i=1,\dots,n_h, \end{cases} \tag{2.3}$$

where $f_{\mathrm{c}}:\mathbb{R}^{n_z}\to\mathbb{R}$ and $h_{\mathrm{c},i}:\mathbb{R}^{n_z}\to\mathbb{R}$ for $i=1,\dots,n_h$, are convex functions, such as defined in [46]. Many convex problems can be solved efficiently in polynomial time by solvers such as `ECOS` [82] and tools for formulating convex problems such as `CVX` [106] and `CVXPY` [79]. However, it is not trivial to check whether a problem is convex. Moreover, convex problems exist that are $\mathcal{NP}$-hard [73]. Within this thesis, the functions f_{c} and $h_{\mathrm{c},i}$, for $i=1,\dots,n_h$ are assumed to be continuously differentiable. In general, convex problems may involve nonsmooth functions and may still be solved efficiently [46].

Nonlinear Programs. An NLP is written in the general form

$$\min_{z\in\mathbb{R}^{n_z}} f_{\mathrm{nlp}}(z) \quad \text{s.t.} \quad \begin{cases} g_{\mathrm{nlp},i}(z)=0, \quad i=1,\dots,n_g, \\ h_{\mathrm{nlp},i}(z)\le 0, \quad i=1,\dots,n_h, \end{cases} \tag{2.4}$$

where $f_{\mathrm{nlp}}:\mathbb{R}^{n_z}\to\mathbb{R}$, $h_{\mathrm{nlp},i}:\mathbb{R}^{n_z}\to\mathbb{R}$ for $i=1,\dots,n_h$ and $g_{\mathrm{nlp},i}:\mathbb{R}^{n_z}\to\mathbb{R}$ for $i=1,\dots,n_g$ are continuously differentiable functions. NLPs are generally nonconvex and in the problem class $\mathcal{NP}$-hard [186] and, therefore, difficult to solve. However, many problem instances can be solved efficiently to local optimality in practice by exploiting derivatives with general-purpose solvers such as `IPOPT` [307]. Solvers that exploit the MPC problem structure are available as open-source software, e.g., `acados` [291], or commercially, e.g., `FORCESPro` [83].

Mixed-Integer Programs. Mixed-integer programs (MIPs) are optimization problems that contain integer variables $y\in\mathbb{Z}^{n_y}$ and continuous variables $z\in\mathbb{R}^{n_z}$. Noteworthy, MIPs can formally be reformulated into NLPs. However, the structure given by the integer variables can be exploited by solvers like `Gurobi` [114] or `MOSEK` [184] to usually achieve a much higher performance than, for example, using a general-purpose NLP solver. An MIP problem can be written as

$$\min_{z\in\mathbb{R}^{n_z},y\in\mathbb{Z}^{n_y}} f_{\mathrm{mi}}(z,y) \quad \text{s.t.} \quad \begin{cases} g_{\mathrm{mi},i}(z,y)=0, \quad i=1,\dots,n_g, \\ h_{\mathrm{mi},i}(z,y)\le 0, \quad i=1,\dots,n_h, \\ [z^\top,y^\top]\in\mathbb{W}, \end{cases} \tag{2.5}$$

with $f_{\mathrm{mi}} : \mathbb{R}^{n_z} \times \mathbb{Z}^{n_y} \to \mathbb{R}$, $h_{\mathrm{mi},i} : \mathbb{R}^{n_z} \times \mathbb{Z}^{n_y} \to \mathbb{R}$ for $i = 1, \dots, n_h$ and $g_{\mathrm{mi},i} : \mathbb{R}^{n_z} \times \mathbb{Z}^{n_y} \to \mathbb{R}$ for $i = 1, \dots, n_g$. All variables are in a convex bounded polyhedral set $\mathbb{W} \subset \mathbb{R}^{n_y + n_z}$. A further refinement of MIPs may specify the structure of the continuous functions of problem (2.5) related to the problem classes above. For example, if the functions f_{mi}, $g_{\mathrm{mi},i}$ for $i = 1, \dots, n_g$ and $h_{\mathrm{mi},i}$ for $i = 1, \dots, n_h$ are continuously differentiable, the problem is referred to as MINLP. Similarly, mixed-integer linear programs (MILPs), mixed-integer quadratic programs (MIQPs), or mixed-integer convex programs (MICPs) can be defined. All MIP problems are hard to solve. Even MILPs or pure integer programs are $\mathcal{NP}$-hard [99]. Nonetheless, some MILPs with a certain structure, such as integer knapsack problems, can in practice be solved faster, their worst-case complexity is still $\mathcal{NP}$-hard [190]. MINLPs with unbounded decision variables are even UNDECIDABLE [130], i.e., simplified, a computer that follows step-by-step instructions cannot solve these problems in finite time, cf. [261] for more details.

Remark 2.1.1. *The term* combinatorial optimization *is not consistently defined in the literature. The authors in [191] state the following: "While there is no generally agreed-upon definition of a combinatorial optimization problem, most problems so named are 0-1 IPs that deal with finite sets and collections of subsets." The class of "0-1 IPs" are pure integer problems without a continuous part and where the integer variables can only take values of zero or one. Pure integer problems are not considered within this thesis. The term "combinatorial" is instead used to highlight nonconvexities that are modeled by integer variables in the optimization problem and desired to be considered in the optimization algorithm.*

2.1.2 Continuous Optimization

In the following, Newton's method, sequential quadratic programming (SQP) and the interior point (IP) methods are described on a high level. Both SQP, IP methods, and mixed-integer programming are extensively used throughout this thesis to solve OCPs.

Newton's Method for Unconstrained Optimization

As a basis for further considerations of constrained optimization, first, Newton's method is explained briefly for unconstrained convex problems

$$\min_{z \in \mathbb{R}^{n_z}} f_{\mathrm{nlp}}(z), \tag{2.6}$$

where $f_{\text{nlp}} : \mathbb{R}^{n_z} \to \mathbb{R}$ is twice differentiable and convex. Consider the Taylor approximation

$$\begin{aligned} f_{\text{nlp}}(z) \approx{} & \tilde{f}_{\text{nlp}}(z; z_0) = \\ & f_{\text{nlp}}(z_0) + \nabla_z f_{\text{nlp}}(z_0)^\top (z - z_0) + \frac{1}{2}(z - z_0)^\top \nabla_z^2 f_{\text{nlp}}(z_0)(z - z_0) \end{aligned}$$

around an initial guess z_0. For convex problems, a first-order necessary condition (FONC) and sufficient condition for optimality is $\nabla \tilde{f}_{\text{nlp}}(z^\star; z_0) = 0$. Applying the optimality condition to the Taylor series and given that the Hessian $\nabla_z^2 f_{\text{nlp}}(z_0)$ is invertible, the optimizer for the approximated problem is

$$z_0^\star = z_0 - \left(\nabla_z^2 f_{\text{nlp}}(z_0)\right)^{-1} \nabla_z f_{\text{nlp}}(z_0),$$

which is set as the new initial guess $z_1 := z_0^\star$. This procedure is repeated until convergence. In fact, the convergence of Newton's method is extremely fast once the initial guess is close enough to the optimizer of the original problem [46]. For convex problems, the FONC $\nabla \tilde{f}_{\text{nlp}}(z^\star; z_0) = 0$ is also a sufficient condition for a global minimizer. For nonconvex problems, the second-order sufficient condition (SOSC) $\nabla^2 \tilde{f}_{\text{nlp}}(z^\star; z_0) \succ 0$ may verify a local minimizer $z^\star$ [194].

Constrained Optimization

Adding constraints to the optimization problem (2.6) requires additional concepts. Since equality constraints can also be formulated as inequality constraints, the following NLP omits equality constraints and reads as

$$\min_{z \in \mathbb{R}^{n_z}} f_{\text{nlp}}(z) \quad \text{s.t.} \quad h_{\text{nlp},i}(z) \leq 0, \quad i = 1, \dots, n_h, \tag{2.7}$$

where $f_{\text{nlp}} : \mathbb{R}^{n_z} \to \mathbb{R}$ and $h_{\text{nlp},i} : \mathbb{R}^{n_z} \to \mathbb{R}$ for $i = 1, \dots, n_h$ are twice continuously differentiable functions. Recall the Lagrangian function of the inequality-constrained NLP (2.7)

$$\mathcal{L}(z, \mu) = f_{\text{nlp}}(z) + \sum_{i=1}^{n_h} \mu_i h_{\text{nlp},i}(z),$$

with the dual variables $\mu_i \in \mathbb{R}$ and $\mu^\top = [\mu_1, \dots, \mu_{n_h}]$. The Lagrangian function can be used to define FONCs for optimality of the primal variables $z^\star$ by utilizing the dual variables $\mu^\star$, known as the Karush-Kuhn-Tucker (KKT) conditions

$$\nabla_z \mathcal{L}(z^\star, \mu^\star) = 0 \tag{2.8a}$$

$$\mu_i^\star h_{\text{nlp},i}(z^\star) = 0, \quad i = 1, \dots, n_h, \tag{2.8b}$$

$$\mu_i^\star \geq 0, \quad i = 1, \dots, n_h, \tag{2.8c}$$

$$h_{\text{nlp},i}(z^\star) \leq 0, \quad i = 1, \dots, n_h. \tag{2.8d}$$

Many optimization algorithms aim to find points that fulfill the FONCs for the Lagrangian function. The two widely used methods SQP and IP are explained below.

Sequential Quadratic Programming (SQP)

The basic idea of SQP is to iteratively approximate the NLP (2.7) by a series of QPs, similar to the standard Newton iterations. An initial guess z_0 for the primal decision variables is chosen as the first linearization point. In order to compute an update of the primary and dual variables as a result of an SQP iteration, the Hessian matrix needs to be computed and inverted, which is computationally expensive. Moreover, if the exact Hessian $\nabla_z^2 \mathcal{L}(z_0, \mu_0)$ is used, the QP subproblems may be nonconvex and hard to solve.

In order to derive convex Hessians and to speed up the computation of the Hessian and its inverse matrix, various approximations exist, known as Newton-type methods. A popular method is the Gauss-Newton Hessian approximation, which is, however, only applicable to NLPs with a nonlinear least-squares objective [217]. The Gauss-Newton Hessian can be computed fast and is always convex. Other computationally favorable Hessian approximations can also be obtained by the Broyden–Fletcher–Goldfarb–Shanno (BFGS) algorithm [53, 95], which can directly approximate the inverse Hessian if desired. For general NLPs, methods described in [290] can be used to get convex Hessian matrices. Hessian approximations may be based on the dual variables μ, requiring also their initialization by μ_0.

After constructing the QP at the linearization point z_0 with the Hessian approximation B_0 obtained from z_0 and, possibly μ_0, the approximating QP is solved to get primal optimizers $z_0^\star$ and dual variables $\mu_0^\star$. After that, the new initial guess or linearization point is set to the previous optimizers with $z_1 \leftarrow z_0^\star$ and $\mu_1 \leftarrow \mu_0^\star$, and another QP is constructed to approximate the NLP at z_1 with a new Hessian approximation B_1. This procedure is repeated for M steps or until convergence with an acceptance criterion to find the final optimizer $z^\star := z_M^\star$ for the NLP.

The QP optimization subproblem for the k-th linearization point depends on the primal variables z_k and the dual variables μ_k. The dual variables may only influence the Hessian approximation B_k, but Hessian approximations independent of dual variables exist as mentioned above. The QP subproblem can be written as

$$\begin{aligned} \min_{z \in \mathbb{R}^{n_z}} \quad & \nabla f_{\text{nlp}}(z_k)^\top (z - z_k) + \frac{1}{2}(z - z_k)^\top B_k (z - z_k) \\ \text{s.t.} \quad & h_{\text{nlp},i}(z_k) + \frac{\partial h_{\text{nlp}}}{\partial z}(z_k)(z - z_k) \leq 0, \quad i = 1, \ldots, n_h. \end{aligned} \tag{2.9}$$

The QP subproblems may be solved by various different algorithms, including active-set solvers such as `qpOASES` [94] or QP interior point solvers such as `HPIPM` [97] for MPC structured problems. Within the SQP algorithm, the individual QP subproblems may have different sets of active constraints that need to be figured out by the QP solver at each iteration if active set QP solvers are used. When the linearization point z_k is close to a strongly regular solution [217], it can be shown that the set of active constraints of the QP subproblem is identical to the set of active constraints of the original NLP [232].

The Hessian approximation is a major design choice since it can provide computationally less expensive iterations. An important question is how the convergence rate of SQP methods is influenced. Consider a sequence z_k that converges to $z^\star$. If there exists a positive integer $\underline{k}$, a positive real number $\bar{c}$, and a sequence $c_k \leq \bar{c} < 1$, a sequence z_k converges "q-linear" if it holds for some $k \geq \underline{k}$ that

$$|z_{k+1} - z^\star| \leq c_k |z_k - z^\star|.$$

If also c_k converges to zero, the convergence is "q-superlinear", and named "q-quadratic", if $c_k = \mathcal{O}\left(|z_k - z^\star|\right)$. While the convergence when using exact Hessian matrices is q-quadratic, given a sufficiently close starting point, methods with other Hessian approximations usually converge slower. For instance, the Gauss-Newton method converges q-linearly, and Hessian update algorithms like the BFGS converge q-superlinearly [217].

Nonlinear Interior Point Method

As introduced above, constraints can conceptually be seen as a mathematical formulation to strictly prevent variables from obtaining specific values. In the context of OCPs, this could be related to conceptually infinite costs or physically impossible values. Infinite values could be approximated by large numbers on a computer. Cost functions that approach infinity for constrained values exhibit a discontinuity at the border of the constraints.

Nonlinear IP methods formulate the constraints within the cost function by a smooth approximation. More precisely, a barrier function is used that also goes towards infinity where the constraints are active but smoothly approximates the transition from the feasible domain. The choice of smoothness alters the original NLP since it assigns wrong costs in the feasible domain. Thus, lowering the smoothness is desired as long as the numerical algorithm is able to solve the problem. In most algorithms, a homotopy is used in consecutive iterations, where the smoothness is varied from initially smooth costs to increasingly nonsmooth costs that approximate the original NLP better.

Regarding the inequality constrained NLP (2.7), the IP method first reformulates the constraints by additional slack variables $\sigma^\top = [\sigma_1, \ldots, \sigma_{n_h}] \in \mathbb{R}^{n_h}$ into the equivalent formulation

$$\min_{z \in \mathbb{R}^{n_z}, \sigma \in \mathbb{R}^{n_h}} f_{\mathrm{nlp}}(z) \quad \text{s.t.} \quad h_{\mathrm{nlp},i}(z) + \sigma_i = 0, \quad \sigma_i \geq 0 \quad i = 1, \ldots, n_h. \tag{2.10}$$

Next, the inequalities for the slack variables are replaced by the logarithmic barrier function within the cost function. By utilizing a smoothness parameter $\tau > 0$, the equality constrained IP problem is

$$\min_{z \in \mathbb{R}^{n_z}, \sigma \in \mathbb{R}^{n_h}} f_{\mathrm{nlp}}(z) - \tau \sum_{i=1}^{n_h} \log(\sigma_i) \quad \text{s.t.} \quad h_{\mathrm{nlp},i}(z) + \sigma_i = 0, \quad i = 1, \ldots, n_h. \tag{2.11}$$

The parameter τ defines the smoothness, where the smoothness is increased by larger values of τ. Therefore, a homotopy varies the smoothness starting from large values and iteratively decreases τ towards zero. The equality-constrained IP problem can be solved by formulating the KKT conditions and solving the related root-finding problem. Notably, the root-finding problem does not contain the more sophisticated complementarity conditions for the inequalities anymore and, thus, can be solved more efficiently [217].

Both the SQP and IP algorithms start from an initial guess of the solution, which determines to which local minimum the algorithm converges if several minima exist. For nonconvex problems, even finding a feasible initial guess or initial guesses that result in acceptable local minima may be challenging.

One mitigation strategy is to reformulate nonconvex problems into MIQPs, as proposed in our contributions [225, 227, 229] and Chapter 6 in this thesis. With this method, the integer variables model the combinatorial problem part related to the nonconvexity and allow the solver to determine globally optimal solutions. The concept of mixed-integer programming is introduced in the next section.

2.1.3 Mixed-Integer Programming

In the following, basic concepts and algorithms for the class of MINLPs are introduced along the lines of [29] and [156]. For the remainder of this section, equality constraints of MINLP (2.5) are formulated as part of the inequality constraints, by

$$\min_{z \in \mathbb{R}^{n_z}, y \in \mathbb{Z}^{n_y}} f_{\mathrm{mi}}(z, y) \quad \text{s.t.} \quad \begin{cases} [z^\top, y^\top] \in \mathbb{W}, \\ h_{\mathrm{mi},i}(z, y) \leq 0, \quad i = 1, \ldots, n_h. \end{cases} \tag{2.12}$$

Moreover, it is assumed that $f_{\mathrm{mi}}(z, y)$ and all $h_{\mathrm{mi},i}(z, y)$ for $i = 1, \dots, n_h$ are twice differentiable, but not necessarily convex. The feasible set is

$$\begin{aligned}\Omega := \big\{(z, y) \in \mathbb{R}^{n_z} \times \mathbb{R}^{n_y} \big| [z^\top, y^\top] \in \mathbb{W},\\ h_{\mathrm{mi},i}(z, y) \leq 0, \quad i = 1, \dots, n_h,\\ y \in \mathbb{Z}^{n_y}\big\},\end{aligned}$$

and its canonical relaxation is

$$\begin{aligned}\Omega^{\mathrm{rel}} := \big\{(z, y) \in \mathbb{R}^{n_z} \times \mathbb{R}^{n_y} \big| [z^\top, y^\top] \in \mathbb{W},\\ h_{\mathrm{mi},i}(z, y) \leq 0, \quad i = 1, \dots, n_h\big\}.\end{aligned}$$

A convex MINLP is an MINLP of the form (2.12) where the NLP resulting from the canonical relaxation of the feasible set to Ω^{rel} is convex, which is the case if the functions $f_{\mathrm{mi}}(z, y)$ and $h_{\mathrm{mi},i}(z, y)$ for $i = 1, \dots, n_h$ are all convex. For the remainder of this section, Problem (2.12) is assumed to be a convex MINLP. It can be stated more concisely as

$$\min_{(z,y)\in\Omega} f_{\mathrm{mi}}(z, y). \tag{2.13}$$

General Concepts

To give a general high-level overview of how integer problems are solved, the basic concepts of branching, relaxations, cutting planes, bounding, heuristics and parallelism are explained in the following.

Branching. Let the feasible set Ω be decomposed into subsets $\Omega_1, \dots, \Omega_P$. The optimization problem (2.13) can equally be solved by

$$\min_{(z,y)\in\Omega} f_{\mathrm{mi}}(z, y) = \min_{i\in[1,\dots,P]} \min_{(z,y)\in\Omega_i} f_{\mathrm{mi}}(z, y), \tag{2.14}$$

where the objective is minimized over each individual partition, and thereafter, the minimum is taken over all individual optimizers. Branching summarizes different methods that decompose the feasible set, mainly related to the integer variable assignments, and solve the individual subproblems. A naive way of solving MINLPs would be to enumerate all integer assignments, exhaustively solve all subproblems with fixed integer variables by NLP solvers, and, finally, take the lowest value of all enumerated subproblems. This scales poorly for an increasing number of integer variables. Even in the case of binary variables, i.e., integer variables in $\{0, 1\}$, the number of enumerations n_{enum} is exponential in the number of binary variables with $n_{\mathrm{enum}} = 2^{n_y}$.

Therefore, strategies exist to exclude as many integer assignments as possible beforehand for a particular problem instance. These strategies are known as presolve techniques. For example, consider the problem constraints

$$y_1 \leq 1, \quad y_2 \leq 1, \quad y_1 + y_2 \geq 2,$$

with $y_1 \in \{0,1\}$ and $y_2 \in \{0,1\}$. Clearly, only the assignment of $y_1 = 1$ and $y_2 = 1$ is feasible. If a solver is able to detect these logical assignment constraints beforehand, the number of possible binary variable assignments can be reduced vastly, leading to faster computations.

Relaxations. Relaxations of the MINLP (2.13) are problems that expand the feasible set to the superset $\hat{\Omega}$, with $\Omega \subseteq \hat{\Omega}$ and value functions $\hat{f}(z,y)$ that are upper bounded by the original value function, with $\hat{f}(z,y) \leq f(z,y)$ for all $(z,y) \in \Omega$. The relaxed NLP is written as

$$\min_{(z,y)\in\hat{\Omega}} \hat{f}_{\mathrm{mi}}(z,y). \tag{2.15}$$

Examples of possible relaxations include the relaxation of integer variables to continuous variables, outer approximations of the feasible set by, e.g., linearizations, dropping some constraints, utilizing duality, the Lagrangian relaxation, and an under-estimation of the objective function, e.g., by linearization. The optimal values of relaxed problems are always lower bounds of the original optimization problem.

Cutting Planes. Consider a solution $(\hat{z},\hat{y})$ that is feasible for the relaxed problem but infeasible for the original problem. The idea of cutting planes is to exclude this solution from the relaxation $\hat{\Omega}$ by adding constraints without excluding feasible solutions of the original problem. This procedure is also referred to as relaxation refinement [29]. Tightening the relaxation $\hat{\Omega}$ can only improve the lower bound, i.e., the lower bound in a consecutive iteration will be equal to or larger than the previous bound.

Bounding. Many algorithms that solve convex MINLPs utilize lower and upper bounds of the objective function to quickly prune integer variable assignments and make arguments about the quality of the intermediate updated solution. For example, suppose the solver finds a feasible assignment of integer variables and a solution of the remaining convex program, i.e., “it cannot get worse”. In that case, the objective value of this particular solution yields an upper bound on the optimal cost. Given such an upper bound, any relaxed solution with a higher optimal objective value can be excluded from further refinements as part of an algorithm. A lower bound can be found, for instance, by solving the relaxed problem (2.15). The lower bound can be used to estimate how close the objective of the currently best feasible solution is to the global optimum of the original problem.

Heuristics. Heuristics play an essential role in solving mixed-integer problems. They aim to find feasible assignments of binary variables quickly. This is beneficial for several reasons. First, upper bounds are helpful to solve the problem, as detailed in the following subsection. Second, if the solver's time is limited, it is often helpful to return a feasible solution, which may not necessarily be the optimal solution.

Parallelism. An essential property of many mixed-integer algorithms is their ability to parallelize large parts of the algorithm. For example, the naive enumeration can be parallelized. If fully parallelized, the overall solution time would only be bounded by the slowest computation time of individual subproblems.

Algorithms for solving mixed-integer programs include some form of tree search on the integer variables. The (nonlinear) branch and bound algorithm was among the first algorithms that were used for various classes of MIPs. The basics are described in the following in order to provide the reader with some intuitive understanding.

Branch and Bound.

The branch and bound algorithm [71] starts by solving the canonically relaxed problem

$$\min_{(z,y)\in\Omega^{\mathrm{rel}}} f_{\mathrm{mi}}(z,y). \tag{2.16}$$

Two possible outcomes of this problem would directly terminate the algorithm. First, if the solution of the relaxed problem lies in the set Ω, i.e., the relaxed integer variables y obtain integer values, the optimizers of (2.16) are also optimizers of the original problem. Second, if the relaxed problem is infeasible, the original MINLP is infeasible since, trivially, adding constraints to an infeasible problem could not make it feasible.

If none of the above scenarios apply, a graph tree is constructed with nodes that correspond to subproblems $\mathbb{P}(\underline{y},\overline{y})$, defined as

$$\min_{(z,y)\in\Omega^{\mathrm{rel}}} f_{\mathrm{mi}}(z,y), \quad \text{s.t. } \underline{y} \le y \le \overline{y}, \tag{2.17}$$

where $\overline{y} \in \mathbb{R}^{n_y}$ and $\underline{y} \in \mathbb{R}^{n_y}$ are bounds on the relaxed integer variables. The root problem does not have additional bounds; thus, it is denoted as $\mathbb{P}(-\infty,\infty)$.

Branching corresponds now to creating two new subproblems $\mathbb{P}(\underline{y}_{\mathrm{a}}, \overline{y}_{\mathrm{a}})$ and $\mathbb{P}(\underline{y}_{\mathrm{b}}, \overline{y}_{\mathrm{b}})$ from a root NLP $\mathbb{P}(\underline{y}, \overline{y})$. Therefore, for any of the integer variables, denoted by y_i, whose solution $\hat{y}_i$ in the relaxed problem is not integer, bounds are added in both of the new subproblems. Particularly, first, the bounds are initialized to be equal to the root node, with $(\underline{y}_{\mathrm{a}}, \overline{y}_{\mathrm{a}}) := (\underline{y}, \overline{y})$ and $(\underline{y}_{\mathrm{b}}, \overline{y}_{\mathrm{b}}) := (\underline{y}, \overline{y})$. Thereafter, bounds for the i-th branching variable are set to the lowest larger integer as a lower bound for one problem, e.g., $\mathbb{P}(\underline{y}_{\mathrm{a}}, \overline{y}_{\mathrm{a}})$, and to the largest lower bound as an upper bound for the other problem $\mathbb{P}(\underline{y}_{\mathrm{b}}, \overline{y}_{\mathrm{b}})$, i.e., $\underline{y}_{\mathrm{a},i} := \lceil \hat{y}_i \rceil$ and $\overline{y}_{\mathrm{b},i} := \lfloor \hat{y}_i \rfloor$.

Every problem $\mathbb{P}(\underline{y}, \overline{y})$ that is solved, and where the solution is integer, is a feasible solution to the original problem and the value of the subproblem $\mathrm{val}\big(\mathbb{P}(\underline{y}, \overline{y})\big)$ is an upper bound U on the optimal value of the original problem. The lowest upper bound, in addition to the particular optimal variables, is stored as the algorithm iterates. As soon as any feasible solution is found, the algorithm can return a feasible suboptimal solution.

Two basic pruning rules are a core part of the branch and bound algorithm. (i) If a problem $\mathbb{P}(\underline{y}, \overline{y})$ is infeasible, any problem in the subtree is infeasible. Therefore, the whole tree can be pruned. (ii) If the value of a problem is larger than the currently lowest upper bound, i.e., $\mathrm{val}\big(\mathbb{P}(\underline{y}, \overline{y})\big) \geq U$, all subtrees and the node can be pruned.

A vital component of the algorithm is the decision on which exact variable the algorithm branches to new subproblems. This choice vastly influences the performance of the algorithm. Usually, the first goal is to quickly find a feasible solution to the original problem since it generates an upper bound to allow related pruning. Furthermore, a feasible solution could be returned if the algorithm was terminated early. The second goal is to decrease the upper bound as much as possible. Many varieties exist for both stages, and details can be found in [29].

Modeling Paradigms

Integer variables can be typically used for three different purposes: (i) discrete quantities, (ii) discrete decision variables that may be related by logical formula, and (iii) indicator variables that get activated if other continuous variables are within a certain value range [304].

Based on the basic principles of the mixed-integer algorithms provided above, a couple of design rules are provided when formulating a mixed integer problem, along the lines of [16, 304].

In this thesis, binary variables, i.e., integer variables that are in $\{0, 1\}$, are extensively used as indicator variables to formulate logical constraints. Some key concepts are presented below. A binary variable $\beta \in \{0, 1\}$ can be used to activate a constraint on a non-negative variable $x \in \mathbb{R}_{\geq 0}$, with bounds $\underline{x} \leq x \leq \overline{x}$, which is expressed by the implication

$$[\beta = 1] \implies [x \geq m]\,,$$

where $m \in \mathbb{R}$ is a constant, with the constraint

$$x - m\beta \geq 0.$$

and the notation $[f(x) < 0]$ denoting the proposition of a formula $f(x) < 0$ which can either be "true" or "false". The other direction of the implication

$$[x > m] \implies [\beta = 1]\,,$$

can be formulated by

$$x - M\beta \leq m,$$

where $M + m$ is an upper bound for x, with $\overline{x} \leq m + M$. Suppose variables β_1 and β_2 are binary variables that are used as indicator variables or logical decisions. Logical formulas can now be formulated in terms of constraints on the binary variables. For example, a logical "or" between propositions related to β_1 and β_2, i.e., $[\beta_1 = 1] \vee [\beta_2 = 1]$, is formulated utilizing constraints by

$$\beta_1 + \beta_2 \geq 1.$$

A logical "and", i.e., $[\beta_1 = 1] \wedge [\beta_2 = 1]$, is formulated by

$$\beta_1 = 1,\ \beta_2 = 1.$$

By using similar equations for disjunctions or negations, logical dependencies can be formulated as part of MIPs. For instance, some nonconvex compact sets of continuous variables can be decomposed into convex sets that imply and are implied by binary variables if and only if the continuous variables are within decomposed convex sets. A disjunctive constraint between the binary variables can be formulated that requires the continuous variable to be within one set. Using this expression to reformulate an NLP with a convex objective function but a nonconvex feasible set into a convex MINLP, this convex MINLP can be solved to global optimality despite the nonconvex feasible set for continuous variables. This formulation was used in our work [225, 227] and [229].

Constraints. When formulating NLPs, an usual heuristic is to reduce the number of constraints since they are a major challenge for the solver. Nevertheless, when formulating MINLPs, adding constraints may often help the solver in the presolve phase and during algorithm iterations to prune larger parts of the search tree.

Number of integer variables. Since, in the worst case, for every node in the branch and bound tree, an NLP needs to be solved, another appealing modeling paradigm when formulating MINLPs may be to reduce the number of integer variables. This is true for most problems. However, a major computational speedup can be achieved if large parts of the branch and bound tree are pruned. Additional integer variables may enable more efficient pruning in some cases. For example, consider properties A, B, C, and D and four disjunctive logic choices that lead to the property combinations AC, BC, AD, and BD. A disjunction among the four logic choices can be modeled by four binary variables $\beta_1, \dots, \beta_4$ and

$$\beta_1 \iff AC,\ \beta_2 \iff BC,\ \beta_3 \iff AD,\ \beta_4 \iff BD,$$

$$\beta_1 + \beta_2 + \beta_3 + \beta_4 = 1.$$

It would not be possible to distinguish between property A and B using the branch and bound algorithm for this formulation. However, it may be useful to branch between property A and B as it could have large implications on the overall cost. Therefore, an additional variable β and the constraints

$$\beta_1 + \beta_3 - \beta = 0, \quad \beta_2 + \beta_4 + \beta = 1,$$

would allow branching on the variable β. The additional branching would allow pruning related to properties A and B. Remarkably, numerous counter-examples exist where reducing the integer variables speeds up the computation [304]. This could be due to symmetries in the model or unnecessarily complicated formulations. For instance, an optimization problem over a convex set could be reformulated by splitting the set into convex partitions and adding disjunctive logic constraints among the partitions with integer variables. Clearly, these reformulations would introduce unnecessary constraints and integer variables.

Further strategies for formulating MIPs, such as different disjunctive formulations, tightening of constraints, or exploiting symmetries, can be found in [304].

2.2 Optimal Control Problem

Optimal control is a branch of applied mathematics that deals with finding a control law for a dynamical system. The aim is to minimize a certain cost and respect constraints related to the system state and the control inputs. The dynamical system model can be stated stochastically or deterministically within the OCP. Note that the concept of Markov decision processes (MDPs) is equal to stochastic discrete-time OCPs and used usually in the field of RL, described in Sect. 2.3.6.

In this thesis, discrete-time OCPs are solved which are approximations of continuous-time OCPs such as the motion planning problem (1.1). Some basic concepts of how to derive discrete-time OCPs from continuous-time OCPs are provided. For a detailed overview of stochastic continuous-time OCPs the reader is referred to [204] and for stochastic discrete-time OCPs, see [35, 37]. The time discretization is illuminated briefly in the following.

A discretization of time is often relevant for real-world systems, where the electronic actuator and sensor systems operate in real-time embedded systems at sampling frequencies. During a sampling period t_Δ, the computations of evaluating sensor inputs and performing operations to provide the subsequent actuator output are performed. The most straightforward control during the embedded system computations is the zero-order-hold setting, where the control $u(t)$ is kept constant with $u(t) = u_k$ between two sampling times kt_Δ and $(k+1)t_\Delta$. The index $k \in \mathbb{N}$ is used to enumerate the sampling intervals. During a sampling interval, the environment state follows the system dynamics. The state is usually not directly accessible. Instead, the state is measured at the discrete times kt_Δ. In simulations, numerical integrators are used to obtain the next discrete state at $(k+1)t_\Delta$ [211]. The discrete-time system inputs, which are the controller outputs, are denoted by the vector $u_k \in \mathcal{U} \subseteq \mathbb{R}^m$. The discrete-time system states are $x_k \in \mathcal{X} \subseteq \mathbb{R}^{n_x}$. The state and control spaces are described by the sets $\mathcal{X}$ and $\mathcal{U}$, respectively.

The fact that real-world electronic systems operate in discrete time and the more tractable direct formulation often leads to the description of the model in a discrete-time version for the stochastic case as

$$x_{k+1} = F(x_k, u_k, w_k), \quad \text{with } w_k \sim P_{\mathcal{W}}(\cdot|x_k, u_k) \tag{2.18}$$

with $F : \mathcal{X} \times \mathcal{U} \times \mathcal{W} \to \mathbb{R}^{n_x}$, the sampling space $\mathcal{W} \subseteq \mathbb{R}^{n_p}$, and where $P_{\mathcal{W}}(\cdot|x_k, u_k)$ is a probability distribution that may depend on the states and controls, but not directly on the previous disturbances w_{k-1}. Note that the term stochastic optimal control often emphasizes stochasticity, whereas deterministic optimal control does not involve random variables. In the following, only Markov systems are considered where the state x_k is assumed to be measured and fully describes the system at a particular time kt_Δ.

The goal is to find a deterministic policy $\pi : \mathcal{X} \to \mathcal{U}$, $\pi \in \Pi$, with Π the space of admissible functions. The policy determines the control $u = \pi(x)$ and optimizes a performance criterion concerning a running cost $l : \mathcal{X} \times \mathcal{U} \to \mathbb{R} \cup [-\infty, \infty]$ with the extended real numbers to allow for infinite costs. A typical performance criterion may be a discounted cost

$$\begin{aligned} J^\pi_\gamma(x) := \mathbb{E}_{w_k \sim P_{\mathcal{W}}} \Bigg[\sum_{k=0}^{\infty} \gamma^k l\Big(x_k, \pi(x_k)\Big) \;\Bigg|\; \\ x_0 = x,\, x_{k+1} = F\big(x_k, \pi(x_k), w_k\big) \Bigg], \end{aligned} \tag{2.19}$$

with the discount factor $\gamma \in (0,1)$ that exponentially discounts costs appearing at an increasing time horizon. Another typical performance criterion is the average cost

$$J^{\pi}(x) := \limsup_{T\to\infty} \frac{1}{T}\,\mathbb{E}_{w_k\sim P_{\mathcal{W}}}\left[\sum_{k=0}^{T} l\Big(x_k,\pi\big(x_k\big)\Big)\,\middle|\; x_0 = x,\ x_{k+1} = F\Big(x_k,\pi\big(x_k\big),w_k\Big)\right] \tag{2.20}$$

that conceptually is similar to the discount factor $\gamma = 1$. The function $J^{\pi}_{\gamma}(x)$ defines the accumulated cost for a policy π when starting at state x, i.e., it defines the cost-related value of a state x under a given policy. Therefore, it is also referred to as a value function. Remarkably, the value function results from the system dynamics $F(\cdot)$, the defined running costs $l(\cdot)$, and the chosen policy $\pi(\cdot)$. It may be emphasized that the system dynamics, the running cost, and the discount factor are defined by the application. The engineering goal is to find a policy that lowers the value function at every point of the state space $x \in \mathcal{X}$. A policy that optimizes the value function to the lowest possible values for all $x \in \mathcal{X}$ is the optimal policy and is defined by

$$J^{\star}_{\gamma}(x) = J^{\pi^{\star}}_{\gamma}(x) \le J^{\pi}_{\gamma}(x),\ \forall x \in \mathcal{X},\ \forall \pi \in \Pi. \tag{2.21}$$

2.3 Optimal Control Algorithms

The ultimate goal of optimal control algorithms is to find a policy that minimizes the performance criterion when applied to the real-world environment. Unfortunately, finding the optimal policy $\pi^{\star}(x)$ as defined in (2.21) is, in general, extremely difficult or intractable.

One strategy is to simplify and approximate the environment. For the approximated environment, the optimal policies are applied to the inherently unknown stochastic and nonlinear real-world environment, hoping to achieve an acceptable, usually sub-optimal, performance. A discussion on environment models is given in the next Sect. 2.3.1, followed by a discussion of, what is called "implicit" and "explicit" approaches in Sect. 2.3.2. Model-based approaches relevant to this thesis are further illustrated in Sect. 2.3.3, utilizing derivative-based optimization, and Sect. 2.3.4, relying on sampling-based optimization.

Two further algorithm classes are elaborated. First, imitation learning (IL) is described in Sect. 2.3.5. In fact, the IL problem setting is different since a further component of the problem setting is assumed, which is an expert policy $\tilde{\pi}^{*}(x)$. The expert could be a human demonstrator or a sophisticated control algorithm that is burdened by its computational complexity and is aimed to be replaced by a more efficient policy. Second, reinforcement learning (RL) is described in Sect. 2.3.6 as an algorithm that starts with policy $\pi(x;\theta)$ that is parameterized by some random vector $\theta \in \mathbb{R}^{n_\theta}$. By interacting online with the environment and evaluating the obtained costs along the state transitions, the parameters are updated to achieve a lower expected value function.

2.3.1 Environment Models

The mathematically modeled environment allows for a theoretical analysis of essential properties related to the performance criteria, such as stability and safety. Nevertheless, as stressed before, the analysis never applies to the real-world system, as this can not be modeled precisely. However, the hope is that the real-world environment behaves similarly enough in that the policy behavior in the real world is similar to the simplified environment. As an example for an approximation, consider the environment to be modeled linearly, i.e.,

$$x_{k+1} = Ax_k + Bu_k + w_k$$

with $A \in \mathbb{R}^{n_x \times n_x}$, $B \in \mathbb{R}^{n_x \times n_u}$ and where $w_k \sim \mathcal{N}(0, \Sigma)$ follows a multivariate normal distribution $\mathcal{N}(0, \Sigma)$ with zero mean and covariance $\Sigma \in \mathbb{R}^{n_x \times n_x}$. Moreover, let the running cost be quadratic with the positive semi-definite matrices $Q \in \mathbb{R}^{n_x \times n_x}$ and $R \in \mathbb{R}^{n_u \times n_u}$ and

$$l(x_k, u_k) = x_k^\top Q x_k + u_k^\top R u_k.$$

For this case, it can be shown that the optimal policy is linear, i.e., $\pi(x_k) = -Kx_k$, cf. [217]. The feedback gain matrix K can be computed by

$$K = (R + B^\top PB)^{-1}(B^\top PA),$$

where P is derived by solving the algebraic Riccati equation

$$P = Q + A^\top PA - A^\top PB(R + B^\top PB)^{-1}B^\top PA$$

Such an analysis of simplified environments allows one to argue about properties for real-world systems that usually approximately fulfill the assumptions.

It ought to be stressed that there are various ways how models can be used in order to ultimately find a policy π for the real-world system, cf., Fig. (2.1). First, a simulation model of the real world can be used for faster and safer interactions. Simulation models also provide insights into hidden states. Secondly, an analysis model can be used to reason about the theoretical properties of a closed-loop system where a controller is assumed to interact with this particular model. Thirdly, a controller model can be used as part of an algorithm that is executed repeatedly online to find the policy output whenever a new state is encountered. For example, controller models are used to plan future trajectories, such as in

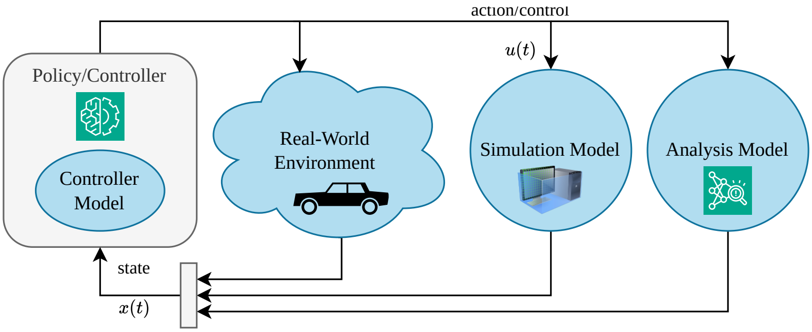

Figure 2.1: Different models are used as approximations of the real-world environment. Simulation models should accurately reproduce relevant real-world behavior. Analysis models are used to understand the theoretical properties of the environment. Often, their complexity is limited due to mathematical challenges in nonlinear and stochastic system analysis. A controller or policy model is used to compute the current control online in a specific time step.

model predictive control (MPC) or other model-based control techniques, such as internal model control [234].

A simulation model is always a simplification of the real-world environment and is based on approximating assumptions. Simplifications in the simulation environment always imply the same simplifications to the policy requirements if the policy performance is evaluated only in simulation. However, within the policy, an even simpler model is usually used to compute actions that achieve a good performance. The controller model trades off the accuracy of modeling the real-world environment with the requirements of the control algorithm. For example, derivative-based MPC requires smooth functions. Similarly, internal model control [234] requires linear models.

An analysis model may be used to theoretically derive specific properties for a control system where the model used in the control algorithm differs from the analysis model. For instance, the effect of using a deterministic approximation of a stochastic model in the controller may be analyzed with a stochastic analysis model. The analysis model usually trades off the complexity of mathematical objects with the actual accuracy of modeling the environment, which makes it also different from the simulation model. The primary purpose of the simulation model is to approximate the environment as accurately as possible.

In the following, general simplifications that are used within this thesis are illuminated. Particular details for simplifications within the policy categories are given in the corresponding sections.

Deterministic models. For the analysis of many problems, it is sufficient to narrow down the environment model to deterministic models

$$\dot{x}(t) = f(x(t), u(t)).$$

Deterministic models often improve the reproducibility [270] and make the optimal control problem (OCP) easier to solve.

Linear models. As indicated in the linear quadratic example above, linear models

$$\dot{x}(t) = Ax(t) + Bu(t) + w(t)$$

are appealing since they simplify solving the OCP, i.e., finding an optimal policy for the linear system.

Approximate costs. The cost $l(x, u)$ can be arbitrary, in general. However, approximations of the cost may be favorable for control algorithms that use an approximation of the OCP as part of the policy algorithm. Essential desired properties are smoothness, convexity, convex quadratic shape, linear shape, or a value of zero at a targeted steady state.

Discrete time. The time-discretization of the inherently infinite-dimensional controls $u(t)$ in time to finitely many controls u_k makes the problem much more tractable. Related deterministic optimal control methods are called *direct approaches* [123].

Discrete states and controls. Besides a discretization of time, the state and control space may also be discretized. Consequently, the system model is then described by a graph, where the vertices are discrete states and the finite directed edges are transitions related to discrete controls. The discretization of the state and control space exhibits several advantages but also significant limitations. An advantage is the applicability of discrete planning algorithms such as Dijkstra's graph search [159] or dynamic programming variants [38]. These algorithms typically are guaranteed to find the global optimum in the discretized search space. However, in the continuous space, they may be suboptimal. Additionally, graph search algorithms scale poorly with the dimension of the state space, which is known as Bellman's "curse of dimensionality" [28].

2.3.2 Implicit and Explicit Policies

In the following, two categories of algorithms are described to obtain acceptable closed-loop behavior. First, the category of *implicit* policies is introduced, which contains an approximate form of a problem formulation within the policy which is solved implicitly. Secondly, the *explicit* policy class is described. Here, explicit algorithms refer to policies that are described through explicit parameterized functions. The parameters are typically learned through an interaction with the real-world environment. Above all, explicit functions, such as a linear control law $\pi(x) = -Kx$ with $K \in \mathbb{R}^{n_u \times n_x}$, are standard and widespread in control systems engineering, but the parameters are typically not “learned” while interacting with the system. In this thesis, explicit policies are parameterized functions, such as neural networks (NNs), that are learned by interactions with the real-world environment or a simulator.

Implicit Policies

The primary control approach of this thesis is MPC [217], which defines the policy $\pi(x)$ implicitly by approximating the “real-world” OCP by a numerically tractable formulation. The optimal policy is not computed offline in advance for the whole state space $\mathcal{X}$, but instead a computationally tractable approximation of the OCP is solved online only for the current state. Several significant simplifications are used.

First, the optimization problem is formulated in discrete time with states x_k and controls u_k. Secondly, the stochastic optimization problem is approximated deterministically by replacing the random noise variable w by the mean value of the noise $\bar{w}$. Next, the optimization problem (2.21) is solved not over policy functions $\pi(\cdot)$ but over particular open-loop controls u_i, for $i = k, k+1, \ldots$ that are computed for the current state x_k at time kt_Δ solely. Since the real-world system is always continuous in time, the running cost $l(x_k, u_k)$ should approximate the states and controls between two sampling times kt_Δ and $(k+1)t_\Delta$. The continuous-time running cost $\tilde{l}\big(x(t), u(t)\big)$ is usually the actual cost desired to be evaluated when designing a system. If the continuous-time running cost is given by the problem definition, the time discretization demands a reformulation to the discrete-time running cost.

The dynamics $F(\cdot)$ are reformulated by a suitable numerical approximation

$$x_{k+1} = F^{\mathrm{mpc}}(x_k, u_k).$$

In fact, the numerical approximation should approximate the underlying continuous-time dynamics, which are expressed in the continuous deterministic case as an ordinary differential equation (ODE)

$$\dot{x} = f\big(x(t), u(t)\big)$$

with starting conditions $x(t=0)=\hat{x}$ and $f:\mathcal{X}\times\mathcal{U}\to\mathbb{R}^{n_x}$. For example, the simple explicit Euler formulation can be used to discretize the continuous system by

$$F^{\mathrm{mpc}}_{\mathrm{euler}}(x_k,u_k)=x_k+t_\Delta f(x_k,u_k).$$

In this case, also the approximated cost can be written as

$$l^{\mathrm{mpc}}_{\mathrm{euler}}(x,u):=t_\Delta\tilde{l}(x,u).$$

In the final approximation, the infinite horizon of the OCP is truncated to only N discrete time steps. The remaining part of the costs after N time steps is replaced by an approximation $J^{\mathrm{mpc}}(x)$ of the value function $J^\star(x)$. To achieve optimality, this cost would be required to be identical to the actual optimal value function. However, since the first part of the OCP is solved implicitly for the current state x, the approximation error at the end of the horizon Nt_Δ is less relevant for the closed-loop behavior, cf., [35]. The final MPC optimization problem that approximates the OCP with discounted costs (2.19) at a state x is

$$\begin{aligned}(u_0^\star,\dots,u_{N-1}^\star)\in\arg\min_{u_0,\dots,u_{N-1}}&\sum_{k=0}^{N-1}\gamma^k l^{\mathrm{mpc}}(x_k,u_k)+\gamma^N J^{\mathrm{mpc}}(x_N),\\ \text{with}\quad & x_0=x,\ x_{k+1}=F^{\mathrm{mpc}}(x_k,u_k),\end{aligned}\tag{2.22}$$

with the control decision variables $u_0,\dots,u_{N-1}$. After solving the optimization problem (2.22) at state x, the first derived optimal control $u_0^\star$ is applied to the environment as the policy output. After the sampling time t_Δ, the MPC optimization problem (2.22) is solved again for the next measured system state.

MPC algorithms do not require offline pre-computations other than identifying an optimization-friendly model. Some information may be passed from one iteration to the next to speed up the computations, cf., Sect 2.3.3. Notably, this procedure is an iterative, local, and implicit control policy.

The overall controller performance is significantly linked to the optimizer's performance. Therefore, dedicated optimization solvers have been developed for MPC problems, e.g., the open-source solver `acados` [291] or the commercial solver `FORCESPro` [83].

Explicit Policies

In the context of this thesis, explicit policies $\pi^\theta(x)$ are referred to as expressive parameterized functions with the parameter $\theta\in\mathbb{R}^{n_\theta}$, where the number of parameters n_θ is usually very high, and the function may be defined through some neural network (NN) architecture. For example, the authors in [306] used more than 10^7 parameters for their policy. Remarkably, the function space is limited through the particular parameterization, which may prohibit the policy $\pi^{\theta}(x)$ from being parameterized such that it is equal to the optimal policy $\pi^{\star}(x)$. However, the hope is that the parameterization is rich enough to get a reasonable closed-loop performance and a close approximation of $\pi^{\star}(x)$ [274].

The determination of the parameters θ is challenging and may follow various algorithms. Here, two approaches are considered. The imitation learning (IL) approach collects data from an expert policy, and while or after collecting data, the parameters are adjusted to approximate the expert behavior. The basics of IL are illuminated in Sect. 2.3.5. Within the reinforcement learning (RL) approach, the policy $\pi^{\theta}(x)$ interacts with the environment to determine which action leads to low costs in each state. By principles of dynamic programming (DP) and Monte Carlo sampling, the parameters θ are modified to acquire a policy that yields a value function for all states x [37, 38, 274]. A model is usually identified first before the implicit MPC policy is applied. Within vanilla RL algorithms, the parameters θ of an explicit policy $\pi^{\theta}(x)$ are continuously updated. The offline learning phase of RL algorithms is the main bottleneck to obtaining a policy that achieves good closed-loop performance. RL has become a significant research field within artificial intelligence, and some core concepts are eluded in Sect. 2.3.6.

The distinction between explicit and implicit policies and their associations to RL is rather conceptual and blurry. For example, within RL, an implicit parameterized optimization layer may be used that resembles an MPC, see, e.g., [110]. Similarly, an MPC layer may be used as a post-processing module to a learned policy in order to guarantee safety, such as the so-called “safety filter” in [280] or used as a fixed part of the environment to transform an auxiliary input to the environment to an actual input with particular guarantees. For example, the auxiliary input could be a desired position or trajectory, which is then used as a reference within MPC, cf. [48, 224].

2.3.3 Derivative-Based Online Optimization

Various algorithms exist to solve the implicit MPC optimization problem locally as defined in (2.22). In the following, the focus is set on derivative-based optimization algorithms, which utilize differentiable models and algorithmic differentiation to iteratively update the decision variables in order to lower the objective of the MPC problem (2.22). The algorithm’s performance is limited by finite computational resources and measured by, for instance, the mean and worst-case computation time and the suboptimality of the solution. As pointed out, the main objective is minimizing the closed-loop performance criterion. Therefore, the algorithm’s performance in solving the numerical optimization problem affects the expected closed-loop performance considerably. Highly suboptimal solutions may perform poorly in the real-world environment. Large computation times imply large sampling intervals, which again imply poor disturbance rejection.

It is pivotal that defining the structure of the optimization problem (2.22) has two contrary effects. First, by assuming a perfect oracle optimization solver that outputs the global optimum of (2.22) in zero time, the driving motivation would be to find the most sophisticated model for the real-world environment in order to achieve the best possible closed-loop performance. However, the expected performance of optimization solvers is highly dependent on the optimization problem structure (2.22), where simpler models allow, in general, the applicability of solvers with better performance and even guarantees.

Depending on the problem simplifications and approximations, problem (2.22) falls into one of the above problem classes. Notably, equivalent formulations in different problem classes usually exist for the same problem. For instance, MIQPs as formulated in (2.5) can be formulated as nonconvex NLPs with disconnected feasible sets by

$$\min_{z\in\mathbb{R}^{n_z},\bar{y}\in\mathbb{R}^{n_y}} f_{\mathrm{mi}}(z,\bar{y}) \quad \text{s.t.} \quad \begin{cases} g_{\mathrm{mi},i}(z,\bar{y}) = 0, & i = 1,\dots,n_g, \\ h_{\mathrm{mi},i}(z,\bar{y}) \geq 0, & i = 1,\dots,n_h, \\ \sin(\pi\bar{y}_i) = 0, & i = 1,\dots,n_y, \end{cases} \tag{2.23}$$

but this formulation would not be useful. It is often an engineer's task to find problem formulations that are aligned with specific optimization problem classes. After finding an appropriate formulation, a particular set of optimization algorithms can be applied. For each category, the different optimization solvers have particular properties that vary among the solvers within one optimization class. Such properties are the average and worst-case online computation time or the ability to return global minima, local minima, or stationary points. Another essential property is the anytime property. It defines whether an algorithm can return a sub-optimal decision variables at any time while solving the problem [330].

From an engineering perspective, all that matters is the finally achieved closed-loop performance, which is indirectly affected by the optimization problem properties and hard to determine a priori. For example, it is hard to determine if a sophisticated solver based on a detailed real-world model that requires a long computation time performs better than a high-sampling-frequency linear controller. Moreover, a sophisticated solver may have an unbounded computation time and, thus, may rely on a safety backup controller. However, a linear control, in contrast, may not approximate the problem complexity of the environment acceptably well. It is sometimes argued that only convex optimization problem structures allow the applicability of solvers that can guarantee a bounded computation time and reliably converge to global minimizers of the convex problem approximation [46]. Nevertheless, it can be argued that strongly nonconvex problems can not be approximated by convex problems well enough to achieve a reasonable closed-loop performance.

Given an approximation of the OCP as in the MPC problem (2.22), further reformulations are often made to make the problem easier to solve for continuous optimization algorithms. Several fundamental reformulations are listed below.

Constraints. First, parts $\mathcal{S}_\infty \subset \mathcal{X} \times \mathcal{U}$ of the state-space $\mathcal{X}$ and control-space $\mathcal{U}$ may have conceptually infinite costs with $l^{\mathrm{mpc}}(x_\infty, u_\infty) \to \infty$ for $(x_\infty, u_\infty) \in \mathcal{S}_\infty$ related to safety critical or infeasibility constraints. While it is impossible to guarantee safety in a real-world environment, it may at least be guaranteed for a simplified simulation model that approximates the real-world system well enough. In continuous optimization algorithms, costs $l^{\mathrm{mpc}}(x, u)$ with potentially infinite values are usually separated into a cost function $\tilde{l}^{\mathrm{mpc}} : \mathcal{X} \times \mathcal{U} \to \mathbb{R}$ with finite values and constraints $h(x, u) \leq 0$, where the sublevel set of $h(x, u)$ at zero defines the feasible set $\mathcal{X} \times \mathcal{U} \setminus \mathcal{S}_\infty$ and the cost $l^{\mathrm{mpc}}(x, u)$ has finite values. Therefore, it holds that $h(x, u) > 0 \Leftrightarrow (x, y) \in \mathcal{S}_\infty$. Similarly, there may exit states $x_\infty \in \mathbb{X}_{\mathrm{t}} \subset \mathcal{X}$ with infinite costs $J^{\mathrm{mpc}}(x_\infty) \to \infty$ of the terminal value function $J^{\mathrm{mpc}}(x)$. The infinite costs in the terminal value function may result from infinite values in the cost that can not be avoided in the future trajectory. For example, consider a car that drives too fast into a curve. At some point, a crash is inevitable despite the collision, i.e., the infinite cost, being several time steps ahead. Therefore, also the terminal value function is separated into a finite valued terminal value function $\tilde{J}^{\mathrm{mpc}} : \mathcal{X} \to \mathbb{R}$ and terminal constraints $h_{\mathrm{t}}(x) \leq 0$ that imply $x \in \mathcal{X} \setminus \mathbb{X}_{\mathrm{t}}$.

Single and Multiple Shooting. It is possible to use an optimization solver for the problem as defined in (2.22). This formulation is called direct single shooting, where *direct* refers to the transcription of the infinite-dimensional OCP to a finite one by discretizing the controls $u(t)$ and states $x(t)$ and single shooting refers to the forward simulation of the system along the whole horizon. In this setting, a state x_k depends on a simulation of the dynamics influenced by all controls u_i, where $0 \leq i < k$. This single shooting formulation only exhibits $(N-1)n_u$ decision variables. However, the resulting optimization problems may be highly nonlinear, and often may be challenging to solve. Better numerical properties can be achieved by reformulating the problem (2.22) into the so-called direct multiple shooting formulation [45]. In this formulation, $n_x N$ additional decision variables $s_k \in \mathcal{X}$ for $k = 0, \ldots, N$ are introduced as variable states that are allowed to violate the model dynamics during solver iterations. Intuitively, the solver acquires more flexibility to provide the optimal solution, and the hope is to improve the numerical properties by increasing the previously low number of decision variables. In fact, the additionally introduced decision variables appear in a favorable structure that optimization problem solvers can exploit. The MPC optimization problem (2.22) is reformulated in the direct multiple shooting formulation as

$$\begin{aligned} \min_{\substack{u_0,\ldots,u_{N-1}\\ s_0,\ldots,s_N}} \; & \sum_{k=0}^{N-1} \gamma^k l^{\mathrm{mpc}}(s_k,u_k) + \gamma^N J^{\mathrm{mpc}}(s_N), \\ \text{s.t.} \quad & s_0 = x, \; s_{k+1} = F^{\mathrm{mpc}}(s_k,u_k), \; k=0,\ldots,N-1. \end{aligned} \tag{2.24}$$

Numerical Integration. So far, the dynamics function was introduced as an explicit computation of a consecutive state x_{k+1} by the function $x_{k+1} = F^{\mathrm{mpc}}(x_k,u_k)$. More precisely, in this thesis, linear single-step explicit and implicit Runge-Kutta methods are used as defined in [211]. The underlying aim is to numerically solve the initial value problem (IVP) posed by the original ODE to get x_{k+1} when starting from x_k and using constant controls $u(t) = u_k$ for $t \in [kt_\Delta, (k+1)t_\Delta)$. Single-step methods only utilize the single current state x_k to compute the next state x_{k+1}, as opposed to multi-step methods that utilize M previous steps $x_{k-M+1},\ldots,x_k$ to compute the next state. In addition to explicit integration schemes, implicit schemes are used for stiff systems. Stiff systems exhibit both fast and slow dynamics. Runge-Kutta integration schemes can be used to vary the integration order for implicit and explicit schemes. Accounting also for implicit integration schemes, the dynamics can be denoted by the implicit equation $0 = \Psi(x_{k+1},x_k,u_k,z_k)$ which involves algebraic states $z_k \in \mathbb{R}^{n_z}$ as intermediate collocation variables or integration steps. The implicit equation may be solved as part of the MPC optimization problem.

After including the numerically favorable concepts of potential implicit integration functions, constraints, and multiple shooting, the final MPC optimization problem template used within this thesis is

$$\begin{aligned} \min_{\substack{u_0,\ldots,u_{N-1}\\ s_0,\ldots,s_N \\ z_0,\ldots,z_{N-1}}} \; & \sum_{k=0}^{N-1} \gamma^k \tilde{l}^{\mathrm{mpc}}(s_k,u_k) + \gamma^N \tilde{J}^{\mathrm{mpc}}(s_N), \\ \text{s.t.} \quad & s_0 = x, \; 0 \geq h_{\mathrm{t}}(s_N), \\ & 0 = \Psi(s_{k+1},s_k,u_k,z_k), \\ & 0 \geq h(s_k,u_k), \qquad k=0,\ldots,N-1. \end{aligned} \tag{2.25}$$

Different specialized optimization solvers are used depending on the classification of the MPC optimization problem (2.25) and the solution's requirements. If the structure of (2.25) corresponds to a quadratic program (QP), quadratic optimization solvers are used. A classification of QP solvers is beyond the scope of this work. It is assumed that convex QPs can be solved efficiently. If the structure of (2.25) corresponds to an nonlinear program (NLP), the sequential quadratic programming (SQP) algorithm iteratively solves a series of QPs that locally approximate the NLP. In some cases, it may be beneficial to reformulate certain nonconvexities of NLPs or nonconvex QPs to mixed-integer nonlinear programmings (MINLPs) or mixed-integer quadratic programs (MIQPs), respectively, by shifting major nonconvexities to integer variables as for example in [213] or our related papers [225, 227]. The reformulation allows mixed-integer optimization solvers to converge to reasonable or even global optimal solutions.

2.3.4 Sampling-Based Online Optimization

This section introduces a class of algorithms that aims at solving the MPC optimization problem (2.22) without the computation of derivatives. This thesis does not apply sampling-based algorithms, yet, for completeness, the basic idea and algorithmic directions are illuminated. The underlying paradigm is to sample, evaluate and rank multiple control trajectories $U^i = (u_0^i, \dots, u_{N-1}^i)$, with $i = 1, \dots, N_{\text{samp}}$. Thereafter, the first control $u_0^\star$ of the lowest-cost trajectory $U^\star$ is chosen as the policy output. Remarkably, only simulations, cost- and constraint evaluations of the model $F^{\text{smpc}}(x_k, u_k, w_k)$ that may also include randomly sampled variable $w_k \sim P_{\mathcal{W}}(\cdot|x_k, u_k)$ are required, which enables more complicated model formulations than in derivative-based algorithms. For example, discontinuous functions, random variables, or logical statements can be used as part of the forward simulation. However, sampling-based algorithms usually suffer from a particularly severe curse of dimensionality.

The straight-forward approach of random shooting [187, 205] independently samples full trajectories $U^1, \dots, U^{N_{\text{samp}}}$ from a distribution $\mathcal{D}_{\text{rs}}$ to obtain the lowest-cost trajectory $U^\star$ by

$$U^\star \in \underset{U \in \{U^1, \dots, U^{N_{\text{samp}}}\}}{\arg\min} \sum_{k=0}^{N-1} \gamma^k l^{\text{mpc}}(x_k, u_k) + \gamma^N J^{\text{mpc}}(x_N), \qquad \text{with} \begin{cases} x_0 = x, \\ x_{k+1} = F^{\text{mpc}}(x_k, u_k, w_k), \\ w_k \sim P_{\mathcal{W}}(\cdot|x_k, u_k). \end{cases} \tag{2.26}$$

While simple to implement and parallelize, this method is only applied to systems with a low number of inputs and short horizons.

The cross-entropy method samples trajectories similar to random shooting. However, the distribution from which the samples are drawn is refined in each iteration [152] based on the costs computed by the previous iteration. Often, Gaussian distributions or Gaussian mixture models are used as parameterized distributions.

A third sampling-based method is model predictive path integral (MPPI) control [136, 281, 282, 303] that considers systems that can be written in the continuous form as the n_w dimensional Wiener process

$$dx = \Big(f_x^{\mathrm{mppi}}(x) + f_u^{\mathrm{mppi}}(x)u\Big)dt + f_w^{\mathrm{mppi}}(x)dw,$$

with $f_x^{\mathrm{mppi}} : \mathcal{X} \to \mathbb{R}^{n_x}$, $f_u^{\mathrm{mppi}} : \mathcal{X} \to \mathbb{R}^{n_x} \times \mathbb{R}^{n_u}$ and $f_w^{\mathrm{mppi}} : \mathcal{X} \to \mathbb{R}^{n_x} \times \mathbb{R}^{n_w}$. The dynamics are nonlinear only in the states and separated into a nonlinearity related to controls f_u^{mppi} and the noise f_w^{mppi}. Similar to the cross-entropy method, MPPI updates the probability distribution from which actions are sampled for the specific model structure above. Despite an involved derivation of the exact sampling strategy [136, 281, 282], the resulting algorithm is straightforward to implement which did not promote the development of dedicated software, such as in derivative-based optimization. MPPI uses a weighted average to update the mean of this distribution where the weights are based on the associated costs.

2.3.5 Imitation Learning

The IL problem setting involves an expert policy $\tilde{\pi}^*(x)$, which could be a human demonstrator or a sophisticated control algorithm that is burdened by its computational complexity. The objective now is to acquire a policy $\pi^\theta(x)$ that approximates the expert policy.

Behavior Cloning

A simple algorithm is behavior cloning where, first, a data set of $i = 1, \dots, N_{\mathrm{exp}}$ expert actions $u_i = \tilde{\pi}^*(x_i)$ are evaluated by the expert policy for random states x_i that are sampled from a distribution $x_i \sim \mathcal{S}$ with the a finite support over the feasible states, i.e., $\mathrm{supp}(\mathcal{S}) = \mathcal{X}$. The samples are used to train a NN, which is formulated as the supervised learning problem

$$\min_\theta \sum_{i=1}^{N_{\mathrm{exp}}} \Big|\Big|\pi^\theta(x_i) - u_i\Big|\Big|_2^2.$$

Although behavior cloning is straightforward to implement, the performance is often rather low, e.g., see the comparison in [172]. One possible reason is that the distribution $\mathcal{S}$ is often chosen uniformly over the state space. When the number of states rises, the sampling becomes inefficient, as the probability of sampling the actual relevant states that are encountered during closed-loop control becomes exponentially small.

Let $\mathcal{S}_k^{\pi}$ be the distribution of states at step k when following policy π for k steps and let

$$\mathcal{S}^{\pi} = \frac{1}{N} \sum_{k=1}^{N} \mathcal{S}_k^{\pi}$$

be the average distribution if policy π is followed for N steps. One way to mitigate the problem of inefficient sampling would be to obtain the samples from a distribution $\mathcal{S}^{\tilde{\pi}^\star}$ that is induced by following the expert policy to obtain states. However, a small approximation error of π^{θ} would accumulate and shift the distribution $\mathcal{S}_k^{\pi^{\theta}}$ away from the training distribution $\mathcal{S}^{\tilde{\pi}^\star}$ [44, 240], also known as covariate shift [239].

Dataset Aggregation

A straightforward algorithm that mitigates the distribution shift is dataset aggregation (DAgger) [240]. The DAgger algorithm alternates between sampling from the distribution $\mathcal{S}^{\tilde{\pi}^\star}$ induced by the expert and a distribution $\mathcal{S}^{\pi^{\theta_i}}$ induced by the currently learned imitation policy π^{θ_i}, where θ_i are the current parameters at iteration i. A disadvantage of DAgger is the required online querying of the expert and policy updates in each iteration, which may be ineffective for large NNs. Moreover, a new data set needs to be created in each overall iteration.

Inverse Reinforcement Learning

In real-world settings, it may be the case that the expert policy $\tilde{\pi}^\star$ is suboptimal. For example, humans may introduce errors due to various reasons. An alternative strategy is to learn the expert's objective under a known environment model. Thereafter, the aim is to imitate the expert by solving an optimization or RL problem based on the learned objective function. Whether methods from numerical optimal control or RL are used determines the wording of inverse RL or inverse optimal control.

A challenge with inverse RL is the ambiguity of the optimal cost function, i.e., equal behavior can be achieved by multiple cost functions [192]. Another challenge is the inner loop of solving an RL or numerical optimization problem. In fact, the structure of inverse reinforcement learning (IRL) often resembles a bi-level programming structure, where the decision variables in the higher-level optimization problem are constrained to be the solution of a lower-level optimization problem. In this thesis, an inverse optimal control algorithm is proposed in Sect. 7.1 and the related work [226] to estimate trajectories assuming a known model.

Another inverse RL algorithm that does not utilize a model is generative adversarial inverse reinforcement learning (GAIL). GAIL borrows ideas from generative adversarial networks (GANs) [103], where trained generative NNs aim to generate data that is similar to a target distribution, i.e., the expert policy distribution, and a trained discriminator NN tries to separate samples from the generative non-expert distribution from samples of the expert distribution.

Recent Advances of Imitation Learning

A state-of-the-art method [44] targets the distribution shift by assuming a certain controlled imitation policy that stabilizes the policy around expert demonstrations. Therefore, the authors in [44] show how supervised learning can be used again to outperform other interactive approaches. Furthermore, multivariate distributions are treated by a generative diffusion model [264, 43].

2.3.6 Reinforcement Learning and Dynamic Programming

RL summarizes a class of methods that aim to iteratively learn a policy π by interacting with the environment. The usually unknown system model is formulated by

$$x_{k+1} \sim P_{\mathcal{X}}(\cdot|x_k, u_k),$$

which is a reformulation of (2.18) but aligned with the common RL literature [274]. Additionally, in the RL literature a stochastic OCP is referred to as Markov decision process (MDP), summarizing the problem by the tuple $\left(\mathcal{X}, \mathcal{U}, P_{\mathcal{X}}, l, \gamma\right)$.

Within RL algorithms, the optimization over the action and state space is required in multiple algorithms. For MDPs with finite action and state dimensions, the optimization over variables can be performed by enumerating the decision variables and evaluating the respective cost to find the lowest value, such as in tabular Q-learning [274]. For continuous state and action spaces, RL is more involved. This thesis focuses on continuous infinite action and state spaces $\mathcal{U}$ and $\mathcal{X}$, respectively. The basic concept for continuous action and state spaces is to use parameterized function approximators such as NNs that generalize to unseen data.

The following concepts are first introduced superficially without discussing the problematic continuous state and action space. These concepts directly apply to discrete state and action spaces. To scale RL to high dimensional or continuous state and action spaces, function approximations are used. A discussion on function approximations is added, along with popular algorithms used within this thesis, i.e., proximal policy optimization (PPO) and soft actor critic (SAC).

Note that for every MDP, a deterministic optimal policy exists under barely restrictive assumptions [274]. So far, the policy $\pi : \mathcal{X} \rightarrow \mathcal{U}$ was deterministic, mapping from states x to actions u. However, in many RL algorithms the policy is defined stochastic as a parameterized distribution $\pi : \mathcal{X} \rightarrow \mathcal{P}(\mathcal{U})$, where $\mathcal{P}(\mathcal{U})$ is a set of probability measures on the control space $\mathcal{U}$. The conditional probability density for the policy in state x is $\pi(u|x)$. The following uses deterministic and stochastic policies depending on the context.

As a performance criterion, typically the discounted cost for the discrete-time case similar to (2.20) is chosen, which defines the value function for the deterministic policy π as

$$J^{\pi}(x) := \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k l\big(x_k, \pi(x_k)\big) \,\middle|\, x_0 = x,\ x_{k+1} \sim P_{\mathcal{X}}(\cdot|x_k, \pi(x_k))\right], \tag{2.27}$$

where the notation $\mathbb{E}\left[g(x,u,x^+) \mid x^+ \sim P_{\mathcal{X}}(\cdot|x,u)\right]$ denotes the expectation of a function $g(x,u,x^+)$ over the conditional distribution of the environment model $P_{\mathcal{X}}(\cdot|x,u)$ for consecutive states x^+. Such as most online optimal control approaches, RL aims to find policies to achieve low values of the value function $J^{\pi}(x)$ for all states $x \in \mathcal{X}$. Usually, the policy is learned by interacting with the environment and observing costs $l(x,u)$ obtained when applying a deterministic control $u = \pi(x)$ or, possibly, sampled control $u \sim \pi(\cdot \mid x)$ at state x.

A particular form of the value function (2.27) is often used in RL, where the first control u_0 is set to a specific value u and all other controls $u_{k>0}$ follow a particular policy π. This function is referred to as action-value function $Q^{\pi} : \mathcal{X} \times \mathcal{U} \rightarrow \mathbb{R}$ and can be written as

$$Q^{\pi}(x,u) := \\ \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k l(x_k, u_k) \,\middle|\, x_0 = x, u_0 = u, u_{k>0} = \pi(x_k),\ x_{k+1} \sim P_{\mathcal{X}}(\cdot|x_k, u_k)\right]. \tag{2.28}$$

Let Π be the function space of all possible policies. Then, the optimal value function is

$$J^{\star}(x) := J^{\pi^{\star}}(x) = \min_{\pi \in \Pi} J^{\pi}(x),$$

which holds for all states x, cf., [35]. The optimal action-value function $Q^{\star}(x,u)$ and the optimal value function are related by

$$J^{\star}(x) = \min_{u \in \mathcal{U}} Q^{\star}(x,u),$$

and

$$Q^{\star}(x,u) = l(x,u) + \gamma\, \mathbb{E}\left[J^{\star}(x^+) \middle| x^+ \sim P_{\mathcal{X}}(\cdot|x,u)\right].$$

A policy minimizing $Q^{\pi}(x,u)$ w.r.t. u in a given state x is called greedy, and, for $Q^{\star}(x,u)$ it is optimal [277]. Therefore, it is sufficient to know $Q^{\star}(x,u)$ to derive an optimal policy. Likewise, when knowing the optimal value function $J^\star(x)$, the cost $l(x,u)$ and the model $P_\mathcal{X}(\cdot|x,u)$, the optimal policy can be found by

$$\pi^\star(x) \in \arg\min_{u\in\mathcal{U}} \mathbb{E}\left[l(x,u) + J^\star(x^+)\middle| x^+ \sim P_\mathcal{X}(\cdot|x,u)\right]. \tag{2.29}$$

Dynamic Programming for Finite Horizon Problems

A fundamental algorithm for finding the optimal policy and value function is dynamic programming (DP) [28, 37], which was adapted to various variants across computer science, mathematics, and engineering. The basic principle of DP is to break down a complex problem into simpler overlapping subproblems, which are solved iteratively. In general, DP can be applied to many different problems. Sequential decision-making problems such as the considered MDP are an important subclass of problems where DP can be applied [69]. In the following, the DP algorithm is stated first for a finite horizon problem, where the MDP terminates after T steps, and a value function $J_k(x)$ is associated with each time step k. At the final step $k = T$, the value function $J_T(x)$ is usually given as part of the problem objective. For example, a certain state $\bar{x}$ should be reached at step $k = T$ which could be expressed by a penalty $J_T(x) = ||x - \bar{x}||^2_{W,2}$ weighted by the matrix $W \in \mathbb{R}^{n_x \times n_x}$. No discount factor is usually considered for finite horizon MDPs, i.e., $\gamma = 1$. As stated above, DP now breaks down the MDP over the full horizon T into simpler subproblems. Particularly, the Bellman equation states that

$$J_k^\star(x_k) = \min_{u_k\in\mathcal{U}} \mathbb{E}\left[l(x_k,u_k) + J_{k+1}^\star(x_{k+1})\middle| x_{k+1} \sim P_\mathcal{X}(\cdot|x_k,u_k)\right]. \tag{2.30}$$

This equation relates the value functions $J_k^\star$, the transition function $P_\mathcal{X}$, and the running cost l between two consecutive steps k and $k+1$. It provides a means to recursively solve the MDP by starting at the final given value function $J_T(x)$ and solving the equation recursively backwards from $k = T-1, \dots, 0$ to get optimal controls $u_{T-1}, \dots, u_0$ and value functions $J_{T-1}^\star, \dots, J_0^\star$.

Dynamic Programming for Infinite Horizon Problems

For infinite horizon problem, the Bellman equation (2.30) applies as a fixed point equation

$$J^\star(x) = \min_{u\in\mathcal{U}} \mathbb{E}\left[l(x,u) + \gamma J^\star(x^+)\middle| x^+ \sim P_\mathcal{X}(\cdot|x,u)\right].$$

The fixed point equation can not be applied recursively since no terminal state and value function are given. Furthermore, there is no sequence of value functions. To find such a value function, the Equation (2.30) can be iteratively applied to the value function by solving

$$J_{i+1}(x) = \min_{u \in \mathcal{U}} \mathbb{E}\left[l(x,u) + \gamma J_i(x^+) \middle| x^+ \sim P_{\mathcal{X}}(\cdot|x,u) \right]$$

in order to generate a series of value function $J_0(x), J_1(x), \ldots$ that converge to the optimal solution $J^\star(x)$, starting from an arbitrary initial value function $J_0(x)$ [277]. This method is called *value-iteration.* From the optimal value function, the optimal policy can be obtained by (2.29).

Another method to derive the optimal policy and value function is named *policy iteration* [35]. Policy iteration iteratively switches between a policy evaluation and a policy improvement step. Consider any given policy $\pi_0 = \pi$. Similar to value iteration, the policy evaluation

$$J^\pi(x) = \mathbb{E}\left[l(x,u) + \gamma J^\pi(x^+) \middle| x^+ \sim P_{\mathcal{X}}(\cdot|x,u) \right]$$

computes the expectation of bootstrapped costs to obtain a value functions J^π under a policy π. In the policy improvement step, the current policy π_i is improved based on the current value function $J^{\pi_i}(x)$ by Bellman's equation

$$\pi_{i+1}(x) \in \arg\min_{u \in \mathcal{U}} \mathbb{E}\left[l(x,u) + \gamma J^{\pi_i}(x^+) \middle| x^+ \sim P_{\mathcal{X}}(\cdot|x,u) \right].$$

It can be shown that the final policy converges to the optimal policy $\pi^\star(x)$ by iterating the policy evaluation and improvement. Policy iteration can be seen as Newton's method for solving Bellman's equation and converges in less iterations than the value iteration [35]. However, each value iteration is usually much faster per iteration. The authors in [208] provided a modified policy iteration algorithm that trades off the value function improvement per iteration and the iteration time. Many related algorithms exist that aim for a speedup of the presented DP algorithms, e.g., state aggregation [36] or multi-grid methods [243].

Value Function Prediction

In the following, two concepts for estimating the value function at a particular state x are introduced. To estimate the value $J^\pi(x)$ of a state x under policy π of an MDP, *Monte Carlo sampling* applies a policy π repeatedly, always starting from the same first state x_0, in order to accumulate the received costs $l_k = l\big(x_k, \pi(x_k)\big)$ in each state $x_k \sim P_{\mathcal{X}}\big(\cdot|x_{k-1}, \pi(x_{k-1})\big)$ to obtain sampled returns

$$G(x) = l_0 + \gamma l_1 + \gamma^2 l_2 + \ldots \Big| x_k \sim P_{\mathcal{X}}\big(\cdot|x_{k-1}, \pi(x_{k-1})\big),\ x_0 = x,\ l_k = l\big(x_k, \pi(x_k)\big).$$

By averaging all received returns G_j for each state, where the index j enumerates the so-called roll-outs, Monte Carlo methods approximate the empirical mean of the value function in (2.27). These roll-outs terminate only for finite MDPs. Nevertheless, the sum over exponentially discounted rewards converges quickly for infinite horizon MDPs. Similar to value iteration, Monte Carlo methods update a value function estimation J_j^π iteratively with iteration $j = 0, 1, \ldots$. By utilizing a parameter $\alpha \in (0, 1)$, which can be determined by methods described in [274], the empirical mean can be incrementally updated by

$$J_{j+1}^\pi(x) = J_j^\pi(x) + \alpha\Big(G(x) - J_j^\pi(x)\Big). \tag{2.31}$$

Monte Carlo methods do not require a model, which makes them appealing. However, samples generated by Monte Carlo sampling exhibit a high variance and resetting a real system to a certain state may be impossible. Additionally, it is often hard and expensive to run experiments multiple times. A detailed overview of Monte Carlo methods that can also be used for off-policy, i.e., following one policy but estimating the value of another, can be found in [274].

An additional concept is temporal differences [273], where the empirical mean $G(x)$ of (2.31) is replaced by the *bootstrapped* current estimate of the value function. Bootstrapping refers to updating the value of a state based on the estimated values of subsequent states instead of waiting for the final outcome. Thereby, bootstrapping blends real experience with current estimates. Different variants of temporal difference learning exist. For the basic TD(0) variant, which predicts one step ahead, the update is defined as

$$J_{j+1}^\pi(x) = J_j^\pi(x) + \alpha\Big(l\big(x, \pi(x)\big) + \gamma J_j^\pi(x^+) - J_j^\pi(x)\Big),$$

where $x^+ \sim P_\mathcal{X}(\cdot|x, u)$. Note that the policy is often stochastic, which requires sampling from the policy distribution by $u \sim \pi(\cdot|x)$. Compared to Monte Carlo sampling, the temporal difference method has a lower variance but a higher bias. Temporal differences combine ideas from Monte Carlo sampling with DP. Like Monte Carlo sampling, it does not assume a known distribution of the environment model $P_\mathcal{X}$. It rather samples the transitions. Similar to DP, temporal differences use an iteratively updated estimate of the true value function.

Q-learning

Along the paradigms of DP, the temporal difference prediction could also be used to iteratively improve the action value function by

$$Q_{j+1}(x, u) = Q_j(x, u) + \alpha\left(l(x, u) + \gamma \min_{u^+ \in \mathcal{U}} Q_j(x^+, u^+) - Q_j(x, u)\right),$$

where the control u can, but is not required to, be taken from the minimum of the previous action-value function $Q_j(x,u)$ over controls u and the next state x^+ drawn from the environment. Since the action u can be taken arbitrarily from $\mathcal{U}$, the method is also referred to as the off-policy method. Under some assumptions [274], it can be shown that $Q_j(x,u)$ converge to $Q^\star(x,u)$ for $j \to \infty$.

Function Approximation

As introduced at the beginning of this section, continuous action and control spaces $\mathcal{U}$ and $\mathcal{X}$, respectively, make it impossible to apply the introduced concepts for each of the uncountable states and controls. A possibility to mitigate the problem uses parameterized functions, such as NNs, for the policies $\hat{\pi}(x;\theta)$ and value functions $\hat{J}(x;\theta)$ and $\hat{Q}(x,u;\theta)$, with parameters $\theta \in \mathbb{R}^{n_\theta}$. The hope is that the functions can be obtained from a finite number of transition samples but generalize to all relevant unseen states.

Remark 2.3.1. *To denote the dependence of a function $f(x,y)$ of a variable x and a variable y, where y serves as a parameter, the notation $f(x;y)$ and, likewise, $f^y(x)$ are used.*

To measure the accuracy of how well a parameterized function $\hat{J}(x;\theta)$ approximates the "true" value function $J^\pi(x;\theta)$, a loss

$$\mathcal{L}_J(\theta) = \int_{x\in\mathcal{X}} \mu(x)\big(J^\pi(x) - \hat{J}^\pi(x;\theta)\big)^2 dx$$

can be formulated using the state visitation probability density $\mu : \mathcal{X} \to \mathbb{R}$. When the policy is optimized during interactions with the environment, the state visitation distribution can be assumed to correspond to $\mu(x)$ [274]. Therefore, the loss $\mathcal{L}_J$ can be minimized by stochastic gradient descent on the samples occurring during interactions by the weight update

$$\theta_{j+1} = \theta_j + \alpha\big(J^\pi(x_j) - \hat{J}^\pi(x_j;\theta_j)\big)\nabla_\theta \hat{J}^\pi(x_j;\theta_j)$$

for closed-loop iterations $j = 1, 2, \ldots$. The true value function $J^\pi(x_j)$ is usually not available. However, it can be shown that an unbiased estimator of $J^\pi(x_j)$, such as the Monte Carlo estimate $G(x)$, converges to the true value function [274]. A bootstrapped value function estimator, such as TD(0), cannot obtain such guarantees since the bootstrapping introduces an error. However, it is still used within the so-called semi-gradient method with the parameter update

$$\theta_{j+1} = \theta_j + \alpha\big(\underbrace{l(x_j,\pi(x_j)) + \gamma\hat{J}^\pi(x_{j+1};\theta_j)}_{\text{bootstrapped approximation of } J^\pi(x_j)} - \hat{J}^\pi(x_j;\theta_j)\big)\nabla_\theta \hat{J}^\pi(x_j;\theta_j),$$

which ignore the gradient of the bootstrapped value function approximation.

The above approaches use stochastic gradient descent to estimate a value function. Stochastic gradient descent could be combined straightforwardly with on-policy Q-learning to derive an approximation $\hat{Q}(x, u; \theta)$ of optimal action-value function. Moreover, off-policy variants require a particular consideration of the distribution shift induced by the policy used to interact with the environment. Notably, the combination of off-policy learning, function approximation, and bootstrapping is known to cause convergence difficulties, cf., “the deadly triad” [274].

Policy Gradient Methods

Policy gradient methods directly optimize a stochastic policy $\pi(u|x;\psi)$ or deterministic policy $u = \pi(x;\psi)$, parameterized by $\psi \in \mathbb{R}^{n_\psi}$. For stochastic policies, the goal is to minimize the objective

$$\min_{\psi} J^{\pi_\psi}(\psi), \tag{2.32}$$

where the value function can be expressed in terms of probability distributions for the policy and the environment as

$$J^{\pi_\psi}(\psi) = \int_{x\in\mathcal{X}} \mu^\pi(x) \int_{u\in\mathcal{U}} \pi(u, x; \psi) l(x, u) \mathrm{d}u\mathrm{d}x = \mathbb{E}_{\substack{u\sim\pi(\cdot|x;\psi),\\ x\sim\mu^\pi}} \left[l(x, u)\right].$$

The expectation is computed over $\mu^\pi(x)$, which is the discounted state distribution under π [258]. Particularly, the discounted state distribution is

$$\mu^\pi(x) = \int_{\xi\in\mathcal{X}} \sum_{k=1}^{\infty} \gamma^{k-1} \mathcal{P}_{\mathcal{X},1}(x) \mathcal{P}_{\mathcal{X}}(\xi \to x, k, \pi) dx,$$

with the distribution $\mathcal{P}_{\mathcal{X},1}(x)$ for the initial state and the probability density $\mathcal{P}_{\mathcal{X}}(\xi \to x, k, \pi)$ at state x after transitions for k steps from state $\xi \in \mathcal{X}$. Remarkably, this state distribution violates some basic properties of probability distributions. The optimization problem in (2.32) is a complicated formulation since it involves the influence of the parameters ψ on the stochastic policy, which, again, influences the next state obtained from the stochastic environment.

The policy gradient theorem [275] is utilized to get an expression for the stochastic gradient

$$\nabla_\psi J^{\pi_\psi}(\psi) = -\,\mathbb{E}_{\substack{u\sim\pi(\cdot|x;\psi),\\ x\sim\mu^\pi}} \left[Q^{\pi_\psi}(x, u) \nabla_\psi \log \pi(u \mid x; \psi)\right]. \tag{2.33}$$

This expression of the policy gradient allows one to directly use control samples $u_j \sim \pi(\cdot|x;\psi)$ and transition samples $x_j \sim \mu^\pi$, that are obtained when applying the policy $\pi(u|x;\psi)$ to the environment, to get a parameter update

$$\psi_{j+1} = \psi_j - \alpha Q^{\pi_\psi}(x_j, u_j) \nabla_\psi \log \pi(u_j \mid x_j; \psi).$$

The action value function $Q^{\pi_\psi}(x_j, u_j)$ is usually approximated. For example, a Monte Carlo roll-out can be used, resulting in the particular REINFORCE algorithm [305]. Monte Carlo roll-outs do not require additional parameters. However, also parameterized action value functions $Q^{\theta}(x, u)$ are used to approximate $Q^{\pi_\psi}(x, u)$, which result in the so-called actor-critic algorithms.

The policy gradients can also be computed for deterministic policies, which is, in fact, more sample efficient but requires a particular exploration strategy [258]. A wide variety of algorithms builds on the main principles described above. The main distinctions are whether they use a stochastic policy, sample online or offline, and how the critic is approximated. Often additional, less fundamental, modifications make them powerful in practice, e.g., experience replay [274], smoothing of value function [98] or using two action value functions [119] to overcome the so-called overestimation bias. Two particular variants used in this thesis are explained briefly, i.e., SAC and PPO.

Soft Actor Critic

The soft actor critic (SAC) algorithm [117] is an off-policy algorithm that utilizes a stochastic policy. A core feature of SAC is a modified cost function $l'(x, u; \theta)$ that is based on the maximum entropy framework for RL, cf. [129, 327]. Particularly the cost

$$l'(x, u; \theta) = l(x, u) + \omega H\left(\pi(\cdot|x; \theta)\right)$$

with weight $\omega \in \mathbb{R}_{\geq 0}$ is used instead of the running cost $l(x, u)$. The entropy

$$H\big(\pi(\cdot|x; \theta)\big) = \mathbb{E}\left[\log \pi(\cdot|x; \theta)\right]$$

is a measure of the information or the randomness of a probability distribution, and therefore, promoting a high entropy favors the exploration. Although the cost function was changed in SAC, it can be shown that by specific algorithm extensions, the optimal value function can be recovered [116]. SAC improves the sample efficiency and leads to a more robust training, i.e., it is less brittle concerning hyper-parameter tuning [117].

Proximal Policy Optimization

The proximal policy optimization (PPO) algorithm [250] is an on-policy RL algorithm to improve a stochastic policy. A key feature of PPO is its ability to update the weights of a parameterized stochastic policy and a parameterized value function by a batch of sampled trajectories. Therefore, PPO can utilize multiple parallel environment simulations and speed up the training. PPO extends on ideas of the *trust region method* [249], where a surrogate loss regularizes the parameter update of the policy by constraining it with the parameters of the previous policy. Like SAC, PPO is less sensitive to hyperparameter tuning than trust region methods.

Chapter 3

Vehicle Models for Motion Planning

This chapter introduces relevant vehicle models for motion planning and simulation as ordinary differential equations (ODEs) in continuous time. Moreover, several obstacle avoidance formulations are introduced.

The remainder of this chapter is outlined as follows. First, in Sect. 3.1, the double-track model is stated as the highest-fidelity model used for simulation within this thesis. Consecutively, the model fidelity is reduced via single-track models in Sect. 3.2 and 3.3 towards a point-mass model in Sect. 3.4. Next, in Sect. 3.5, general vehicle components such as tire modeling, steering models, and powertrain concepts are introduced on a high level. The models are compared in Sect. 3.6. In Sect. 3.7, the projection of the Cartesian vehicle model on a reference curve is elaborated in detail. The final Sect. 3.8 of this chapter focuses on collision avoidance formulations.

The illustrated models differ in fidelity, i.e., how accurately they describe the actual physical motion, and play a pivotal role in accurately representing real-world dynamics. Similarly to [13, 318], four distinct vehicle model categories – a point-mass, kinematic single-track, dynamic single-track, and double-track model – are compared, highlighting their respective advantages, limitations, and applications in optimization-based motion planning.

Above all, the classification of vehicle models to these proposed categories is common but does not precisely define the models. Instead, a conceptual group of models is considered for each classification. For example, this classification does not specify which tire force model is used or how the steering dynamics are chosen. Hence, additional classifiers may be added to the model description, such as in [318]. In general, vehicle dynamics modeling can become arbitrarily complex and is not the main topic of this thesis. Therefore, the classification relates to the main characteristics relevant to this work, and simplifying assumptions are made on the categorization.

Using higher fidelity vehicle models as dynamic functions within optimization problems comes at the cost of more states and usually increases nonlinearity. The point-mass model, higher-order linear models, e.g., [104], or a linearization around a set point allows for a linear formulation of the dynamics and, therefore, to stay in the realm of quadratic programs (QPs). Solving highly nonlinear optimization problems with a large number of states comes with an increased computational burden and requirements for the optimization problem solver. Suppose the online computation time of the controller or planner is the limiting factor of the sampling frequency on the embedded platform. In that case, the resulting lower sampling frequency may yield worse closed-loop performance, as shown, e.g., in [153].

Moreover, higher fidelity models require more parameters. Obtaining some of these model parameters may be highly nontrivial and subject to expensive system identification procedures.

The following introduction of vehicle models uses location index identifiers $* \in \{\mathrm{r}, \mathrm{f}\}$ to refer to either the rear or front location of a vehicle and, likewise, $\bullet \in \{\mathrm{rl}, \mathrm{rr}, \mathrm{fl}, \mathrm{fr}\}$ to refer to the rear left, rear right, front left or front right location-related variable. For instance, the tire velocities are referred to as $\omega_{\mathrm{t},\bullet}$ to reference each of the possible velocities in the text for the four wheels.

3.1 Double-Track Model

As the double-track model is only used for simulation in this thesis and not within any of the contributed planners or controllers, it is only described conceptually. The authors in [13] provide a comprehensive overview of the double-track model formulation used in this thesis.

The double-track model is the lowest-fidelity model that uses all four wheels to describe the physical vehicle motion. The modeling of all four wheels allows for considering different load distributions during dynamic maneuvers, significantly affecting the transmitted force between the tires and the road. For example, the load on the outer wheels is higher in curves due to the centrifugal force. Additionally, all four wheels' torques and steering angles may be actuated individually to achieve high performance in critical driving scenarios [319]. Even if the wheels can not be controlled individually, most vehicles have different steering angles on the wheels to achieve specific handling properties. For example, the *Ackermann steering* uses different front steering wheel angles in order to have no longitudinal slip caused by cornering [113, p. 88]. In a more straightforward planar form, the double-track model assumes a rigid chassis, as in [96] or [319]. More modeling details can be added by including the spring and dampers of each wheel.

3.2 Dynamic Single-Track Model

The dynamic single-track vehicle model is a medium-fidelity vehicle model that captures a vehicle's dynamic motion, such as drifting and tire force relations, but neglects lateral load transfer and spring-damper dynamics. Nevertheless, it offers a compromise for control systems due to its fewer states, moderate nonlinearities, and often sufficient modeling of important physical aspects such as tire forces.

Three different coordinate frames are used to describe individual parts of the vehicle, i.e., the Cartesian "earth" global frame (C) with unit vectors ($x_\mathrm{e} \in \mathbb{R}^2, y_\mathrm{e} \in \mathbb{R}^2$), the vehicle frame (V) with unit vectors ($x_\mathrm{v} \in \mathbb{R}^2, y_\mathrm{v} \in \mathbb{R}^2$) and the tire frame (T) with unit vectors ($x_\mathrm{t} \in \mathbb{R}^2, y_\mathrm{t} \in \mathbb{R}^2$). An overview of different symbols is given in Fig. 3.1. In fact, the velocity in the longitudinal direction in the vehicle coordinate frame at the center of gravity (CG) is $v_x \in \mathbb{R}$ and $v_y \in \mathbb{R}$, respectively, for the lateral vehicle frame direction. The vehicle velocity vector in the Cartesian coordinate frame at the CG is denoted as $v \in \mathbb{R}^2$, at the front tire as $v_\mathrm{f} \in \mathbb{R}^2$, and at the rear tire as $v_\mathrm{r} \in \mathbb{R}^2$. The heading angle of the vehicle in the Cartesian frame is denoted as φ and the related yaw rate as ω. The deviation of the vehicle body frame heading angle to the angle of the actual velocity vector is denoted by the slip angles for the respective position, with β at the CG and $\beta_{\{f,r\}}$ at the two tires.

The vehicle model comprises a minimum of six states, with three states

$$x_\mathrm{p} = \begin{bmatrix} p_x & p_y & \varphi \end{bmatrix}^\top \in \mathbb{R}^3$$

that describe Cartesian position (p_x, p_y) and yaw or heading angle φ and three states related to their velocities

$$x_{\dot{\mathrm{p}}} = \begin{bmatrix} v_x & v_y & \omega \end{bmatrix}^\top \in \mathbb{R}^3,$$

with the vehicle frame velocity components (v_x, v_y) and the yaw rate φ. The state vector is

$$x = \begin{bmatrix} x_\mathrm{p}^\top & x_{\dot{\mathrm{p}}}^\top \end{bmatrix}^\top . \tag{3.1}$$

In automotive systems, a cascade of low-level controllers for the subsystems involved in the overall vehicle actuation is often used, and therefore, the particular choice of controls varies. Here, it is assumed that the revolution speed $\omega_{\mathrm{t},*}$ of the wheels are control inputs. Alternatively, the drive and brake torque acting on the two modeled wheels may be used as four independent controls. In this case, the tires are modeled as subsystems that introduce new states for the revolution speed. Another common alternative would be to directly take the longitudinal acceleration force acting on the vehicle frame as an input, e.g., the vehicle model in [293]. This ignores the tire-road interaction in the longitudinal direction. In addition to the tire revolution speeds $\omega_{\mathrm{t},*}$, the steering angle δ is used as control. Therefore, the input vector reads as

$$u = \begin{bmatrix} \omega_{\mathrm{t,f}} & \omega_{\mathrm{t,r}} & \delta \end{bmatrix}^\top . \tag{3.2}$$

The vehicle parameters include its mass m, its inertia I_z at the CG, the wheel base l, the front and rear wheelbases $l_\mathrm{r}, l_\mathrm{f}$ relative to the CG, with $l = l_\mathrm{r} + l_\mathrm{f}$, and tire parameters which are described in Sect 3.5. Figure 3.1 shows the velocity and force vectors related to the vehicle model. The tire forces within the tire coordinate frame are denoted as $F_{\mathrm{l,f}}(v_x, v_y, \omega, \omega_{\mathrm{t,f}}, \delta)$ and $F_{\mathrm{l,r}}(v_x, v_y, \omega, \omega_{\mathrm{t,r}})$ for the longitudinal tire forces in the y_t direction, and $F_{\mathrm{c,f}}(v_x, v_y, \omega, \delta)$ and $F_{\mathrm{c,r}}(v_x, v_y, \omega)$ for the lateral cornering tire forces in the x_t direction and are functions of several vehicle states and the controls.

The tire forces for the front wheel are projected onto the vehicle coordinate frame related to the steering angle δ by

$$\begin{aligned} F_{x,\mathrm{f}}(v_x, v_y, \omega, \omega_{\mathrm{t,f}}, \delta) &= F_{\mathrm{l,f}}(v_x, v_y, \omega, \omega_{\mathrm{t,f}}, \delta) \cos(\delta) + F_{\mathrm{c,f}}(v_x, v_y, \omega, \delta) \sin(\delta), \\ F_{y,\mathrm{f}}(v_x, v_y, \omega, \omega_{\mathrm{t,f}}, \delta) &= F_{\mathrm{l,f}}(v_x, v_y, \omega, \omega_{\mathrm{t,f}}, \delta) \sin(\delta) - F_{\mathrm{c,f}}(v_x, v_y, \omega, \delta) \cos(\delta), \end{aligned}$$

and for rear wheels without active steering, the tire forces act directly on the vehicle frame by

$$\begin{aligned} F_{x,\mathrm{r}}(v_x, v_y, \omega, \omega_{\mathrm{t,r}}) &= F_{\mathrm{l,r}}(v_x, v_y, \omega, \omega_{\mathrm{t,r}}), \\ F_{y,\mathrm{r}}(v_x, v_y, \omega) &= F_{\mathrm{c,r}}(v_x, v_y, \omega). \end{aligned}$$

From Newton's laws of motion the dynamics $\dot{x} = f(x, u)$ can be obtained as

$$\begin{aligned} \dot{p}_x &= v_x \cos\varphi - v_y \sin\varphi, \\ \dot{p}_y &= v_x \sin\varphi + v_y \cos\varphi, \\ \dot{\varphi} &= \omega, \\ m\dot{v}_x &= 2F_{x,\mathrm{f}}(\cdot) + 2F_{x,\mathrm{r}}(\cdot) + m\omega v_y + F_{\mathrm{res}}(v_x), \\ m\dot{v}_y &= 2F_{y,\mathrm{f}}(\cdot) + 2F_{y,\mathrm{r}}(\cdot) - m\omega v_x, \\ I_z\dot{\omega} &= 2F_{y,\mathrm{f}}(\cdot) l_\mathrm{f} - 2F_{y,\mathrm{r}}(\cdot) l_\mathrm{r}. \end{aligned}$$

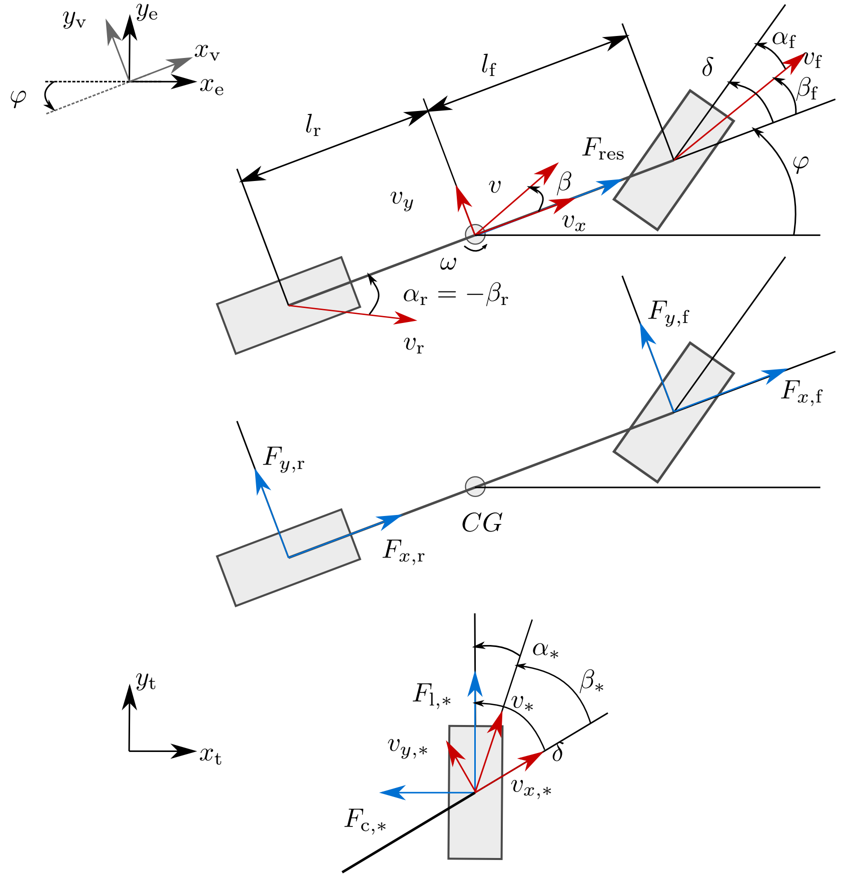

Figure 3.1: Dynamic single-track vehicle model in the Cartesian coordinate frame and a tire view in the tire coordinate frame.

The velocity states are expressed in the vehicle body frame (V), and the corresponding tire, friction, and drive forces are explained in Sect. 3.5. Since the single-track model only uses one tire per axle, the tire forces must be multiplied by 2. Particularly, this has to be considered within the tire and drive train models to relate the force to single tires. The forces from the tires $F_{\{x,y\},*}$ act on the vehicle body frame. As a simplification, all driving resistance forces $F_{\text{res}}(v_x)$ act on the CG and only into longitudinal vehicle coordinate direction.

3.3 Kinematic Single-Track Model

The kinematic single-track model and the dynamic counterpart use the same geometric model, i.e., the model uses two instead of four tires at the wheelbase distance l, see Fig. 3.2. However, the fundamental difference is the negligence of tire slips. Therefore, the longitudinal tire force can instantly and directly transmit to the road surface without considering the road friction parameters or tire characteristics. Moreover, irrespective of the lateral acceleration, the tires move kinematically in their heading direction with zero lateral slip. This clearly is a significant simplification, but, according to [206], the motion is described sufficiently well for lateral accelerations a_{lat} below a certain threshold of $a_{\text{lat}} \leq 0.5\mu g$, where g is the gravitational constant and μ is the road friction parameter. This limitation can be used in model-based planners and controllers as a constraint to obtain a reasonable performance, even at higher speeds. The authors in [151] used the kinematic single-track model even for a racing low-level controller in a miniature race track at the performance limits of the vehicle.

The kinematic single-track model uses a minimum of four states

$$x^\top = \begin{bmatrix} p_x & p_y & \varphi & v \end{bmatrix}.$$

A common choice of the reference point (p_x, p_y) is the rear axle since it simplifies the dynamic equations to

$$\dot{p}_x = v\cos(\varphi), \quad \dot{p}_y = v\sin(\varphi), \quad \dot{\varphi} = \frac{v}{l}\tan(\delta), \quad m\dot{v} = F_{\text{d}} + F_{\text{res}}.$$

Due to the assumption of a direct force transfer without losses between the tires and the road, the acceleration or braking force is not related to one specific tire and lumped to the force F_{d}. In addition, a resistance force F_{res} is modeled as a longitudinal force, similarly to the dynamic single-track model.

3.4 Point-Mass Model

The point-mass model can only describe the actual vehicle motion with stark limitations. It is usually used to plan a trajectory in the two-dimensional

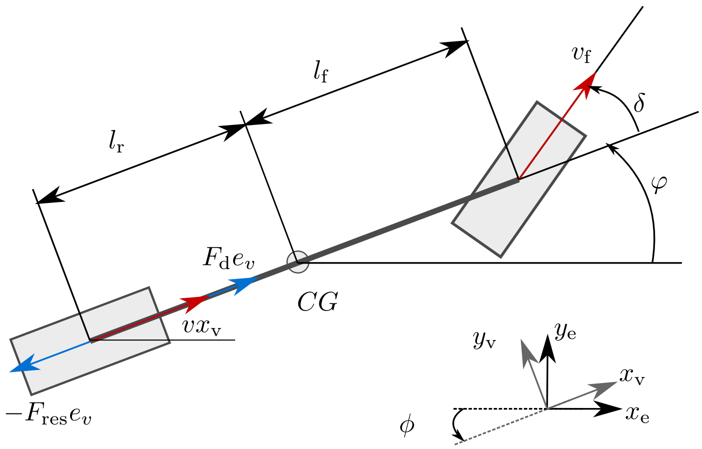

Figure 3.2: Kinematic vehicle model.

coordinate frame by a chain of integrators in both coordinates. A lower-level controller then tracks the trajectory. When deploying an integrator chain, the smoothness of the trajectory is expressed by its derivatives and can be constrained or weighted in the objective of an optimization-based planner. Usually, a chain of one to four integrators is used, with the inputs resembling the snap, the jerk [209], the acceleration [227] or the speed [213]. Here, the acceleration (a_x, a_y) is used as input and defines the following governing equations in the Cartesian coordinate frame

$$\dot{p}_x = v_x, \quad \dot{p}_y = v_y, \quad m\dot{v}_x = a_x, \quad m\dot{v}_y = a_y.$$

For optimization-based vehicle motion planning, the point-mass model is often used to provide a linear model formulation and, subsequently, a convex or quadratic optimization problem. A convex formulation of boundary constraints further requires a transformation into curvilinear coordinates, which are discussed in Sect. 3.7. Unfortunately, the projection into curvilinear coordinates makes the point-mass model highly nonlinear again. Mitigation strategies for a convexification are proposed rigorously in [86], or simplified in [213]. For highway motion planning, the curvature can be assumed to be close to zero for motion planning not at the handling limits [213].

3.5 Tires and Longitudinal Dynamics

In the following, the specifics of tires and longitudinal dynamics, i.e., steering, braking, powertrain modeling, and resistance forces, are summarized into a small but necessary discussion for this thesis. These components are typically separable parts of the general vehicle model.

3.5.1 Tire Models

Tires are typically modeled using a vertical load force F_z and a road friction coefficient μ, with $\mu \geq 0$. The vertical load may depend on vehicle states since the load transfer may distribute the load force unequally during dynamic maneuvers. However, often, the vertical force is assumed to be constant and computed by equally distributing the vehicle mass m among the tires to obtain for each tire $F_z = \frac{1}{4}mg$, where g is the gravitational acceleration. The external road friction parameter depends on the road condition and is close to $\mu = 1$ for dry roads and positive but closer to 0 for icy road conditions.

Tire models now aim to describe the forces transmitted from the road to the vehicle body as functions of the vehicle state and the road parameters. Omitting transient tire behavior leads to static functions of the states [113] that are usually split into a lateral (cornering) tire force component $F_{c,\bullet}$ and a longitudinal tire force component $F_{l,\bullet}$ in the tire coordinate system. This force can be computed for all four tires individually or approximated for the two tires on an axle in single-track models.

Many different tire models emerged, each with distinct advantages. Among the most popular models are the Pacejka model [198], the Dugoff model [84] or the Burckhart model [220]. While the Dugoff and Burckhart model focus on computational efficiency by reducing complexity, the Pacejka model is more flexible and capable of fitting a broad range of data. In this thesis, only the Pacejka model is used. The model is briefly explained in the following.

The Pacejka model describes the tire forces using parameterized functions designed to fit measured data. The tire model uses longitudinal slip variables

$$\sigma_\bullet = \frac{v_{\mathrm{t},\bullet} - \omega_{\mathrm{t},\bullet} r_{\mathrm{t},\bullet}}{v_{\mathrm{t},\bullet}},$$

for each tire, with the vehicle body velocity $v_{\mathrm{t},*}$ at a specific tire, the tire angular speed $\omega_{\mathrm{t},*}$ and the effective tire radius $r_{\mathrm{t},*}$. Moreover, lateral slip angles $\alpha_{\mathrm{t},*}$ are used that can be approximated as functions of the general vehicle state x [113]

with $* \in \{\mathrm{l}, \mathrm{r}\}$ by

$$\alpha_{*\mathrm{f}} = \delta_{*f} - \beta_{\mathrm{f}} = \delta - \arctan\left(\frac{v_y + l_{\mathrm{f}}\omega}{v_x}\right),$$

$$\alpha_{*\mathrm{r}} = -\beta_{\mathrm{r}} = -\arctan\left(\frac{v_y - l_{\mathrm{r}}\omega}{v_x}\right),$$

and define the angle between the tire heading and the speed vector at a specific tire. Here, it is assumed that the Pacejka model can provide a sufficient model for the tire forces with $F_{l,*} = f_{\mathrm{pac,l},*}(\sigma_*, F_z, \mu)$ and $F_{c,*} = f_{\mathrm{pac,c},*}(\alpha_*, F_z, \mu)$, with more details given in the original paper [198] and an exhaustive tire modeling description in [113].

The tire force functions $f_{\mathrm{pac,l},*}(\sigma_*, \cdot)$ and $f_{\mathrm{pac,l},*}(\alpha_*, \cdot)$ are nearly linear for values close to $\sigma_* = 0$ and $\alpha_* = 0$, respectively. At certain threshold values, these forces saturate, and the tires start drifting. However, the maximum transmitted force is achieved under usual conditions in the transition from the linear to the drifting phase. Due to the linearity around the origin that coincides with the usual operating range, these force functions are often linearized. When used within an optimization problem, the slip variables are constrained to the linear region, preventing the tire from drifting.

3.5.2 Longitudinal Forces

Longitudinal forces refer to separate sources that influence the longitudinal motion of a vehicle [317], which include the powertrain, resistance forces such as air drag, and the braking force.

The vehicle's powertrain comprises the motor and the transmission system, which allocates torque to the wheels. As these dynamics are highly complex, they are often considered a controlled subsystem with specific attributes provided by the manufacturer. It is usually assumed that the torque at the wheels can be provided instantaneously. However, it is limited to a particular nonconvex set in the space that is spanned by the rotational tire speeds ω_{t}, or, similarly, the vehicle speed v, and the torque [169, 64, 317]. The set constraining the possible torques describes an inverse dependency on vehicle speed, where at high velocities, less torque can be provided [169]. The engine type has implications for the transmission system. Most prominently, either electric, combustion or hybrid engines are used. It may also be necessary to control the gear in manual transmission systems, where each gear has its own characteristic torque allocation set. Besides feasibility, the vehicle speed and the allocated torque vastly determine the engine's efficiency, which may be used to optimize energy consumption.

The braking system has a much lower time constant than the allocation of a driving force. Therefore, the braking force is modeled as an instantaneous force. When no tire models are used, the braking force is modeled as a deceleration force on the vehicle frame.

External longitudinal forces $F_{\text{res}}(v)$ include friction forces, which are the rolling resistance and the air drag, modeled as a function of the vehicle velocity v by

$$F_{\text{res}}(v) = -c_{\text{roll}}\text{sign}(v) - c_{\text{drag}}v^2, \tag{3.6}$$

with friction coefficients c_{roll} and c_{drag}. Moreover, a term $-mg\sin(\alpha_{\text{road}})$ may be added to model the gravitational force on inclined or declined roads.

3.6 Comparison of Models

The introduced models, i.e., the double-track, dynamic single-track, kinematic single-track and point-mass model were compared in several works [113, 321, 311, 153, 206, 317]. While [113] compares the models mainly from the perspective of simulation, the others compare the models primarily when used as part of a planning and control algorithm. The author in [113] argues that the double-track model is preferable to single-track models besides academic purposes and, presumably, for simulation. The author motivates it by the inability of single-track models to account for different steering angles, among other inaccurate approximations.

The proposed models have different "degrees of freedom", which are the number of configuration states that define the configuration, i.e., the position and orientation, of the model completely. Velocities describe the rate of change of the configuration states over time but do not determine the configuration of the system. In general, a mechanical system with n degrees of freedom can be described by n second order differential equations in time, with $2n$ constraints for positions and velocities [101].

From the planning perspective, the double-track model may be necessary and proper if the particular influence of each tire is vital, e.g., in over-actuated electric vehicles at the handling limits [319]. Of course, this requires high computational resources.

For many applications, the dynamic single-track model may be sufficient even for higher speeds within a planner or controller [293, 167, 317]. It still allows for different loads, brake, and torque forces on the axles, as well as a yaw angle that is defined through dynamic relations rather than rigid kinematic relations. Depending on the particular tire model, the dynamics may still be highly nonlinear and, thus, pose challenges to optimization-based control techniques. Moreover, the parameters of the tire may be hard to identify.

The kinematic single-track model does not allow the distribution of the torques. Neither the braking forces between wheels nor tire slips are considered. However, the lateral acceleration may be constrained so that the resulting slips can be omitted [151]. Additionally, tire models are not applicable since only kinematic relations are considered. An advantage of this model is the few required parameters, few states, and the rather *mild* nonlinearities, which make it appealing for optimization-based planning and control. Optimization problems based on computationally simpler models may achieve higher sampling frequencies. This model may be particularly favorable for long-horizon planning at a higher hierarchical level that assumes a lower-level tracking and stabilizing controller.

Comparing the dynamic and kinematic single-track models for model predictive control (MPC) was repeatedly performed in the literature, e.g., [318, 311, 153, 206]. Most authors conclude that the dynamic single-track model is necessary when high lateral accelerations and low friction coefficients μ are encountered. The authors in [206] specifically name a decision criterion $a_{\text{lat}} \geq 0.5\mu g$ for the lateral force a_{lat}. The authors in [153] compare the two models by including the sampling times and find that the kinematic model with a higher sampling time may outperform the dynamic model. Notably, the authors use an Euler integration scheme with relatively large sampling times.

Obviously, the point-mass model is quite different from an actual vehicle in the real world, and the control inputs do not even match. However, the computed trajectory can provide a smooth trajectory with a constrained acceleration, which may be sufficient for higher-level planners [213, 181]. In autonomous driving (AD), finding a collision-free, feasible trajectory to a point-mass model may already be challenging. A considerable advantage of the point-mass model is its linear dynamical model, allowing linear MPC formulations or, as part of a combinatorial problem, for mixed-integer quadratic program (MIQP) formulations [213, 227]. A disadvantage for particularly low speeds is the missing kinematics and the heading angle.

Tab. 3.1 compares features and their applicability to different vehicle models relevant to this thesis.

3.7 Coordinate Frames for Motion Planning

The vehicle models introduced so far are formulated in the Cartesian coordinate frame (CCF). In the following, a coordinate frame is proposed that is specifically suited for vehicle motion planning and control on roads. An optimal control problem (OCP) for vehicle motion planning and control is considered, such as formulated in (1.1). Moreover, a structured environment is assumed, i.e., a detailed road map is available, and the task related to OCP (1.1) is restricted

	PM	KST	DST	DT
degrees of freedom	2	3	≥ 3	≥ 3
		Mechanical Influence		
steering kinematic, yaw motion	✗	✓	✓	✓
air drag	✓	✓	✓	✓
rolling resistance	✗	✓	✓	✓
powertrain model	✗	(✓)	✓	✓
lateral forces	✓	✗	✓	✓
longitudinal tire models	✗	(✓)	✓	✓
lateral tire models	✗	✗	✓	✓
different steering angles on axles	✗	✗	✗	✓
chamber angles	✗	✗	✗	✓
axle differential	✗	✗	✗	✓
chassis roll	✗	✗	✗	✓
chassis pitch	✗	✗	(✓)	✓
suspension	✗	✗	✗	✓
		Application		
path planning	✓	✓	✗	✗
trajectory planning	✓	✓	✓	✗
vehicle control	✗	(✓)	✓	✓
simulation	✗	(✓)	✓	✓

Table 3.1: Comparison of vehicle models: point-mass model (PM), kinematic single-track model (KST), dynamic single-track model (DST), double-track model (DT). The following symbols are used: ✓... common/possible, (✓)... uncommon/possible, ✗... impossible.

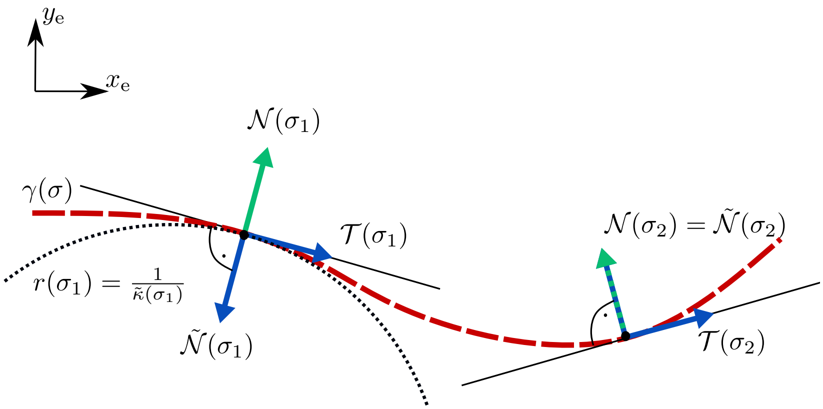
y_e
x_e
$\mathcal{N}(\sigma_1)$
$\gamma(\sigma)$
$\mathcal{T}(\sigma_1)$
$r(\sigma_1) = \frac{1}{\tilde{\kappa}(\sigma_1)}$
$\tilde{\mathcal{N}}(\sigma_1)$
$\mathcal{N}(\sigma_2) = \tilde{\mathcal{N}}(\sigma_2)$
$\mathcal{T}(\sigma_2)$

Figure 3.3: Tangent, normal and signed normal vector along a curve $\gamma(\sigma)$.

to an environment with a road. This road could potentially be a race track for autonomous racing or a highway consisting of several lanes.

In order to include the road alignment specifications in the OCP formulation, the road geometry is taken into account. Therefore, let the road center curve

$$\Gamma := \{\gamma(\sigma) | \sigma \in [0, \infty)\}$$

be specified by a function $\gamma^\top(\sigma) = \begin{bmatrix} p_x(\sigma) & p_y(\sigma) \end{bmatrix}$ with $\gamma : \mathbb{R}_{\geq 0} \rightarrow \mathbb{R}^2$, parameterized by its path length $\sigma \in \mathbb{R}_{\geq 0}$.

Remark 3.7.1. *The spatial derivative* $\frac{\partial f(\sigma)}{\partial \sigma}$ *of a function* $f : \mathbb{R}_{\geq 0} \rightarrow \mathbb{R}$*, where* σ *is the path length, is denoted in the following by* $f'(\sigma) := \frac{\partial f(\sigma)}{\partial \sigma}$.

Consider the tangent vector

$$\mathcal{T}(\sigma) := \gamma'(\sigma)$$

of a curve which points in the direction of the curve $\gamma(\sigma)$ that progresses with σ, cf., Fig. 3.3. Due to the parameterization by its path length, the tangent vector is a unit vector, where $||\mathcal{T}(\sigma)||_2 = 1$ holds. The change of the tangent vector along the reference path $\mathcal{T}'(\sigma)$ is the curvature vector. It is perpendicular to the tangent vector, with $\mathcal{T}^\top(\sigma)\mathcal{T}'(\sigma) = 0$ and points "inside" the curve. The magnitude of the curvature vector is the curvature $\tilde{\kappa}(\sigma)$ with

$$\tilde{\kappa}(\sigma) := ||\mathcal{T}'(\sigma)||_2 = ||\gamma''(\sigma)||_2 = \sqrt{p_x''(\sigma)^2 + p_y''(\sigma)^2}$$

and the normalized curvature vector is the normal unit vector $\tilde{\mathcal{N}}(\sigma)$, with

$$\tilde{\mathcal{N}}(\sigma) := \frac{\mathcal{T}'(\sigma)}{\tilde{\kappa}(\sigma)}.$$

The vectors $\mathcal{T}$ and $\tilde{\mathcal{N}}$ define an orthonormal basis that moves along the reference curve with σ. They are also referred to as the Frenet-Serret coordinate system in two dimensions. The kinematic progression of the coordinate frame is described by the Frenet-Serret formulas

$$\mathcal{T}'(\sigma) = \tilde{\kappa}(\sigma)\tilde{\mathcal{N}}(\sigma),$$

$$\tilde{\mathcal{N}}'(\sigma) = -\tilde{\kappa}(\sigma)\mathcal{T}(\sigma).$$

In order to avoid the flipping of the normal unit vector $\tilde{\mathcal{N}}(\sigma)$ emerging from a change of the "turning" direction of a curve, the normal unit vector can be formulated as a 90 degree perpendicular counter-clockwise rotation of the tangent vector with the rotation matrix R^{90}. This yields the slightly modified, signed normal unit vector $\mathcal{N}(\sigma) = R^{90}\mathcal{T}(\sigma)$, cf. [207] for details. By using the signed curvature $\kappa(\sigma)$, which is positive if the curve turns *left* in the direction of motion and negative otherwise, the Frenet-Serret formulas can also be stated as

$$\mathcal{T}'(\sigma) = \kappa(\sigma)\mathcal{N}(\sigma),$$

$$\mathcal{N}'(\sigma) = -\kappa(\sigma)\mathcal{T}(\sigma).$$

The curvature can be obtained from the signed curvature by

$$\tilde{\kappa}(\sigma) = |\kappa(\sigma)|$$

and, vice versa, by

$$\kappa(\sigma) = \tilde{\kappa}(\sigma)\mathcal{N}^\top(\sigma)\tilde{\mathcal{N}}(\sigma).$$

Notably, the curvature can also be related to the turning angle $\varphi^\gamma(\sigma)$, which is the angle of the tangent vector $\mathcal{T}(\sigma)$, with $\mathcal{T}(\sigma) = \begin{bmatrix}\cos\varphi^\gamma(\sigma) & \sin\varphi^\gamma(\sigma)\end{bmatrix}^\top$. By taking the derivative of the tangent vector, it follows that

$$\mathcal{T}'(\sigma) = \varphi^{\gamma\prime}(\sigma)\begin{bmatrix}-\sin\varphi^\gamma(\sigma) & \cos\varphi^\gamma(\sigma)\end{bmatrix}^\top = \varphi^{\gamma\prime}\mathcal{N}(\sigma). \tag{3.7}$$

Relating (3.7) to the Frenet-Serret formulas, the signed curvature can be expressed by the change of the turning angle by

$$\kappa(\sigma) = \varphi^{\gamma\prime}(\sigma).$$

An interpretation of the curvature can be given by considering the concept of the osculating circle, cf., Fig. 3.4. At any point σ of the curve where the curvature is nonzero, i.e., $\kappa(\sigma) \neq 0$, a circle with the center

$$c(\sigma) = \gamma(\sigma) + \frac{1}{\kappa(\sigma)}\mathcal{N}(\sigma)$$

and radius

$$r(\sigma) = \frac{1}{|\kappa(\sigma)|}$$

can be found that has the same tangent and normal vector and has the same curvature as the reference path Γ at $\gamma(\sigma)$ and locally approximates Γ at σ. The curve defined by $c(\sigma)$ is called the evolute of Γ.

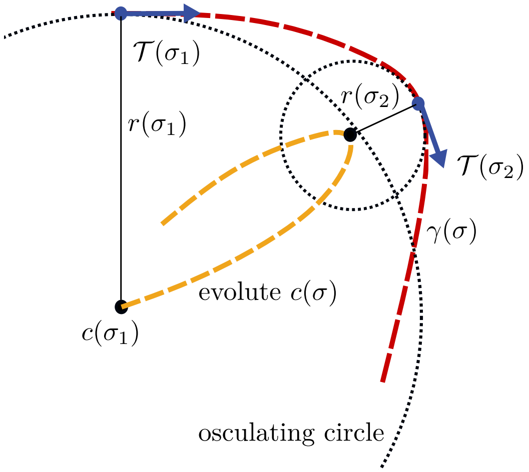

Figure 3.4: Conceptual drawing of an evolute and osculating circle.

3.7.1 Projection of Configuration States

At this point, several important concepts related to a planar curve Γ have been introduced. In the context of AD, this curve may be the center line of a road. This curve can be used as part of an OCP that is used to formulate the motion planning problem. The position states of a planned trajectory in OCP (1.1) are typically closely aligned with the road. In other words, the trajectory and, potentially, boundary constraints are curvilinear to the road's center line. This alignment is also manifested in the cost specifications. For example, a typical cost in autonomous racing would involve driving as far as possible within a given time interval within the road boundaries. In highway driving scenarios, part of the cost may be driving close to the current lane center. Altogether, four objectives in autonomous driving may be related to the road geometry: (i) the road-aligned progress, (ii) the road-aligned velocity, (iii) the road-aligned tracking costs, and (iv) road-aligned boundary constraints. The following shows how these objectives can be formulated numerically efficiently in an OCP.

Assume the two-dimensional Cartesian position state $p(t) = \begin{bmatrix} p_x(t) & p_y(t) \end{bmatrix}^\top \in \mathbb{R}^2$ as part of the Cartesian state vector $x(t) \in \mathbb{R}^{n_x}$. For the Cartesian position state $p(t)$, a distance $d(\sigma, t) = p(t) - \gamma(\sigma)$ is formulated between the vehicle position and the curve at some position σ, see Fig. 3.5. The closest point $\gamma(s(t))$ on the path $\gamma(\sigma)$ can be found by solving the unconstrained optimization problem

$$s(t) = \arg\min_{\sigma} \left\| d(\sigma, t) \right\|_2 . \tag{3.8}$$

The longitudinal path progression $s(t)$ describes the projected position along

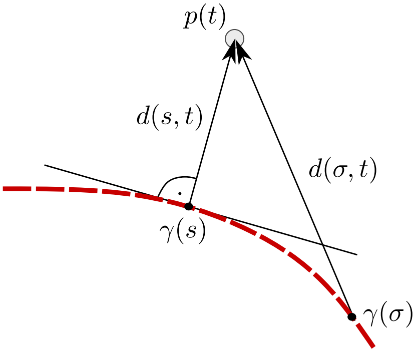

Figure 3.5: Distance of a point $p(t)$ to a curve $\gamma(\sigma)$ parameterized by its path length σ.

the path $\gamma(\sigma)$ and can be used along with its derivative $\dot{s}(t)$ to specify the costs related to (i) and (ii). For example, a linear cost -$w_{\mathrm{s}}s$ could be used to formulate a maximization of progression, as, for instance, in [236].

The closest distance to the path $d(t) = d(s(t), t)$ can be used to formulate a center line tracking cost (iii) and boundary constraints (iv). In many applications, the distance to the path is also referred to as contouring cost, and its minimization is a primary objective in the considered problem, e.g., in [157] for biaxial contouring machines, in [167] for autonomous vehicle racing and in [236] for drone racing. It can be stated as a multiple of the signed normal vector $\mathcal{N}(t)$ by $d(t) = n(t)\mathcal{N}(t)$, where $n \in \mathbb{R}$ is the signed distance which is positive if the point $p(t)$ is left of the curve in the direction of progress. In the following, time dependency is dropped for a more concise notation.

Most vehicle states of Chapter 3 comprise a heading angle φ. The heading angle can be related to the turning angle of the curve $\gamma(\sigma)$ by defining the heading angle mismatch α as

$$\alpha := \varphi - \varphi^{\gamma}(s).$$

In the following, two variants of a state vector are defined. The symbol $\bullet = \{F, C\}$ is used to refer to the Frenet coordinate frame (FCF) or the CCF. One state vector is defined in the CCF, while the other state vector is defined by substituting the Cartesian states with road-aligned states. Let the position $p^{\top} = \begin{bmatrix} p_x & p_y \end{bmatrix}$ and the heading angle φ be part of a CCF configuration state vector $x^{\mathrm{c,C}} = \begin{bmatrix} p_x & p_y & \varphi \end{bmatrix}^{\top}$. The configuration space usually has the same dimension as the degrees-of-freedom of a vehicle or, in general, a robot, cf., [159]. Simplified, it comprises all possible position and orientation states of a vehicle. Usually, the three states of the position and the heading angle are formulated in the Cartesian or Frenet space. Similar to the Cartesian states, let the projected position s on the reference curve Γ, the lateral distance n and the heading angle mismatch α be part of an FCF configuration state vector $x^{\mathrm{c,F}} = \begin{bmatrix} s & n & \alpha \end{bmatrix}^\top$. Either the CCF state $x^{\mathrm{c,C}}$ or the FCF state $x^{\mathrm{c,F}}$ along with the reference curve Γ fully define the vehicle configuration. Therefore, the full state vector can be composed either with FCF or CCF configuration states and coordinate frame independent states $x^{\neg c} \in \mathbb{R}^{n_x - 3}$, with

$$x^{\bullet} = \begin{bmatrix} x^{\mathrm{c},\bullet} \\ x^{\neg \mathrm{c}} \end{bmatrix}.$$

The full CCF state vector x^{C} corresponds to the usual vehicle state x in Cartesian coordinates and comprises also the coordinate independent states $x^{\neg c}$. Including the controls $u \in \mathbb{R}^{n_u}$, the ODE $\dot{x} = f(x, u)$ in the CCF describes the vehicle motion, as in Chapter 3. For example, in the kinematic model of Sect. 3.3, the state $x^{\neg \mathrm{c}}$ would comprise the velocity v and the steering angle δ and the controls would be acceleration force F_x and the steering angle rate r.

Note that the FCF states together with Γ define the configuration uniquely almost everywhere, excluding the state space covered by the evolute $c(\sigma)$, as detailed in the course of this chapter. The FCF configuration state can be obtained from the CCF configuration state by what is called here the Frenet transformation

$$x^{\mathrm{c,F}} = \mathcal{F}(x^{\mathrm{c,C}}) = \mathcal{F}(P^{\mathrm{c}} x) = \begin{bmatrix} s \\ \big(p - \gamma(s)\big)^\top \mathcal{N}(s) \\ \varphi - \varphi^{\gamma}(s) \end{bmatrix}$$

for

$$\big(p - \gamma(s)\big)^\top \mathcal{N}(s) \kappa(s) \neq 1$$

and

$$s = \arg\min_{\sigma} ||d(\sigma)||_2.$$

The matrix $P^{\mathrm{c}} \in \mathbb{R}^{3 \times n_x}$ selects the position and heading angle states of x, or likewise, of the FCF state x^{F}. Similarly, the longitudinal state s is selected by $P^{\mathrm{s}} \in \mathbb{R}^{1 \times n_x}$ and the lateral state n is selected by $P^{\mathrm{n}} \in \mathbb{R}^{1 \times n_x}$ from the FCF states. The inverse Frenet transformation is easier and given as

$$x^{\mathrm{c,C}} = \mathcal{F}^{-1}(x^{\mathrm{c,F}}) = \begin{bmatrix} \gamma(s) + n\mathcal{N}(s) \\ \varphi^{\gamma}(s) + \alpha \end{bmatrix}.$$

OCP (1.1) can now be reformulated to a general form that includes the Frenet transformation as the projection on the reference path by

$$\min_{x(\cdot),\,u(\cdot)} \quad J^{\neg\mathrm{c}}\big(x^{\neg\mathrm{c}}(\cdot)\big) + J^{\mathrm{c,F}}\big(\mathcal{F}(P^{\mathrm{c}}x(\cdot))\big) + J^{u}\big(u(\cdot)\big) \tag{3.9a}$$

s.t.

$$x(t_0) = x_0, \tag{3.9b}$$

$$\dot{x}(t) = f(x(t), u(t)), \qquad t \in [t_0, \infty), \tag{3.9c}$$

$$P^{\mathrm{c}}x(t) \in \mathbb{X}_{\mathrm{free}}(t), \qquad t \in [t_0, \infty), \tag{3.9d}$$

$$P^{\mathrm{n}}\mathcal{F}\big(P^{\mathrm{c}}x(t)\big) \in \mathbb{X}_{\mathrm{rd}}\big(P^{\mathrm{s}}\mathcal{F}(P^{\mathrm{c}}x(t))\big), \quad t \in [t_0, \infty), \tag{3.9e}$$

$$x^{\neg\mathrm{c}}(t) \in \mathbb{X}^{\neg\mathrm{c}}, \qquad t \in [t_0, \infty), \tag{3.9f}$$

$$u(t) \in \mathbb{U}, \qquad t \in [t_0, \infty), \tag{3.9g}$$

and represents many vehicle motion planning problems on roads. The objective (3.9a) contains a cost $J^{\neg\mathrm{c}}\big(x^{\neg\mathrm{c}}(\cdot)\big)$ for states that are coordinate frame independent. For example, the yaw rate or the difference between the vehicle and reference velocities are typically penalized. The CCF states $x^{\mathrm{c},C} = P^{\mathrm{c}}x$ are rarely directly part of the cost in a structured road environment, besides in a trajectory tracking formulation. If the cost involves trajectory tracking, it can be likewise formulated in the FCF. Trajectory and, more common, path tracking costs related to the FCF states $x^{\mathrm{c},F} = \mathcal{F}(P^{\mathrm{c}}x)$ are often considered as main objective, e.g., [167, 85, 151]. For instance, the path progression state s is often used in autonomous racing to maximize progress as an approximation of time-optimal driving [167, 41]. The lateral distance to the reference lane Γ, which may be the lane center, is accounted for with a cost on the lateral state n [181, 213]. Collision avoidance constraints are usually defined on Cartesian earth frame states $x^{\mathrm{c,C}} = P^{\mathrm{c}}x$ and denoted by the nonconvex set $\mathbb{X}_{\mathrm{free}} \subseteq \mathbb{R}^3$ in (3.9d) but can also be defined using road aligned states $x^{\mathrm{c,F}}$ using over-approximations [228] of the obstacles in the FCF. In the constraints (3.9e) road-aligned boundary constraints are formulated by means of the compact set or tube $\mathbb{X}_{\mathrm{rd}}(s) \subset \mathbb{R}$, defining constraints on the lateral state n. The constraints on the lateral state depend on the projected longitudinal position s along the path Γ. Equation (3.9f) comprises coordinate frame independent state constrains $\mathbb{X}^{\neg\mathrm{c}} \subseteq \mathbb{R}^{n_x-3}$, e.g., constraints on the maximum velocity.

Notably, the projected states $x^{\mathrm{c,F}}$ appear in the cost (3.9a) and road boundary constraints (3.9e), whereas the CCF states appear in the dynamics and the collision avoidance constraints (3.9d).

Besides the potential nonlinearity and nonconvexity, several structures of OCP (3.9) make it hard to solve and require both simplifications and efficient numerical solvers. The collision avoidance constraints (3.9d) are nonconvex, and the planning space is nonhomeomorphic. Therefore, the problem comprises a combinatorial subproblem [159] and requires techniques from discrete optimization. Finding appropriate formulations that include the subtask of solving the combinatorial subproblem and solving them efficiently is a major part of this thesis. It is considered in our related works [225, 227, 229] and Sect. 6.1, 6.2 and 6.3 and the introduction Sect. 3.8.

Secondly, obtaining road aligned states $x^{\mathrm{c,F}} = \mathcal{F}(P^{\mathrm{c}}x)$ from CCF states x provided by most measurement systems involves solving the optimization problem (3.8). Different approaches of simplifying or avoiding this projection inside of optimization problem (3.9) were proposed in related literature. Two major variants involve either an *additional* path progression state [157, 85], referred to as *contouring control formulation* or a projection of the vehicle dynamics onto the reference path [292, 151, 22], referred to as *projected formulation.* The two major variants are explained in the following. A third variant that builds on the projection of vehicle dynamics utilizes a formulation of the dynamics in the road-aligned FCF and, additionally, in the CCF. The third variant is a major contribution of this thesis and subject of Chapter 5.2 and our related paper [228].

3.7.2 Contouring Control Formulation

One approach to include the projection of the CCF state onto the reference path $\gamma(\sigma)$ in OCP (3.9) involves adding the projected position $\tilde{s}$ as an auxiliary decision variable to optimization problem (3.9) and including the distance to the path $\gamma(\tilde{s})$ at the auxiliary state $\tilde{s}$ to the objective (3.9a), see [85, 157, 167, 236]. This results in a formulation that trades off the original objective (3.9a) with the projection onto the path (3.8). The possibly conflicting objectives result in a multi-objective optimization problem. The distance $p(t) - \gamma(\tilde{s})$ is split into a lateral component

$$(p(t) - \gamma(\tilde{s}))^\top \mathcal{N}(\tilde{s}),$$

i.e., a contouring error [157, 236], and a longitudinal component

$$(p(t) - \gamma(\tilde{s}))^\top \mathcal{T}(\tilde{s}),$$

i.e., a lag error, by utilizing the tangent and normal unit vectors $\mathcal{T}(\tilde{s})$ and $\mathcal{N}(\tilde{s})$, respectively. The lag error corresponds to “tracking” the auxiliary state, while the contouring error approximates minimizing the distance to the path Γ. This formulation can also be interpreted as a specific approximation of the Frenet transformation at $\tilde{s}$, written as

$$\tilde{\mathcal{F}}(x^{\mathrm{c,C}}, \tilde{s}) = \begin{bmatrix} \tilde{s} \\ (p - \gamma(\tilde{s}))^\top \mathcal{N}(\tilde{s}) \\ \varphi - \varphi^\gamma(\tilde{s}) \end{bmatrix}. \tag{3.10}$$

The projection within the optimization problem (3.9) may be wrong if constraints or costs were defined on any other functions dependent on the projected position $\tilde{s}$, such as in constraints (3.9e) or the costs $J^{\mathrm{c,F}}(x^{\mathrm{c,F}})$. Unfortunately, these constraints are essential to the overall motion planning problem formulation. Therefore, a small mismatch between $\tilde{s}$ and the actual projected state s is accepted in related algorithms. The safety-relevant constraint for the road boundaries (3.9e) can be safely approximated by defining a set

$$\mathcal{B}(\tilde{s}) \subseteq \mathbb{R}^3$$

along the path Γ for which the following implication holds

$$P^{\mathrm{c}}x \in \mathcal{B}(\tilde{s}) \implies P^{\mathrm{n}}\mathcal{F}\big(P^{\mathrm{c}}x\big) \in \mathbb{X}_{\mathrm{rd}}\big(P^{\mathrm{s}}\mathcal{F}(P^{\mathrm{c}}x)\big). \tag{3.11}$$

For example, the set $\mathcal{B}(\tilde{s})$ can be a 2-norm ball around the Cartesian path position $\gamma(\tilde{s})$ and the heading angle mismatch $\varphi - \varphi^{\gamma}$. This ball can be chosen such that all configuration states $P^{\mathrm{c}}x$ are within the road boundaries. The state $\tilde{s}$ can be seen as a tracking state [236] which may even have its own dynamics by an additional velocity state $\tilde{v}_s$. Finally, problem (3.9) can be reformulated for this particular approximation as

$$\min_{x(\cdot),\, u(\cdot),\, \tilde{s}(\cdot)} \quad J^{\neg\mathrm{c}}\big(x^{\neg\mathrm{c}}\big) + J^{\mathrm{c,F}}\big(\tilde{\mathcal{F}}(P^{\mathrm{c}}x), \tilde{s}\big) + J^{u}\big(u\big) \tag{3.12a}$$

$$+\, J^{\mathrm{lag}}\Big(\big(P^{\mathrm{p}}x - \gamma(\tilde{s})\big)^{\top}\mathcal{T}(\tilde{s})\Big) \tag{3.12b}$$

s.t.

$$x(t_0) = x_0, \tag{3.12c}$$

$$\dot{x}(t) = f(x(t), u(t)), \qquad t \in [t_0, \infty), \tag{3.12d}$$

$$P^{\mathrm{c}}x(t) \in \mathbb{X}_{\mathrm{free}}(t), \qquad t \in [t_0, \infty), \tag{3.12e}$$

$$P^{\mathrm{c}}x(t) \in \mathcal{B}(\tilde{s}(t)), \qquad t \in [t_0, \infty), \tag{3.12f}$$

$$x^{\neg\mathrm{c}}(t) \in \mathbb{X}^{\neg\mathrm{c}}, \qquad t \in [t_0, \infty), \tag{3.12g}$$

$$u(t) \in \mathbb{U}, \qquad t \in [t_0, \infty). \tag{3.12h}$$

The formulation (3.12) involves an additional cost $J^{\mathrm{lag}}(\cdot)$ for the lag error in the longitudinal path direction $\mathcal{T}(\tilde{s})$.

3.7.3 Projected Formulation

Another approach of reformulating (3.9) in order to be able to efficiently solve the related OCP involves formulating the dynamics solely in terms of projected states $x^{\mathrm{c,F}}$ and coordinate frame independent states $x^{\neg\mathrm{c}}$, with the Frenet state $x^{\mathrm{F}\top} = \begin{bmatrix} x^{\mathrm{c,F}\top} & x^{\neg\mathrm{c}\top} \end{bmatrix}$. In problem formulation (3.9), only the collision avoidance constraints (3.9d) are formulated using CCF states. Fortunately, for common obstacle dimensions of passenger vehicles, the vehicle shape can be over-approximated by a convex shape. Therefore, the collision avoidance constraints can be under-approximated by an obstacle-free set $\mathbb{X}^{\mathrm{F}}_{\mathrm{free}}(t)$. However, the over-approximation of the obstacles (or under-approximation of the obstacle-free set) requires some conservativeness. A discussion is given in Chapter 5.2 and our related paper [228].

As shown in the following, the dynamics can be projected onto the reference by formulating the first-order necessary condition (FONC) of (3.8), solving them at the initial time and finding a condition to establish them for all future times [287]. The objective of (3.8) can be squared and multiplied by $\frac{1}{2}$ without changing the optimal solution. The FONC for (3.8) defines a necessary condition on the path progression σ such that the point $\gamma(\sigma)$ is closest to $p(t)$, with the distance $d(\sigma, t) = p(t) - \gamma(\sigma)$ and

$$\nabla_{\sigma} \frac{1}{2} \left|\left| d(\sigma, t) \right|\right|_2^2 = d(s, t)^\top \mathcal{T}(s) = 0. \tag{3.13}$$

Assuming that $s(t = 0) = \arg\min_{\sigma} ||d(\sigma, t = 0)||_2^2$, the condition (3.13) can be enforced for all times by setting the first derivative of the FONC to zero, i.e.,

$$\frac{d}{dt} d(s, t)^\top \mathcal{T}(s) = 0. \tag{3.14}$$

Computing the derivative as defined in (3.14) yields

$$\dot{s}(t) = \frac{\left(\dot{p}(t)\right)^\top \mathcal{T}(s(t))}{1 - \kappa(s) d(s, t)^\top \mathcal{N}(s)}. \tag{3.15}$$

The term $\left(\dot{p}(t)\right)^\top \mathcal{T}(s(t))$ can be identified as the velocity in the direction of the path. It can be expressed by the coordinate independent velocity $v(t)$ and heading angle mismatch $\alpha(t)$ as

$$\left(\dot{p}(t)\right)^\top \mathcal{T}(s(t)) = v(t) \cos\left(\varphi(t) - \varphi^{\gamma}(s(t))\right) = v(t) \cos\left(\alpha(t)\right). \tag{3.16}$$

Additionally, the term $d(s, t)^\top \mathcal{N}(s(t))$ was defined previously as the signed lateral distance to the path Γ by

$$n(t) := d(s, t)^\top \mathcal{N}(s(t)), \tag{3.17}$$

which yields the ODE of the longitudinal path state

$$\dot{s} = \frac{v \cos\left(\alpha\right)}{1 - \kappa(s) n}, \tag{3.18}$$

expressed by Frenet states and omitting the dependency on time t. Taking the time derivative of $n(t)$ in (3.17) and simplifying the equations yields the following ODE for $n(t)$:

$$\dot{n} = \big(\dot{p} - \frac{\partial \gamma(s)}{\partial s}\dot{s}\big)^\top \mathcal{N}(s) + \big(p - \gamma(s)\big)^\top \frac{\partial \mathcal{N}(s)}{\partial s}\dot{s} \tag{3.19a}$$

$$= \big(\dot{p} - \mathcal{T}(s)\dot{s}\big)^\top \mathcal{N}(s) + n\mathcal{N}(s)^\top \big(- \kappa(s)\mathcal{T}(s)\big)\dot{s} \tag{3.19b}$$

$$= \dot{p}^\top \mathcal{N}(s) \tag{3.19c}$$

$$= v \sin(\alpha). \tag{3.19d}$$

The time derivative of the heading angle mismatch α, expressed by Frenet states x^F is

$$\dot{\alpha} = \dot{\varphi} - \dot{\varphi}^\gamma(s) \tag{3.20a}$$

$$= \dot{\varphi} - \frac{\partial \varphi^\gamma(s)}{\partial s}\dot{s} \tag{3.20b}$$

$$= \dot{\varphi} - \kappa(s)\frac{v\cos\big(\alpha\big)}{1-\kappa(s)n}. \tag{3.20c}$$

The dynamics $\dot{x}^\mathrm{F}$ of the Frenet states x^F can now be expressed by utilizing the results from (3.18), (3.19a) and (3.20a) to obtain $\dot{x}^\mathrm{F} = f^\mathrm{F}(x^\mathrm{F}, u)$. Notably, the curvature $\kappa(s)$ becomes part of the dynamics function $f^\mathrm{F}(x^\mathrm{F}, u)$ in this formulation, making the dynamics function potentially more nonlinear. The OCP (3.9) can be stated in terms of FCF states in the *projected formulation* as

$$\min_{x^\mathrm{F}(\cdot),\, u(\cdot)} \quad J^{\neg\mathrm{c}}\big(x^{\neg\mathrm{c}}\big) + J^{\mathrm{c},\mathrm{F}}\big(P^\mathrm{c}x^\mathrm{F}\big) + J^u\big(u\big) \tag{3.21a}$$

$$\text{s.t.}$$

$$x^\mathrm{F}(t_0) = x_0^\mathrm{F}, \tag{3.21b}$$

$$\dot{x}^\mathrm{F}(t) = f^\mathrm{F}(x^\mathrm{F}(t), u(t)), \quad t \in [t_0, \infty), \tag{3.21c}$$

$$P^\mathrm{c}x^\mathrm{F}(t) \in \mathbb{X}^\mathrm{F}_\mathrm{free}(t), \quad t \in [t_0, \infty), \tag{3.21d}$$

$$P^\mathrm{n}x^\mathrm{F}(t) \in \mathbb{X}_\mathrm{rd}\big(P^\mathrm{s}x^\mathrm{F}(t)\big), \quad t \in [t_0, \infty), \tag{3.21e}$$

$$x^{\neg\mathrm{c}}(t) \in \mathbb{X}^{\neg\mathrm{c}}, \quad t \in [t_0, \infty), \tag{3.21f}$$

$$u(t) \in \mathbb{U}, \quad t \in [t_0, \infty). \tag{3.21g}$$

This formulation requires $1 - n\kappa(s) \neq 0$ in order to be well defined. The subproblems as part of numerical algorithms for solving (3.21) may even get ill-conditioned in the vicinity of the subset defined by

$$\{(s,n)|1-n\kappa(s)=0\}.$$

A mitigation strategy is presented in our related paper [222] and Chapter 5.1. Since the states obtained by measurement systems are typically in the CCF, the initial state x_0^{F} requires the projection on the path by solving (3.8) in advance. This can be typically solved efficiently. Notably, this is not done as part of the optimization problem (3.21) and just for a single state.

Vehicle models and coordinate frames used in this thesis. In Tab 3.2, an overview of the vehicle models used within the main contributions of this thesis is given. Remarkably, the kinematic single-track model was mostly used within the controllers due to its sufficient modeling depth for less dynamic maneuvers. The planner used even simpler models, like the point-mass model or model-free methods, i.e., reinforcement learning.

Sect., Ref.	Vehicle Models		
	Higher-Level Planner	Low-Level Controller	Environment
5.1, [222]	N/A	KST-FCF	KST
5.2, [228]	N/A	KST-FCF	KST
6.1, [225]	spatial model FCF	KST-FCF	real-world vehicle DevBot [233], DST
6.2, [227]	PM-FCF	KST-FCF	DST
6.3, [229]	PM-FCF	KST-FCF	DT
7.1, [226]	KST-FCF	split longitudinal / lateral	real-world vehicle DevBot [233], DST
7.2, [224]	model-free	KST-FCF	KST

Table 3.2: Comparison of vehicle models used within the main contributions of this thesis. Models are related to the optimization problems of the planner, the controller, and the possibly simulated environment. The following abbreviations are used: PM (point-mass model), KST (kinematic single-track model), DST (dynamic single-track model), DT (double-track model), FCF (Frenet coordinate frame).

Remark 3.7.2. *A transformation of the vehicle dynamics was presented for the planar, two-dimensional case. The general Frenet-Serret transformation is defined in the three-dimensional space [42]. Therefore, three-dimensional motion planning problems occurring for drones or robotic manipulators can also be* *formulated similarly [20]. However, the third dimension requires an additional curve property called* torsion*, and the transformation is more sophisticated. The Frenet-Serret coordinate frame shows several disadvantages in three dimensions, such as singularities and the twist over the tangent component. Consequently, alternative coordinate frames are also considered, such as the Euler Rodrigues frame [19, 21].*

3.8 Collision Avoidance

In the following section, the collision avoidance constraint (3.9d), given so far in the deterministic and non-interactive form

$$P^{\mathrm{c}} x(t) \in \mathbb{X}_{\mathrm{free}}(t),$$

is discussed in more detail. The collision avoidance constraint is significant for providing safety and is always considered as part of planning problems. In control problems related to (1.1), collision avoidance constraints are occasionally considered since the major task is usually the stabilization of the vehicle and tracking of a reference.

In the following, collision avoidance is introduced step-by-step by increasing the complexity of the underlying optimization problem (1.1). First, the general objective is introduced for deterministic predictions or static obstacles. Next, a distinction is drawn between interactive and non-interactive collision avoidance. Thereafter, more elaborate formulations involving stochastic and game-theoretic reasoning are introduced. Finally, some well-known behavior models for highway driving are provided and used within the simulation frameworks. In this thesis, collision avoidance is related mainly to surrounding vehicles (SVs). However, the concepts apply similarly to general obstacles, as, for example, the obstacles considered in Chapter 6.1 and our related work [225]. Chapter 6.1 considers the additional concept of rewards, where a rectangular region reduces the cost if traversed by the ego vehicle.

3.8.1 Obstacle Shapes and Deterministic Formulation

Consider the three configuration states $x^{\mathrm{c}} = \begin{bmatrix} p_x & p_y & \varphi \end{bmatrix}^\top$ in Cartesian coordinates. A rectangular shape that defines the occupied set of the ego vehicle $\mathcal{O}$ is centered at the geometric vehicle center (p_x, p_y) and rotated by the heading angle φ. The rectangular shape is assumed to overapproximate the true vehicle shape tightly, cf., Fig. 3.6. The vehicle chassis specifications determine the size with the chassis length l_{ch} and width w_{ch}. Likewise, the occupied set for an SV is defined by the equivalent configuration states $p_x^{\mathrm{sv}}, p_y^{\mathrm{sv}}$

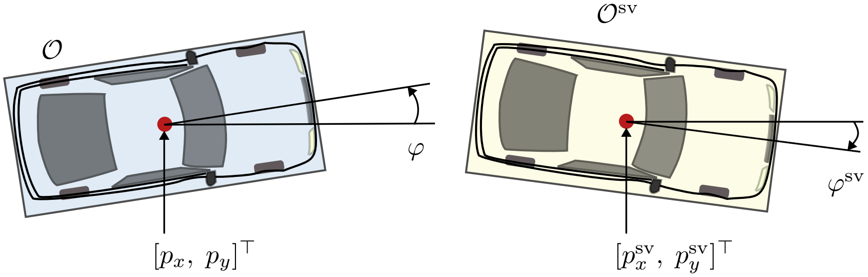
$\mathcal{O}$
$\mathcal{O}^{\mathrm{sv}}$
φ
φ^{sv}
$[p_x,\ p_y]^\top$
$[p_x^{\mathrm{sv}},\ p_y^{\mathrm{sv}}]^\top$

Figure 3.6: Vehicle configurations with rectangular approximated shapes.

and φ^{sv}. The occupied set of a rectangular obstacle can be defined in terms of vehicle configuration states x^{c} by

$$\mathcal{O}(x^{\mathrm{c}}) := \left\{ \begin{bmatrix} x \\ y \end{bmatrix} \in \mathbb{R}^2 \middle| \left\| \left(\frac{1}{2} \begin{bmatrix} l_{\mathrm{ch}} & 0 \\ 0 & w_{\mathrm{ch}} \end{bmatrix} \right)^{-1} R^{-\varphi} \left(\begin{bmatrix} x \\ y \end{bmatrix} - \begin{bmatrix} p_x \\ p_y \end{bmatrix} \right) \right\|_\infty \leq 1 \right\}, \tag{3.22}$$

where R^β is the rotation matrix for a rotation by angle β. This rectangular obstacle shape related to vehicles is also referred to as $\mathcal{O}^{\mathrm{v}}(t) := \mathcal{O}\left(x^{\mathrm{c}}(t)\right)$.

Other time-varying geometric shapes may also represent obstacles unrelated to vehicle shapes. The time-dependent occupied area of arbitrary obstacle shapes is $\mathcal{O}^\star(t) \subset \mathbb{R}^2$. For example, the infinity norm in(3.22) can be modified to a smooth function using the sublevel set of any scaled high-order norm with $\alpha_p(t) \geq 2$ by

$$\mathcal{O}^{\mathrm{n}}(t) := \left\{ p \in \mathbb{R}^2 \middle| \left\| (p - p_0(t)) R^{\varphi(t)} \mathrm{diag}\big([s_x(t),\ s_y(t)]^\top \big) \right\|_{\alpha_p(t)} \leq 1 \right\}.$$

The parameters may be time varying and comprise the offset $p_0(t)$, the rotation angle $\varphi(t)$, the scaling $s_x(t), s_y(t)$ or the norm parameter $\alpha_p(t)$, e.g., as in [221].

Commonly, these shapes are convex. This it allows for a safe over-approximation within sequential quadratic programming (SQP) iterations [221]. However, nonconvex shapes were also proposed, such as a union of convex shapes. Particularly, a union of circles is a well-known approximation of planar shapes, e.g., see [297, 328]. Another common representation of obstacle shapes is the intersection of hyperplanes

$$\mathcal{O}^{\mathrm{hp}}(t) := \left\{ p \in \mathbb{R}^2 | A(t) p \leq b(t) \right\},$$

with possibly time dependent hyper-plane parameters $A \in \mathbb{R}^{n_p \times 2}$ and $b \in \mathbb{R}^{n_p}$.

Given a time-varying obstacle shape $\mathcal{O}^\star(t)$, the task of deterministic collision avoidance involves planning a trajectory $\tau = \{x^{\mathrm{c}}(t) | t \geq 0\}$ with configuration states $x^{\mathrm{c}}(t)$ that satisfy

$$\mathcal{O}(x^{\mathrm{c}}(t)) \cap \mathcal{O}^\star(t) = \emptyset \tag{3.23}$$

for all $t \geq 0$. If collision avoidance between the ego vehicle and an SV, given its configuration states $x^{\mathrm{c,sv}}$, is considered, the obstacle avoidance can be stated as

$$\mathcal{O}(x^{\mathrm{c}}(t)) \cap \mathcal{O}(x^{\mathrm{c,sv}}(t)) = \emptyset. \tag{3.24}$$

Note that the prediction of non-interactive SVs may involve a physical model $f^{\mathrm{sv}}(x^{\mathrm{sv}}(t), u^{\mathrm{sv}}(t))$ of the SV, which describes the motion by the ODE

$$\dot{x}^{\mathrm{sv}}(t) = f^{\mathrm{sv}}(x^{\mathrm{sv}}(t), u^{\mathrm{sv}}(t)), \tag{3.25}$$

with an input $u^{\mathrm{sv}}(t)$. By assuming a model of the driver that decides on the inputs $u^{\mathrm{sv}}(t)$ and the current state $x_0^{\mathrm{sv}} = x^{\mathrm{sv}}(0)$ the SV states can be obtained by forward simulation.

Static obstacle avoidance refers to avoiding obstacles without time dependency of the obstacle shape or configuration, thus $\mathcal{O}^\star(t) = \mathcal{O}^\star$.

Simplifications and Formulations

It is nontrivial to represent the constraint $\mathcal{O}\big(x^{\mathrm{c}}(t)\big) \cap \mathcal{O}^\star(t) = \emptyset$ in the OCP (3.9), cf., [248]. One possibility involves adding a constraint on the signed distance between the sets $\mathcal{O}(x^{\mathrm{c}}(t))$ and $\mathcal{O}^\star(t)$. The signed distance is negative if the obstacles overlap and is nonnegative otherwise. However, the computation of the signed distance is still computationally demanding [248] and requires reformulations by, for example, exploiting the strong duality of convex optimization and additional decision variables [320].

A major simplification of the collision avoidance constraint can be achieved by over-approximating the obstacle shape $\mathcal{O}^\star(t)$ by a shape $\tilde{\mathcal{O}}^\star(t)$, with $\mathcal{O}^\star(t) \subseteq \tilde{\mathcal{O}}^\star(t)$, such that the implication

$$\begin{bmatrix} p_x(t) \\ p_y(t) \end{bmatrix} \notin \tilde{\mathcal{O}}^\star(t) \implies \mathcal{O}\big(x^{\mathrm{c}}(t)\big) \cap \mathcal{O}^\star(t) = \emptyset \tag{3.26}$$

holds. This over-approximation is often named shape inflation and allows one to plan a trajectory without considering the particular ego vehicle orientation. For example, if two rectangular shapes for the SV and ego vehicle are considered as in Fig. 3.6, the over-approximated shape $\tilde{\mathcal{O}}^\star(t)$ can be chosen to be a circle with

$$\tilde{\mathcal{O}}^\star(t) = \big\{p \in \mathbb{R}^2 \big| ||p - p^{\mathrm{sv}}(t)||_2 \leq \tilde{r}\big\},$$

and radius

$$\tilde{r} = \sqrt{(l_{\mathrm{ch}}/2)^2 + (w_{\mathrm{ch}}/2)^2} + \sqrt{(l_{\mathrm{ch}}^{\mathrm{sv}}/2)^2 + (w_{\mathrm{ch}}^{\mathrm{sv}}/2)^2},$$

centered at the SV position $(p_x^{\text{sv}}, p_y^{\text{sv}})$. For numerical optimization algorithms, it is favorable if the shape $\tilde{\mathcal{O}}^\star(t)$ is convex and smooth, cf. [221].

The typically convex definition of obstacles leads to nonconvex optimization problems when considering the obstacle-free space [159]. In fact, avoiding multiple obstacles imposes an NP-hard problem [159] and requires techniques from discrete optimization. Many popular algorithms simplify the problem by discretizing the state-space and transforming the motion planning problem to a graph-search problem [159, 199]. However, this inevitably suffers from the curse of dimensionality and imposes the reduction of the number of states and controls in the vehicle model, e.g., as in [8]. Clearly, this may increase the suboptimality. Alternatively, the combinatorial part of the problem, i.e., the nonconvexity, can be reformulated by using discrete optimization variables, leading to a mixed-integer program (MIP) problem structure. Since the problem at hand is in the challenging NP-hard complexity class, it may not be surprising that MIPs are equally hard to solve. Particularly, real-time environments are challenging. However, the subclass of MIQPs, which requires convex quadratic costs and linear constraints, can be solved more efficiently thanks to powerful commercial solvers that automatically exploit problem structures and utilize heuristics. MIQP planning formulations and the corresponding solvers are on the brink of being real-time feasible in practice. For example, the authors in [213] have proposed a decision-making and planning framework based on MIQPs with evaluations on embedded hardware. Since MIQPs require linear models, the constraints in OCP (3.9d) need to be linearized. An MIQP problem formulation requires a linear model, which could be a point-mass model in Frenet coordinates such as in Sect. 3.4, or linearizing higher fidelity models at a set point.

The linearization of the model and other constraints limit the use of these formulations to specific scenarios where the linearization captures the dynamics well enough. Notably, the computation time can still be too high for specific scenarios. Two strategies to sufficiently decrease the computation time in highway scenarios are proposed in Chapter 6.2 and 6.3 and our related works [227] and [229], respectively. In Chapter 6.2, the fundamental idea is to decrease the discrete variables by an efficient MIQP formulation. In Chapter 6.3, a speedup is obtained by utilizing machine learning methods to predict the discrete variables of the problem through extensive simulations and tailored neural network (NN) architectures.

Interactive Vehicles

In this thesis, collision avoidance for interactive vehicles in the deterministic case considers a dependency of the SV configuration state $x^{\text{c,sv}}$ on the ego vehicle configuration state x^{c}, such as used, e.g., in [93]. What is here referred to as "interactive vehicles" may also be referred to as "planning by forward simulation”, cf. [59]. It can be modeled by a time-varying ODE

$$\dot{x}^{\mathrm{c,sv}}(t) = f^{\mathrm{sv}}\big(x^{\mathrm{c,sv}}(t), x^{\mathrm{c}}(t), t\big).$$

This model is referred to as the deterministic interactive behavior model. Regarding the consideration of an interactive SV in the planning problem (3.9), the OCP needs to be augmented by the SV dynamics. However, besides the requirement of such a behavior model and the additional states, the complexity for solving the OCP does not increase fundamentally. Notably, if the SV is assumed to optimize its own cost and can change its policy, the problem is considered a game-theoretic problem or multi-agent problem. Game-theoretic or multi-agent problems are considerably harder to solve and described in Sect. 3.8.3. For example, the “intelligent driver model (IDM)” [286] or the “minimizing overall braking induced by lane change (MOBIL)” model [142] are considered interactive vehicle models.

3.8.2 Stochastic Collision Avoidance

So far, the predictions of SVs were assumed to be known deterministically. However, the uncertainty of the prediction may also be accounted for by a stochastic prediction. In the following, the discrete-time notation is used because stochasticity is rarely expressed in continuous-time in the related literature. Similar to [326], a probability distribution $\Delta(\cdot|x_k^{\mathrm{c,sv}}, x_k)$ describes an interactive consecutive SV configuration state $x_{k+1}^{\mathrm{c,sv}}$ at time $(k+1)t_\Delta$, with

$$x_{k+1}^{\mathrm{c,sv}} \sim \Delta(\cdot|x_k^{\mathrm{c,sv}}, x_k).$$

The distribution may also be modeled non-interactively by $\Delta(\cdot|x_k^{\mathrm{c,sv}})$, only depending on the SV state x_k^{cv}, cf. [26, 102, 263, 323]. Stochastic collision avoidance now requires that the constraints are satisfied by a certain probability $\eta \in [0,1]$, also referred to as confidence level. The so-called chance-constrained collision avoidance constraint may then be formulated by

$$\mathrm{prob}\big(\mathcal{O}(x_k^{\mathrm{c}}) \cap \mathcal{O}(x_k^{\mathrm{c,sv}}) = \emptyset\big) \leq 1 - \eta$$

when the SV state follows the distribution $x_k^{\mathrm{c,sv}} \sim \Delta(\cdot|x_{k-1}^{\mathrm{cv}}, x_{k-1})$. If the distribution Δ has finite support, the confidence level η may be equal to one, corresponding to the so-called robust formulation, cf., [263, 26]. The robust formulation accounts for the worst case of all configuration states following the distribution Δ.

SVs can be modeled hierarchically by a driver model, expressed by a probability distribution $\pi^{\mathrm{sv}}(\cdot|x^{\mathrm{c,sv}}, x^{\mathrm{c}})$ that determines the SV controls $u_k^{\mathrm{sv}} \sim \pi^{\mathrm{sv}}(\cdot|x^{\mathrm{c,sv}}, x^{\mathrm{c}})$ of the vehicle model, cf., [323]. The distribution $\pi^{\mathrm{sv}}(\cdot|x^{\mathrm{c,sv}}, x^{\mathrm{c}})$ is assumed to model the behavior of the SV, therefore, referred to as behavior model.

Commonly, Gaussian processes are used to model the distribution Δ, e.g., [102] with a non-interactive prediction model and [326], using an interactive prediction model based on a Gaussian process.

3.8.3 Game-Theoretic Collision Avoidance

In real-world scenarios, other drivers' behavior may change arbitrarily at each time step by strategic long-term reasoning [59, 161]. Game theory provides a framework to describe the behavior of multiple vehicles or, more generally, multiple agents. Each agent is assumed to optimize its own objective, given possibly limited information about the overall scenario, i.e., the dynamic game. Depending on the assumptions, the game-theoretic framework can have many particularities and mathematical challenges. Note that the aim here is to give a relatively high-level introduction. Game-theoretic methods pose significant challenges to the online optimization framework.

In general, the individual costs of the agents can be arbitrary, leading to so-called general sum games of multiple agents, which are hard to solve. Solving a game usually relates to finding the lowest-cost Nash equilibria. Nash equilibria occur if, for chosen agent policies or player strategies, none of the players can improve their cost by changing solely their own policy. For example, consider two vehicles passing through a narrow gap where only one fits at a time. Furthermore, consider that there is a small negative cost for traversing quickly through the gap and a high cost for a collision. The two vehicles' policies allow either waiting and letting the other driver pass or proceeding with driving. The two Nash equilibria are the two combinations of policies where one vehicle waits and the other passes. This example clearly shows that some problems may arise in this context. First, it is assumed that all agents know the game's structure and act rationally. Secondly, if there are several, there needs to be an agreement on which Nash equilibria is played.

For competitive scenarios, the agents' objective may be simplified from a general sum game to a zero-sum type corresponding to a zero-sum over all individual costs of the agents. This type of game is often used in race car scenarios, e.g., in [299, 168].

Another simplification may be applied in driving scenarios, where one may assume that all agents cooperate as opposed to acting against each other. More realistic than pure cooperation would be a semi-cooperative modeling using a Social Value Orientation [57], which originated from social psychology.

If the game is modeled as a Stackelberg game, a leader and follower structure is assumed, and the arising equilibria are named Stackelberg equilibria, c.f, [58, 316, 90]. In this setting, the power of the ego agent is overestimated in the

Chapter	Obstacle or SV Type		
	Planner	Controller	Environment
Ch. 5.1, [222]	N/A	N/A	N/A
Ch. 5.2, [228]	N/A	non-interactive, deterministic, moving	non-interactive, deterministic, moving
Ch. 6.1, [225]	non-interactive, deterministic, static, maze-like shapes	non-interactive, deterministic, static, tunnel	non-interactive, deterministic, static, maze-like shapes
Ch. 6.2, [227]	interactive, deterministic, moving	non-interactive deterministic moving	interactive, stochastic, moving
Ch. 6.3, [229]	interactive, deterministic, moving	non-interactive deterministic moving	interactive, stochastic, moving
Ch. 7.1, [226]	non-interactive, deterministic, moving, adaptive	N/A	non-interactive, deterministic, moving, adaptive
Ch. 7.2, [224]	interactive, stochastic, moving	interactive, deterministic, moving	interactive, stochastic, moving

Table 3.3: Comparison of the types of SV and obstacles.

context of games related to autonomous driving but still yields a reasonable approximation, as, for instance, in [90].

Obstacle and surrounding vehicle types used in this thesis. In Tab 3.3, a comparison of the obstacle types used within the main contributions of this thesis is given. Mostly, deterministic obstacles were used, possibly with interaction models. Stochastic obstacle models often have the disadvantage of a higher computational burden, which increases the required sampling time.

Chapter 4

Software and Hardware Environments

This thesis comprises contributions that are specific to optimization-based motion planning and collision avoidance on highways, in racing scenarios, or generic for general motion planning and control problems on roads. Therefore, also the testing environments differ. For highway scenarios, the testing environment `CommonRoad` [13] was used that utilizes the traffic simulator `SUMO` [155] as a backend. Algorithms that targeted problems in racing environments were tested on an embedded autonomous driving (AD) software stack of the Autonomous Racing Graz (ARG) in the real-world racing series Roborace [233] and on a custom simulation environment. The racing series organizer Roborace [233] stopped operating in 2022.

The following introduces and summarizes the different environments in the overview Table 4.1.

4.1 Autonomous Racing Graz Stack

In the years 2020 and 2021, the race car series Roborace [233] conducted two racing series for autonomous vehicles on real race tracks where seven international teams competed with their software solutions. The event organizers provided the teams with real-sized vehicles named Devbot 2.0, including the electrical and mechanical framework, cf, Fig. 4.1. Moreover, an onboard computing platform was provided with basic functionality. The teams had to develop their compatible software stack. The two works of Sect. 6.1 and 7.1, corresponding to the papers [225] and [226], emerged from these racing series

Chapter, Ref.	Testing Environment	Type
Sect. 5.1, [222]	custom Python-based	generic
Sect. 5.2, [228]	custom Python-based	generic
Sect. 6.1, [225]	Nvidia Drive PX2, Autoware-ARG ROS [247]	racing
Sect. 6.2, [227]	CommonRoad [13], SUMO [155]	traffic
Sect. 6.3, [229]	CommonRoad [13], SUMO [155]	traffic
Sect. 7.1, [226]	Nvidia Drive PX2, Autoware-ARG ROS [247]	racing
Sect. 7.2, [224]	custom Python-based	racing

Table 4.1: Comparison of simulation environment is given for each Chapter and contribution. The following abbreviations are used: ARG (Autonomous Racing Graz). The testing environments and Chapters target different objectives of AD and are clustered into three types: generic, i.e., independent of the environment, racing, and traffic scenarios.

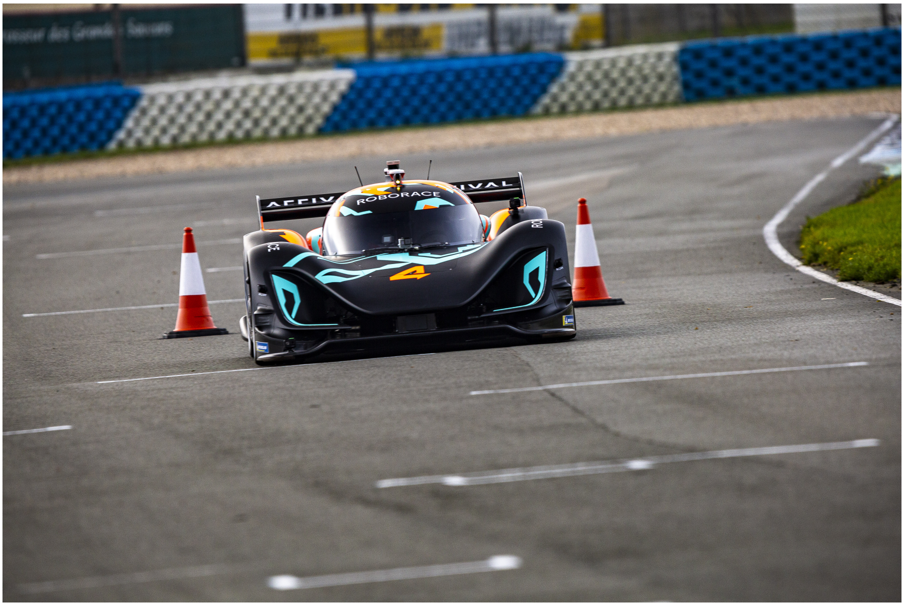

Figure 4.1: The Roborace vehicle Devbot 2.0 used in real-world races in the years of 2020 and 2021.

within the ARG team and were tested in real-world events. In the following, the software stack of the ARG team is introduced along the lines of a more detailed overview in [247].

Hardware. The Devbot 2.0 vehicle was equipped with sensors and actuators that were controlled by electrical control units (ECUs) and provided by the organizers of Roborace. The data was transmitted via Ethernet and the control area network (CAN) bus. Two distinct platforms were used. First, a real-time platform with a Speedgoat Unit Real-Time Target Machine was established that meets the desired real-time requirements for the low-level control and estimation algorithms. Secondly, the central ARG computing platform comprises an NVIDIA Drive PX2 hardware. The NVIDIA hardware is based on an ARM64 controller architecture.

Software. The functional components of the software are split into sensing, perception, planning, and control. The control is part of the real-time software for the Speedgoat Unit Real-Time Target Machine and was developed in Matlab/Simulink utilizing the Matlab code generation tools. The sensing, perception, and planning modules are part of the ARG software stack [247]. The sensing module comprises the interaction with the sensor interfaces, including GPS and V2X communication. The V2X communication was used to broadcast information about virtual obstacles. The received data was processed via an object detection module. The perception module involves processing sensor data to obtain a state estimation of the ego vehicle and surrounding vehicles (SVs). Additionally, the perception module involves object prediction, which is the central part of Sect. 7.1 and the publication [226]. The planner modules comprise the trajectory planner based on the solver `acados` [291] for optimal control problem (OCP)-structured nonlinear programs (NLPs). For combinatorial challenging obstacle avoidance problems, an additional mixed-integer linear programming planner was used to determine the homotopies relevant for the lower-level planner, cf., Sect. 6.1. In addition to the functional modules introduced above, a substantial system diagnosis module was employed to monitor and diagnose the software framework. The ARG software stack operates on an Ubuntu 22.04 operating system, utilizing ROS 2 [174] Humble Hawksbill and CycloneDDS [4] and Iceoryx [5] for shared memory communication.

Autonomous driving stack configurations. Software development requires various settings for rapid development, as real-world experiments are expensive and labor-intensive. Therefore, the AD stack was developed in four different phases that differ mainly in the simulated vehicle and the computing platform. First, a ROS-based version of the stack using a custom simple kinematic single-track model was utilized. The vehicle model was simulated by integrating the dynamics with an RK4 integration step. This framework was utilized on the local machines of the developers in a Docker container for rapid development. In the second step, the simple kinematic single-track model was replaced by the Hive Simulator, a higher-fidelity simulator that the Roborace organizers provided. In the next step, the hardware was aligned with the real-world environment. Therefore, the ARG stack was executed on the destination hardware NVIDIA Drive PX2, and the low-level control was executed on the Speedgoat platform. The final and most expensive stage involved testing the software on the real vehicle. This testing was usually performed remotely with local support from the Roborace development team.

4.2 CommonRoad Interactive Simulation Environment

Interactive traffic was simulated using the higher-level software framework `CommonRoad` [13], which provides a Python-based interface to the traffic simulator `SUMO` and a library of planning problem scenarios. The `CommonRoad` environment comprises several modules for motion planning tasks. Its main objective is to provide composable benchmarks for researchers to evaluate their motion planners. The main modules involve motion planning utilities, scenario creation, implemented motion planners, driving stack interfaces, simulator interfaces, and a reinforcement learning environment. In this thesis, mainly the simulator interface [150] to `SUMO` [170], the provided benchmarks and utility functions for the evaluation and animation of those scenarios were used. The finite number of benchmarks of `CommonRoad` are typically composed of fixed motion planning problems that involve deterministic traffic, a start, and a goal region or task. The problem setup requires a strong alignment of the user software with the `CommonRoad` workflow. This allows to compare ego motion planning results with other motion planners from the community.

For most motion planning problems relevant to this work, no previous benchmark results were available. The rigorous evaluation of the proposed planners required testing on a large number of randomized scenarios to detect potential crashes and evaluate the average closed-loop performance. Consequently, the interactive simulations were adapted to randomize the traffic and the initial state. Moreover, a custom goal formulation according to the specific problem of lane-changing in [227] or maintaining traffic rules and a target speed at multi-lane highway scenarios [229] was implemented.

4.3 Custom Python-Based Environment Vehiclegym

As part of this thesis, a custom Python-based development environment was developed that features a modular approach to rapidly test and evaluate motion planning and control algorithms. The environment is open-source available at github.com/RudolfReiter/vehicle_gym.

The environment consists of a core library with dedicated object-oriented modules that maintain certain targets of the framework, cf., Fig. 4.2 and a reinforcement learning (RL) module that targets interfaces to machine learning libraries and learning-based decision-making algorithms.

Core vehiclegym environment. The core `vehiclegym` environment is used to test numerical algorithms related to motion planning without utilizing learning-based interaction. In this thesis, the core environment is used in Sect. 5.1 and 5.2. Two types of input data files are used. One file defines the road geometry by waypoints, and another set of files defines vehicle parameters, such as their geometry, weight, and friction coefficients. The road definition files comprise real-world race tracks obtained from the Roborace [233] ecosystem. Vehicle parameters are either obtained from Roborace for their custom vehicle Devbot 2.0 or from the `CommonRoad` framework [13]. At the framework's core, a simulator can simulate an arbitrary number of interacting vehicles. The simulator requires several objects, i.e., a road object, MPC modules for each simulated vehicle, and a simulation model which can either be a symbolic `CasADi` [14] expression or a `CommonRoad` vehicle module. The simulator simulates one time step of each vehicle model and subsequently broadcasts all vehicle states to the corresponding MPC modules.

The MPC modules then perform a prediction of each other vehicle and compute their subsequent controls. If a symbolic model is used, the model is simulated by an RK4 step with the specified sampling time. In case a `CommonRoad` model was used, the `CommonRoad` simulator is utilized. Each MPC module requires a symbolic `CasADi` expression in order to construct an NLP-MPC solver with `acados` [291] which is compiled into a c-code encapsulation. The MPC module, the simulator, and, possibly, the vehicle model in Frenet coordinates require to be instantiated with the road object. The road object contains splines that represent the road center line and the left and right road boundaries.

Moreover, the road object provides convenience functions. For example, a random road can be generated by randomizing the curvature and the road boundaries in a specific interval. The road randomization, along with the randomization of initial vehicle positions is essential to evaluate the corresponding algorithms in this thesis in various random settings. Evaluation

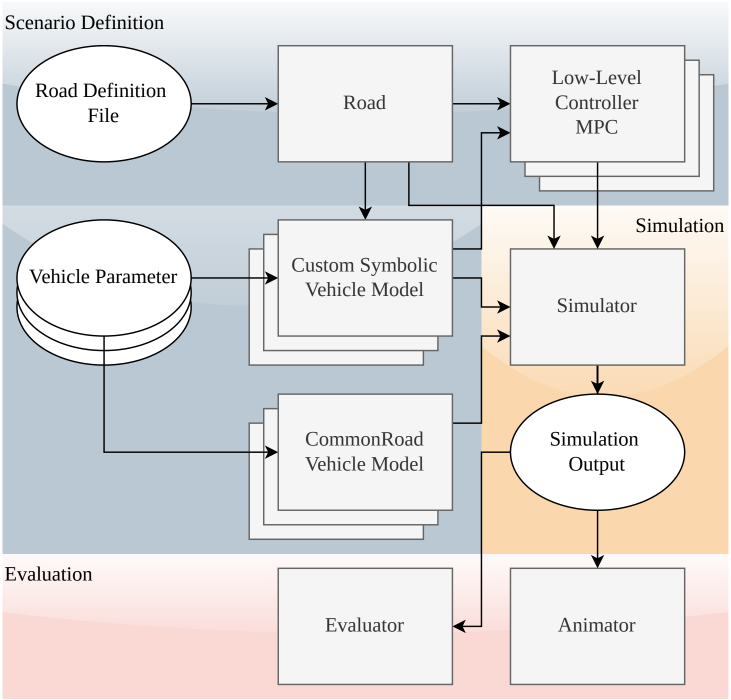

Figure 4.2: Modules involved in the `vehiclegym` environment. Data files are shown as round ellipses, and classes within the modules are rectangular. The RL environment arranges modules from the core `vehiclegym` environment in a classical RL composition.

is performed by two modules, an animator for qualitative comparison and quantitative evaluations of different performance indicators such as the CPU computation time of the MPC optimizer, the final progress and potential collisions.

Vehiclegym reinforcement learning environment. The `vehiclegym-rl` environment uses the modules from the `vehiclegym` to compose an OpenAI gym environment [50], cf., Fig. 4.3. The `vehiclegym-rl` environment comprises various different agents that are simulated simultaneously. Two agent types, i.e., a conventional MPC agent as described above and a lane-keeping agent with a decoupled speed and lateral lane-keeping control, are simulated as part of the environment without trainable parameters. Two further agent types contain learnable parameters θ, implemented as `PyTorch` [202] neural networks (NNs). First, a hierarchical agent type along Chapter 7.2 and the related work [224] is provided, comprising a model predictive control (MPC) lower-level and an RL higher-level planner. Secondly, a pure NN-based RL agent is provided. The `vehiclegym-rl` framework is derived from the OpenAI gym environment [50] and, thus, provides the basic RL functions to interact with the environment. Therefore, a reward function and an observation function are deployed within the `vehiclegym-rl` framework. Both functions are crucial for the training performance of the learnable agents. The RL interaction is further wrapped in a `hydra` decorator to manage the RL training, which involves an abundance of hyper-parameters. The RL algorithms are obtained from the `StableBaselines-3` library [214].

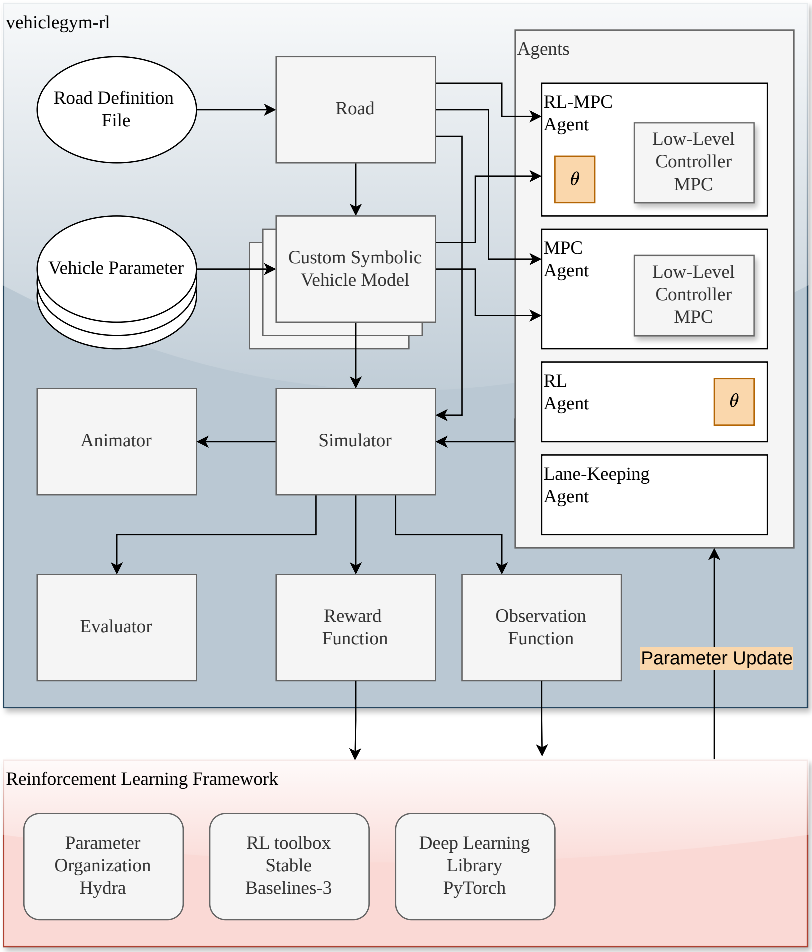

Figure 4.3: Modules involved in the core `vehiclegym-rl` environment. Data files are shown as round ellipses, and classes within the modules are rectangular. The modules are clustered into scenario definition modules, simulation modules, and evaluation modules.

Chapter 5

Model Formulations for Optimization-Based Motion Planning

Using derivative-based numerical optimization for solving motion planning and control problems requires appropriate model formulations. Linear models would be favorable since they pose a necessary condition to formulate convex programs, which usually can be solved efficiently [46]. However, linear models do not sufficiently approximate the real-world model to achieve the required closed-loop performance. Smooth nonlinear models can be used as part of a nonlinear program (NLP). The performance of NLP solvers significantly depends on the particular nonlinearities imposed by the chosen model. In Sect. 5.1, an approach to parameterizing a curvilinear model is presented, which improves the convergence properties of the NLP solver and, therefore, the closed-loop performance of a model predictive control (MPC) that requires the solution of the NLP in each iteration. Furthermore, the feasible state-space related to obstacle avoidance constraints is often formulated nonconvex to allow for a larger feasible planning space. However, when the infeasible set covering the obstacle is formulated convex, the nonconvex formulation of the feasible set can yield favorable numerical properties for NLP solvers. In Sect. 5.2, a novel model formulation is proposed that includes convex obstacle sets, along with a curvilinear projection on the Frenet coordinate frame (FCF). This curvilinear projection enables a straightforward formulation of typical autonomous driving (AD) objectives. The model extends the state-space to two coordinate frames.

5.1 Parameterization Approach of the Frenet Transformation for Model Predictive Control of Autonomous Vehicles

---

In this section, the paper published in [222] is reprinted with permission of Moritz Diehl. Note that the formatting of some formulas, terms, and numbers has been slightly adjusted for consistency without changing their meaning or content.

The contributions of each author are listed in the following.

Rudolf Reiter:	*idea, programming, design of the experiments, writing of the document*
Moritz Diehl:	*mathematical corrections, stylistic corrections, linguistic improvements*

©2021 European Control Association. DOI: 10.23919/ECC54610.2021.9655053

Abstract. Model predictive control (MPC) and nonlinear optimization-based planning for autonomous vehicles are often formulated in a transformed coordinate frame, namely the curvilinear Frenet frame. Mostly, the center line of the road is used as a transformation curve, but the choice of the transformation curve might have properties that make the optimization problem hard or even infeasible to solve in the whole search space. This paper proposes an optimization-based parameterization approach to establish an alternative transformation curve that yields favorable numerical properties for the consecutive use of numerical optimization approaches such as MPC. The optimization objective minimizes the change of curvature and pushes the evolute (i.e., singular region) of the transformation curve outside the feasible region. The convergence improvement of the proposed parameterization approach in terms of integrator precision, optimization time, and iteration counts is compared in simulation examples using a time-optimal nonlinear optimization formulation.

5.1.1 Introduction

In the last decade, nonlinear optimization-based approaches for both trajectory planning and control of autonomous vehicles have been investigated actively in scientific research and real-world applications ([100, 199, 167]). Nonlinear optimization helps to either perform control tasks which respect nonlinearities

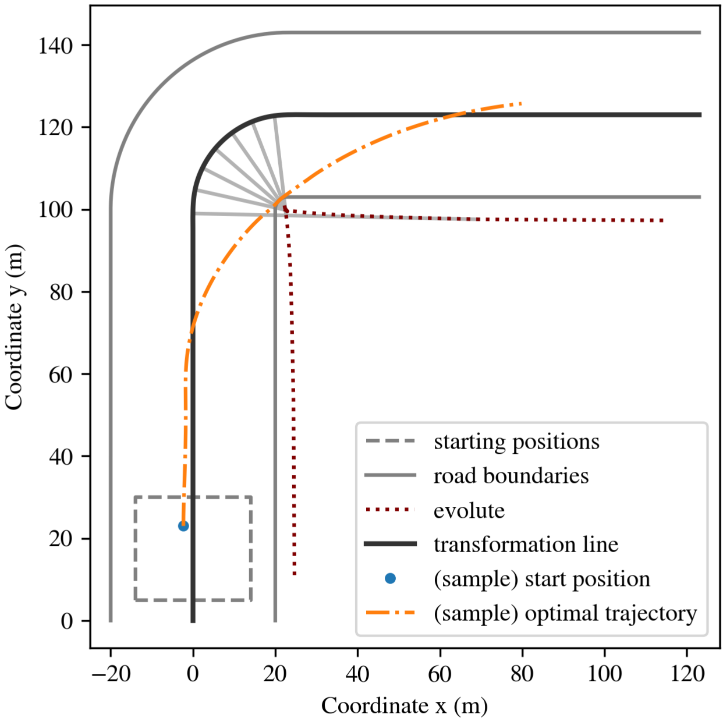

Figure 5.1: Center line transformation.

and constraints or plan optimal trajectories or paths for motion control systems. Many of the existing approaches use a particular state transformation, which maps the Cartesian states to a road-aligned path formulated as a curve in the Cartesian frame. The transformed frame is referred to as "Frenet frame" and used for example in [151, 195, 65, 293, 292]. This leads to favorable properties of the resulting system model regarding optimization objectives and the structure of the resulting NLP. The formulation of maximum path progress with a constant time horizon, which corresponds to time-optimal planning, is straightforward by maximizing the system state of path progress, and the tracking of the transformation curve can be achieved easily by minimizing the lateral distance state. Also, the motion of other traffic participants is typically aligned with the road, which can be integrated easily into the constraints. The choice of the road center line as a transformation curve comes naturally since the reference curve and the road boundaries are often parallel to the center line. Other traffic participants also often move along center-lane-aligned trajectories. This usual choice of the transformation curve is shown in Fig. 5.1. Due to these advantages, the center line is widely taken as the transformation to the road geometry, and restrictions are put on the road geometry itself to make this transformation unique and free of singularities [151, 292]. In contrast to academic cases, real-world scenarios come with several differences, which make the choice of the transformation curve a design parameter or even require a necessary adaption so that the whole road space is feasible for the system states. So far, no work has considered this transformation as a design parameter; it was rather taken as given input. Given a point on a curve, an osculating circle describes a circle that has the same tangent as the curve in this point and which has the same curvature (Fig. 5.7 shows the osculating circle in point p_i). The curve, describing the evolution of the center of the osculating circles, is referred to as evolute. As a hard constraint for the choice of the transformation curve, the road boundary perpendicular to the transformation curve must be closer than its oriented radius of all osculating circles of the planar transformation curve. Since an NLP is an approximation of the optimal control problem (OCP) and approximated with a discretization in time, the distance of the road boundary to the evolute must even be raised by a certain factor to achieve numerical robustness. By transforming the vehicle model into the Frenet coordinates, the curvature becomes part of the dynamic system. Consequently, the nonlinearity of the curvature also becomes part of the vehicle model. Besides of the hard constraint regarding the singularity emerging from the curvature, this work also addresses further favorable properties of the transformation choice. The dynamic state equations (including the curvature) enter the NLP by equality constraints and at least one derivative is used by the solver. Good convergence properties are achieved if higher-order derivatives of the NLP constraints can be lowered, and this is performed by the presented parameterization of the transformation curve. An example of the parameterized curve is shown in Fig. 5.2. For known tracks (e.g., race tracks, known road networks), the transformation can be computed offline in advance.

5.1.2 System Model

Single Track Model

A kinematic model in a curvilinear reference frame is used, which was presented first in [302] and leads to a slip-free tire model. The direction of the movement of the center of gravity (CG) with the vehicle mass m is given by the angle $\psi + \beta$, where ψ is the vehicle orientation. The side-slip angle β is defined as depicted in Fig. 5.3 and gives the relative angle of motion related to the vehicle coordinate system. The side-slip angle is given by

$$\beta = \arctan\left(\frac{l_\mathrm{r}}{l_\mathrm{r} + l_\mathrm{f}} \tan\delta\right). \tag{5.1}$$

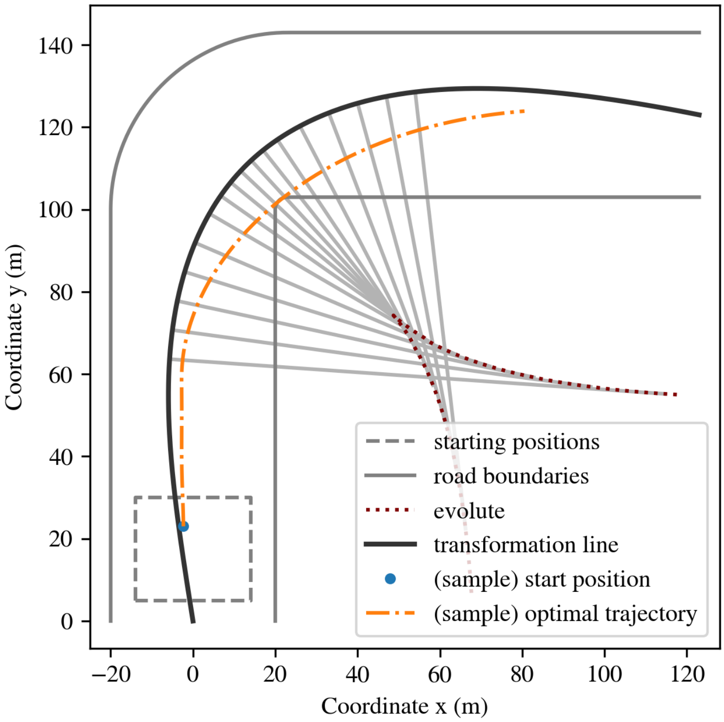

Figure 5.2: Suggested transformation with objective as in (5.13).

The system model can then be described by the following Equations (5.2) in the Cartesian coordinate frame. The velocity vector v denotes the velocity related to the CG.

$$\dot{p}_{\mathrm{X}} = v\cos(\psi+\beta) \tag{5.2a}$$

$$\dot{p}_{\mathrm{Y}} = v\sin(\psi+\beta) \tag{5.2b}$$

$$\dot{\psi} = \frac{v}{l_{\mathrm{r}}}\sin\beta \tag{5.2c}$$

$$\dot{v} = \frac{F_{\mathrm{x}}^{\mathrm{d}}}{m}\cos\beta \tag{5.2d}$$

The geometry of the vehicle is simply described by the longitudinal position of the CG with the front distance l_{f} and the rear distance l_{r}. The input force $F_{\mathrm{x}}^{\mathrm{d}}$ only acts on the rear wheel, whereas the steering angle δ only deflects the

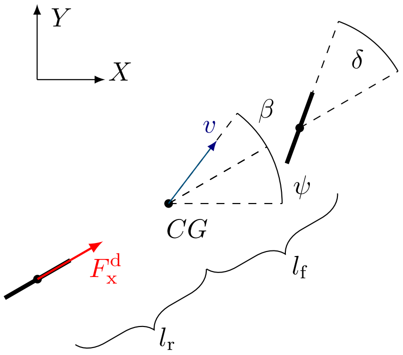

Figure 5.3: Kinematic single-track model.

front wheel, which is an arbitrary choice and does not influence the proposed algorithm.

Curvilinear Transformation

So far the system model is independent of any road geometry, but as pointed out, a useful transformation leads to the vehicle system equations in the Frenet frame. It leads to the dynamic system (5.3) with the states $x = \begin{bmatrix} s, & n, & \alpha, & v \end{bmatrix}^T$ and controls $u = \begin{bmatrix} F_{\mathrm{x}}^{\mathrm{d}}, & \delta \end{bmatrix}^\top$, which now depend on the curvature. Here, path-aligned states are used, which describe the progress on the transformation path $s(t)$, the normal distance to the transformation path $n(t)$, and the heading angle mismatch $\alpha(s,t) = \psi(t) - \psi^{\mathrm{c}}(s)$. In many works, e.g., [292, 151], this transformation is performed along the center line, which is generally not the case here.

$$\dot{s} = \frac{v\cos(\alpha+\beta)}{1-n\kappa(s)} =: f_s(x) \tag{5.3a}$$

$$\dot{n} = v\sin(\alpha+\beta) \tag{5.3b}$$

$$\dot{\alpha} = \dot{\psi} - \kappa(s)\dot{s} =: f_\alpha(x) \tag{5.3c}$$

$$\dot{v} = \frac{F_{\mathrm{x}}^{\mathrm{d}}}{m}\cos(\beta) \tag{5.3d}$$

The equations can be summarized by the nonlinear dynamic system equations of first order.

$$\dot{x} = f(x,u) \tag{5.4}$$

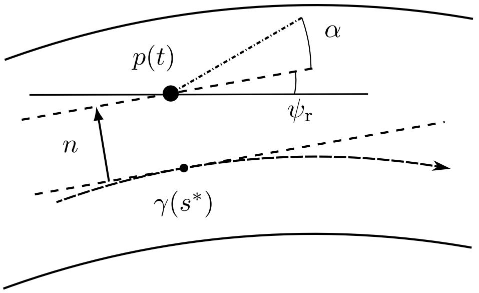

Figure 5.4: Path-parametric model as in [151].

Note that the curvature $\kappa(s)$ together with bounds on the normal distance state n now fully describe the road geometry. Fig. 5.4 shows the transformation of a point to the curve $\gamma(s)$, normal distance n, the error angle α and the heading angle of the reference ψ_{r}.

5.1.3 Newton-Type Optimization

As described in [151], the time-optimal racing problem can be described very generally by the following multiple shooting NLP and allows the usage of a Gauß-Newton Hessian approximation but might also be solved with an exact Hessian [217]. It is important to emphasize that the presented approach shifts the reference line, which might not be desired for certain applications. For those applications, a reference offset could be used, but it is out of the scope of this work. The time dependence of the states x and controls u is discretized with fixed time intervals Δt and an integration scheme $F(x_k, u_k, \Delta t)$. Equation (5.5e) puts constraints on the states, which can depend on the path variable s as well as on the time index k, to account for time-varying constraints like moving obstacles.

$$\min_{\substack{x_0,\dots,x_N,\\ u_0,\dots,u_{N-1}}} \quad \sum_{k=0}^{N-1} \|x_k - x_{k,\mathrm{ref}}\|_Q^2 + \|u_k\|_R^2 + \|x_N - x_{N,\mathrm{ref}}\|_{Q_{\mathrm{N}}}^2 \tag{5.5a}$$

$$\text{s.t.} \quad x_0 = x_{\mathrm{c}}, \tag{5.5b}$$

$$x_{k+1} = F(x_k, u_k, \Delta t), \quad k = 0, \dots, N-1, \tag{5.5c}$$

$$\underline{u} \le u_k \le \overline{u}, \quad k = 0, \dots, N-1, \tag{5.5d}$$

$$\underline{x}_k(s_k) \le x_k \le \overline{x}_k(s_k), \quad k = 0, \dots, N-1, \tag{5.5e}$$

Newton-type algorithms use first-order (Gauß-Newton Hessian) or second-order derivatives (exact Hessian) of the constraints, which are particularly interesting for (5.5c) and (5.5e), since both equations are influenced by the chosen representation of the road geometry $\kappa(s)$. Since (5.5e) is, in general, driving scenarios not exactly known a priori, it can just be assumed that obstacles move and their geometry is aligned along the center line. Nevertheless, the model integration (5.5c) is of fixed structure, which can be exploited in order to parameterize the transformation of the road geometry, resulting in $\kappa(s)$. By taking a closer look at the system dynamics 5.3a, the function value is heading towards infinity if the normal distance of the transformation line n is close to the reciprocal value of $\kappa(s)$, which corresponds to the radius of curvature. The curvature ratio $\rho(s,n)$ can be defined as the ratio of the lateral distance n to the radius of the osculating circle $R(s) = \frac{1}{\kappa(s)}$, which yields $\rho(s,n) = \frac{n}{R(s)} = n\kappa(s)$.

5.1.4 Singularity and Smoothness Problem

By symbolically computing the first (5.6) and second (5.7) partial derivatives of the state s, it is further obvious that the denominators have quadratic dependence on the nonlinearity described above, which make the resulting Jacobian of the constraints arbitrarily ill-conditioned if the curvature ratio $\rho(s,n)$ is sufficiently close to 1. Equation (5.6) shows that higher order derivatives for the differential equations of the integrator in (5.5c) can be lowered by reducing the maximum possible value for the denominator, which is the main subject of the presented parameterization approach. According to [194], it can generally be assumed that lowering higher order derivatives increases convergence properties of usually applied numerical solvers.

$$\frac{\partial f_s(x)}{\partial s} = \frac{v\cos(\alpha+\beta)n\kappa(s)'}{(1-n\kappa(s))^2} \tag{5.6a}$$

$$\frac{\partial f_\alpha(x)}{\partial s} = -(\kappa(s)' f_s(x) + \kappa(s)\frac{\partial f_s(x)}{\partial s}) \tag{5.6b}$$

$$\frac{\partial^2 f_s(x)}{\partial s^2} = \frac{v\cos(\alpha+\beta)n((1-n\kappa(s))\kappa(s)'' + 2n(\kappa(s)')^2)}{(1-n\kappa(s))^3} \tag{5.7a}$$

$$\frac{\partial^2 f_\alpha(x)}{\partial s^2} = -\kappa(s)'' f_s(x) - 2\kappa(s)'\frac{\partial f_s(x)}{\partial s} - \kappa(s)\frac{\partial^2 f_s(x)}{\partial s^2} \tag{5.7b}$$

Also the partial derivative $\kappa(s)'$ should stay small to reduce higher order derivatives.

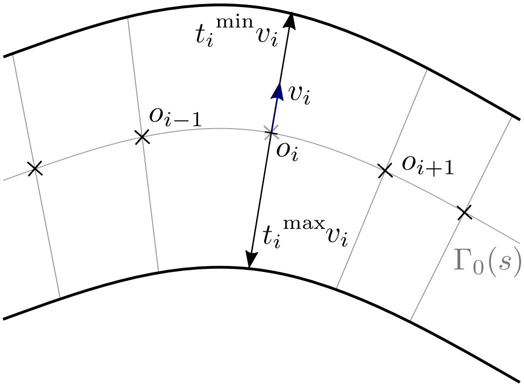

Figure 5.5: Geometry of waypoint borders.

5.1.5 Optimal Curvilinear Parameterization

Using the previously defined weaknesses of the center line transformation approach (i.e. mainly the singularities close or inside the feasible region) as costs, a novel parameterization approach is presented, which is formulated as an NLP. By means of discrete geometric considerations, the singularity is pushed outside the feasible region, and the curvature along the path length variable is smoothed while the distance to the center line is kept as close as possible. The solution transformation not only guarantees that no singularities are located inside the feasible region but also pushes these very nonlinear regions outside the feasible region as far as possible. Here, the waypoints $o_i \in \mathbb{R}^2$ describe N discrete points of a given piece-wise linear reference curve $\Gamma_o(s)$ in the Cartesian frame, depending on the path length s. The vector $v_i \in \mathbb{R}^2$ represents the tangent norm vector, with $\|v_i\| = 1$ pointing "left" facing the positive road direction, as shown in Fig. 5.5. The scalar value $t_i \in \mathbb{R}$ is used as the i-th element of the optimization variables $t = [t_0, ..., t_N]^T \in \mathbb{R}^N$ and shifts the reference point towards the road boundaries, resulting in the shifted point p_i (Fig. 5.6). The road borders are given as maximum or minimum deflection t_i of point p_i into direction v_i, which are denoted by $t_i^{\max}$ or $t_i^{\min}$. At the discrete total path length s_i of the piece-wise linear reference path, it holds that $\Gamma(s_i) = p_i$. The distance vector between point p_i and p_{i+1} is given by h_i, where $\Delta s_i = s_{i+1} - s_i = \|h_i\|$.

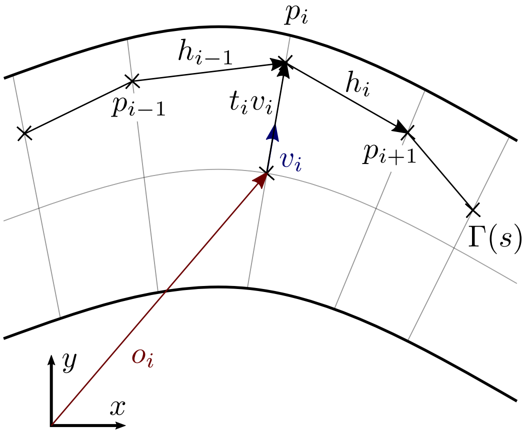

Figure 5.6: Geometry of waypoint displacement.

$$p_i = \begin{bmatrix} p_{\mathrm{x},i} \\ p_{\mathrm{y},i} \end{bmatrix} = o_i + t_i v_i \tag{5.8}$$

$$h_i = \begin{bmatrix} h_{\mathrm{x},i} \\ h_{\mathrm{y},i} \end{bmatrix} = p_{i+1} - p_i \tag{5.9}$$

As shown in [259], the discrete curvature κ_i for unequally spaced points can be computed by means of the first central and second finite differences, with the notation of D_i^1 for first, D_i^2 for second discrete derivatives and the composition of $D_i^n = \begin{bmatrix} D_{x,i}^n, & D_{y,i}^n \end{bmatrix}^T$. Another method to compute the discrete curvature would be the method of osculating circles as shown in [124].

$$\begin{aligned} D_i^1 :=& -\frac{\Delta s_i}{\Delta s_{i-1}(\Delta s_i + \Delta s_{i-1})} p_{i-1} - \\ & \frac{\Delta s_i + \Delta s_{i-1}}{\Delta s_i \Delta s_{i-1}} p_i + \frac{\Delta s_{i-1}}{\Delta s_i(\Delta s_i + \Delta s_{i-1})} p_{i+1} \end{aligned} \tag{5.10a}$$

$$D_i^2 := 2\frac{\Delta s_i p_{i-1} - (\Delta s_i + \Delta s_{i-1}) p_i + s_{i+1} p_{i+1}}{\Delta s_{i-1} \Delta s_i (\Delta s_i + \Delta s_{i-1})} \tag{5.10b}$$

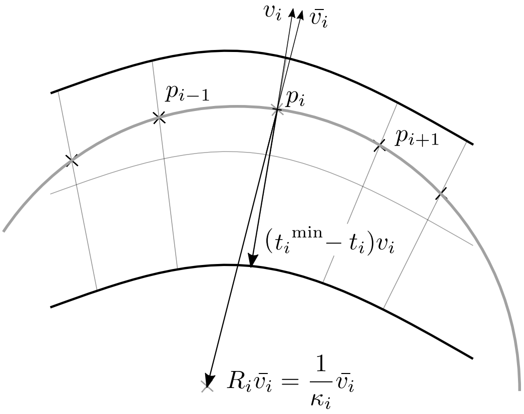

Figure 5.7: Geometry of osculating circle.

$$\kappa_i(t_{i-1}, t_i, t_{i+1}) = \frac{D^1_{\mathrm{x},i}D^2_{\mathrm{y},i} - D^1_{\mathrm{y},i}D^2_{\mathrm{x},i}}{((D^1_{\mathrm{x},i})^2 + (D^1_{\mathrm{y},i})^2)^{\frac{3}{2}}} \tag{5.11}$$

The discrete curvature is now used to formulate an overall objective, which is the sum of several objectives. First, the maximum value of the ratio of the "inner" (the side where the curve bends) boundary distance $t_i^{\min} - t_i$ if $\kappa_i < 0$ or $t_i^{\max} - t_i$, if $\kappa_i > 0$, to the signed radius of curvature $R_i = 1/\kappa_i$ is minimized by means of the slack variable $\bar{\rho}$. The osculating circle to point p_i is sketched in Fig. 5.7, which also shows the maximum deflection $t_i^{\min} - t_i$ of the reference curve $\Gamma(s)$ at point $s = s_i$. Note that the vector to the center of the osculating circle, which is a multiple of the new tangent vector $\bar{v}_i$ to $\Gamma(s_i)$ generally does not need to be parallel to the tangent vector v_i of the initial curve $\Gamma_0(s_i)$. Nevertheless, it is assumed that the vectors v_i and $\bar{v}_i$ are equal since the mismatch is generally small if the curve Γ_0 is initialized properly, i.e. close to the center line. The slack variable $\bar{\rho}$ bounds the maximum allowed value of the curvature ratio $\rho = n\kappa$ and is also bounded by a value $\bar{\rho}_{\max}$, which should be between 0.5 and 0.95 and penalized by a cost $T_\rho(\bar{\rho})$. The optimization figuratively pushes the evolute shown in Fig. 5.2 off the boundaries and guarantees that they do not intersect by the constraint on $\bar{\rho}$. The parameter w_ρ contributes to the shape of the penalty function for relevant values $0 < \bar{\rho}_i < \bar{\rho}_{\max}$ and might also be set to zero, if $\bar{\rho}_{\max}$ is used conservatively, i.e., low values. It is noteworthy that this objective alone would lead to a flat minimum, which would result in nonsingular solutions. Secondly, the discrete derivative of the curvature is minimized and weighted by $w_{\mathrm{d}\kappa}$ in objective $T_{\mathrm{d}\kappa}$. Thirdly, a term $T_{\mathrm{dc}}(t)$ is added, which can be used to force the resulting transformation line towards the road center. This might be useful if obstacles parallel to the center line are added.

$$T_{\mathrm{d}\kappa}(t) = \sum_{i=1}^{N-2} \left(\frac{\kappa_{i+1}(t_i, t_{i+1}, t_{i+2}) - \kappa_i(t_{i-1}, t_i, t_{i+1})}{\|h_i(t_i, t_{i+1})\|} \right)^2 \tag{5.12a}$$

$$T_{\mathrm{dc}}(t) = \sum_{i=0}^{N} \left(\frac{t_i^{\min} + t_i^{\max}}{2} - t_i \right)^2 \tag{5.12b}$$

$$T_{\rho}(\bar{\rho}) = \sum_{i=1}^{N-1} \frac{\bar{\rho}_i}{1 - \bar{\rho}_i} \tag{5.12c}$$

$$\begin{aligned} \min_{\bar{\rho} \in \mathbb{R}^{N-1}, t \in \mathbb{R}^N} \quad & w_{\rho} T_{\rho}(\bar{\rho}) + w_{\mathrm{d}\kappa} T_{\mathrm{d}\kappa}(t) + w_{\mathrm{dc}} T_{\mathrm{dc}}(t) \\ \text{s.t.} \quad & t_i^{\min} \leq t_i \leq t_i^{\max} \qquad i = 0, \ldots, N, \\ & (t_i^{\min} - t_i)\kappa_i \leq \bar{\rho}_i \qquad i = 1, \ldots, N-1, \\ & (t_i^{\max} - t_i)\kappa_i \leq \bar{\rho}_i \qquad i = 1, \ldots, N-1, \\ & \bar{\rho}_i \leq \bar{\rho}_{\max} \qquad i = 1, \ldots, N-1 \end{aligned} \tag{5.13}$$

5.1.6 Simulation Results

Integration Error

Since the integration scheme which is used in order to obtain an NLP out of the OCP by direct multiple shooting is one of the most important contributors to the overall accuracy of the OCP solution, the improvement of the integration performance is shown in a simplified setting (Fig. 5.8). The vehicle model ($l_{\mathrm{r}} = 2\,\mathrm{m}$ and $l_{\mathrm{f}} = 1\,\mathrm{m}$) in the Cartesian frame (5.3) as well as the vehicle model in the Frenet frame (5.3a) are forward simulated with a constant speed of $1\,\frac{\mathrm{m}}{\mathrm{s}}$ and a constant steering angle of $\delta = -0.15\,\mathrm{rad}$. The "Frenet model" is transformed into the Frenet frame with respect to a transformation curve, given as a circle. This circular transformation curve leads to a constant curvature, and the center point of the circle represents the whole singularity. In subsequent experiments, this circle is displaced along a line that crosses the motion path of the vehicle. The displacement is characterized by the maximum curvature ratio $\rho_{\max}$, which is the maximum value of the curvature ratio $\rho(s, n)$ along the exactly simulated trajectory. This value would be equal to zero if the integrated

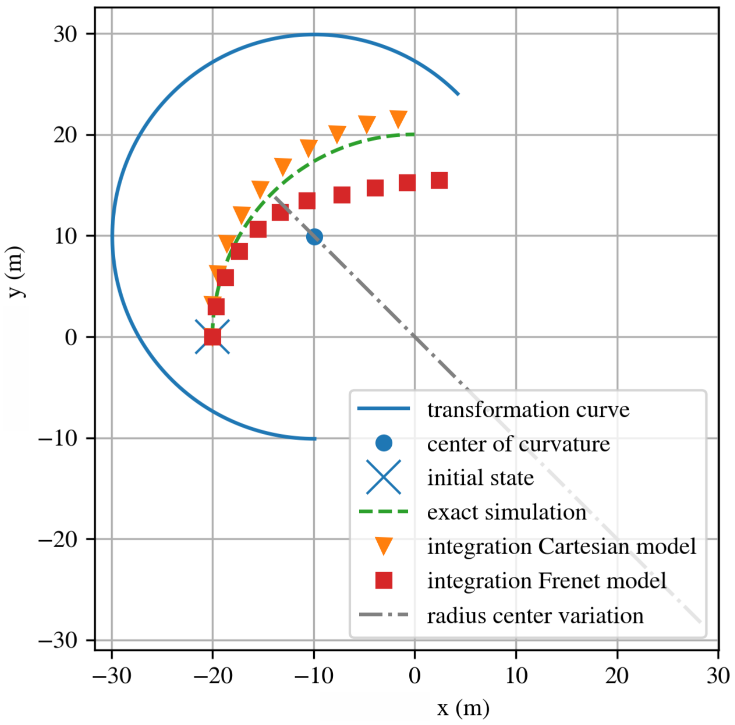

Figure 5.8: Integration setting.

trajectory lies exactly on the transformation curve (circle perimeter) and equal to one if the trajectory crosses the radius center (singularity). Two different integration schemes are used (Euler and Runge-Kutta-4 (RK4)) in order to demonstrate the superior integration result in terms of the end-point error to simulate a quarter-circle path exactly. Additionally, each integration scheme is performed with different step lengths. As a step size $\Delta t_0 = 2.6\,\mathrm{s}$ is used. In Fig. 5.9, the integration schemes are compared, dependent on the curvature ratio ρ for a certain step size. The Euler integration was performed with four times the number of steps than the RK4 integration to compare them related to their number of function evaluations. It can easily be verified that at some value of the curvature ratio ρ, each integration scheme shows high errors, and at some ratio ρ, the end-point simulation error is rather randomly close to the exact solution. Based on these observations, for a given integration scheme, it is obvious to set an upper border for the curvature ratio ρ in order to define the transformation curve.

The integration performs best if the simulated path is located exactly at the

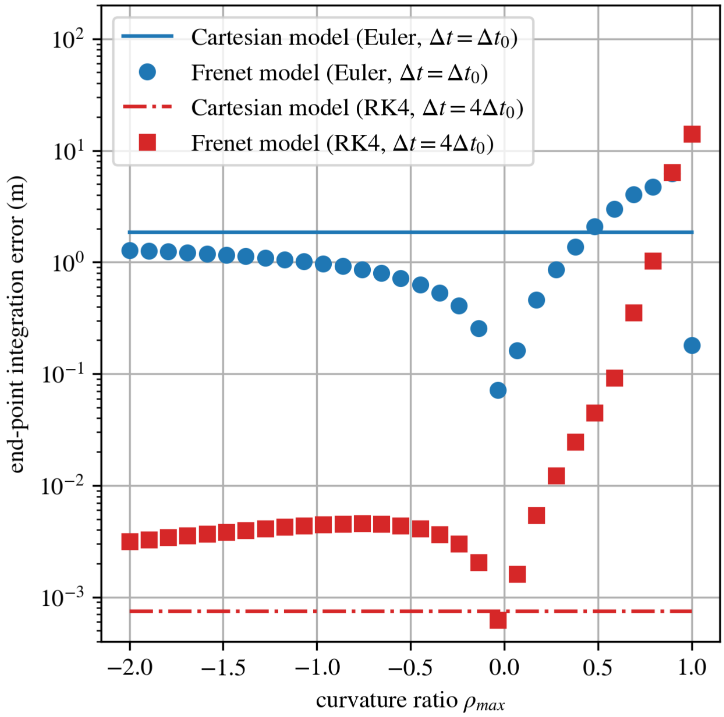

Figure 5.9: Integration errors.

transformation curve (curvature ratio $\rho = 0$) and performs poorly if the path is located close to the singularity (curvature ratio $\rho = 1$). For negative curvature ratios ρ, Euler integration always performs better with the Frenet model. Although, if the Runge-Kutta-4 integration scheme is used, the Cartesian model is superior, despite a curvature ratio of $\rho = 0$.

Model Predictive Control

To point out the superior properties of the presented approach for parameterizing the transformation curve, an exemplary task for time-optimal MPC is solved. The time-optimal trajectory for the curve in Fig.5.2 is obtained by solving an NLP of the form (5.5). Two different transformation curves are compared as shown in Fig. 5.1 for the center line and Fig. 5.2 for the numerically better behaving parameterized optimal transformation curve, which was obtained by solving (5.13) ($w_{\mathrm{dc}} = 10$, $w_\rho = 10$, $\bar{\rho}_{\max} = 0.7$ and $w_{\mathrm{d}\kappa} = 10^8$). The

Table 5.1: Comparison of transformation curves for MPC.

Statistical NLP value	center line	optimal transformation curve
solution time (mean)	611.1ms	470.9ms
SQP iterations (mean)	11.0	8.2
QP iterations (mean)	22.3	11.1
QP solver fails	40%	0%

parameters of the vehicle model were taken from a real race car model with wheel bases $l_\mathrm{r} = 1.4\mathrm{m}$, $l_\mathrm{f} = 1.6\mathrm{m}$, a maximum lateral and longitudinal acceleration of $5\frac{\mathrm{m}}{\mathrm{s}^2}$ for both. The OCP was discretized with a horizon $T = 9\mathrm{s}$ and a discretization time of $\Delta T = 0.05$ s. Other values are equivalent to [151]. For solving the NLP, `acados` [291] was used with an RK4 integrator performing one step per multiple shooting interval. The problem was solved with 40 different starting positions as marked in Fig. 5.1, and the resulting numerical differences are shown in Table 5.1. The presented parameterization results in superior numerical properties for all relevant performance measures. Note that the so-called “center curve” was also re-parameterized slightly in order to obtain smooth system differential equations ($w_\mathrm{dc} = 10^3$, $w_\rho = 10$, $\bar{\rho}_\mathrm{max} = 0.9$ and $w_{\mathrm{d}\kappa} = 10^6$). Without the presented parameterization, the solver failed in a significant number of simulation runs, which is another indicator that some form of parameterization, like the presented approach, is even necessary for most applications of the Frenet model MPC.

5.1.7 Conclusions

The paper presents a parameterization approach that is suggested for the use with Frenet transformations related to numerical optimization approaches for autonomous driving. It formalizes the problem of finding the transformation curve, which must not have singularities in the feasible driving region. A parameterization stated as an optimization problem, is presented, which ensures feasibility as well as superior numerical properties for Newton-type optimization. The performance increase is shown by means of a plain integration and a small but relevant test example. Further considerations and future work might focus on the related nonconvexity of the constraints, the implementation as a real-time capable NLP, or the consecutively needed transformation of the borders.

5.2 Frenet-Cartesian Model Representations for Automotive Obstacle Avoidance within NMPC

In this section, the paper published in [228] is reprinted with permission of Armin Nurkanović, Jonathan Frey and Moritz Diehl. Note that the formatting of some formulas, terms, and numbers has been slightly adjusted for consistency without changing their meaning or content.

The contributions of each author are listed in the following.

Rudolf Reiter:	*idea, programming, development of the mathematical foundations, design of the algorithm, design and evaluation of the experiments, writing of the document*
Armin Nurkanović:	*mathematical corrections, stylistic corrections, linguistic improvements*
Jonathan Frey:	*mathematical corrections, stylistic corrections, linguistic improvements, support with programming using the "acados" software tool*
Moritz Diehl:	*corrections to content, mathematical corrections, stylistic corrections, linguistic improvements*

©2023 European Control Association. Published by Elsevir Ltd. All rights reserved.
DOI: https://doi.org/10.1016/j.ejcon.2023.100847

Abstract. In recent years, nonlinear model predictive control has been extensively used for solving automotive motion control and planning tasks. In order to formulate the nonlinear model predictive control problem, different coordinate systems can be used with different advantages. We propose and compare formulations for the nonlinear MPC related optimization problem, involving a Cartesian and a Frenet coordinate frame in a single nonlinear program. We specify costs and collision avoidance constraints in the more advantageous coordinate frame, derive appropriate formulations, and compare different obstacle constraints. With this approach, we exploit the simpler formulation of opponent vehicle constraints in the Cartesian coordinate frame, as well as road-aligned costs and constraints related to the Frenet coordinate frame. Comparisons to other approaches in a simulation framework highlight the advantages of the proposed methods.

5.2.1 Introduction

Trajectory optimization with obstacle avoidance is a major challenge of motion planning and control in autonomous driving. Trajectories need to be feasible to kinodynamic equations and avoid collisions with objects that are often hard to predict. Collision avoidance and the related generation of a reference trajectory or collision avoidance as part of the controller, e.g., nonlinear model predictive control (NMPC), is often formulated as a nonlinear optimal control problem [151, 215, 167, 49]. Through a carefully chosen NLP formulation and by using dedicated real-time optimization solvers [291, 245], the problem can be solved efficiently. The transformation of the dynamics into a road-aligned coordinate frame (CF), namely the FCF, has shown many advantages, such as the simplification of references and road boundaries [151, 225, 302]. Nevertheless, the transformed coordinates also come with the disadvantage of transformed geometric obstacle shapes [310], cf. Sec. 5.2.2. Typical convex geometric shapes, such as boxes, ellipses, or circles, are easier to describe in the Cartesian reference CF and become nonconvex after transformation into the FCF. The shapes of objects in both frames are shown in Fig. 5.10. In nonlinear optimization, "lifting" is a technique where the optimization problem is formulated and solved in a higher dimensional space, which offers advantages regarding convergence rates and the region of attraction [10]. We contribute by an approach to extend and lift the state space of the vehicle model by including both CFs and formulate constraints and costs in the more appropriate CF. We show an increase in the overall performance due to the improved description of the obstacle shapes with various deterministic obstacle avoidance formulations in simulation. Despite the increased state dimensions, even the computation time can be lowered compared to a pure FCF representation. Additionally, references can be set in any of the two CFs, which allows for flexible combinations with planning modules that use either CF, e.g., [293].

Related Work

The effectiveness of NMPC using the FCF related to automotive tasks was shown in numerous works [151, 215, 225, 302, 293, 297, 60, 226, 166, 238, 301]. None of them explicitly considers the shape transformation of objects, which are rather over-approximated with convex shapes in the FCF. Convex obstacle shapes in the Cartesian coordinate frame (CCF) are considered in [216] with potential fields, in [297, 328] with covering circles and Euclidean distance constraints, in [49] with ellipses, in [52, 188] with separating hyperplanes and in [245, 92] with a formulation related to a conjunction of convex planes covering the obstacle. The most prominent variants are compared within this paper in the Frenet and the lifted formulation. More importantly, the shape-fitting problem with transformed objects in the FCF and an approach with surrogate representations in both CFs were recently considered in a related way in [310]. However, [310]

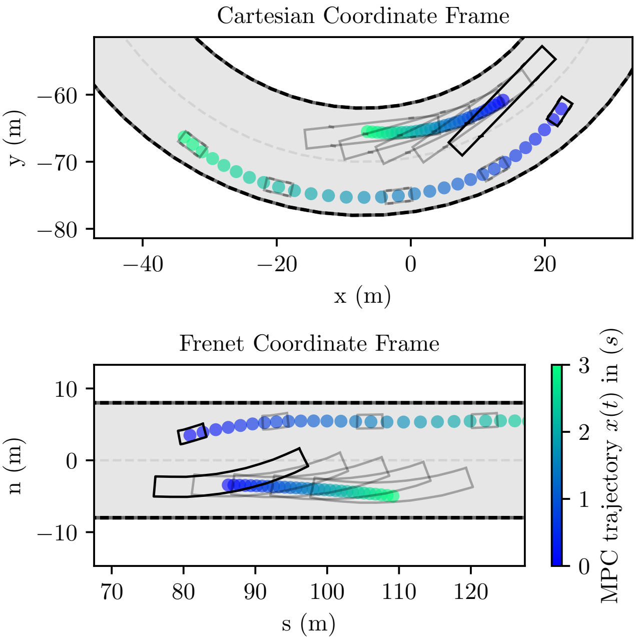

Figure 5.10: Simulated overtaking of the same maneuver shown in the Cartesian (top plot) and the Frenet CF (bottom plot). Planned trajectories plotted with $\Delta t = 0.1$s and snapshots of boundary box alignments every 0.7s.

focuses on linear MPC and does not consider dedicated obstacle formulations. Furthermore, it integrates an approximation of the transformation itself into the model, i.e., an approximation of a differential algebraic equation (DAE). In contrast, in our formulation, we provide a reduced index formulation, which constitutes an ordinary differential equation (ODE), cf. Sec. 5.2.2. Secondly, we eliminate algebraic variables directly. Another variant of tracking along a reference path stems from [157] and was extended to vehicles in [167]. It uses a method called *contouring control*, which uses a state on a path-length parameterized reference curve and an additional state for its path position. Similarly to [310], it considers the transformation implicitly, which involves approximating a bi-level optimization problem for finding the closest point on the reference curve.

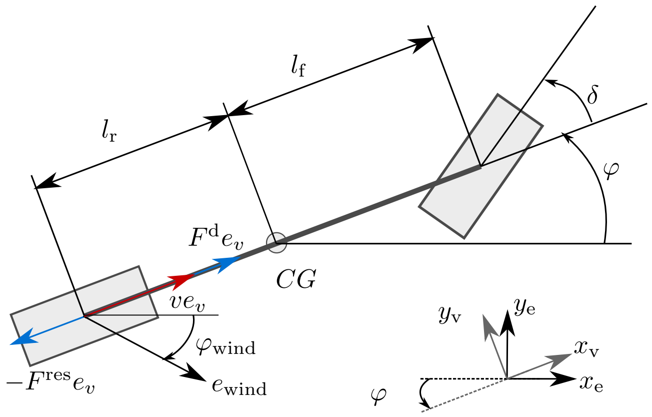

Figure 5.11: Kinematic vehicle model including wind drag in wind direction e_wind.

Contribution

We propose novel NMPC formulations that extend the state space to two CFs and allow for more efficient consideration of the occurring costs and constraints. Thereby, the usually convex and simple obstacle shapes in the CCF can be directly used in the NMPC formulation. We show in simulation that we outperform the conventional approach of over-approximation [166, 238, 215, 301] in terms of computation time and performance. Furthermore, the obstacle shapes are independent of the states and the road (up to Euclidean transformations), which is not valid in a conventional Frenet representation. Additionally, we contribute by making an extensive comparison of common obstacle avoidance formulations in the proposed formulations.

5.2.2 Vehicle Models

In order to formulate the NMPC problem, we use a rear axis referenced kinematic vehicle model of [151, 222], shown in Fig. 5.11. Given a high enough sampling time and excluding highly dynamic maneuvers (e.g., emergency turns), kinematic vehicle models perform similarly to dynamic models for many automotive motion planning problems [153]. The model comprises three states x^c related to the CF. Particularly, we use $x^\mathrm{c,C} = [x \quad y \quad \varphi]^\top \in \mathbb{R}^3$ for the Cartesian states and $x^\mathrm{c,F} = [s \quad n \quad \alpha]^\top \in \mathbb{R}^3$ for the Frenet states. We use the Cartesian (earth) position states x_e, y_e and the heading angle φ. Similarly, in the FCF, we use longitudinal and lateral position states s and n, together with the difference angle α (cf. Sec. 5.2.2 and Fig. 5.10). The FCF position states are curvilinear coordinates along the reference road. Further states $x^{\neg c} = [v \quad \delta]^\top \in \mathbb{R}^2$ are used for both, CCF and FCF, where v is the absolute value of the velocity at the rear axis and δ is the steering angle. For the full CCF model we use the state $x^{\mathrm{C}} = [x^{\mathrm{c,C}\top} \quad x^{\neg c\top}]^\top$ and for the FCF model we use the state $x^{\mathrm{F}} = [x^{\mathrm{c,F}\top} \quad x^{\neg c\top}]^\top$. We assume a rear wheel drive force F^{d} as input, including the acceleration and braking force, which is a valid approximation for small steering angles, cf. [151, 222]. The most prominent resistance forces for wind $F^{\mathrm{wind}}(v, \varphi) = c_{\mathrm{air}} v_{\mathrm{rel}}(v, \varphi)^2$ and the rolling resistance $F^{\mathrm{roll}}(v) = c_{\mathrm{roll}} \mathrm{sign}(v)$ are included, with the total resistance force $F^{\mathrm{res}}(v, \varphi) = F^{\mathrm{wind}}(v, \varphi) + c_{\mathrm{roll}} \mathrm{sign}(v)$. The air drag depends on the vehicle speed v in relation to the wind speed v_{wind} with the air friction parameter c_{air}. The rolling resistance is proportional to $\mathrm{sign}(v)$ by the constant c_{roll}. We drop the sign function since we only consider strictly positive speeds. We model the relative speed related to the air drag, which we assume constant and known by $v_{\mathrm{rel}}(v, \varphi) = v - v_{\mathrm{wind}} \cos(\varphi - \varphi_{\mathrm{wind}})$, where φ_{wind} is the angle of the direction e_{wind} in which the wind asserts force and φ is the heading of the vehicle in the CCF. The wind speed was included in recent works [179, 180], particularly when it comes to energy-efficient trajectory planning. Real-time wind data can be obtained by weather service providers, such as shown in [179]. The wind speed demonstrates an influence that can be easily modeled in the FCF but is difficult to model in the CCF.

The input of our model is given by $u = [F^{\mathrm{d}} \quad r]^\top \in \mathbb{R}^2$, where $r = \frac{\mathrm{d}\delta}{\mathrm{d}t}$ denotes the steering rate. The dynamics of the coordinate unrelated states are written as

$$\dot{x}^{\neg c} = f^{\neg c}(x^{\neg c}, u, \varphi) = \begin{bmatrix} \frac{1}{m}(F^{\mathrm{d}} - F^{\mathrm{wind}}(v, \varphi) - F^{\mathrm{roll}}(v)) \\ r \end{bmatrix}, \tag{5.14}$$

where m denotes the vehicle mass. The lateral acceleration $a_{\mathrm{lat}}(x^{\neg c})$ at the rear wheel axis is given by

$$a_{\mathrm{lat}}(x^{\neg c}) = \frac{v^2 \tan(\delta)}{l}, \tag{5.15}$$

where l is the total wheelbase length of the vehicle. The wheelbase length can be expressed as the sum of the distance between the center of gravity (CG) to the rear wheel axis l_{r} or front wheel axis l_{f} by $l = l_{\mathrm{f}} + l_{\mathrm{r}}$.

Cartesian Coordinate Frame Vehicle Model

By using simple kinematic relations, the dynamics of the Cartesian states can be written as

$$\dot{x}^{\mathrm{c,C}} = f^{\mathrm{c,C}}(x^{\mathrm{C}}, u) = \begin{bmatrix} v \cos(\varphi) \\ v \sin(\varphi) \\ \frac{v}{l} \tan(\delta) \end{bmatrix}. \tag{5.16}$$

The full five-state Cartesian vehicle model is given by

$$\dot{x}^{\mathrm{C}} = \begin{bmatrix} f^{\mathrm{c,C}}(x^{\mathrm{C}}, u) \\ f^{\neg c}(x^{\neg c}, u, \varphi) \end{bmatrix}. \tag{5.17}$$

FCF Vehicle Model

Since in usual vehicle motion control tasks, the vehicle moves mainly close to a reference curve $\gamma : \mathbb{R} \to \mathbb{R}^2$, i.e., the street center line, the transformation into a curvilinear CF is a natural choice. The reference curve $\gamma(\sigma) = [\gamma_x(\sigma) \quad \gamma_y(\sigma)]^\top$ is parameterized by its path length σ and can be fully described by one initial transformation offset $\gamma(\sigma_0)$, an initial orientation φ_0 and the curvature $\kappa(\sigma) = \frac{\mathrm{d}\varphi^\gamma}{\mathrm{d}\sigma}$ along its path. We use $\varphi^\gamma(\sigma)$ for the tangent angle in each point of the curve. As part of the Frenet transformation, we project the Cartesian vehicle reference point $p^{\mathrm{veh}} \in \mathbb{R}^2$ on the closest point along the reference curve with

$$s^*(p^{\mathrm{veh}}) = \arg\min_{\sigma} \left\| p^{\mathrm{veh}} - \gamma(\sigma) \right\|_2^2. \tag{5.18}$$

W.l.o.g., we always set the initial reference point of the transformation to zero. The position s along the curve is used as a longitudinal FCF state. The vector $(p^{\mathrm{veh}} - \gamma(s^*))$ is the difference of the closest point on the curve to the vehicle. By using the 90 degree rotation matrix R^{90} and projection to the normal unit vector of the curve $e_n = R^{90}\gamma'(s^*)$, we obtain the Frenet state $n = (p^{\mathrm{veh}} - \gamma(s^*))^\top e_n$. The third Frenet state α relates the tangent angle of the curve to the heading of the vehicle with $\alpha = \varphi - \varphi^\gamma(s^*)$. The transformation relations are shown in Fig. 5.12. We write the transformation of a Cartesian state $x^{\mathrm{c,C}} = [x \quad y \quad \varphi]^\top$ to a Frenet state $x^{\mathrm{c,F}} = [s \quad n \quad \alpha]^\top$ by means of the transformation

$$x^{\mathrm{c,F}} = \mathcal{F}_{\tilde{\gamma}}(x^{\mathrm{c,C}}) = \begin{bmatrix} s^* \\ (p^{\mathrm{veh}} - \gamma(s^*))^\top e_n \\ \varphi^\gamma(s^*) - \varphi \end{bmatrix}, \tag{5.19}$$

and its inverse by

$$x^{\mathrm{c,C}} = \mathcal{F}_{\tilde{\gamma}}{}^{-1}(x^{\mathrm{c,F}}) = \begin{bmatrix} \gamma_x(s) - n\sin(\varphi^\gamma(s)) \\ \gamma_y(s) + n\cos(\varphi^\gamma(s)) \\ \varphi^\gamma(s) - \alpha \end{bmatrix}. \tag{5.20}$$

The existence and uniqueness of the transformation are guaranteed under mild assumptions, which are discussed in detail in [222]. As shown in [151], we obtain the ODE for the kinematic motion in the FCF as

$$\dot{x}^{\mathrm{c,F}} = f^{\mathrm{c,F}}(x^{\mathrm{F}}, u) = \begin{bmatrix} \frac{v\cos(\alpha)}{1 - n\kappa(s)} \\ v\sin(\alpha) \\ \frac{v}{l}\tan(\delta) - \frac{\kappa(s) v \cos(\alpha)}{1 - n\kappa(s)} \end{bmatrix}. \tag{5.21}$$

The Cartesian state φ is needed in order to formulate the wind disturbance. It is not available in the FCF. Consequently, it needs to be computed by evaluating the tangent angle $\varphi^\gamma(s)$ of the current position s on the reference curve $\gamma(\sigma)$.

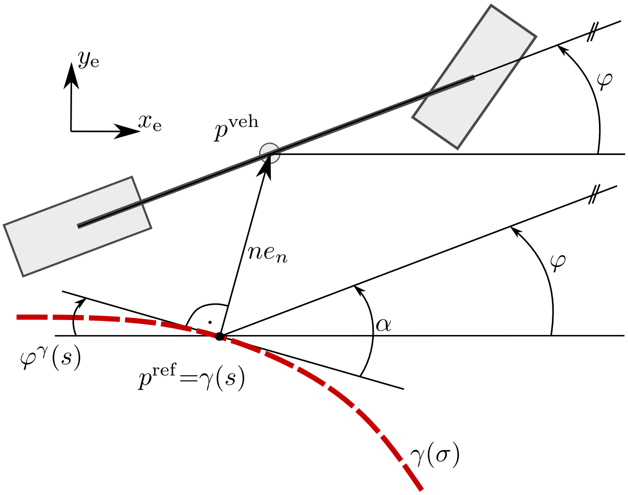

Figure 5.12: State relations between the CFC and FCF

This can be approximated by a spline function $\hat{\varphi}^\gamma(s)$ that is computed for the road layout and yields an approximation $\hat{\varphi}(s,\alpha) = \hat{\varphi}^\gamma(s) + \alpha$ of the heading angle. The full FCF vehicle model is consequently given by the five-state model

$$\dot{x}^\mathrm{F} = \begin{bmatrix} f^{\mathrm{c},\mathrm{F}}(x^\mathrm{F}, u) \\ f^{\neg\mathrm{c}}(x^{\neg\mathrm{c}}, u, \hat{\varphi}) \end{bmatrix}. \tag{5.22}$$

Model Comparison

As indicated in Sec. 5.2.1 and Tab. 5.2, the CF models have different advantages when used in a NMPC formulation. The definition of road boundaries and the reference curve, which are often lane-aligned curves, are straightforward to define in the FCF but hard to define in the CCF. However, the definition of an obstacle in the FCF is cumbersome for several reasons. Despite nonconvex obstacle shapes in the FCF, safety cannot be guaranteed when using sequential quadratic programming (SQP) to solve the NMPC problem with the Frenet model. Convex obstacle shapes cannot be guaranteed to be convex if transformed into the FCF. This fact can be seen from the following counterexample. Consider a straight line, which is a convex set, and a circular road. Let the line intersect the road at coordinates $\gamma(\sigma_1) = [x_1 \quad y_1]^\top$ and $\gamma(\sigma_2) = [x_1 \quad y_1]^\top$. The transformed Frenet states n_1, n_2 are zero in either point. At $\sigma_3 \in (\sigma_1, \sigma_2)$, the transformed state $n_3 \neq 0$, thus the transformed set is not convex. Considering Lemma 5.2.1, it can be shown that convex obstacles are guaranteed to be a subset of the linearized constraints within an SQP iteration, thus safely over-approximated.

Feature	CCF	FCF
reference definition	✗	✓
boundary constraints	✗	✓
obstacle specification	✓	✗
disturbance specification	✓	✗

Table 5.2: Comparison of the two model representations

Lemma 5.2.1. *Regard the set* $\mathcal{C} = \{x \in \mathbb{R}^n \mid g(x) \geq 0\}$ *and* $\mathcal{C}^{\text{lin}}(x^*) = \{x \in \mathbb{R}^n \mid g(x^*) + \nabla g(x^*)^\top (x - x^*) \geq 0\}$*. Suppose that the function* $g : \mathbb{R}^n \to \mathbb{R}$ *is convex, then* $\mathcal{C} \subseteq \mathcal{C}^{\text{lin}}(x^*)$ *for any* x^**.*

Proof. Due to convexity, $g(x^*) + \nabla g(x^*)^\top (x - x^*) \leq g(x)$ and therefore, it follows that $\mathcal{C} \subseteq \mathcal{C}^{\text{lin}}$. ☐

Nonconvex obstacles, which result from the transformation of convex shapes into the FCF, are not safely over-approximated within SQP algorithms.

Another problem that arises with objects in the FCF is the dependence of the shape on the state. Consequently, if the obstacle constraints are defined along a discretized time horizon, at each time step $i = 0, \dots, N$, the shape has to be transformed separately, cf. Fig.5.10. In typical applications, this could lead to N transformations for each obstacle in every NMPC iteration, followed by a convexification (e.g., bounding boxes, convex polygons, covering circles) to guarantee safety. Alternatively, an over-approximation could be used to capture all possible transformed shapes. However, this would lead to a striking conservatism, especially for long vehicles and small curve radii.

Overview of CF Lifting Formulations

As outlined in Sec. 5.2.2, having states of both CFs in the NLP formulation is beneficial to simplify the constraints. Several different ways of including both CFs are possible, and a summary is given in Tab. 5.3.

First, one can choose the primary CF ODE and introduce the states related to the other CF as algebraic variables that are determined by the primary CF and obtain a DAE of index 1. One could either have a CCF based DAE

$$\dot{x}^{\mathrm{C}} = f^{\mathrm{C}}(x^{\mathrm{C}}, u), \quad 0 = x^{\mathrm{c,F}} - \mathcal{F}_{\tilde{\gamma}}(x^{\mathrm{c,C}}), \tag{5.23}$$

or an FCF-based DAE

$$\dot{x}^{\mathrm{F}} = f^{\mathrm{F}}(x^{\mathrm{F}}, x^{\mathrm{c,C}}, u), \quad 0 = x^{\mathrm{c,C}} - \mathcal{F}_{\tilde{\gamma}}{}^{-1}(x^{\mathrm{c,F}}). \tag{5.24}$$

Formulation	ODE CF	Obstacle CF	Cost CF	n_x	n_z	Practical Relevance	Issues
Conventional Frenet	Frenet	Frenet	Frenet	5	0	yes	nonconvex state-dependent obstacle shapes usually over-approximated [166, 238, 215, 301]
Direct Elimination Frenet	Frenet	Cartesian	Frenet	5	0	yes	additional nonlinearities (objective, constraints)
Lifted ODE Frenet	Frenet	Cartesian	Frenet	8	0	yes	redundant states
DAE Frenet	Frenet	Cartesian	Frenet	5	3	no	bad convergence in our experiments
Conventional Cartesian	Cartesian	Cartesian	Cartesian	5	0	no	nonconvex boundary constraints [167]
Cartesian with Frenet States	Cartesian	Cartesian	Frenet	$\{5, 8\}$	$\{0, 3\}$	yes	Difficult bi-level problem. Approximations, e.g. [167]

Table 5.3: Comparison of CF Formulations in NMPC. Formulations are compared among their CF specifications, the required number of differential/algebraic states n_x/n_z, and their relevance. Bold-typed formulations are compared within this paper.

The inverse transformation ${\mathcal{F}_{\tilde{\gamma}}}^{-1}$ is computationally cheap since it just needs explicit function evaluations, whereas the forward transformation $\mathcal{F}_{\tilde{\gamma}}$ requires solving an NLP as in (5.18), resulting in a computationally expensive bi-level problem in the final NMPC formulation. Therefore, we choose the FCF of (5.24) as a basis and exclude CCF formulations (5.23) from further comparisons. The DAE of index 1 (*Lifted DAE Frenet*) is one possible way to formulate the problem and was similarly used in [310] for a linearized model. Another possible formulation (*Direct Elimination Frenet*) is to directly eliminate the algebraic variables in (5.24) by using the inverse Frenet transformation in the nonlinear constraint formulation. If the objective includes Cartesian states with quadratic costs and lifted constraints, the direct elimination would lead to a nonlinear objective and constraints. Alternatively, we can perform an index reduction of (5.24), which is obtained by differentiation of the algebraic constraint, leading to

$$\dot{x}^{\mathrm{c,F}} = f^{\mathrm{c,F}}(x^{\mathrm{c,F}}, u) \tag{5.25a}$$

$$0 = \dot{x}^{\mathrm{c,C}} - \frac{\partial {\mathcal{F}_{\tilde{\gamma}}}^{-1}(x^{\mathrm{c,F}})}{\partial x^{\mathrm{c,F}}} f^{\mathrm{c,F}}(x^{\mathrm{c,F}}, u), \tag{5.25b}$$

$$\text{and} \quad x^{\mathrm{c,C}}(0) := {\mathcal{F}_{\tilde{\gamma}}}^{-1}(x^{\mathrm{c,F}}). \tag{5.25c}$$

Detailed computation (not presented here) shows the equivalence of (5.25b) to

$$\dot{x}^{\mathrm{c,C}} = f^{\mathrm{c,C}}(x^{\mathrm{c,C}}, u) \tag{5.26}$$

The approach in (5.25) or (5.26) (*Lifted ODE Frenet*) results in redundant states in both CFs, which are coupled through the inputs and the initial state.

5.2.3 Obstacle Avoidance Formulations

Different formulations for obstacle avoidance constraints are used in NMPC and visualized in Fig. 5.13. We assume that rectangles represent real vehicle shapes. Often, simple geometric covering shapes (circles [145] or ellipses [49]) and related distance functions are used. Alternatively, covering polygons and restrictions on edges or vertices (hyperplanes) are formulated in [245, 52, 92]. Furthermore, the road boundaries can be deflected in order to cover the obstacle by the boundary constraints [151]. The latter approach is not within the scope of this work due to the generally different formulations that, for instance, include a combinatorial planner for choosing the passing side [225]. We compare the formulation of obstacle avoidance constraints with an *ellipse* [49], *covering circles* [297, 145] and *separating hyperplanes* [52]. We also implemented a formulation introduced in [245], which we refer to as *set-vertices-exclusion*, but which poorly converged in our experiments. We assume a rectangular shape of the vehicles with the rear/front chassis length $l_{\mathrm{ch}} = l_{\mathrm{r,ch}} + l_{\mathrm{f,ch}}$ related to the vehicle CG and chassis

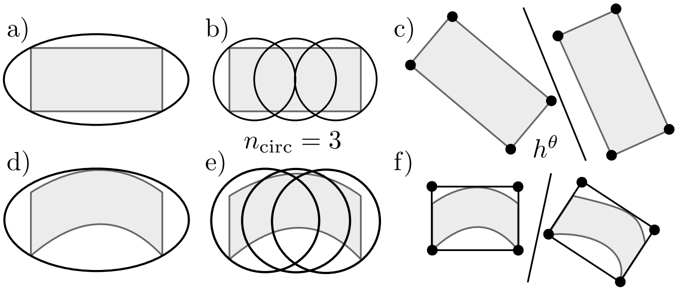

Figure 5.13: Schematic drawing of obstacle constraints. (a: ellipse CCF, b: covering circles CCF, c: separating hyperplanes CCF, d: ellipse FCF, e: covering circles FCF, f: separating hyperplanes FCF)

width w_{ch}. The separating hyperplane formulation does not require increased obstacle sizes beyond their actual rectangular shape, whereas circle and ellipse formulations require over-approximations.

Obstacle Approximation by an Ellipse

Constraining the distance between a circle and an ellipse is less complex than constraining the distance between two ellipses. This can be easily argued by the rotational invariance of a circle which allows for a simple shape inflation of the ellipse by the radius of the circle, followed by a level set constraint. The distance between two ellipses depends on the orientation of both, thus shape inflation of the one ellipse and a point reduction of the other one is inhibited. Thus, we cover the ego car with a circle. The main axes a, b of an ellipse covering a rectangle are computed by $a = \frac{1}{\sqrt{2}}(l_{\text{f,ch}} + l_{\text{r,ch}})$ and $b = \frac{1}{\sqrt{2}} w_{\text{ch}}$. Increased by the ego radius r^{ego}, this leads to the extended ellipse matrix $D = \text{diag}([a + r^{\text{ego}}, b + r^{\text{ego}}])$. With the rotation matrix $R(x^{\text{c,opp}}) \in \mathbb{R}^{2\times 2}$ and a translation vector $t(x^{\text{c,opp}}) \in \mathbb{R}^2$ related to the orientation and position of the obstacle vehicle, we can formulate the collision avoidance constraint with the matrix $\Sigma(x^{\text{c,opp}}) = R(x^{\text{c,opp}}) D R(x^{\text{c,opp}})^{\top}$ via the feasible set

$$\mathcal{P}^{\text{ell}}(x^{\text{c,opp}}) = \left\{ x^{\text{c}} \in \mathbb{R}^3 \,\middle|\, \|x^{\text{c}} - t(x^{\text{c,opp}})\|^2_{\Sigma^{-1}(x^{\text{c,opp}})} \geq 1 \right\}.$$

Obstacle Approximation by Covering Circles

We use the union of a *set of circles* to cover the vehicle shape as shown in [297, 328, 145]. For $l_{\text{ch}} \geq w_{\text{ch}}$, the number of covering circles n_{circ} must be larger than $\lceil \frac{l_{\text{ch}}}{w_{\text{ch}}} \rceil$ and have a radius $r^{\{\text{ego,opp}\}}$ of $w_{\text{ch}} \frac{1}{\sqrt{2}}$. For each combination of the $n_{\mathrm{circ,ego}}$ and $n_{\mathrm{circ,opp}}$ circles a distance constraint must be satisfied, leading to $n_{\mathrm{circ,ego}} n_{\mathrm{circ,opp}}$ inequality constraints. The covering circle center points are computed according to [328], which gives us a function $p^i : \mathbb{R}^3 \to \mathbb{R}^2$ for the circle center i that computes the center positions $p_i = p^i(x^{\mathrm{c,C}})$ from the states $x^{\mathrm{c,C}}$. With $\Delta r = r^{\mathrm{ego}} + r^{\mathrm{opp}}$ and $x := x^{\mathrm{c,C}}$, we can denote the free set as

$$\mathcal{P}^{\mathrm{circ}}(x^{\mathrm{opp}}) = \Big\{ x \in \mathbb{R}^3 \big| \left\| p^i(x) - p^j(x^{\mathrm{opp}}) \right\|_2 \geq \Delta r, \\ \text{for } 1 \leq i \leq n_{\mathrm{circ,ego}}, 1 \leq j \leq n_{\mathrm{circ,opp}} \Big\} \tag{5.27}$$

Obstacle Approximation by Separating Hyperplanes

When formulating collision avoidance with separating hyperplanes, we optimize for a feasible solution of the parameters $\theta \in \mathbb{R}^3$ of a hyperplane $h^{\theta}(p)$. The parameterized hyperplane needs to separate all four vertices $p_i^{\{\mathrm{ego,opp}\}}(x^{\mathrm{c}\{\mathrm{ego,opp}\}})$ of either vehicle's bounding box. We write the feasible region using the related hyperplane existence problem with points $\bar{p}^{\{\cdot\}\top} = [p^{\{\cdot\}\top} \quad 1]$ as

$$\mathcal{P}^{\mathrm{hp}}(x^{\mathrm{opp}}) = \Big\{ x \in \mathbb{R}^3, \theta \in \mathbb{R}^3 \Big| \; \theta_1^2 + \theta_2^2 = 1, \\ \theta^\top \bar{p}_i^{\mathrm{ego}}(x) \leq 0, \; \theta^\top \bar{p}_i^{\mathrm{opp}}(x^{\mathrm{opp}}) \geq 0, \; \forall i = 0, \dots, 3 \Big\}. \tag{5.28}$$

We avoid a degenerate solution with the constraint $\theta_1^2 + \theta_2^2 = 1$ for the hyperplane parameters.

5.2.4 NMPC Formulation

The NMPC aims to plan a feasible trajectory for a vehicle to drive on a road with bounded curvature on a reference lane parallel to the center line and with a desired reference speed. Furthermore, the NMPC must avoid static and dynamic obstacles. As motivated in Sec. 5.2.2, we use two variants of an FCF-based ODE to obtain Cartesian states, i.e., the *direct elimination* and *lifted ODE* formulation and compare it to the *conventional* formulation with over-approximation, such as shown in [238, 215]. First, we define the costs and constraints.

General Costs & Constraints

Some constraints are unrelated to the CF, such as the lower and upper bounds for states $x^{\neg\mathrm{c}}$ and inputs u. For control costs $u^\top R u$, we use the positive semi-definite weight matrix $R \in \mathbb{R}^{2\times 2}$.

FCF Related Costs & Constraints

State costs are related to FCF states since there is no practical advantage of including CCF state costs. A cost related to a desired reference path parallel to the road center line is accounted for by a square penalty of the deviation of the Frenet lateral coordinate n to its reference n_{ref}. For a reference speed v_{ref}, a square penalty with positive weight w_s, as well as a penalty on precomputed longitudinal reference positions $s_{\mathrm{ref},i} = \hat{s}_0 + i\Delta t v_{\mathrm{ref}}$ is used, with the measured state $\hat{s}_0$ and sampling time Δt. Since we assume a road with constant width, boundary constraints simplify in the FCF to box constraints for an upper bound $\overline{n}$ and a lower bound $\underline{n}$.

Cartesian Coordinate Frame Related Costs & Constraints

We use the collision avoidance formulations, which could be one out of $\mathcal{O} = \{\mathrm{ell}, \mathrm{circ}, \mathrm{hp}\}$ in the CCF, thus have the constraint $x^{\mathrm{c,C}} \in \mathcal{P}^{\{\mathrm{ell,circ,hp}\}}$. FCF costs are defined via the positive weight matrix $Q = \mathrm{diag}(q)$ with the weight vector $q \in \mathbb{R}^5$ and the reference states $x^{\mathrm{F}}_{\mathrm{ref}}$. We use a terminal cost $Q_N = \mathrm{diag}(q_N)$ with the weight vector $q_N \in \mathbb{R}^5$ after N discrete time steps. We can, therefore, write the objective function as

$$\begin{aligned} &J(x_0^{\mathrm{F}}, \dots, x_N^{\mathrm{F}}, u_0, \dots, u_{N-1}) = \\ &\sum_{k=0}^{N-1} \|u_k\|_R^2 + \left\|x_k^{\mathrm{F}} - x_{\mathrm{ref},k}^{\mathrm{F}}\right\|_Q^2 + \left\|x_N^{\mathrm{F}} - x_{\mathrm{ref},N}^{\mathrm{F}}\right\|_{Q_N}^2 . \end{aligned} \tag{5.29}$$

Direct Elimination NMPC Formulation

With the direct formulation, we can directly use the inverse transformation $x^{\mathrm{c,C}} = \mathcal{F}_{\tilde{\gamma}}{}^{-1}(x^{\mathrm{c,F}})$ to eliminate the Cartesian states in the constraint formulation. Consequently, we obtain fewer states but "more" nonlinear constraints. We discretize the continuous trajectory with $N-1$ control intervals and use direct multiple shooting [45] with one step of an RK4 integration function $\Phi^{\mathrm{F}}(x^{\mathrm{F}}, u, \Delta t)$ for the ODE in (5.22) and the NLP formulation

$$\min_{\substack{x_0^{\mathrm{F}},\dots,x_N^{\mathrm{F}},\\ u_0,\dots,u_{N-1}\\ \theta_1,\dots,\theta_{n_{\mathrm{opp}}}}} \quad J(x_0^{\mathrm{F}},\dots,x_N^{\mathrm{F}},u_0,\dots,u_{N-1}) \tag{5.30a}$$

s.t.

$$x_0^{\mathrm{F}} = \hat{x}_0^{\mathrm{F}}, \tag{5.30b}$$

$$x_{i+1}^{\mathrm{F}} = \Phi^{\mathrm{F}}(x_i^{\mathrm{F}}, u_i, \Delta t), \qquad i = 0,\dots,N-1, \tag{5.30c}$$

$$\underline{u} \le u_i \le \overline{u}, \qquad i = 0,\dots,N-1, \tag{5.30d}$$

$$\underline{x}^{\mathrm{F}} \le x_i^{\mathrm{F}} \le \overline{x}^{\mathrm{F}}, \qquad i = 0,\dots,N, \tag{5.30e}$$

$$\underline{x}^{\mathrm{c,C}} \le \mathcal{F}_{\tilde{\gamma}}^{-1}(x^{\mathrm{c,F}}) \le \overline{x}^{\mathrm{c,C}}, \qquad i = 0,\dots,N, \tag{5.30f}$$

$$\underline{a}^{\mathrm{lat}} \le a_{\mathrm{lat}}^{\mathrm{F}}(x_i) \le \overline{a}^{\mathrm{lat}}, \qquad i = 0,\dots,N, \tag{5.30g}$$

$$v_N \le \overline{v}_N, \tag{5.30h}$$

$$\mathcal{F}_{\tilde{\gamma}}^{-1}(x^{\mathrm{c,F}}) \in \mathcal{P}(x_i^{\mathrm{c,opp,j}}, \theta_j), \qquad i = 0,\dots,N-1, \quad j = 1,\dots,n_{\mathrm{opp}}. \tag{5.30i}$$

Decision variables $\theta_1,\dots,\theta_{n_{\mathrm{opp}}}$, where $\theta_j = [\theta_j^0,\dots,\theta_j^N] \in \mathbb{R}^{3\times N}$ are only used for the separating hyperplanes formulation.

Lifted ODE NMPC Formulation

In this formulation we use the extended state $x^{\mathrm{d}} = [x^{\mathrm{F}\top} \quad x^{\mathrm{c,C}\top}]^\top$ and the extended ODE (5.26). The additional states increase the size of the state space to eight states in our case, three of which stem from either CF and two of which are CF-independent states. In this formulation, we use the RK4 integration function $\Phi^{\mathrm{d}}(x^{\mathrm{d}}, u, \Delta t)$ of dynamics (5.26). We can write the final NLP for the *lifted ODE* formulation as

$$\min_{\substack{x_0^{\mathrm{d}},\ldots,x_N^{\mathrm{d}},\\ u_0,\ldots,u_{N-1}\\ \theta_1,\ldots,\theta_{n_{\mathrm{opp}}}}} \quad J(x_0^{\mathrm{F}},\ldots,x_N^{\mathrm{F}},u_0,\ldots,u_{N-1}) \tag{5.31a}$$

s.t.

$$x_0^{\mathrm{d}} = \hat{x}_0^{\mathrm{d}}, \tag{5.31b}$$

$$x_{i+1}^{\mathrm{d}} = \Phi^{\mathrm{d}}(x_i^{\mathrm{d}}, u_i, \Delta t), \quad i = 0,\ldots,N-1, \tag{5.31c}$$

$$\underline{u} \leq u_i \leq \overline{u}, \quad i = 0,\ldots,N-1, \tag{5.31d}$$

$$\underline{x}^{\mathrm{d}} \leq x_i^{\mathrm{d}} \leq \overline{x}^{\mathrm{d}}, \quad i = 0,\ldots,N, \tag{5.31e}$$

$$\underline{a}^{\mathrm{lat}} \leq a_{\mathrm{lat}}(x_i^{\mathrm{d}}) \leq \overline{a}^{\mathrm{lat}}, \quad i = 0,\ldots,N, \tag{5.31f}$$

$$v_N \leq \overline{v}_N, \tag{5.31g}$$

$$x_i^{\mathrm{c,C}} \in \mathcal{P}(x_i^{\mathrm{c,opp,j}}\theta_j), \quad i = 0,\ldots,N-1, \quad j = 1,\ldots,n_{\mathrm{opp}}. \tag{5.31h}$$

5.2.5 Numerical Experiments

In order to evaluate the performance of the proposed approach, we simulate two randomized scenarios that constitute three non-ego vehicles in front of the ego vehicle with a lower cruise speed. The scenario is simulated for 20 seconds, where usually three overtakes are possible. In total, 500 full simulation runs are evaluated for each scenario type. We record the solution times of the NMPC and the final driven distance after the simulation ends, which we take as a performance indicator. We use different types of obstacles, particularly long ones in the dimensions of a truck (truck-sized), as well as short ones resembling normal cars (car-sized). We make several simplifications in order to avoid performance influences of sources unrelated to our formulation. Firstly, there is no model-plant mismatch, i.e., the simulation framework and the NLP use the same kinematic vehicle model and discretization. Secondly, the ego NMPC has complete knowledge of the other vehicles' planned trajectories to avoid the influence of prediction errors. Finally, we model non-ego participants to be non-interactive. They aim at driving along a reference line parallel to the center line. The simulations were run on an Alienware m-15 Notebook with an Intel Core i7-8550 CPU (1.8 GHz). The parameters for the environment and the NMPC are shown in Tab. 5.2.5 and Tab. 5.2.5, respectively. We use the NLP solver `acados` [291] with `HPIPM` [97], RTI iterations and a partial condensing

Module	Name	Variable	Value
Road	road bounds[2]	$\underline{n}, \overline{n}$	±10, ±8.5
	curvature[1]	κ	[-0.05, 0.05]
	wind speed	v_{wind}	20
	wind direction	φ_{wind}	0
Ego vehicle	length wheelbase	$l_{\text{r}}, l_{\text{f}}$	1.7
	length chassis	$l_{\text{r,ch}}, l_{\text{f,ch}}$	2
	width chassis	w_{ch}	1.9
	mass	m	1160
	lateral acc. bound	$\underline{a}_{\text{lat}}, \overline{a}_{\text{lat}}$	±5
	input bounds	$\underline{u}, \overline{u}$	±[10kN, 0.39]
	velocity bound	$\overline{v}$	40
	steering angle bound	$\underline{\delta}, \overline{\delta}$	±0.3
	starting position[1]	$x_0^{\text{c,F}}$	[0, -5, 0]- [0, 5, 0]
	reference speed	v_{ref}	40
Opp. vehicles	length wheelbase[2]	$l_{\text{r}}, l_{\text{f}}$	10
	length chassis[2]	$l_{\text{r,ch}}, l_{\text{f,ch}}$	13
	width chassis[2]	w_{ch}	4
	mass[2]	m	3000
	input bounds[2]	$\overline{u}$	[30kN, 0.39]
	input bounds[2]	$\underline{u}$	[-45kN, -0.39]
	starting position[1] i	s_0^i	$50(i+1)$
		n_0^i	[-5, 5]
	reference speed	v_{ref}	15

Table 5.4: Environment parameters.[1] Randomized with uniform distribution.[2] Parameters only differ in long vehicle scenarios. The parameters are equal for all vehicles if not noted explicitly. We use SI units if they are not stated explicitly.

horizon of $\frac{N}{2}$. We use obstacle constraint formulations of Sec. 5.2.3. Besides the different obstacle dimensions, the proposed NMPC formulations *conventional*, *direct elimination*, and *lifted ODE* were evaluated with the different obstacle formulations of Sec. 5.2.3. We use the ellipse ("EL"), the n covering circles ("Cn"), and the separating hyperplane ("HP") formulation. In Fig. 5.2.5, the computation times and the maximum achieved progress of the randomized scenarios are shown for truck- and car-sized vehicles. Clearly, the final progress after overtaking in the truck-sized scenario is significantly increased by the proposed formulation due to the more accurate representation of the obstacle shape. For car-sized vehicles, the extended states do not yield a prominent advantage since, in this case, the Frenet transformation does not deform the obstacles vastly. The maximum progress is nearly equal for both proposed

Name	Variable	Value
nodes / disc. time	$N/\ \Delta t$	40/ 0.1
terminal velocity	$\overline{v}_N$	15
state weights	q	$[1, 500, 10^3, 10^3, 10^4]\Delta t$
terminal state weights	q_N	$[10, 90, 100, 10, 10]$
control weights	R	$\mathrm{diag}([10^{-3}, 2 \cdot 10^6])\Delta t$

Table 5.5: Parameter for NMPC in SI units.

approaches since the obstacle constraint formulations based on Cartesian states are equal. A striking difference between the two proposed formulations can be seen in the computation times, shown detailed in Tab. 5.6. While the *lifted ODE* formulation even decreases the average computation time for nearly all obstacle formulations, the *direct elimination* formulation increases the computation time by around 30%. Remarkably, the ellipsoidal obstacle formulation in the proposed lifted ODE formulation outperforms all other obstacle formulations in both the computation time as well as the performance measured in the average progress after overtaking, which highlights the advantage of the formulation. Contrary to our expectations, the separating hyperplane formulation shows weaker performance in computation time and average progress. In theory, separating hyperplanes should be more accurate in capturing the obstacle shape. Nevertheless, due to the disadvantageous linearizations within the SQP iterations, the shape is not captured well. This might be mainly due to the nonconvex and nonlinear constraint in (5.28). Note that the proposed lifting approach is not limited to kinematic vehicle models. It extends to higher fidelity models since the lifting is limited to the six coordinate-related states that appear equally in high fidelity models [293], namely CCF positions x and y, the CCF heading angle θ, the FCF position states s and n, as well as the FCF angle α.

5.2.6 Conclusions

We have presented two novel FCF-based formulations of NMPC for vehicles that include states of the CCF in order to gain numerical advantages. Simulated evaluations and the comparison of several wide-spread obstacle constraint formulations show that the proposed approaches are capable of representing the obstacle shapes more suitably and that with the *lifted ODE* formulation, even the computation time was decreased. Furthermore, our evaluations show that an ellipsoidal obstacle representation outperforms all other obstacle formulations in computation time. In conclusion, the combination of the ellipsoidal obstacle constraint formulation with the *lifted ODE* formulation yields superior results in all categories.

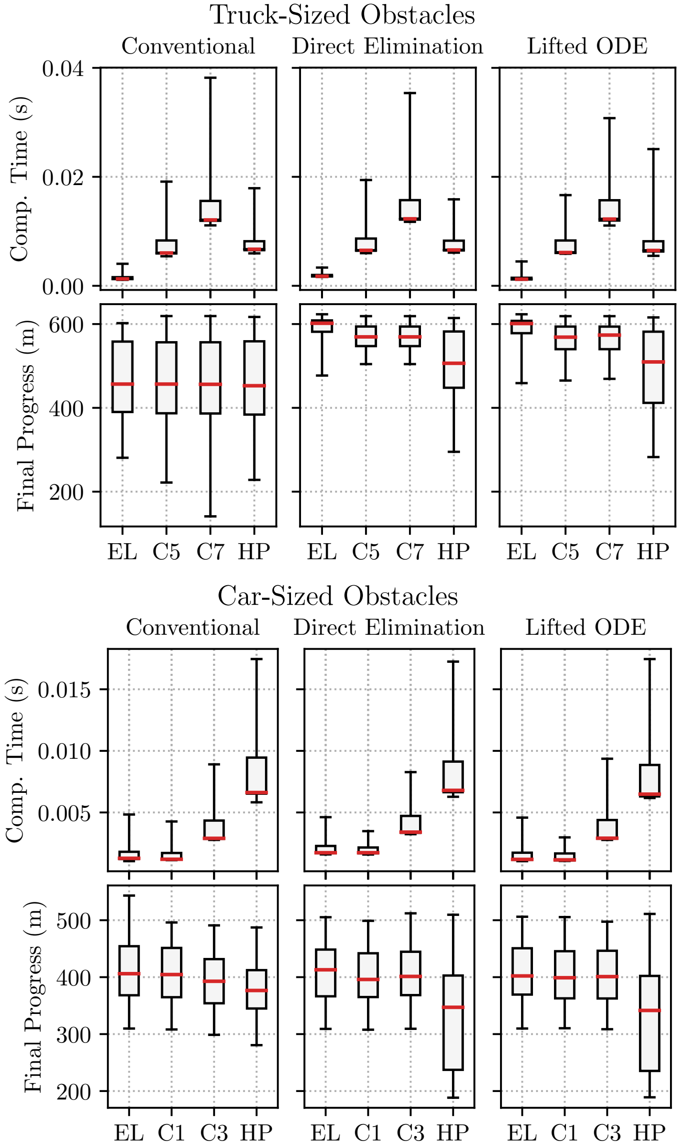

Figure 5.14: Box-plot comparison of the NMPC solution timings for each real-time iteration and the final progress after 20 seconds for different obstacle formulations for truck- and car-sized vehicles.

Computation Times (ms) for Truck-Sized Obstacles

	Conventional	Direct Elimination		Lifted ODE	
EL	1.5 ± 0.4	1.9 ± 0.2	28.9%	1.4 ± 0.3	−6.6%
C5	7.2 ± 1.9	7.6 ± 1.7	5.5%	7.2 ± 1.8	−0.0%
C7	14.0 ± 3.2	14.0 ± 2.8	−0.1%	13.9 ± 2.9	−0.4%
HP	7.5 ± 1.5	7.5 ± 1.5	−0.1%	7.4 ± 1.7	−1.6%
Car-Sized Obstacles					
EL	1.5 ± 0.5	2.0 ± 0.4	29.6%	1.4 ± 0.4	−5.7%
C1	1.4 ± 0.4	1.9 ± 0.4	34.0%	1.4 ± 0.4	−3.5%
C3	3.6 ± 1.1	4.0 ± 1.0	12.4%	3.6 ± 1.1	0.6%
HP	8.0 ± 2.3	7.9 ± 1.9	−0.6%	7.7 ± 2.0	−4.0%

Table 5.6: Mean and standard deviation of computation times for different scenarios, obstacle formulations, and lifting formulations. Additionally, the difference in percent to the conventional formulation is given.

5.3 Critical Discussion

The previous two sections showed two significant contributions for Frenet frame vehicle models used as part of an optimization problem.

In Sect. 5.1 and the related paper [222], a preprocessing algorithm is established that guarantees a singularity-free state space and smoothens nonlinearities for the Frenet frame-based vehicle model. Offline computations are not real-time critical. However, online computations have strict real-time requirements. Empirical comparisons in closed-loop simulations show that the preprocessed model lowers the number of required online quadratic program (QP) iterations by half, the number of sequential quadratic programming (SQP) iterations by 25% and the online computation time by 23%. Moreover, the QP solver failed in 40% of the simulations where the singularity was close to the feasible state space. By pushing the singularity outside the feasible domain by utilizing our proposed preprocessing algorithm, no QP failures were encountered.

As a disadvantage, the proposed algorithm requires knowledge of the track in advance, such as in racing scenarios or fixed routes. Recently, the authors in [308] proposed a modification to apply the same idea for fast online adaptions. While the contribution of the proposed algorithm was linked to models used within online optimization like model predictive control (MPC), the method also provides a way to generate curves on known tracks that trade off distance and curvature. By parameterizing the optimization problem, a smooth transition of shortest-path curves to minimum-curvature curves can be obtained.

In Sect. 5.2 and the related paper [228], a model formulation for MPC was introduced that guarantees safe and tight over-approximations of obstacle shapes when using SQP algorithms. The novel formulation uses a “lifted” model for the configuration states in the Frenet coordinate frame (FCF) and, redundantly, configuration states in the Cartesian coordinate frame (CCF). The model states are prevented from drifting by transforming the initial state in each iteration into both coordinate frames and specifying a constraint on the initial state within the nonlinear program (NLP) of the MPC. An alternative “direct” formulation was proposed that uses the transformation of the states within the NLP constraint definition. The empirical test involved overtaking truck-sized and car-sized vehicles in simulations. Various obstacle avoidance formulations were compared using the proposed formulation. MPC formulations with the ellipsoidal obstacle avoidance constraints achieved the best performance, which was measured in the maximum driven distance after a time interval, requiring overtaking three other vehicles. The proposed novel “lifted” formulation achieved a 30% increased maximum progress while reducing the computation time by 5.7% for cars and 6.6% for trucks. Also, an alternative “direct” formulation increased the maximum progress by 30% but also increased the online computation time by 30%. The higher computational burden may be due to increased nonlinearities. The proposed “lifted” and “direct” methods were deployed in an industrial autonomous robot as part of a Master’s thesis by the student Akash John Subash, using a slightly different vehicle model. Moreover, the experimental work of this Master’s thesis was accepted for the IEEE/RSJ International Conference on Intelligent Robots and Systems 2024 [271].

Chapter 6

Mixed-Integer Optimization for Collision Avoidance

Collision avoidance involves avoiding bounded obstacle shapes such as rectangles or ellipsoids. These shapes are convex, and therefore, the planning problem is inherently nonconvex. By decomposing the nonconvex planning space into convex partitions, binary variables can be added as decision variables to the optimization problem to formulate a disjunction between convex partitions. The disjunctive formulation allows mixed-integer solvers to find globally optimal solutions, cf., the formulation in [213].

An exhaustive formulation uses binary variables for each of the four convex partitions resulting from rectangular obstacles, for each obstacle and for every discrete time step along a prediction horizon of N. The resulting number of binary variables in the exhaustive formulation is $4N_{\text{obs}}N$, where N_{obs} is the number of obstacles. Since the computational burden depends on the number of binary variables, the number of binary variables must be kept low. The high number of binary variables in this formulation requires reducing the planning horizon and the number of considered obstacles [213].

In the following sections, several approaches are proposed to speed up the online computation time. In Sect. 6.1, the number of binary variables is reduced for problems with static obstacles to N_{obs} by decomposing the problem into a coarse mixed-integer linear program (MILP) approximation that decides on which side an obstacle is passed, and a subsequent nonlinear program (NLP) that uses smooth nonconvex obstacle shapes, where the passing side is fixed by the MILP solution. Additionally, rewards are modeled, which are convex regions that lower the closed-loop cost when passed.

In Sect. 6.2, the number of binary variables is reduced to $\mathcal{O}(N_{\text{obs}} + N)$ for multi-lane highways. Multi-lane highways exhibit a particular structure where obstacles mostly follow their current lane.

Sect. 6.3 does not assume a particular environment specification other than the general formulation in [213]. However, it reduces the computation time by replacing the combinatorial part of the mixed-integer formulation with a neural network (NN) predictor. The NN is trained on randomized simulated data and predicts only the binary variables of the expert mixed-integer quadratic program (MIQP) [213]. However, the remaining quadratic program (QP) is solved online since its computational burden is low compared to the MIQP. To improve the overall performance and enhance safety, an additional NLP is solved in each iteration to project potentially unsafe trajectories to the feasible region.

6.1 Mixed-Integer Optimization-Based Planning for Autonomous Racing with Obstacles and Rewards

---

In this section, the paper published in [225] is reprinted with permission of Martin Kirchengast, Daniel Watzenig and Moritz Diehl. Note that the formatting of some formulas, terms, and numbers has been slightly adjusted for consistency without changing their meaning or content.

The contributions of each author are listed in the following.

Rudolf Reiter:	*Idea, programming of the published algorithm and the interface to the overall system, programming on the overall system (various program modules of the "Autonomous Driving Stack"), design of the experiments, writing of the document*
Martin Kirchengast:	*Programming of the overall system (embedded system, Autonomous Racing Graz software stack), mathematical corrections, stylistic corrections, linguistic improvements*
Daniel Watzenig:	*Leading the "Autonomous Racing Graz" team, design of the overall system, operational management and organization of the competitions*
Moritz Diehl:	*Mathematical corrections, stylistic corrections, linguistic improvements*

©2022 The Authors. Originally published in IFAC-PapersOnLine 54-6 (2021), pp. 99-106. https://doi.org/10.1016/j.ifacol.2021.08.530.

Abstract. Trajectory planning with the consideration of obstacles is a classical task in autonomous driving and robotics applications. This paper introduces a novel solution approach for the subclass of autonomous racing problems, which is additionally capable of dealing with reward objects. This special type of object represents particular regions in state space whose optional reaching is somehow beneficial (e.g., results in bonus points during a race). First, a homotopy class is selected, which represents the left/right and catch/ignore decisions related to obstacle avoidance and reward collection, respectively. For this purpose, a linear mixed-integer problem is posed such that an approximated combinatorial problem is solved and repetitive switching decisions between solver calls are avoided. Secondly, an optimal control problem (OCP) based on a single-track vehicle model is solved within this homotopy class. In the corresponding nonlinear program, homotopy iterations are performed on the

race track boundaries which correspond to the previously chosen homotopy class. This leads to an improved convergence of the solver compared to the direct approach. The mixed-integer method's effectiveness is demonstrated within a real-world test scenario during the autonomous racing competition `Roborace`. Furthermore, its combination with the OCP, as well as the performance gain resulting from the homotopy iterations, are shown in simulation.

6.1.1 Introduction

Autonomous racing poses great challenges for the development of planning and control algorithms. As the vehicles move with high velocities close to their physical limits, nonlinear effects on their system dynamics arise and must already be considered during planning. Sudden obstacles or race opponents with uncertain behavior require evasion or overtaking maneuvers planned in real-time on hardware with typically rather limited computational power. As part of the racing series `Roborace`, the participating teams develop software for the fully autonomous operation of electric race cars and are confronted with increasingly demanding objectives from one event to the other. This work addresses real-time trajectory planning for lap time minimization while avoiding obstacles and collecting rewards. Whereas the vehicle moves on a real race track, the purely virtual obstacles and rewards, which translate into lap time penalties and bonuses, respectively, are provided to the car's software about 200 m in advance. The planning algorithm not only has to compute a trajectory that eludes obstacles and is feasible regarding vehicle kinematics and dynamics as well as race track geometry. It also must decide whether rewards are worth gathering, distinguishing the considered task from related problems in literature. The proposed algorithm was successfully used in the `Roborace` competition at the Bedford race circuit in December 2020. Figure 6.1 shows the race car collecting a virtual reward object, which was live-streamed as augmented-reality animation during the race.

Related Work

Trajectory planning with obstacle avoidance occurs in many domains, and consequently, different approaches exist. Typically, in a certain situation, there are infinitely many trajectories that fulfill the feasibility conditions. Therefore, cost functions are used to choose an optimal one, depending on the application. Graph-based methods perform a global search on the time- and space-discretized configuration space to find the best trajectory. However, their ability to find the global optimum strongly depends on the chosen discretization, and if that is fixed, even finding a feasible solution may fail, e.g., in narrow passages. Common graph search techniques include Dijkstra's algorithm, A*, and D* with their variants [199]. Exemplary, [231] describes the graph construction for racing

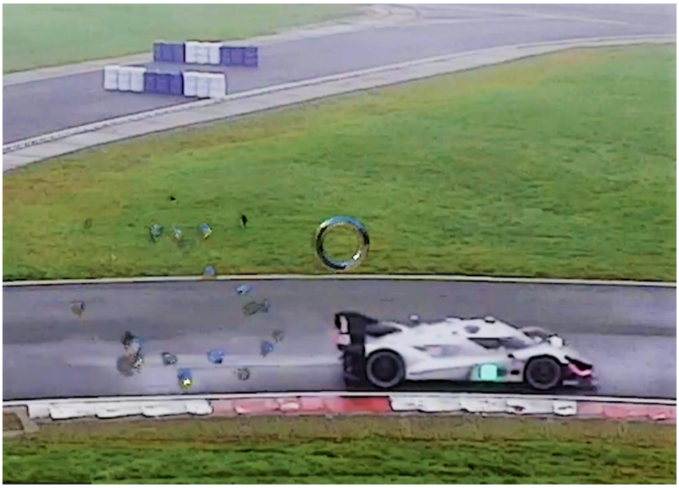

Figure 6.1: Race car hitting an augmented reality reward at the competition in Bedford, England.

applications and uses a Dijkstra-like search algorithm. Depending on the vehicle model fidelity and the discretization grid size, the computation times of graph search quickly rise. Techniques based on nonlinear continuous optimization (aka variational methods) tend to perform better in these cases, although they only provide locally optimal solutions unless the initial guess is sufficiently close to the global optimum [199]. An application within the racing domain is shown in [120], where first, a minimum curvature path is computed via solving a quadratic program (QP). Afterwards, the velocity profile is generated so that the acceleration limits are not violated. Many variational methods have strong similarities to nonlinear model predictive control (NMPC), e.g., [132], but instead of directly applying their predicted model inputs to the plant, their computed trajectories serve as references for underlying controllers. [11] demonstrate how to express an NMPC formulation of the trajectory planning problem in a linear parameter varying form. In [167], a formulation as model predictive contouring control problem leads to a progress maximization on the race track's center line. [31] introduce a homotopy strategy to account for the strong nonlinearity of the obstacles. In both papers, obstacles are considered via variations of the track boundaries. Thereby, the decision on which side an obstacle should be bypassed is delegated to a higher-level planner. [33] and [32] consider the combinatorial problem by combining optimal control with lattice-based path planning. Their work addresses the same underlying problem but focuses on unstructured environments like parking lots. [246] and [230] illustrate how to integrate these binary decisions into a single MILP and show its application to solve obstacle avoidance for vehicles and spacecraft. In the context of controller design [128] use similar ideas in an MPC based on MIQP for avoiding obstacles. [201] first decompose the collision-free space into convex cells, which are connected to distinct homotopy classes, and subsequently solve MIQPs for each of them. Thereby, the globally optimal solution is selected from the local optima of each homotopy class.

Contribution

This paper proposes a planning procedure that considers obstacles and rewards and consists of both offline and online steps. Prior to the actual race, an optimal *racing line* for the given track geometry is computed, assuming that there are no obstacles. This is done with an approach similar to [120]. However, since it is not the focus of this paper, further details are omitted. The online part is twofold: First, a MILP is formulated whose solution selects a homotopy class, i.e., which rewards are collected and on which side obstacles are eluded. This homotopy class is represented by deformed boundaries of the original track. Secondly, a continuous optimization problem is stated where deflections from the racing line are minimized, considering the modified boundaries from the MILP and vehicle constraints. The NMPC-like optimization problem is not directly solved. Instead, homotopy iterations are performed, which gradually put more weight on the modified boundary constraints and improve the convergence of the solver. The paper contributes with a motion planning approach that is capable of solving time-optimal motion planning problems with a combinatorial structure without fully discretizing the state-space, as opposed to graph-based state-of-the-art algorithms.

6.1.2 Vehicle and Object Models

Separating the trajectory optimization from the combinatorial homotopy class selection facilitates using different models. An utterly simplified geometric model is utilized to solve the combinatorial obstacle and reward problem, representing a deviation from the racing line with limited maximum steepness. The continuous optimization problem is stated on the basis of a single-track model.

Single-Track Model for Continuous Optimization

The utilized kinematic model is given, for instance, in [151] and does not consider tire slip. In order to reduce the computation time of the optimizer later on, drifting motion is not considered during planning. Instead, the subsequent trajectory-following controller is assumed to consider lateral and longitudinal tire slip. The movement direction of the center of gravity (CG) with the vehicle mass m is given by the angle $\psi + \beta$, where ψ is the vehicle orientation. The

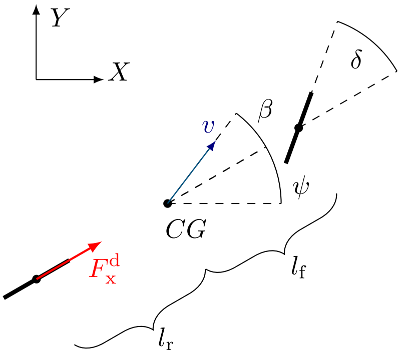

Figure 6.2: Kinematic single-track model.

side slip angle

$$\beta = \arctan\left(\frac{l_{\mathrm{r}}}{l_{\mathrm{r}} + l_{\mathrm{f}}} \tan\delta\right)$$

is defined as depicted in Fig. 6.2 and gives the relative angle of motion related to the vehicle coordinate system [151]. The vehicle motion is governed by

$$\dot{p}_{\mathrm{X}} = v\cos(\psi + \beta), \tag{6.1a}$$

$$\dot{p}_{\mathrm{Y}} = v\sin(\psi + \beta), \tag{6.1b}$$

$$\dot{\psi} = \frac{v}{l_{\mathrm{r}}}\sin\beta, \tag{6.1c}$$

$$\dot{v} = \frac{F_{\mathrm{x}}^{\mathrm{d}}}{m}\cos\beta, \tag{6.1d}$$

in the Cartesian coordinate frame, where v is the longitudinal velocity in the movement direction of the CG. The vehicle's geometry is described by the longitudinal position of the CG with front distance l_{f} and rear distance l_{r}. The input force $F_{\mathrm{x}}^{\mathrm{d}}$ only acts on the rear wheel, whereas the steering angle δ only deflects the front wheel. As a second input, the steering rate $r = \dot{\delta}$ is utilized to avoid discontinuities in the steering angle δ, which would arise with directly choosing δ as an input.

Frenet Transformation

System (6.1) is transformed into Frenet coordinates as described in [151] with the difference that this transformation is performed along the racing line instead

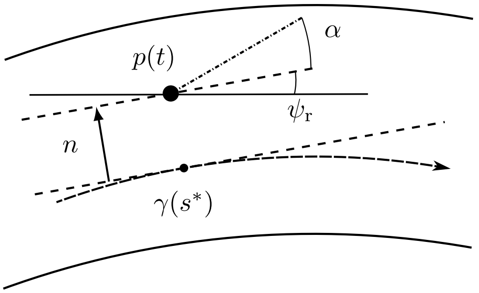

Figure 6.3: Path-parametric model as in [151].

of the center path. The resulting nonlinear dynamic system $\dot{x} = f(x, u)$ now depends on the curvature $\kappa(s)$ and reads as

$$\dot{s} = \frac{v\cos(\alpha+\beta)}{1-n\kappa(s)}, \tag{6.2a}$$

$$\dot{n} = v\sin(\alpha+\beta), \tag{6.2b}$$

$$\dot{\alpha} = \dot{\psi} - \kappa(s)\dot{s}, \tag{6.2c}$$

$$\dot{v} = \frac{F_{\mathrm{x}}^{\mathrm{d}}}{m}\cos(\beta), \tag{6.2d}$$

$$\dot{\delta} = r, \tag{6.2e}$$

with states $x = \begin{bmatrix} s, & n, & \alpha, & v, & \delta \end{bmatrix}^\top$, and controls $u = \begin{bmatrix} F_{\mathrm{x}}^{\mathrm{d}}, & r \end{bmatrix}^\top$. Path-aligned states are used which describe the progress on the transformation path $s(t)$, the normal distance to the transformation path $n(t)$ and the heading angle mismatch $\alpha(s,t) = \psi(t) - \psi_{\mathrm{r}}(s)$. Note that $\kappa(s)$ together with bounds on the normal distance state n fully describe the road geometry. Fig. 6.3 shows the transformation of a point p with respect to the curve $\gamma(s)$, normal distance n, the error angle α and the heading angle ψ_{r} of $\gamma(s)$. The path parameter s^* corresponds to the parameter of the closest point $\gamma(s^*)$ to the given position p of the vehicle model. The lateral acceleration of this model can be stated as

$$a_{\mathrm{lat}} = \frac{1}{l_{\mathrm{f}}+l_{\mathrm{r}}}v^2\delta + \frac{F_{\mathrm{x}}^{\mathrm{d}}}{m}\sin\left(\frac{\delta l_{\mathrm{r}}}{l_{\mathrm{f}}+l_{\mathrm{r}}}\right). \tag{6.3}$$

Linear Model for Mixed-Integer Programming

For the integer problem, an utterly simplified model is used, which basically accounts for a geometric offset from the Frenet transformation line and is formulated in the Frenet frame as well. The model lacks time dependency. It rather depends on the path progress s, which is discretized on a grid related to the obstacles and written as s_k. The discretization is performed to represent the obstacle shape related to the racing line. The lateral distance n_k is limited based on the road constraints $n_{k,\mathrm{left}}$ and $n_{k,\mathrm{right}}$. The continuous control variable u relates the only state variables n_k to each other and is split into a negative and positive part due to the integer formulation of the optimization objective. The control variable u expresses the steepness of the path deviation from the racing line between two nodes and is limited by $u_{\max}$. The model can be written as

$$n_{k+1} = f_k(u_k^{\mathrm{pos}}, u_k^{\mathrm{neg}}) = n_k + (u_k^{\mathrm{pos}} - u_k^{\mathrm{neg}})(s_{k+1} - s_k) \tag{6.4a}$$

$$0 \leq u_k^{\mathrm{pos}} \leq u_{\max} \tag{6.4b}$$

$$0 \leq u_k^{\mathrm{neg}} \leq u_{\max} \tag{6.4c}$$

$$n_{\mathrm{l,right}} \leq n_l \leq n_{l,\mathrm{left}} \tag{6.4d}$$

for $k = 0...N_{\mathrm{d}} - 1$ and $l = 0...N_{\mathrm{d}}$, where N_{d} is the number of discretizations of the longitudinal path variable s. This model describes a piece-wise linear path in Frenet coordinates.

Object Representation

Objects are modeled in the two-dimensional Frenet frame by means of polygons as shown in Fig. 6.4. A particular obstacle i of totally N_{O} obstacles is represented by a polygon with $N_{\mathrm{p}}^{\mathrm{O}i}$ points and a reward d of N_{R} rewards with $N_{\mathrm{p}}^{\mathrm{R}d}$ points respectively. The object's vertices are aligned with the discretization of the longitudinal coordinate, which is chosen such that it resembles the shape of the object “well enough”. Therefore, a vertex always has an opposite side point with the same longitudinal coordinate. Right or left sides are noted with $\{r, l\}$. The coordinate points in the Frenet frame of the polygon characterizing each reward or obstacle are written as

$$p_k^{\mathrm{R}d,\{\mathrm{r,l}\}} = \begin{bmatrix} s_k^{\mathrm{R}d} \\ n_k^{\mathrm{R}d,\{\mathrm{r,l}\}} \end{bmatrix}, \quad p_k^{\mathrm{O}i,\{\mathrm{r,l}\}} = \begin{bmatrix} s_k^{\mathrm{O}i} \\ n_k^{\mathrm{O}i,\{\mathrm{r,l}\}} \end{bmatrix}$$

with k as the global spatial index of the longitudinal Frenet axis.

6.1.3 Combinatorial Optimization

Binary Object-Boundary Relation

In order to find an optimal path through the obstacle setting in the Frenet frame with the reduced model (6.4), binary integer variables are used to construct

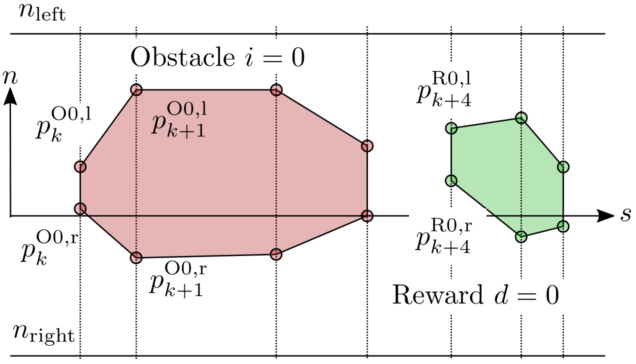

Figure 6.4: Object configuration in Frenet frame.

a linear mixed-integer problem. For each obstacle i, one binary variable ω^i indicates the passing side (left/right). Rewards are treated differently because they could be catched on different positions. For "short" rewards (where "short" is related to the longitudinal extension of the reward), one binary variable β^d sufficiently specifies whether this reward should be caught or ignored. For long rewards, "gates" with several binary variables β_k^d are used. For each long reward d, totally $N_{\mathrm{p}}^{\mathrm{R}d}/2$ binary variables are used, which are indexed by $l = 0 \dots (N_{\mathrm{p}}^{\mathrm{R}d}/2 - 1)$. Each binary variable β_l^d indicates if boundaries are set to gate the corresponding lateral state variable at the particular longitudinal position of the reward. A logical OR connective of the binary gate variables

$$\bar{\beta}^d = \beta_k^d \vee \cdots \vee \beta_{k+N_{\mathrm{p}}^{\mathrm{R}d}-1}^d \tag{6.5}$$

indicates the final binary state for reward d, which is then used to specify the associated cost. That means if at least one gate is "closed" (meaning the binary variable sets the boundaries active), the final path crosses the reward polygon at some point. This formulation leads to a high number of decision variables, which is necessary if a reward can be caught at multiple positions. For example, a very long "reward zone" aligned with the road might be entered in very different positions. With known shapes of the rewards, particularly where the longitudinal length in the Frenet frame is small (e.g., smaller than 5 meters in the setting used for the competition) (6.5) can be simplified by taking the left-most and the right-most polygon points to define a "gate" that is either switched active or inactive by just one integer variable β^d.

The binary variables are subsequently used to adjust boundaries, forming a homotopy class for the gradient-based optimization. Detailed considerations about homotopy classes in this context are shown in [31]. Obstacle boundary values $n_k^{\mathrm{O}i,\{\mathrm{r,l}\}}$ and its binary variables ω_i are related to state constraint on n_k

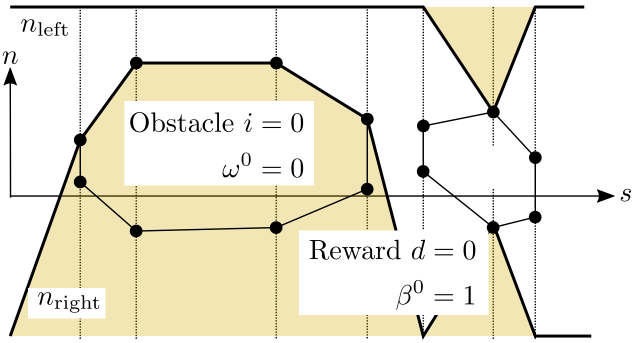

Figure 6.5: Road bounds aligned to object integer variables.

with the inequalities

$$n_k \geq \omega^i n_k^{\mathrm{O}i,\mathrm{r}} + (1 - \omega^i) n_{k,\text{right}} = \mathcal{H}_{\text{low}}(k, \omega),$$

$$n_k \leq (1 - \omega^i) n_k^{\mathrm{O}i,\mathrm{l}} + \omega^i n_{k,\text{left}} = \mathcal{H}_{\text{upp}}(k, \omega),$$

for $k = 0 \ldots N_\mathrm{d}$ where a binary state equal zero is defined as “passing left”. For relating the borders to the reward polygon, the equations

$$n_k \geq \beta_k^d n_k^{\mathrm{R}d,\mathrm{r}} - (1 - \beta_k^d) n_{k,\text{right}} = \mathcal{I}_{\text{low}}(k, \beta),$$

$$n_k \leq \beta_k^d n_k^{\mathrm{R}d,\mathrm{l}} - (1 - \beta_k^d) n_{k,\text{left}} = \mathcal{I}_{\text{upp}}(k, \beta),$$

are used, where the binary variable equal to one is defined as “catch”. Fig. 6.5 shows the changed bounds due to the integer variables.

Decision-Flickering Avoidance

During real-world conditions on vehicle hardware, the perception system receives slightly shifted versions of the same problem, where new objects appear infrequently. Very often, two solutions nearly have the same cost (e.g., an obstacle in the middle of the road), which could lead to jumping binary variables. For that reason, a penalty for changes to previous decisions is introduced that increases with the number of decisions computed previously by the algorithm and decreases with the distance of the object to the vehicle. In other words, binary re-decisions for objects far from the vehicle that have just appeared are “cheaper” than re-decisions for objects that are closer and longer present. To account for re-decisions, the binary values of c_i^R and c_i^O are used to form a logic XOR connective between the binary decision variables β_i or ω_i and the previous decisions $\hat{\beta}_i$ or $\hat{\omega}_i$, where

$$c_i^{\mathrm{R}} = \beta_i(1-\hat{\beta}_i) + (1-\beta_i)\hat{\beta}_i \tag{6.6}$$

$$c_i^{\mathrm{O}} = \omega_i(1-\hat{\omega}_i) + (1-\omega_i)\hat{\omega}_i \tag{6.7}$$

A parameter p_i accounts for the weight that this decision should be kept. The weight is updated iteratively and related to the mentioned criteria of distance and decision account by

$$p_i = w_{0,\mathrm{bin}} \max\left(1 - \frac{\Delta s_i}{d_0}, 0\right) \frac{\exp(a_0 c_i)}{1+\exp(a_0 c_i)}, \tag{6.8}$$

with $w_{0,\mathrm{bin}}, d_0$ and a_0 as scaling constants, c_i as the counter of decision repetitions for each binary variable, and Δs_i as the longitudinal distance of an obstacle corresponding to the binary variable i to the vehicle position. In (6.16), the related cost function for re-decisions is stated.

Cost Functions

The final cost function consists of several parts. First, the L1-norm of the deviation from the Frenet-transformation line (i.e., the optimal racing line) is stated. It is proportional by a weight w_{n} to the area of absolute lateral error in the Frenet frame and computed as

$$C_{\mathrm{n}} = w_{\mathrm{n}} \sum_{k=0}^{N_d} |n_k|. \tag{6.9}$$

The L1-norm cannot be directly formulated as a linear function. Therefore (6.4) is used, with the splitting of u into a positive and negative part according to a reformulation shown in [46]. With $\Delta s_k := s_{k+1} - s_k$, the variable $n_k = n_k^{\mathrm{pos}} - n_k^{\mathrm{neg}}$ and the cost C_{n} can also be written as

$$C_{\mathrm{n}} = w_{\mathrm{n}} \sum_{k=0}^{N_d} n_k^{\mathrm{pos}} + n_k^{\mathrm{neg}}, \tag{6.10}$$

$$\text{with, } n_k^{\mathrm{pos}} \geq 0,\ n_k^{\mathrm{neg}} \geq 0. \tag{6.11}$$

The positive and negative parts of n_k can be directly obtained by summing up the associated controls u_k. The inequalities (6.11) are therefore fulfilled with (6.4). The initial value n_0 is a constant and can also be split into a positive n_0^{pos} and a negative n_0^{neg} with $n_0 = n_0^{\mathrm{pos}} - n_0^{\mathrm{neg}}$. Consequently, n_{k+1} can be decomposed according to the following steps.

$$n_{k+1} = n_k + \Delta s_k u_k \tag{6.12a}$$

$$n_{k+1} = n_0 + \sum_{i=0}^{k} \Delta s_i u_i \tag{6.12b}$$

$$n_{k+1} = n_0 + \sum_{i=0}^{k} \Delta s_i (u_i^{\text{pos}} - u_i^{\text{neg}}) \tag{6.12c}$$

$$n_{k+1} = n_{k+1}^{\text{pos}} - n_{k+1}^{\text{neg}} \tag{6.12d}$$

The result (6.12d) is used to state (6.10) and therefore the cost in (6.9).

Secondly, the steepness of the deviation from the racing line defines the cost term $C_{\Delta\text{n}}$ with its associated cost $w_{\Delta\text{n}}$ in a similar way by using the approach of [46] for the "basis pursuit problem". Since the controls u_k represent the steepness of deviation, the decomposed parts of (6.4) can be added as costs for the steepness by

$$C_{\Delta\text{n}} = w_{\Delta\text{n}} \sum_{k=0}^{N_d} (u_k^{\text{pos}} + u_k^{\text{neg}}).$$

Thirdly, negative costs are added for catching a reward by

$$C_{\text{rew}} = \begin{cases} C_{\text{rew}}^{\text{ext}}, & \text{if } l_{\text{R}} \geq \bar{l}_{\text{R}} \\ C_{\text{rew}}^{\text{short}}, & \text{otherwise.} \end{cases}$$

These reward costs with their associated negative weight w_{rew} are split into extended costs $C_{\text{rew}}^{\text{ext}}$ for long rewards (with long referring to the longitudinal dimension l_{R} in the Frenet frame) and simplified ones $C_{\text{rew}}^{\text{short}}$ for short rewards. A threshold $\bar{l}_{\text{R}}$ is used for their classification. The simpler version of the problem for longitudinally short rewards reads as

$$C_{\text{rew}}^{\text{short}} = w_{\text{rew}} \sum_{i=0}^{N_{\text{R}}-1} \beta^i. \tag{6.13}$$

In the extended reward formulation (6.15) the logic OR connective is needed. Auxiliary continuous optimization variables $x^{\text{R}i}$ for the cost reduction of reward i are introduced. These auxiliary variables realize the OR connective between the l reward gates associated with their binary gating variable β_l^i when being minimized in the overall optimization problem. The set $\mathcal{K}_{\text{rew}}$ restricts the auxiliary cost variable to the (negative) weight value w_{rew} as a general lower bound or to 0 if all gates are open. Thus, their binary values are zero. This set reads as

$$\mathcal{K}_{\text{rew}}(\beta) = \left\{ x^{\text{R}} \in \mathbb{R}^{N_{\text{R}}} \middle| \, x^{\text{R}i} \geq w_{\text{rew}}, x^{\text{R}i} \geq w_{\text{rew}} \sum_{l=0}^{N_{\text{R}i}/2-1} \beta_l^i, \, i = 0 \ldots N_{\text{R}} - 1 \right\} \tag{6.14}$$

with variables summarized by

$$\beta = \left[\beta^0, \ldots, \beta^{(N_{\text{R}}-1)}\right], \quad \omega = \left[\omega^0 \ldots, \omega^{(N_{\text{O}}-1)}\right].$$

The extended reward costs

$$C_{\text{rew}}^{\text{ext}} = \sum_{i=0}^{N_{\text{R}}-1} x^{\text{R}i}, \text{ with } x^{\text{R}} \in \mathcal{K}_{\text{rew}}(\beta), \tag{6.15}$$

are simply the sum of auxiliary variables realizing the OR connective under their minimization.

Finally, the “re-decision costs” C_{bin} penalize the toggling of binary variables. Here, the weights p_i are different for every binary variable, as described in (6.8). The variable $c_i^{\text{O,R}}$ for obstacles and rewards indicates if the binary variable for the object has changed, which is equal to an XOR logic connective in (6.6) and (6.7). Please note that there might be several binary variables for each reward. Therefore, $\bar{N}_R$ is used to account for all binary variables related to rewards.

$$C_{\text{bin}} = \sum_{i=0}^{\bar{N}_{\text{R}}-1} p_i c_i^{\text{R}} + \sum_{i=0}^{N_{\text{O}}-1} p_i c_i^{\text{O}} \tag{6.16}$$

Final Problem Formulation

For the ease of notation, the continuous optimization variables are combined in

$$u = \begin{bmatrix} u_0^{\text{neg}} & \cdots & u_{N_d-1}^{\text{neg}} \\ u_0^{\text{pos}} & \cdots & u_{N_d-1}^{\text{pos}} \end{bmatrix}. \tag{6.17}$$

With the set of binary numbers $\mathcal{B} = \{0, 1\}$, the final mixed-integer problem (6.18) is stated by combining the model (6.4) with the cost model function

(6.10) by using the result of (6.12c) splitting the race path deviation n_k.

$$\begin{aligned}
\min_{\substack{u\in\mathbb{R}^{2\times N_d},\beta\in\mathcal{B}^{N_\beta},\\ \omega\in\mathcal{B}^{N_\omega},x^{\mathrm{R}}\in\mathbb{R}^{N_{\mathrm{R}}}}} \quad & C_{\mathrm{n}}(u)+C_{\Delta\mathrm{n}}(u)+C_{\mathrm{rew}}(x^{\mathrm{R}})+C_{\mathrm{bin}}(\beta,\omega) && \text{(6.18a)}\\
\text{s.t.} \quad & n_{k+1}=f_k(u_k^{\mathrm{pos}},u_k^{\mathrm{neg}}), && \text{(6.18b)}\\
& 0\le u_k^{\mathrm{pos}}\le u_{\max}, && \text{(6.18c)}\\
& 0\le u_k^{\mathrm{neg}}\le u_{\max} \qquad k=0,\dots,N_d-1, && \text{(6.18d)}\\
& \mathcal{I}_{\mathrm{low}}(k,\beta)\le n_k\le \mathcal{I}_{\mathrm{upp}}(k,\beta), && \text{(6.18e)}\\
& \mathcal{H}_{\mathrm{low}}(k,\omega)\le n_k\le \mathcal{H}_{\mathrm{upp}}(k,\omega) \quad k=0,\dots,N_d, && \text{(6.18f)}\\
& x^{\mathrm{R}}\in\mathcal{K}_{\mathrm{rew}}(\beta) && \text{(6.18g)}
\end{aligned}$$

Also, the reduced costs for reward catching (6.13) as well as the costs for keeping binary variables in (6.16) are included. N_β and N_ω denote the total count of all binary decision variables for rewards and obstacles.

6.1.4 Trajectory Optimization

After solving (6.18), a homotopy class is computed from the object polygons and their associated binary states. The road bounds are described by $\underline{n}(s)$ and $\overline{n}(s)$, which linearly interpolate the original road boundary points or the associated object boundaries (see Fig. 6.5) according to the homotopy class. As described in [151], the optimal control problem (OCP) of time-optimal racing can be described very generally by the following multiple shooting nonlinear program (NLP)

$$\min_{\substack{x_0,\dots,x_N,\\ u_0,\dots,u_{N-1}\\ \zeta_0,\dots,\zeta_N}} \quad \sum_{k=0}^{N-1} \|x_k - x_{k,\mathrm{ref}}\|_Q^2 + \|u_k\|_R^2 + \tag{6.19a}$$

$$\mu_i \left\|\zeta_k\right\|^2 + \nu_i \left\|\zeta_k\right\|_1 + \|x_N - x_{N,\mathrm{ref}}\|_{Q_\mathrm{N}}^2$$

$$\text{s.t.} \qquad x_0 = \bar{x}_0, \tag{6.19b}$$

$$x_{k+1} = F(x_k, u_k, t_\Delta), \tag{6.19c}$$

$$\underline{u} \le u_k \le \overline{u} \qquad k = 0, \dots, N-1, \tag{6.19d}$$

$$\underline{x} \le x_k \le \overline{x}, \tag{6.19e}$$

$$\underline{n}(s_k) - \zeta_k \le n_k \le \overline{n}(s_k) + \zeta_k, \tag{6.19f}$$

$$-\overline{a}_\mathrm{lat} \le a_\mathrm{lat}(x_k) \le \overline{a}_\mathrm{lat}, \tag{6.19g}$$

$$\zeta_k \ge 0 \qquad k = 0, \dots, N. \tag{6.19h}$$

It represents a tracking problem of a vehicle model, where the final reference point $x_{N,\mathrm{ref}}$ is set “out-of-reach” to obtain approximate time-optimal trajectories. Here, the 1.2-fold maximum achievable distance was chosen for the final reference point, where the maximum distance would correspond to the distance obtained by the maximum speed driven for the given time interval $(N+1)t_\Delta$. Its structure of a tracking problem and the associated quadratic cost function allows the usage of a fast Gauss-Newton Hessian approximation. As opposed to [151], the reference in (6.19) is set as a previously approximated optimal racing path rather than the center line of the road. The final values for the cost function weightings were tuned by experiments and are shown in Table 6.2. The vehicle model (6.2) is discretized with an integration scheme $F(x_k, u_k, t_\Delta)$ with fixed time intervals t_Δ and incorporated as (6.19c) into the NLP. Inequality (6.19e) puts box constraints on the states, which include maximum velocity and steering angle. The lateral state constraints (6.19f) represent the road boundary and are dependent on the longitudinal state variable s. Slack variables ζ_k for violating (6.19f) are used. An iterative procedure increases their penalty weighting in consecutive optimization iterations, which we call homotopy iterations. The lateral acceleration (6.3) is limited via (6.19g).

After n_SQP sequential quadratic programming (SQP) iterations of the solver, the weighting parameters μ_i and ν_i are increased as shown in Algorithm (1). Strictly increasing scheduling functions $\alpha_\mu(i)$ and $\alpha_\nu(i)$ govern the weighting of the boundary slack variables. This lets the weights for the boundary slack variables “grow”, which leads to a smooth transition of the boundary nonlinearity. With this procedure, the convergence time is reduced, which is shown in the results

Section 6.1.5. As a drawback, the re-weighting might be unnecessary for smooth obstacle boundaries and takes additional time.

Algorithm 1: Homotopy iterations for the NLP.

```
$i = 0$;
while $i \leq i_{\max}$ do
    $\mu \leftarrow \alpha_\mu(i)$;
    $\nu \leftarrow \alpha_\nu(i)$;
    solve NLP with $n_{\mathrm{SQP}}$ iterations;
    $i \leftarrow i + 1$;
end
```

6.1.5 Real-World and Simulation Results

The presented strategies for combinatorial optimization by mixed-integer programming (COMIP) and the trajectory optimization of Section 6.1.4 (TO) are two independent algorithms which were tested in two settings. Both COMIP and TO were tested in simulations together. The COMIP algorithm was further used on embedded hardware in the third Roborace competition in its so-called “season beta” (second series of competitions in 2020), where the presented TO was replaced by a simpler decoupled trajectory planning algorithm. The competition took place on the Bedford race circuit (England) in December 2020.

Field Test on the Bedford Race Circuit (COMIP only)

The proposed COMIP algorithm was tested on the NVIDIA DRIVE PX2, with Ubuntu 16.04. This electronic control unit (ECU) provides two CPUs (4x ARM Denver, 8x ARM Cortex A57) and two GPUs (2x Tegra X2, 2x Pascal GPU). The open-source Coin-OR CBC solver was used in a mixed Python/C++ `ROS` framework for solving the problem (6.18) with a nominal rate of 0.5 Hz. A varying amount between 0 and 50 virtual objects was received in generally random sizes but rectangular shapes in Cartesian coordinates. The race car “devbot” is described in [233]. This algorithm was also tested in simulations with the second setup, namely an HP Elitebook with an Intel Core i7-8550 CPU (1.8 GHz), which turned out to be faster by a factor of 4-20. In this setup, the output of the COMIP was used together with a subsequent algorithm for curvature minimization (which is a simplified and decoupled approach compared to the one described in Section 6.1.5). It obtained a minimum curvature path approximation as well as a final analytical speed maximization based on this minimum curvature path and road friction parameters as shown in [272] and [135]. According to [47] and [120], the minimum curvature path is a

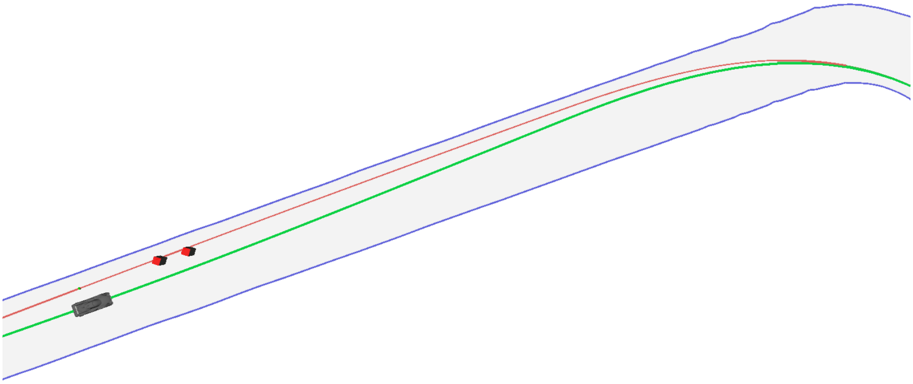

Figure 6.6: Visualization of recorded data of an evasion maneuver on the Bedford race circuit. The track boundaries are shown in blue, and the racing line (Frenet transformation line) in red. The car follows the computed trajectory, which is shown in green.

	ECU	Simulation
av. (max.) comp. time	1.9s (6.0s)	0.2s (0.4s)

Table 6.1: Maximum computation times of the overall optimization algorithm

good approximation for the optimal race path. The curvature approximation was computed with the algorithm described in [120], but with only first-order differences for race line deviations. Fig. 6.6 illustrates planning results from this event. Table 6.1 shows the maximum computation times for the combined algorithm (due to logging limitations by the embedded system), where the COMIP part accounts for 70% of the computation time on average.

Simulation in Virtual Environments (COMIP + TO)

The COMIP algorithm was also tested extensively on different race tracks with different object settings, including up to 40 differently shaped objects simultaneously within a prediction horizon of 300 meters. The measured times for the simulated algorithm on the Bedford race circuit are shown in Table 6.1.

The proposed subsequent TO was tested using the same simulation setup. After COMIP, the boundaries in (6.19f) were approximated by linear splines with a spatial discretization interval of the longitudinal coordinate s of 1 meter. For

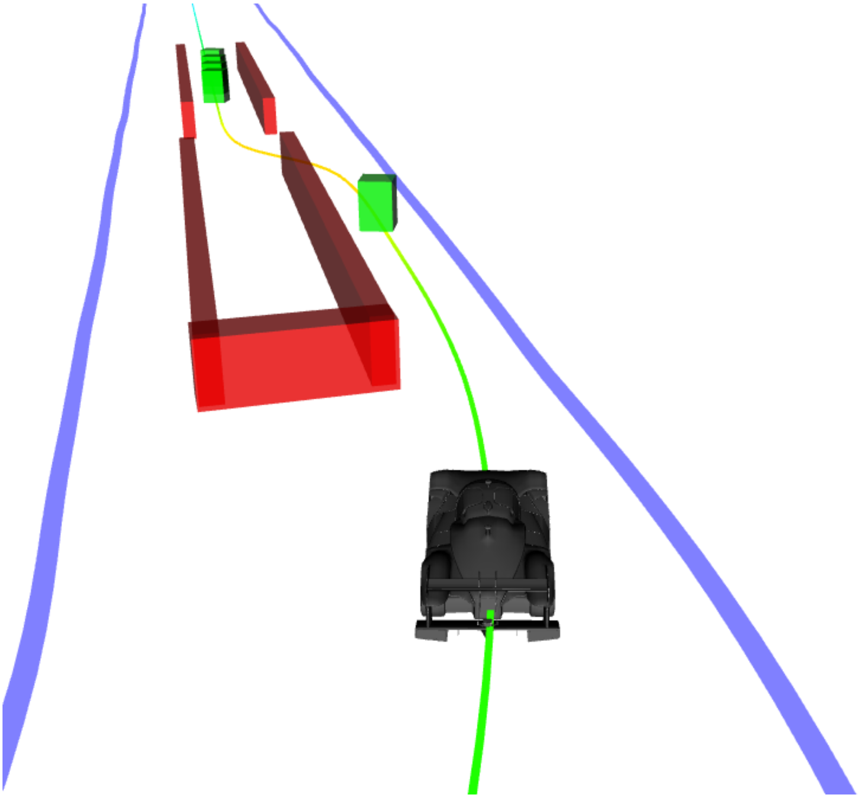

Figure 6.7: Trajectory obtained by the presented algorithms in a `ROS` simulation framework with obstacles (red) and rewards (green).

solving the NLP in (6.19), the kinematic vehicle model of Section 6.1.2 was used with a center of gravity at lengths $l_\mathrm{r} = 1.4\mathrm{m}$ and $l_\mathrm{f} = 1.6\mathrm{m}$ and parameters according to Table 6.2.

The NLP was solved with `acados` [291], where a Gauss-Newton Hessian approximation was used together with a two-stage implicit Runge-Kutta integration scheme. For solving the quadratic program (QP), the interior point solver `HPIPM` [97] was used. Altogether, with Algorithm (1) 8 SQP iterations are performed, with different weights for the slack variables, accounting for the homotopy iterations. The algorithm was compared to a standard setting with 8 SQP iterations on the final slack weights according to $i = 3$ in Table 6.2. With the constant slack weight setting, the NLP solver `acados` [291] could not find solutions for several obstacle configurations, where either QP iterations failed or the solution trajectory got stuck in front of obstacles. With the presented

Parameter Name	Value
Q	$\begin{bmatrix}10^{-6} & 10^{-3} & 1 & 10^{-4} & 10^{-1}\end{bmatrix}^\top$
Q_N	$\begin{bmatrix}10^{-2} & 10^{-1} & 10^{-2} & 10^{-4} & 2 \cdot 10^{-3}\end{bmatrix}^\top$
R	$\begin{bmatrix}10^{-3} & 10^{-2}\end{bmatrix}^\top$
$\alpha_\mu(i)$, $\alpha_\nu(i)$	10^i, $0.1 \cdot 10^{0.7i}$
i_max, n_SQP	3, 2
$\overline{a}_\mathrm{lat}$	$5 \, \frac{\mathrm{m}}{\mathrm{s}}$
N, t_Δ	100, 0.05 s

Table 6.2: Parameters for Algorithm 1

Name	Value
QP comp. time (min/mean/max)	6/9/20 ms
QP iterations (min/mean/max)	6/8/10
Total SQP iterations	8
Total NLP solution time (mean)	72 ms

Table 6.3: Results for Algorithm 1

homotopy iterations, both problems were mitigated. The results are summarized in Table 6.1.5.

6.1.6 Conclusion

This work presents two contributions to the sophisticated subproblems of trajectory planning for autonomous racing. First, a novel approximation of the combinatorial problem as a Frenet-frame-based linear mixed-integer problem is derived, which allows a fast and robust computation of a distinct homotopy class. Secondly, a homotopy strategy is presented to obtain robust convergence of a consecutive NLP. Using these approaches on a real embedded setup has been verified to achieve high performance in novel autonomous race car competitions and provide an alternative to the full state-space discretization of graph-search methods. Further considerations may include the combination of homotopy iterations with the integer problem and the extension to time-varying objects.

6.2 A Long-Short-Term Mixed-Integer Formulation for Highway Lane Change Planning

---

In this section, the paper published in [227] is reprinted verbatim. Note that the formatting of some formulas, terms, and numbers has been slightly adjusted for consistency without changing their meaning or content.

The contributions of each author are listed in the following.

Rudolf Reiter:	*idea (long term formulation starting from Chebeychev centering, short term formulation and combination), programming, simulation, experiment design and validation, proposing mathematical formulations, algorithm design, creation of the document*
Armin Nurkanović:	*mathematical corrections, grammar corrections, spelling style improvements, coherence review and improvement, simplification suggestions, rebuttal proof reading*
Daniele Bernardini:	*discussions in initial algorithm design phase, proof reading, grammar corrections, mathematical corrections*
Moritz Diehl:	*mathematical corrections, grammar corrections, spelling style improvements, coherence review and improvement, simplification suggestions*
Alberto Bemporad:	*idea (Chebeychev centering for long term formulation, prediction of surrounding vehicles), algorithm design, mathematical corrections, grammar corrections, spelling style improvements, coherence review and improvement, simplification suggestions, rebuttal proof reading*

Copyright ©2024 IEEE. Reprinted, with permission from Armin Nurkanović, Daniele Bernardini, Moritz Diehl and Alberto Bemporad. A Long-Short-Term Mixed-Integer Formulation for Highway Lane Change Planning. May/2024. DOI: 10.1109/TIV.2024.3398805.

Abstract. This work considers the problem of optimal lane changing in a structured multi-agent road environment. A novel motion planning algorithm that can capture long-horizon dependencies as well as short-horizon dynamics is presented. Pivotal to our approach is a geometric approximation of the long-horizon combinatorial transition problem, which we formulate in the continuous time-space domain. Moreover, a discrete-time formulation of a short-horizon optimal motion planning problem is formulated and combined with the long-horizon planner. Both individual problems, as well as their combination, are

formulated as MIQPs and solved in real-time by using state-of-the-art solvers. We show how the presented algorithm outperforms two other state-of-the-art motion planning algorithms in closed-loop performance and computation time in lane-changing problems. Evaluations are performed using the traffic simulator `SUMO`, a custom low-level tracking model predictive controller, and high-fidelity vehicle models and scenarios provided by the `CommonRoad` environment.

6.2.1 Introduction

In recent years many approaches have been proposed for vehicle motion planning in structured multi-lane road environments. However, considering combinatorial long-term dependencies and providing optimal trajectories subject to dynamic constraints in real-time remains a challenging problem. In fact, even deterministic two-dimensional motion planning problems with rectangular obstacles are NP-hard [219, 159].

This work proposes a novel iterative planning algorithm, referred to as long short term motion planner (LSTMP) that reduces the combinatorial complexity by splitting the problem into a short-term motion planning formulation (STF) and a long-term motion planning formulation (LTF), both solved by one MIQP, cf. Fig. 6.8. The STF aims at optimizing a four-state discrete-time trajectory of a point-mass model, including obstacle constraints, similar to the formulations of [213, 181]. The STF trajectory is computed for a shorter horizon to approximate a maximum of one lane change. In contrast, the LTF aims at obtaining optimal lane transitions, defined by the transition times and longitudinal transition positions, which are both continuous variables. These lane transitions are used for long-term planning, i.e., the choice of gaps between vehicles on several consecutive lanes. Reachability and the choice of transition gaps on consecutive lanes are modeled by disjunctive programming.

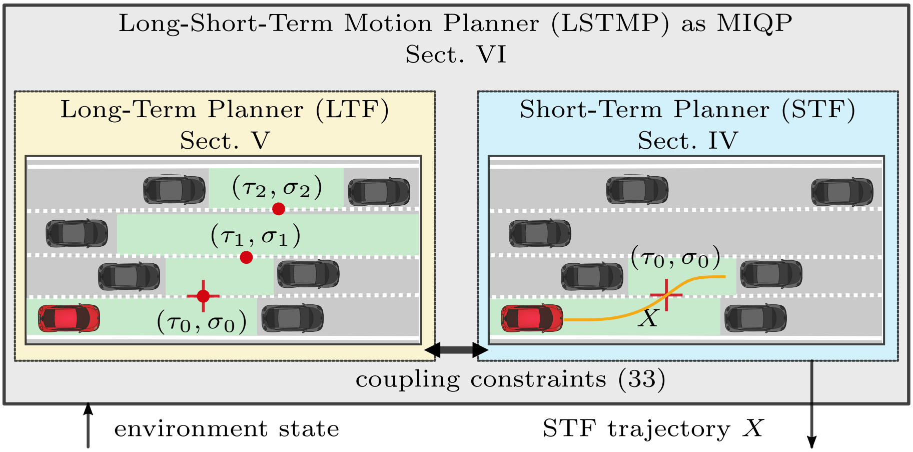

Figure 6.8: Overview of the proposed MIQP formulation for motion planning, referred to as LSTMP. The MIQP consists of long-term and short-term planning formulations where the decision variables of both are coupled through *consistency* constraints. The short-term decision variables include a continuous point-mass model trajectory to approximate a single lane change. The long-term decision variables account for selecting gaps between SVs on each lane.

The planned trajectory of the STF and the transitions of the LTF are formulated consistently, i.e., a transition point constrains the point-mass model trajectory to the corresponding lane. Contrary to strict hierarchical decomposition, the coarser approximation of the high-level plan cannot be infeasible for the low-level planner.

A challenge of state-of-the-art motion planners is the scaling of computational complexity with the horizon length [213], which makes long-horizon planning most often intractable. Within the formulation of the LTF, the locations of transitions in time and position are continuous. The proposed modeling uses integer variables only related to the gaps between vehicles on each lane. Therefore, the number of integer variables does not scale with the horizon length within the LTF. Consequently, the constant small number of integer variables, even for long-term predictions, allows for fast computation times of the algorithm.

Evaluations of the proposed approach are performed with both deterministic and interactive closed-loop simulations that involve a `CommonRoad` [13] vehicle model, a low-level NMPC, and interactive traffic that is simulated with `SUMO`.

Related Work

An abundance of fundamentally different techniques address highway motion planning and were reviewed by [199] and, more recently, by [67, 218] and particularly for deep learning in [17, 148]. The authors in [199] identified geometric, variational, graph-search, and incremental search methods as fundamental planning categories, whereas the more recent and exhaustive survey [67] adds a major part on artificial intelligence (AI), among further refinements.

Historically, geometric and rule-based approaches were more dominant. Parametric curves are used in highly structured environments, such as highways, due to their simplicity and ease of alignment with the road geometry [185, 257, 149]. However, the motion plans are usually conservative and unable to cope with complex environments [67].

In several works, the state space is discretized, and some sort of graph search is performed [199, 218]. Probabilistic road maps [139, 137], Djikstra graph search and similar, rapidly exploring random trees [23, 300, 296] are common in highway motion planning. Nonetheless, they suffer from the curse of dimensionality, the inability to handle dynamics, a poor connectivity graph, and a poor repeatability of results [67]. By using heuristics, hybrid A* [183, 8, 146] aims to avoid the problem of high-dimensional discretization in graph-search. Choosing an admissible heuristic is challenging, and a time-consuming graph generation is performed in each iteration. Moreover, authors try to improve graph- or sampling-based algorithms by combining them with learning-based methods [295, 237].

Optimization-based methods can successfully solve motion planning problems in high-dimensional state spaces in real-time [80]. They are appealing due to numerous advantages, e.g., consideration of dynamics and constraints, adaptability to new scenarios, finding and keeping solutions despite environmental changes, and taking into account complex scenarios. Pure derivative-based methods are often restricted to convex problem structures or sufficiently good initial guesses. Highway motion planning is highly non-convex. Nonetheless, by introducing integer variables, the problem can be formulated as an MIQP [209, 181, 164, 213] and solved by dedicated high-performance solvers, such as `Gurobi` [114]. Yet, the planning horizon of MIQP with a fixed discrete-time trajectory is still limited due to the increasing number of integer variables for increased horizon lengths. Therefore, keeping the computation time limited remains a challenge, [67].

One successful structure exploitation for solving the highway motion planning problem faster is the decomposition of the state space into *spatio-temporal* driving corridors [133, 30, 181, 173, 164, 77] with *simple* obstacle predictions. Still highly non-convex, the region can be decomposed into convex cells [173, 77] or used in a sampling-based planner [164].

To leverage the computational burden further, without sacrificing significantly the overall performance, the presented approach uses a short-horizon planning similar to [213] and [181], adding a long-horizon coarse geometric approximation in the spatio-temporal domain (see Fig. 6.8). The proposed STF differs from [181] by using only one binary variable per time step for the first lane change and more accurately modeling occupied regions that also consider braking due to preceding obstacles. Moreover, the consecutive gap is not fixed as in [181] but determined by the LTF. The idea of combining two horizons was presented in [158], yet not related to combinatorial motion planning and hierarchically decomposed in [146] using a graph-based planner.

AI-based methods often use exhaustive simulations to train NNs by reinforcement learning (RL) [148] or use expert data, such as data collected from human divers, to perform imitation learning [295, 51]. They often struggle to consider safety critical constraints and adapt to environment changes [67, 17, 51]. Furthermore, sim-to-real challenges apply [276] since these methods are mainly trained within simulations. An advantage includes the capability of using raw sensor inputs, such as camera images, and the low computational requirements of trained NNs [17].

The performance of the LSTMP is compared to a state-of-the-art hybrid A* method [8], which can be classified as both a deterministic planning AI and graph-search method [241], and to the mixed-integer programming-based decision maker (MIP-DM) [213] which is a comparable state-of-the-art optimization-based method.

Contribution

This paper contributes a novel algorithm for optimal lane-changing highway maneuvers. Compared to other highway motion planning algorithms, the proposed lane change motion planner approximates long-term dependencies in the spatio-temporal (ST)-space, where the computational burden is independent of the position and the time a lane change occurs. Moreover, we solve the problem involving the long-term approximation as a consistent single problem and, therefore, avoid a problematic decoupling. The closed-loop performance is improved by 15% compared to [213] and [8], and the average computation time is lowered in randomized interactive simulations by two orders of magnitude. Compared to [213], the number of integer variables used within the underlying optimization problem reduces from $O(N_{\text{veh}}N)$ to $O(N_{\text{veh}} + N)$ for a number of N_{veh} SVs and N discrete-time prediction steps, with a comparable closed-loop performance on the evaluated scenarios.

Outline

In Sect. 6.2.2, important background concepts are defined that are used throughout the paper. In Sect. 6.2.3, the general problem definition, related assumptions, and simplifications are introduced. Next, the planning approach is described in Sect. 6.2.4 to 6.2.6 and evaluated in Sect. 6.2.7. The approach is discussed, and conclusions are drawn in Sect. 6.2.8.

6.2.2 Preliminaries and Notation

The set of non-negative real numbers is denoted by $\mathbb{R}_{\geq 0} = \{x \in \mathbb{R} \mid x \geq 0\}$ and non-negative integers by $\mathbb{N}_0 = \{x \in \mathbb{Z} \mid x \geq 0\}$. Integer sets are written as $\mathbb{N}_{[m:n]} = \{z \in \mathbb{N}_0 \mid m \leq z \leq n\}$ with $m < n$. By using the notation $f(x; y)$ in the context of optimization problems, we denote the dependency of function f on variables x and parameters y. The convex hull $C \subseteq \mathbb{R}^n$ of two polygons $A \subseteq \mathbb{R}^n$ and $B \subseteq \mathbb{R}^n$ is $C = \operatorname{conv}(A, B)$. We use the floor function $\lfloor x \rfloor$ for rounding a number $x \in \mathbb{R}$ to the largest smaller integer and the ceil function $\lceil x \rceil$ for rounding to the smallest larger integer.

Propositional logic and mixed-integer notation

For a given compact set $\mathcal{X} \subset \mathbb{R}$ and continuous function $f : \mathcal{X} \to \mathbb{R}$, let $\overline{M} \geq \max_{x \in \mathcal{X}} f(x)$ and $\underline{M} \leq \min_{x \in \mathcal{X}} f(x)$ denote an upper and lower bound of $f(x)$ on $\mathcal{X}$, respectively. The following properties hold for a given $f : \mathcal{X} \to \mathbb{R}$ [283, 304].

Property 6.2.1. *For a product $y = \beta f(x)$, with $y \in \mathbb{R}$, the following equivalence holds for all $x \in \mathcal{X}$ and $\beta \in \mathbb{N}_{[0:1]}$:*

$$y = \beta f(x) \Leftrightarrow \begin{cases} y & \leq \overline{M}\beta, \\ y & \geq \underline{M}\beta, \\ y & \leq f(x) - \underline{M}(1-\beta), \\ y & \geq f(x) - \overline{M}(1-\beta). \end{cases} \tag{6.20}$$

Property 6.2.2. *The implication $[\beta = 1] \implies [f(x) \geq 0]$ of a binary variable $\beta \in \mathbb{N}_{[0:1]}$ that activates constraint $f(x) \geq 0$, is formulated as*

$$f(x) \geq \underline{M}(1-\beta). \tag{6.21}$$

Property 6.2.3. *The implication $[f(x) > 0] \implies [\beta = 1]$ of a binary variable $\beta \in \mathbb{N}_{[0:1]}$ that gets activated if constraint $f(x) > 0$ is valid, is formulated as*

$$f(x) \leq \overline{M}\beta. \tag{6.22}$$

Property 6.2.4. *The disjunction* $\bigvee_{i=1}^{N}[f_i(x) \geq 0]$ *is formulated by adding* N *binary variables* $\beta_i \in \mathbb{N}_{[0:1]}$ *with* $i \in \mathbb{N}_{[1:N]}$*, and the conditions*

$$\begin{aligned}
&[\beta_i = 1] \implies [f_i(x) \geq 0], \forall i \in \mathbb{N}_{[1:N]} \\
&\sum_{i=1}^{N} \beta_i \geq 1
\end{aligned} \tag{6.23}$$

Chebychev Center

The Chebychev center (CC) of a polyhedron $P = \{x \mid Ax \leq b\}$, with $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$, is the center $x^\star \in \mathbb{R}^n$ of the largest ball $B(x^\star, r^\star) = \{x \mid \|x^\star - x\| \leq r^\star\}$ contained in P [46]. The radius $r^\star$ is called the Chebychev radius. With A_i and b_i being the i-th row of A and b, respectively, the CC and Chebychev radius can be computed by solving the linear program

$$\min_{r,\, x} \quad -r \tag{6.24a}$$

$$\text{s.t.} \quad A_i x + r\|A_i\|_2 \leq b_i \quad i \in \mathbb{N}_{[1:m]}, \tag{6.24b}$$

$$r \geq 0 \tag{6.24c}$$

6.2.3 General Lane Changing Problem

The general problem for lane changing is stated as an optimal control problem (OCP), similar to [325]. For a multi-lane environment, a total of L lanes are defined by curvilinear center curves and a lane width d_{lane}. Moreover, we assume the existence of a parametric function $\gamma : \mathbb{R}_{\geq 0} \to \mathbb{R}^2$ for the rightmost reference lane that maps a longitudinal path coordinate s to a Cartesian point. The reference lane is parameterized by a vector θ_γ which is included in the road geometry parameters $\theta := \big(\theta_\gamma, d_{\text{lane}}\big)$. We consider a vehicle model in the Frenet coordinate frame [328, 302, 228] with states $x(t) \in \mathbb{R}^{n_x}$ and inputs $u(t) \in \mathbb{R}^{n_u}$, whose trajectories are governed by the nonlinear ordinary differential equation (ODE) $\dot{x} = \xi(x(t), u(t))$ with the initial condition $x(t_0) = x_0$. Using Frenet coordinates poses mild assumptions on the maximum value of the curvature, cf. [86]. Among others, the state of the Frenet model includes a longitudinal position state s, a lateral position state n, a velocity v, and a heading angle mismatch α [228].

States and controls are constrained by physical limitations depending on θ, which are expressed by admissible sets $\mathbb{X}(t;\theta)$ and $\mathbb{U}(t)$.

For M vehicles on each lane, we consider $N_{\text{veh}} = LM$ SVs with states $x_i^{\text{sv}}(x(t), t)$ for $i \in \mathbb{N}_{[1:N_{\text{veh}}]}$ that depend on the planned ego trajectory $x(t)$. Note that the dependency of the states $x_i^{\mathrm{sv}}(x(t), t)$ on the ego state $x(t)$ is due to the interaction of the ego vehicle with SVs and is a major source of complexity [298, 161]. We assume that the obstacle-free set can be approximated by $\mathbb{X}_{\mathrm{free}}(x(t), t)$. In the latter sections 6.2.4 and 6.2.5, we explain how to define the set $\mathbb{X}_{\mathrm{free}}(x(t), t)$ in an MIQP model.

In compliance with [77], multiple general objectives are proposed in the Frenet coordinate frame in order to define the desired behavior. A goal lane index $\tilde{l} \in \mathbb{N}_{[1:L]}$ and a reference velocity $\tilde{v} \in \mathbb{R}_{\geq 0}$, define the goal parameters

$$\Theta := (\tilde{l}, \tilde{v}).$$

Curvilinear reference paths are expressed as constant lateral references $\tilde{n}_i$ for $i \in \mathbb{N}_{[1:L]}$. One desired behavior is to track the lateral lane reference the vehicle is currently driving on. The current reference lane index $l(n)$ w.r.t. the current lateral state n is uniquely determined by

$$l(n) = \left\lceil \frac{n}{d_{\mathrm{lane}}} + \frac{1}{2} \right\rceil. \tag{6.25}$$

Note that determining the lane as in (6.25) within an optimization problem is not trivial and requires, for instance, the use of additional integer variables, as shown in Sect 6.2.4. By using the weights w_n and w_v, the cost of tracking the reference lane index $l(n(t))$ and longitudinal reference speed $\tilde{v}$ is

$$\begin{aligned} g_{\mathrm{ref}}\big(x(t), u(t); \theta, \Theta\big) = & w_n \Big(n(t) - \big(l(n(t)) - 1\big) d_{\mathrm{lane}}\Big)^2 + \\ & w_v \Big(v(t) - \tilde{v}\Big)^2 + u^\top(t) R u(t), \end{aligned} \tag{6.26}$$

which includes a quadratic penalty on the input u, with the positive definite weighing matrix $R \in \mathbb{R}^{n_u \times n_u}$.

Next, a cost for the distance to the goal lane $\tilde{l} \in \mathbb{N}_{[1:L]}$ with a weight $w_{\mathrm{g}} \in \mathbb{R}_{\geq 0}$ is added, which is the main objective of the presented planner and written as

$$g_{\mathrm{lane}}(x(t); \Theta) = w_{\mathrm{g}} \big| l(n(t)) - \tilde{l} \big|. \tag{6.27}$$

In the proposed approach, only lane changes towards the goal lane $\tilde{l}$ are considered.

Finally, the objective functional is

$$J\big(x(\cdot), u(\cdot); \theta, \Theta\big) := \int_{t=t_0}^{\infty} \Big(g_{\mathrm{ref}}(x(t), u(t); \theta) + g_{\mathrm{lane}}(x(t); \Theta)\Big) \mathrm{d}t, \tag{6.28}$$

and the considered general optimal control problem that is approximately solved by the proposed approach is

$$\min_{x(\cdot),\,u(\cdot)} \quad J\big(x(\cdot), u(\cdot); \theta, \Theta\big) \tag{6.29a}$$

s.t.

$$x(t_0) = x_0, \tag{6.29b}$$

$$\dot{x} = \xi(x(t), u(t)), \qquad t \in [t_0, \infty), \tag{6.29c}$$

$$x(t) \in \mathbb{X}_{\text{free}}(x(t), t; \theta) \cap \mathbb{X}(t; \theta), \quad t \in [t_0, \infty), \tag{6.29d}$$

$$u(t) \in \mathbb{U}(t), \qquad t \in [t_0, \infty). \tag{6.29e}$$

Assumptions and Simplifications

Several assumptions and simplifications are made for the proposed planning approach in order to approximate (6.29) by an MIQP. As a major simplification, the vehicle dynamics are formulated by a point-mass model with mass m in a Frenet coordinate frame [86], including the longitudinal and lateral position states s and n, as well as associated velocities v_s and v_n, with $x = [s, n, v_s, v_n]^\top$, and acceleration inputs a_s and a_n, with $u = [a_s, a_n]^\top$. The dynamics are modeled by

$$\dot{x} = [v_s, v_n, \frac{1}{m}a_s, \frac{1}{m}a_n]^\top. \tag{6.30}$$

We assume the absolute value of the curvature $\kappa(s)$ and its derivative $\kappa'(s)$ to be small for highway roads and, therefore, the acceleration in the Frenet coordinate frame is approximately equal to the acceleration in Cartesian coordinate frame [86]. This model was empirically shown to be valid for the cases where vehicles are not driving at their dynamical limits [181] and motivated in several other works, e.g., [213, 158, 58, 122, 255]. Critical evasion maneuvers are passed to a NMPC within the presented structure.

Assumption 6.2.1. *Lane changes of SVs can be detected.*

Ass. 6.2.1 can be satisfied by perception techniques described in [112, 199] or by vehicle-to-vehicle communication.

Assumption 6.2.2. *Considering two vehicles in the same lane, the rear vehicle is responsible for avoiding collisions. The leading vehicle must maintain general deceleration limits. Vehicles that change lanes must give way to vehicles on the lane they are changing to.*

Taking into account interactions among traffic participants within $\mathbb{X}_{\text{free}}(x(t), t; \theta)$ is essential for certain maneuvers in order to avoid prohibitive conservatism [285].

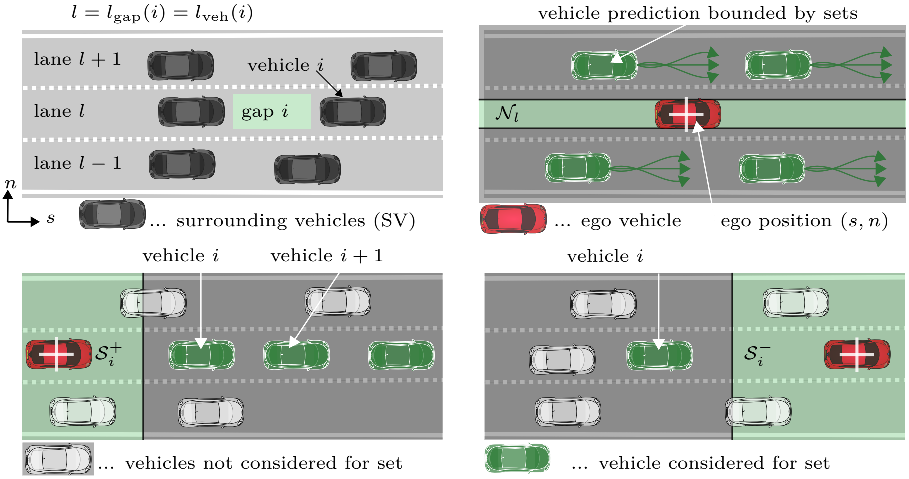

Figure 6.9: The first figure shows the enumeration of lanes and gaps, and the three rightmost figures show the sets related to free spaces. Surrounding vehicles (SVs) are uniquely enumerated. Gaps are the free spaces on a lane w.r.t. SVs and are enumerated according to the leading vehicles. An additional index is used for each frontmost gap. The sets $\mathcal{N}_l, \mathcal{S}_i^+$ and $\mathcal{S}_i^-$ define half-spaces in the SLT-space and are plotted in *green* for the position dimensions. The sets are *tightened* to include all configurations of the SVs and ego vehicle to allow collision-free planning with a point-mass model. All leading vehicles on the same lane are considered to construct the set $\mathcal{S}_i^+$ since any slower vehicle requires all following vehicles to brake. For the following vehicle set $\mathcal{S}_i^-$, only the closest vehicle to gap i is considered since preceding ones are assumed not to influence leading vehicles.

However, the interdependence of plans among interactive agents leads to computationally demanding game-theoretic problems [160, 56]. Similar to [181], the leader-follower interaction is simplified by ignoring collision constraints of followers on the same lane at the current state, leaving the responsibility for collision avoidance to the follower. Other SVs that are not following on the current lane are considered obstacles independent of the ego plan as long as these SVs are on adjacent lanes or in front of the ego vehicle.

The following assumptions consider constraints in the three-dimensional SLT-space [77], i.e., the space of the longitudinal and the lateral Frenet position states s and n and time t.

On each lane, a maximum of M vehicles are considered. We use indices $i \in \mathbb{N}_{[1:LM]}$ for the enumeration of the resulting maximum LM vehicles on the lanes in ascending order, starting from lane $l = 1$ from rear to front. The free space on the back of each vehicle along the lane is referred to as *gap* and enumerated according to the leading vehicle index, cf., Fig. 6.9. A number of L indices are added for the gaps in front of the first vehicle on each of the L lanes. Therefore, the number of gap indices is $L(M+1)$. The function $l_{\text{veh}}(i)$ returns the lane index of vehicle i. The function $l_{\text{gap}}(i)$ returns the lane index of gap i. The function $M_{\text{lane}}(i)$ returns the total number of vehicles on lane $l = l_{\text{veh}}(i)$ for a vehicle with index i.

In the following, we assume that $\mathbb{X}_{\text{free}}(x(t), t)$ can be partitioned into sets $\mathbb{X}_{\text{free}}^{\mathcal{I}}(x(t), t)$ related to SVs $\hat{x}_i^{\text{sv}}$ within indices in the set $i \in \mathcal{I} \subseteq \mathbb{N}_{[1:LM]}$, with $\mathbb{X}_{\text{free}}(x(t), t) \subseteq \mathbb{X}_{\text{free}}^{\mathcal{I}}(x(t), t)$ and $\mathbb{X}_{\text{free}}(x(t), t) = \mathbb{X}_{\text{free}}^{\mathbb{N}_{[1:LM]}}(x(t), t)$.

By inflating the obstacle shapes and lane boundaries by a safe distance according to all allowed configurations of the ego and the SVs, the planning problem can be formulated by a point-wise set exclusion of the curvilinear ego position states $s(t)$ and $n(t)$ [159].

Assumption 6.2.3. *For all SVs not driving at the current ego lane l, defined by the index set $\mathcal{I}(l) = \{k \in \mathbb{N}_{[1:LM]} \mid l \neq l_{\text{veh}}(k)\}$, an obstacle-free set $\mathcal{N}_l = \{n \in \mathbb{R} \mid \underline{n}_l \leq n \leq \overline{n}_l\}$ w.r.t. the ego lateral state n can be found such that*

$$n \in \mathcal{N}_l \implies x(t) \in \mathbb{X}_{\text{free}}^{\mathcal{I}(l)}(x(t), t).$$

Ass. (6.2.3) is used to define collision avoidance constraints to vehicles on adjacent lanes by formulating constraints on the lateral state n. Without further details, it is assumed that the bounds in Ass. 6.2.3 are *tight* enough to contain *most* of the adjacent lanes as free space, i.e., $\mathcal{N}_l \neq \emptyset$.

Assumption 6.2.4. *Given an SV with index i, upper position bounds $\overline{s}_i^{\text{sv}}$ and velocity bounds $\overline{v}_i^{\text{sv}}$ can be found that define the collision-free set*

$$\mathcal{S}_i^- = \left\{(s,t) \in (\mathbb{R}_{\geq 0} \times \mathbb{R}_{\geq 0}) \;\middle|\; s \geq \overline{s}_i^{\text{sv}} + t\overline{v}_i^{\text{sv}}\right\}. \tag{6.31}$$

such that it holds that

$$(t, s(t)) \in \mathcal{S}_i^- \implies x(t) \in \mathbb{X}_{\text{free}}^{\{i\}}(x(t), t).$$

Assumption 6.2.5. *For all SVs on lane l, with indices $i \in \mathbb{N}_{[1:M]}$, lower position bounds $\underline{s}_i^{\text{sv}}$, velocity bounds $\underline{v}_i^{\text{sv}}$ and leading vehicle distances Δs_i can be found that define the set*

$$\mathcal{S}_i^+ = \left\{(s,t) \in (\mathbb{R}_{\geq 0} \times \mathbb{R}_{\geq 0}) \;\middle|\; s \leq \underline{s}_i^{\text{sv}} + t\underline{v}_i^{\text{sv}} + \sum_{k=i+1}^{M_{\text{lane}}(i)} \underline{s}_k^{\text{sv}} + t\underline{v}_k^{\text{sv}} - \Delta s_{k-1}\right\}. \tag{6.32}$$

such that for $0 \leq t \leq \overline{t}$ it holds that

$$(t, s(t)) \in \mathcal{S}_i^+ \implies x(t) \in \mathbb{X}_{\text{free}}^{\{i,\ldots,M\}}(x(t), t).$$

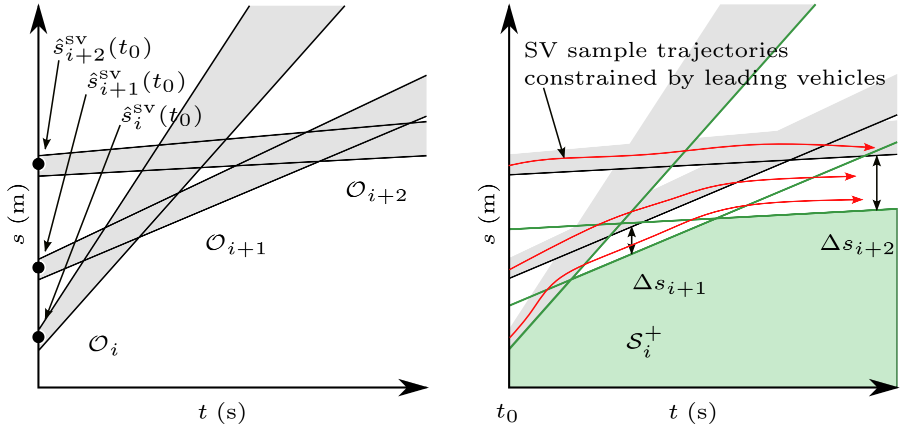

Figure 6.10: Construction of longitudinal ostacle-free space $\mathcal{S}_i^+$ for an SV with index i and two leading vehicles. The left plot shows the nominal prediction sets $\mathcal{O}_i$. The right plot shows the blocking lower-bound set enforced on the following vehicles. Red trajectories correspond to samples of actually driven trajectories.

Similar assumptions are made in related work, e.g., [133, 181], and with a more accurate lateral shape in [213]. The bounds in Ass. 6.2.5 approximate a distribution that is generated by the *intelligent driver model* [286], where a vehicle either drives or approaches a range around a reference velocity $\tilde{v}_i^{\text{sv}}$, with $\underline{v}_i^{\text{sv}} \leq \tilde{v}_i^{\text{sv}} \leq \overline{v}_i^{\text{sv}}$, or drives within a certain distance Δs_i to a slower leading vehicle [133, 181], cf., Fig. 6.10. The set

$$\mathcal{O}_i = \left\{(s,t) \in (\mathbb{R}_{\geq 0} \times \mathbb{R}_{\geq 0}) \;\middle|\; \underline{s}_i^{\text{sv}} + t\underline{v}_i^{\text{sv}} \leq s \leq \overline{s}_i^{\text{sv}} + t\overline{v}_i^{\text{sv}}\right\}$$

is referred to the nominal SV prediction set in the absence of leading vehicles. In each planning step, the bounds of $\mathcal{O}_i$ are updated based on the current SV state $\hat{x}_i^{\text{sv}}$, where for the velocity bounds it additionally holds that $\underline{v}_i^{\text{sv}} \leq \hat{v}_i^{\text{sv}} \leq \overline{v}_i^{\text{sv}}$ and for the position bounds $\underline{s}_i^{\text{sv}} \leq \hat{s}_i^{\text{sv}} \leq \overline{s}_i^{\text{sv}}$ holds.

Assumption 6.2.6. *The duration of a lane-change* t_{lc} *is upper-bounded by* $t_{\text{lc}} \leq \overline{t}_{\text{lc}}$.

For a concise notation, no offsets are assumed, i.e., the current lane and gap index are 1, the current planning time is assumed at zero seconds and the initial lateral reference and longitudinal estimated state are set to 0, therefore $\tilde{n}_0 = 0$ and $\hat{s} = 0$.

Obstacle-Free Set Approximations

In the following, convex obstacle-free sets in the SLT-space are defined as intersections of the sets $\mathcal{S}_i^+, \mathcal{S}_i^-$ and $\mathcal{N}_l$, which serve as a basis for the proposed LSTMP.

Two convex three-dimensional sets in the SLT-space are used to formulate a lane change from lane l and gap index g on the same lane, i.e., $l = l_{\text{gap}}(g)$, to gap index g^+ on the next lane $l+1$, i.e., $l+1 = l_{\text{gap}}(g^+)$. First, for lane-keeping, obstacle avoidance reduces to the problem of staying within the current lane boundaries (Ass. 6.2.3), ignoring following SVs on the same lane (Ass. 6.2.2) and consider leading SVs, with an upper-bound related to (6.32), stated as the convex obstacle-free set over the longitudinal and lateral position and time

$$\mathcal{F}_g^+ = \left\{(t,s,n) \mid (t,s) \in \mathcal{S}_g^+, \quad n \in \mathcal{N}_{l_{\text{gap}}(g)}\right\}. \tag{6.33}$$

Next, the free set for a lane change is defined in the two-dimensional ST-space, which is a subspace of the SLT-space, as

$$\mathcal{S}_{g,g^+}^{\text{lc}} = \left\{(t,s) \in \mathcal{S}_g^+ \cap \mathcal{S}_{g^+}^+ \cap \mathcal{S}_{g^+}^-\right\}. \tag{6.34}$$

Finally, as shown in Fig. 6.11, the free space related to a lane change from lane l and the related gap index g to lane $l+1$ and the related gap index g^+ is

$$\mathcal{F}_{g,g^+}^{lc} = \left\{(t,s,n) \in \mathcal{S}_{g,g^+}^{\text{lc}} \times \mathbb{R} \;\middle|\; n \in \operatorname{conv}(\mathcal{N}_{l_{\text{gap}}(g)} \cup \mathcal{N}_{l_{\text{gap}}(g^+)})\right\}. \tag{6.35}$$

For a lane change, both lanes are required to be free of SVs, and for the next lane $l+1$, also rear vehicles need to be considered for the duration of the lane change, cf. Ass. 6.2.2. Only the closest rear vehicle on the next lane needs to be considered since more distant vehicles are constrained by preceding ones, cf. Fig. 6.11.

The convexity of (6.33), (6.34) and (6.35) stems from the fact that each set is an intersection of hyperplanes, which implies convexity [46]. In case of a detected lane change of an obstacle, which we assume to be detectable (Ass. 6.2.1), both lanes are considered to be blocked for the whole prediction horizon, cf. Ass. 6.2.1.

6.2.4 Short-Horizon Approximations

The short-term motion planning formulation (STF) approximates the vehicle dynamics for a prediction horizon t_{f} and a maximum of one lane change towards the goal lane $\tilde{l}$, similar to [181, 213]. The selection of the particular gap index g^+ on the next lane is part of the long-term motion planning formulation (LTF), which is vice versa constrained by the trajectory of the STF in the final LSTMP formulation.

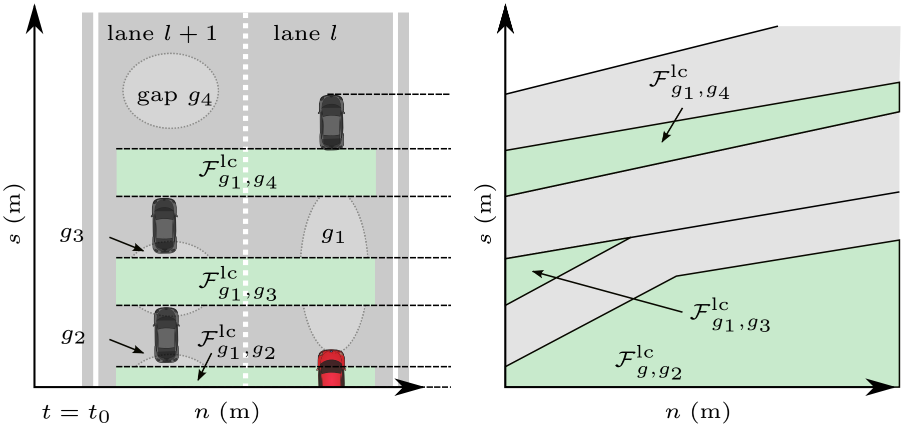
lane $l+1$
lane l
gap g_4
$\mathcal{F}^{\mathrm{lc}}_{g_1,g_4}$
g_3
g_1
$\mathcal{F}^{\mathrm{lc}}_{g_1,g_3}$
g_2
$\mathcal{F}^{\mathrm{lc}}_{g_1,g_2}$
s (m)
$t = t_0$
n (m)
$\mathcal{F}^{\mathrm{lc}}_{g_1,g_4}$
s (m)
$\mathcal{F}^{\mathrm{lc}}_{g_1,g_3}$
$\mathcal{F}^{\mathrm{lc}}_{g,g_2}$
n (m)

Figure 6.11: Sketch of obstacle-free sets $\mathcal{F}^{\mathrm{lc}}$ (green) for lane changing related to three SVs on the two lanes l and $l+1$. The left plot shows the curvilinear space with coordinates s and n. The right plot shows the ST-space. Three possible gaps with indices g_2, g_3, and g_4 on the consecutive lane $l+1$ are available for a transition from gap index g_1 and lane l.

Discretizing the model (6.30) with a discretization time t_d yields the linear discrete-time model

$$x_{k+1} = Ax_k + Bu_k. \tag{6.36}$$

A prediction horizon $t_\mathrm{f} = Nt_\mathrm{d}$ with N steps is used to approximate the infinite horizon in (6.29).

We define the acceleration bounds on the Frenet coordinate frame accelerations by the admissible control set

$$\mathbb{U} = \left\{u \in \mathbb{R}^{n_u} \;\middle|\; \underline{u} \leq u_k \leq \overline{u}\right\}, \tag{6.37}$$

where $\underline{u} = [\underline{a}_{\mathrm{lon}}, -\overline{a}_{\mathrm{lat}}]^\top$ and $\overline{u} = [\overline{a}_{\mathrm{lon}}, \overline{a}_{\mathrm{lat}}]^\top$. Nonetheless, higher curvatures and its derivatives can be inner-approximated by convex sets according to [86]. Moreover, the constraint

$$\alpha_\mathrm{l} v_s \leq v_n \leq \alpha_\mathrm{r} v_s \tag{6.38}$$

limits the lateral velocity in order to approximate the nonholonomic motion of a kinematic vehicle model.

The reference tracking cost (6.26) is approximated for the STF, whereas the remaining costs of the objective (6.28) are approximated as part of the LTF. Binary variables $\lambda_k \in \mathbb{N}_{[0:1]}$, with $\Lambda = [\lambda_0, \ldots, \lambda_N]$, are used to indicate whether the planned position is on the current lane, $\lambda_k = 0$, or on the next lane, $\lambda_k = 1$.

The lane indices are always updated w.r.t. the current state such that $\lambda_k = 0$ corresponds to the current lane. A lateral reference can, therefore be expressed by

$$\tilde{n}_k = d_{\text{lane}} \lambda_k,\ k \in \mathbb{N}_{[0:N]}, \tag{6.39a}$$

$$\lambda_{k+1} \geq \lambda_k,\ k \in \mathbb{N}_{[0:N-1]}. \tag{6.39b}$$

Constraint (6.39b) is used to cut off binary assignments to ease the solution of the MIQP problem. The constraint

$$\tilde{n}_k - \frac{d_{\text{lane}}}{2} \leq n_k \leq \tilde{n}_k + \frac{d_{\text{lane}}}{2}$$

is added to guarantee that from $\tilde{n}_k > 0$ the planned ego vehicle state is located on the next lane. Cost (6.26) can consequently be approximated with Frenet states as

$$g^{\text{st}}_{\text{ref}}(x_k, u_k, \lambda_k) = w_n (d_{\text{lane}} \lambda_k - n_k)^2 + w_v (\tilde{v} - v_{\text{s},k})^2 + u_k^\top R u_k. \tag{6.40}$$

Cost (6.40) includes the term $(d_{\text{lane}}\lambda_k - n_k)^2 = (d_{\text{lane}}\lambda_k)^2 - 2d_{\text{lane}}\lambda_k n_k + n_k^2$ with the bilinear term $-2d_{\text{lane}}\lambda_k n_k$ which cannot directly be handled by MIQP solvers [114]. Thus, this bilinear term is reformulated by introducing continuous auxiliary variables $q_{\text{bin},k} \in \mathbb{R}_{\geq 0}$, the additional constraint $q_{\text{bin},k} = \lambda_k n_k$ and further related constraints according to Property 6.2.1.

Finally, safety constraints approximating the set $\mathbb{X}_{\text{free}}(x(t), t)$ for the current and the next lane are formulated by considering M vehicles on the current lane, which is always set to $l = 1$ and the next lane $l^+ = 2$ and a chosen gap index $g^+ \in \mathbb{N}$ on the next lane, with $2 = l_{\text{gap}}(g^+)$.

Changing lane at time τ_1 follows three stages (cf. also [181]) where, in each stage, the constraints can be formulated as convex sets, cf., Fig. 6.12. First, at time $t \leq \tau_1 - \frac{1}{2}\bar{t}_{\text{lc}}$, the ego lane is tracked, second the lane is changed until the time limit $\tau_1 + \frac{1}{2}\bar{t}_{\text{lc}}$, and thirdly constraints for driving on the next lane hold for $t \geq \tau_1 + \frac{1}{2}\bar{t}_{\text{lc}}$. The lane change time on either lane is approximated by the upper-bound related to the time indices, $n_{\text{lc}} = \lceil \frac{\bar{t}_{\text{lc}}}{2t_{\text{d}}} \rceil$. Consequently, the lane change phases can be formulated in terms of index shifts of λ_k, with $[(1 - \lambda_{k+n_{lc}}) = 1]$ indicating the first stage, $[(\lambda_{k+n_{lc}} - \lambda_{k-n_{lc}}) = 1]$ indicating the transition phase and $[\lambda_{k-n_{lc}} = 1]$ indicating the last stage on the next lane, cf. Fig. 6.12. For out-of-range indices, i.e., $k < 0$ or $k > N$, the first value λ_0 or the last value λ_N are padded. For each position and time tuples (t_k, s_k, n_k) with $k \in \mathbb{N}_{[0:N]}$ and

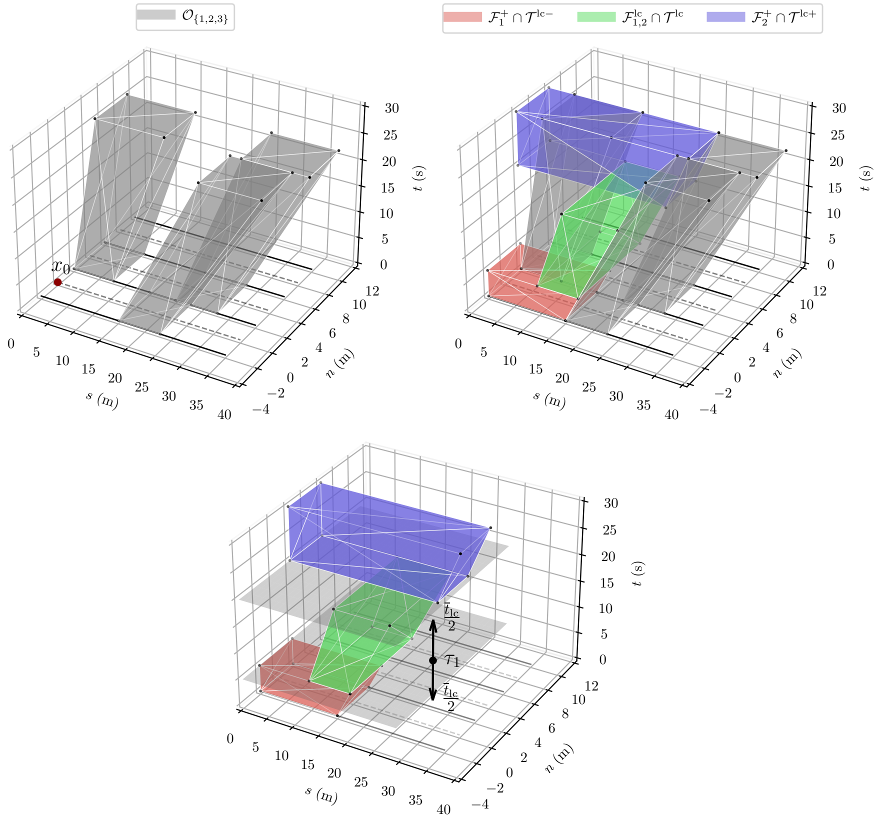

Figure 6.12: Visualization of SVs sets $\mathcal{O}_{\{1,2,3\}}$ in the SLT-space and consecutive convex free regions between gap index 1 and gap index 2. The time subspaces $\mathcal{T}^{\mathrm{lc}-} = \{(t,s,n) | t \leq \tau_1 - \bar{t}_{\mathrm{lc}}/2\}$, $\mathcal{T}^{\mathrm{lc}} = \{(t,s,n) | \tau_1 - \bar{t}_{\mathrm{lc}}/2 \leq t \leq \tau_1 + \bar{t}_{\mathrm{lc}}/2\}$ and $\mathcal{T}^{\mathrm{lc}+} = \{(t,s,n) | t \leq \tau_1 + \bar{t}_{\mathrm{lc}}/2\}$ define the consecutive time-related spaces on the planning horizon. The set $\mathcal{F}_1^+ \cap \mathcal{T}^{\mathrm{lc}-}$ is the obstacle-free space on the first lane before the lane change, $\mathcal{F}_{1,2}^{\mathrm{lc}} \cap \mathcal{T}^{\mathrm{lc}}$ is the free space during the lane change and $\mathcal{F}_2^+ \cap \mathcal{T}^{\mathrm{lc}+}$ is the obstacle-free space on the next lane after the lane change. Rear vehicles in the same lane are ignored, i.e., a vehicle is always allowed to brake. The binary variables λ_k determine which set constraints are active for each x_k.

the current gap index $g = 1$, we require

$$[1 - \lambda_{k+n_{lc}} = 1] \implies (t_k, s_k, n_k) \in \mathcal{F}_1^+, \tag{6.41a}$$

$$[\lambda_{k-n_{lc}} - \lambda_{k+n_{lc}} = 1] \implies (t_k, s_k, n_k) \in \mathcal{F}_{1,g^+}^{\mathrm{lc}}, \tag{6.41b}$$

$$[\lambda_{k-n_{lc}} = 1] \implies (t_k, s_k, n_k) \in \mathcal{F}_{g^+}^+, \tag{6.41c}$$

where the implications are reformulated according to Property 6.2.2.

Recursive feasibility requires disjunctive terminal velocity constraints depending on the final lane, which is either the current lane implying $[\lambda_N = 0]$ or the next lane, implying $[\lambda_N = 1]$. Therefore, the terminal set is expressed by

$$[\lambda_N = 1] \implies v_{s,N} \leq \underline{s}_{g^+}^{\mathrm{sv}} + t_{\mathrm{d}} N \underline{v}_{g^+}^{\mathrm{sv}}, \tag{6.42a}$$

$$[1 - \lambda_N = 1] \implies v_{s,N} \leq \underline{s}_1^{\mathrm{sv}} + t_{\mathrm{d}} N \underline{v}_1^{\mathrm{sv}}, \tag{6.42b}$$

$$v_{n,N} = 0. \tag{6.42c}$$

Note that this terminal set is restrictive since it upper-bounds the final velocity with the velocity of the preceding vehicle on the respective lane. An increased terminal safe set could be formulated by piece-wise linear approximations of deceleration constraints, which requires further binary variables, cf. [181].

So far, constraints and costs have been introduced that are used as part of the STF to plan a collision-free discrete-time trajectory from the current lane to a certain gap at the next lane. Noteworthy, this trajectory is constrained such that it is always safe w.r.t. the obstacle constraints. The selection of the possible gap indices and also all further gaps towards the goal lane are formulated in the LTF and explained in the next section. The STF and the LTF are formulated in the final MIQP with mutual constraints, such that the rather approximate LTF cannot plan transitions that are infeasible w.r.t. the STF.

6.2.5 Long-Horizon Approximations

Within the LTF, costs and constraints are formulated that select collision-free transition gaps between two adjacent lanes. For long horizons, a fixed discretization in time is prohibitive since the number of variables increases with the horizon length and would make the optimization problem hard to solve [213]. To circumvent the computational scaling with the prediction time, we propose a formulation in the two-dimensional continuous ST-space, where we exclusively model the transitions as points in time and longitudinal position for each lane change, with the transition times $T = [\tau_1, \ldots, \tau_{L-1}]^\top$ and longitudinal transition positions $\Sigma = [\sigma_1 \ldots, \sigma_{L-1}]^\top$.

In the following, three synergetic concepts are formulated to approximate the transitions, i.e., constraints that approximate reachability, a formulation for guaranteeing and maximizing the distance to SVs and a disjunctive formulation for choosing among gaps between vehicles for each lane. Binary variables are used to indicate whether transitions are invalid, resulting in the tuples of valid transitions for $\bar{L} \leq L$ lanes.

Approximate Reachability

Reachability between transitions is approximated by the set $\mathcal{R}(\tau_l, \sigma_l)$ using constraints defined by operating velocity bounds $\underline{v}_{\text{op}}$ and $\overline{v}_{\text{op}}$ and an approximation $\tilde{t}_{\text{lc}}$ of the time required to traverse a lane. The operating velocity bounds are artificially added to approximate the *true* reachable set around the expected velocity range of the vehicle.

The approximated set is used to define constraints for the next transition $(\tau_{l+1}, \sigma_{l+1})$, cf. Fig. 6.13. Each reachable set depends on the last transition (σ_l, τ_l) by the shifted cone

$$\mathcal{R}(\tau_l, \sigma_l) = \left\{ (\tau_{l+1}, \sigma_{l+1}) \middle| \begin{matrix} \sigma_{l+1} \leq \sigma_l + \overline{v}_{\text{op}}(\tau_{l+1} - \tau_l - \tilde{t}_{\text{lc}}) \\ \sigma_{l+1} \geq \sigma_l + \underline{v}_{\text{op}}(\tau_{l+1} - \tau_l + \tilde{t}_{\text{lc}}) \end{matrix} \right\}. \tag{6.43}$$

The convex reachable set (6.43) is an approximation using the velocity bounds $\overline{v}_{\text{op}}$ and $\underline{v}_{\text{op}}$. Using bounds on the acceleration would result in nonconvex quadratic constraints, which could still be approximated by the problem-specific parameters $\tilde{t}_{\text{lc}}, \overline{v}_{\text{op}}$ and $\underline{v}_{\text{op}}$.

Chebychev Centering for Transitions

Next, criteria for determining the locations of continuous transition points are defined. Transitions require an obstacle-free area for a minimum of the duration of the lane-change $\bar{t}_{\text{lc}}$, as defined for the STF. Beyond the minimum required time, an approach based on the Chebychev center (CC) is proposed that centers the transition in the ST-space related to obstacle constraints, i.e., the transition should be planned at a maximum weighted distance to constraints.

The CC formulation of (6.24) is used as a basis for further considerations. Centering constrains for the polytope defined by $[\tau,\ \sigma] \in \mathcal{S}_g^+$ are written as $h_g^+(\tau, \sigma, r) \leq 0$ according to (6.24), which includes the centering radius r. For the polytope defined by $[\tau,\ \sigma] \in \mathcal{S}_g^+ \cap \mathcal{S}_g^-$, the centering constraints are denoted by $h_g(\tau, \sigma, r) \leq 0$.

Notice that the ST-space has different units, namely longitudinal distance and time. To achieve a meaningful distance measure, the time coordinate is scaled

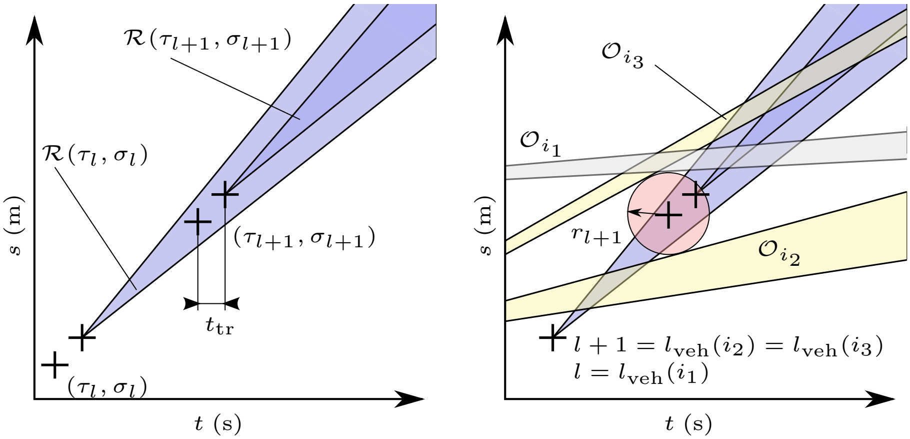
$\mathcal{R}(\tau_{l+1}, \sigma_{l+1})$
$\mathcal{R}(\tau_l, \sigma_l)$
s (m)
$(\tau_{l+1}, \sigma_{l+1})$
t_{tr}
(τ_l, σ_l)
t (s)
$\mathcal{O}_{i_3}$
$\mathcal{O}_{i_1}$
s (m)
r_{l+1}
$\mathcal{O}_{i_2}$
$l + 1 = l_{\text{veh}}(i_2) = l_{\text{veh}}(i_3)$
$l = l_{\text{veh}}(i_1)$
t (s)

Figure 6.13: The left plot shows the reachable sets after the transition to lane l and after the transition to lane $l + 1$. The reachable set has an offset related to the estimated traversal time $\tilde{t}_{\text{lc}}$. The right plot shows the Chebychev center (CC) of the transition to lane $l + 1$ from gap index i_1 to gap index i_3 with two SVs on the next lane $l + 1$ (yellow) and one SV on the current lane l (grey). The time axis is scaled by the reference velocity $\tilde{v}$, which is here assumed to be 1.

by the reference velocity $\tilde{v}$. Therefore, the unit of the radius is measured in meters.

The constraint (6.24c) is tightened to $r \geq \underline{r}$ to guarantee the minimum distance to obstacle constraints in the ST-space, i.e., the centering is only feasible for a centering radius higher than a threshold $\underline{r}$. Using the sequence of gap indices $[g_1, \ldots, g_{L-1}]$, with $l = l_{\text{gap}}(g_l)$, for each lane transition and the accumulated cost

$$G_{\text{safe}}(R) = -w_{\text{safe}} \sum_{l=1}^{L-1} r_l,$$

with transition radii $R = [r_1, \ldots, r_{L-1}]^\top$ and a weight w_{safe} to promote a further safety distance beyond the hard constraints, all transitions can be formulated in the shared linear program

$$\min_{R,\Sigma,T} \quad G_{\text{safe}}(R) \tag{6.44a}$$

$$\text{s.t.} \qquad h^{+}_{g_l}(\tau_l,\sigma_l,r_l) \leq 0, \quad l \in \mathbb{N}_{[1:L-2]}, \tag{6.44b}$$

$$h_{g_{l+1}}(\tau_l,\sigma_l,r_l) \leq 0, \quad l \in \mathbb{N}_{[1:L-2]}, \tag{6.44c}$$

$$\underline{r} \leq r_l, \quad l \in \mathbb{N}_{[1:L-1]}. \tag{6.44d}$$

Fig. 6.13 shows the centering of a transition $(\tau_{l+1},\sigma_{l+1},r_{l+1})$ from lane l to lane $l+1$ in the presence of a leading SV i_1 on lane l with $l = l_{\text{veh}}(i_1)$ and two SVs i_2, i_3 on the next lane $l+1$ with $l+1 = l_{\text{veh}}(i_2) = l_{\text{veh}}(i_3)$. Besides the SVs constraints, the center of the transition $(\tau_{l+1},\sigma_{l+1})$ is constrained by the approximated reachable set.

The linear program (6.44a) is not directly solved, but its cost and constraints are included in the final LSTMP MIQP. Notice that, therefore, the centering is solved as a weighted trade-off to other constraints, such as the duration of the lane change. The sequence of gap indices is determined by the disjunctive formulation including the constraints within a "big-M" formulation, cf. Property 6.2.4, and explained in the next Section 6.2.5. Notice that computing the transitions purely by maximizing the distance to SVs, without including a measure along the time axis, would ignore the safety distance related to the relative velocity of vehicles.

Disjunctions Among Gaps

A fundamental combinatorial aspect of lane change planning is the choice of gap indices on each lane. In Sect. 6.2.5, it was shown how to constrain transitions to an approximate reachable set, and in Sect. 6.2.5, a formulation to center a transition in the ST-space was introduced, given a sequence of gap indices. As an essential final component of the LTF, a disjunctive formulation of choosing a single gap on each lane is proposed according to Property 6.2.4. To activate constraints related to a certain gap, on each transition (τ_l,σ_l) binary variables $\beta_i \in \mathbb{N}_{[0:1]}$ are used and summarized in the vector $B \in (\mathbb{N}_{[0:1]})^{(L-1)(M+2)}$. The activation of gaps starts on the second lane since the current lane gap is trivially fixed. In each lane, one additional binary variable is added to account for the option of *no transition* or *lane-keeping*. Therefore, this particular binary variable with index $\hat{g}$ implies the variables (τ_l,σ_l) to be unconstrained by defining $h_{\hat{g}}(\tau_l,\sigma_l,r_l) \leq M$, where M is a large number. For the following definitions, we define the set $\mathcal{G}_l := \{g \mid l = l_{\text{gap}}(g)\}$, i.e., the set of all ´gap indices on a particular lane, including the additional virtual unconstrained one and the set $\hat{\mathcal{G}}$ that contains all indices of unconstrained added gaps. The disjunctions

$$\bigvee_{g^+\in\mathcal{G}_{l+1}} \begin{bmatrix} \beta_{g^+} \\ h_{g^+}(\tau_l,\sigma_l,r_l)\leq 0 \end{bmatrix}, \quad \forall l\in\mathbb{N}_{[1:L-1]}, \tag{6.45}$$

constrain the transition (τ_l, σ_l) onto lane $l+1$ by the leading and following vehicles of a selected gap index g^+, where $\beta_{g^+} = 1$ and the disjunctions

$$\bigvee_{g\in\mathcal{G}_{l}} \begin{bmatrix} \beta_{g} \\ h_{g}^{+}(\tau_l,\sigma_l,r_l)\leq 0 \end{bmatrix}, \quad \forall l\in\mathbb{N}_{[2:L-1]}, \tag{6.46}$$

constrains the transition (τ_l, σ_l) from lane l to the next lane only by the leading vehicles. In the case of *no transition*, i.e., the additional binary variables $\beta_{\hat{g}}$ is activated, where $\hat{g}$ the index of the virtual added unconstrained gap, a high cost $G_{\text{lane}}(\Sigma, T, B)$ is added that approximates (6.27) for not changing the lane.

Note that the binary variables B are related to the gaps on each lane, starting with the second lane $l = 2$. The transitions (τ_l, σ_l) are related to two adjacent lanes l and $l+1$, starting with the transition from the first lane $l = 1$. This distinction is crucial to the disjunctive constraints for each transition. For a transition l related to the departing lane l, only leading vehicle constraints $h_g^+(\tau_l, \sigma_l, r_l) \leq 0$ related to gap index g are considered according to Ass. 6.2.2. For the next lane gap index g^+ both, the preceding constraints and the leading vehicle constraints in $h_{g^+}(\tau_l, \sigma_l, r_l) \leq 0$ are used. This formulation models interactive behavior by allowing to slow down SVs on the current lane to reach a certain gap on the next lane.

The following further constraint on the binary variables

$$\sum_{g\in\mathcal{G}_l} \beta_g = 1, \quad \forall l\in\mathbb{N}_{[2:L]}, \tag{6.47}$$

reduces the search space for the mixed-integer (MI) solver. For each pair of consecutive virtual gaps $\hat{g}$ and $\hat{g}^+$, with $l_{\text{gap}}(\hat{g}^+) = l_{\text{gap}}(\hat{g}) + 1$, the physical constraint

$$\beta_{\hat{g}^+} \geq \beta_{\hat{g}}, \tag{6.48}$$

sets all further lane-changes $l > l_1$ to *lane-keeping*, if a lane is blocked.

Cost Approximations

In the following, a lane changing cost $G_{\text{lane}}(\Sigma, T, B)$ that approximates (6.27) and a reference velocity cost $g_{\text{ref}}^{\text{lh}}(\Sigma, T)$ that approximates (6.26) and penalizes transitions with deviations of the reference velocity $\tilde{v}$ are formulated.

Cost (6.27) linearly penalizes the number of lanes distant from the goal lane and is integrated over time in the final objective. This integral can be approximated over a horizon t_f by the following sum

$$\int_{t=t_0}^{t_f} g_{\text{lane}}(x(t);\Theta)dt \approx G_{\text{lane}}(\Sigma, T, B) := w_g \sum_{\hat{g}\in\hat{\mathcal{G}}} \tau_l(1-\beta_{\hat{g}}) + t_f\beta_{\hat{g}}, \qquad (6.49)$$

which penalizes the duration τ_l on each lane l, if there was a valid transition, i.e., $\beta_{\hat{g}} = 0$. If *no transition* was computed for lane l, i.e., $\beta_{\hat{g}} = 1$, the cost for the full horizon staying on the lane is summed in (6.49). Note that (6.49) contains bilinear terms of binary variables $\beta_{\hat{g}}$ and σ_l, both decision variables. The terms are treated by introducing an additional variable $q_{\text{bi},l}$ for each bilinear term, cf., Property 6.2.1. All auxiliary variables related to bilinear terms are summarized by $Q_{\text{bi}} = [q_{\text{bi},1}, \dots, q_{\text{bi},L}, q_{\text{bin},1}, \dots, q_{\text{bin},N-1}]$

Finally, the difference of the reference speed according to (6.26) is penalized for two consecutive transitions by

$$G_{\text{ref}}(T,\Sigma) = \frac{w_v t_f}{l_g}\left(\sum_{l=2}^{l_g}\left((\sigma_l - \sigma_{l-1}) + (\tau_l - \tau_{l-1})\tilde{v}\right)^2\right). \qquad (6.50)$$

Cost (6.50) approximates the duration between lane transitions with the constant value $\frac{t_f}{l_g}$, starting from the second transition as the reference cost approximation of (6.26) for the first transition is included in the STF cost (6.40).

Notably, this cost approximation for the reference velocity neglects the planned time driving on each lane.

6.2.6 Long-Short-Horizon Motion Planner

In the following, we complete the final motion planning MIQP problem with necessary additional formulations for combining the STF of Sect. 6.2.4 and the LTF of Sect. 6.2.5.

First, the formulations of the STF and the LTF are combined *consistently* according to the following definition for the first transition (τ_1, σ_1).

Definition 6.2.1. *A transition (τ, σ) is consistent with the longitudinal states s_k and the lateral states n_k of a trajectory, with $k \in \mathbb{N}_{[0:N]}$, if and only if the* *following inequalities hold*

$$n_k \leq \frac{d_{\text{lane}}}{2}, \forall k \in \{i \in \mathbb{N}_{[0:N]} \mid it_\Delta \leq \tau\}, \tag{6.51a}$$

$$n_k > \frac{d_{\text{lane}}}{2}, \forall k \in \{i \in \mathbb{N}_{[0:N]} \mid it_\Delta > \tau\}, \tag{6.51b}$$

$$s_k \leq \sigma, \forall k \in \{i \in \mathbb{N}_{[0:N]} \mid it_\Delta \leq \tau\}, \tag{6.51c}$$

$$s_k > \sigma, \forall k \in \{i \in \mathbb{N}_{[0:N]} \mid it_\Delta > \tau\}. \tag{6.51d}$$

Def. 6.2.1 states that the position states of the STF trajectory must be located on the current lane, closer than the longitudinal position σ and before the transition time τ and on the consecutive lane and position, thereafter.

In the following, the consistency formulation for the first transition (τ_1, σ_1) of a feasible solution of the LSTMP is shown. Therefore, constraints among the discrete decision variables λ_k, the transition (τ_1, σ_1) and positions s_k of the STF are defined by the pair-wise *exclusive* disjunctions according to Property 6.2.4,

$$\begin{bmatrix} [\lambda_k = 1] \\ kt_d \geq \tau_1 \\ s_k \geq \sigma_1 \end{bmatrix} \vee \begin{bmatrix} [\lambda_k = 0] \\ kt_d < \tau_1 \\ s_k < \sigma_k \end{bmatrix}, \quad \forall k \in \mathbb{N}_{[1:N]}, \tag{6.52}$$

For each pair k, the disjunctions (6.52) use the same binary variable λ_k, yet, with the opposite indication, i.e., either $[\lambda_k = 0]$ or $[\lambda_k = 1]$, which makes it exclusively choosing the related constraints.

Moreover, a terminal set formulation for the STF is required to reach a transition (τ_1, σ_1) with $\tau_1 \geq Nt_{\text{d}}$, i.e., the transition time τ_1 is further distant than the final STF prediction time Nt_{d}. It holds that $\tau_1 \geq Nt_d \Leftrightarrow [\lambda_N = 0]$, so the reachable set can be conditioned on λ_N by

$$[\lambda_N = 0] \implies (\tau_1, \sigma_1) \in \mathcal{R}(Nt_d, s_N). \tag{6.53}$$

The final LSTMP, formulated as an MIQP, can be stated by decisions variables, costs, and constraints of the STF, the LTF, and with the additional coupling constraints (6.52) and (6.53).

The STF decision variables are $X_{\text{s}} = (X, U, \Lambda) \in V_{\text{s}}$, where

$$V_{\text{s}} = \mathbb{R}^{N \times n_x} \times \mathbb{R}^{N-1 \times n_u} \times (\mathbb{N}_{[0:1]})^N,$$

and the LTF decision variables are $X_{\text{l}} := (\Sigma, T, R, Q_{\text{bi}}, B) \in V_{\text{l}}$, where

$$V_{\text{l}} := \mathbb{R}^{L-1} \times \mathbb{R}^{L-1} \times \mathbb{R}^{L-1} \times \mathbb{R}^{L-1+N} \times (\mathbb{N}_{[0:1]})^{L-1 \times M+2}.$$

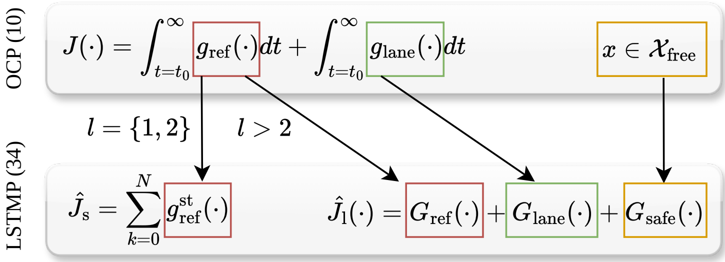
OCP (10)
$J(\cdot) = \int_{t=t_0}^{\infty} g_{\mathrm{ref}}(\cdot) dt + \int_{t=t_0}^{\infty} g_{\mathrm{lane}}(\cdot) dt$
$x \in \mathcal{X}_{\mathrm{free}}$
$l = \{1, 2\}$
$l > 2$
LSTMP (34)
$\hat{J}_{\mathrm{s}} = \sum_{k=0}^{N} g_{\mathrm{ref}}^{\mathrm{st}}(\cdot)$
$\hat{J}_{\mathrm{l}}(\cdot) = G_{\mathrm{ref}}(\cdot) + G_{\mathrm{lane}}(\cdot) + G_{\mathrm{safe}}(\cdot)$

Figure 6.14: Overview of the approximations of the general OCP objective function (6.29) by the LTF cost $\hat{J}_{\mathrm{l}}(\cdot)$ and the STF cost $\hat{J}_{\mathrm{s}}(\cdot)$. The reference cost $g_{\mathrm{ref}}(\cdot)$ is approximated for the first two lanes within the STF and, thereafter, by the LTF.

In total, $3(L-1)$ continuous and $(L-1)(M+2)$ binary variables are used to model the transitions for LM SVs. Another $(N-1)(n_x+n_u)+n_x$ continuous and N binary variables model the first lane change for a horizon of $t_{\mathrm{d}}N$. A total of $L+N-1$ variables are used as auxiliary variables.

Remarkably, the total number of binary variables is $N_{\mathrm{bin}} = (L-1)(M+2)+N$, which is with $O(LM+N)$ usually a much lower number in contrast to $O(LMN)$ of [213].

The cost function (6.28) is approximated by the cost of the STF trajectory

$$\hat{J}_{\mathrm{s}}(X_{\mathrm{s}}) = \sum_{k=0}^{N} g_{\mathrm{ref}}^{\mathrm{st}}(x_k, u_k, \lambda_k),$$

and the cost of the long horizon is

$$\hat{J}_{\mathrm{l}}(X_{\mathrm{l}}) = G_{\mathrm{lane}}(\Sigma, T, B) + G_{\mathrm{ref}}(T, \Sigma) + G_{\mathrm{safe}}(R).$$

The relations of the general OCP objective in (6.29) approximated by the LSTMP, comprising the LTF cost $\hat{J}_{\mathrm{l}}(\cdot)$ and the STF cost $\hat{J}_{\mathrm{s}}(\cdot)$ are shown in Fig. 6.14.

Including a constraint $x_0 = \hat{x}$ that constrains the decision variable x_0 to the current state $\hat{x}$, the constraints of the STF are summarized by $g_{\mathrm{s}}(X_{\mathrm{s}}) \leq 0$ and include the discrete dynamic model (6.36), the control and state constraints (6.37) and the constraints related to the first lane-change (6.39), (6.40), (6.41) and (6.42).

For the LTF, a constraint $g_{\mathrm{l}}(X_{\mathrm{l}}) \leq 0$ summarizes the reachability constraints (6.43), the CC constraints (6.44a), and the constraints used to formulate the disjunction among gaps in (6.45), (6.46), (6.47), and (6.48).

The coupling constraints (6.52) and (6.53) between states of the LTF and STF are concisely written as $g_{\mathrm{c}}(X_{\mathrm{s}}, X_{\mathrm{l}}) \leq 0$.

Ultimately, the LSTMP approximates the solution of the OCP (6.29) by solving the following MIQP in each iteration

$$\min_{\substack{X_{\mathrm{s}} \in V_{\mathrm{s}},\\ X_{\mathrm{l}} \in V_{\mathrm{l}}}} \hat{J}_{\mathrm{s}}(X_{\mathrm{s}}) + \hat{J}_{\mathrm{l}}(X_{\mathrm{l}}) \quad \text{s.t.} \quad \begin{cases} g_{\mathrm{s}}(X_{\mathrm{s}}) \leq 0, \\ g_{\mathrm{l}}(X_{\mathrm{l}}) \leq 0, \\ g_{\mathrm{c}}(X_{\mathrm{s}}, X_{\mathrm{l}}) \leq 0. \end{cases} \tag{6.54}$$

The output $X^* = (X_{\mathrm{s}}, X_{\mathrm{l}})$ of the LSTMP is always safe w.r.t. the obstacle constraints (6.33) and (6.35). This follows directly from the constraint formulations of the STF, including the terminal safe set (6.42). Approximation errors in the LTF formulations may lead to sub-optimal behavior. However, they do not influence safety related to the feasibility of the trajectory X_{s}^*.

6.2.7 Evaluation

We evaluate the proposed LSTMP approach in two different setups. First, deterministic SVs are simulated as they are modeled in the LSTMP, and exact tracking of the provided plan is assumed. A second setup includes more realistic scenarios, where the traffic is simulated interactively by the traffic simulator `SUMO` [170], based on benchmark scenarios provided by the `CommonRoad`-framework [13], cf., Fig. 6.15. Moreover, the LSTMP is integrated into an autonomous driving (AD)-stack with a low-level NMPC tracking controller of [228] that tracks the LSTMP trajectory X^* by controlling a simulated single-track *BMW 320i* medium-sized passenger car model provided by `CommonRoad`. The SV states $\hat{X}^{\mathrm{SV}}$ and the current estimated point-mass state $\hat{x}$ are the inputs of the planner. The point-mass state $\hat{x}$ is obtained from the six-dimensional simulated single-track vehicle state $\hat{z}$.

For both setups, the LSTMP is compared against the MIP-DM of [213] and a hybrid A* formulation according to [8]. Rendered simulations can be found on the website `https://rudolfreiter.github.io/lstmp_vis/`

Implementation Details

We describe the setup used for evaluation in the following. Parameters are chosen according to Tab. 6.4.

Preprocessing. The SVs states X^{SV} are processed before either planner is executed. First, SVs that drive closer to each other than a longitudinal threshold

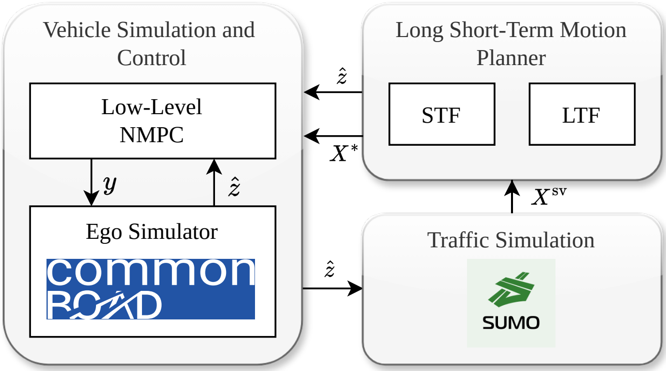

Figure 6.15: Overview of the adopted simulation architecture. After obtaining the ego vehicle state $\hat{z}$ and the SVs states X^{sv}, the LSTMP solves in each planning iteration the MIQP (6.54). The computed plan X^* related to the point-mass model is forwarded to the low-level NMPC for tracking. For the ego vehicle simulation, a *BMW 320i* vehicle model provided by `CommonRoad` is used. The position of the ego vehicle $\hat{z}$ is passed to the `SUMO` traffic simulator.

Table 6.4: Parameter for evaluations.

General parameters	
t_d, M	300ms, 7
$w_n, w_v, w_g, w_\mathrm{safe}$	$10^{-2}, 10^{-1}, 200, 10^{-5}$
R	$\mathrm{diag}\big([5 \cdot 10^{-4},\ 2 \cdot 10^{-3}]\big)$
d_lane	3.75m (Germany), 12 feet (US)
$[\underline{a}_\mathrm{lon}, \underline{a}_\mathrm{lat}]$	$[-8,\ -3]\frac{\mathrm{m}}{\mathrm{s}^2}$
$[\overline{a}_\mathrm{lon}, \overline{a}_\mathrm{lat}]$	$[5,\ 3]\frac{\mathrm{m}}{\mathrm{s}^2}$
LSTMP - deterministic scenario	
t_d, N, M	300ms, 15, 7
t_f, $\bar{t}_\mathrm{lc}$	10^5s , 2.7s
$\overline{v}^{\mathrm{sv}}, \underline{v}^{\mathrm{sv}}$	$\hat{v}^{\mathrm{sv}}, \hat{v}^{\mathrm{sv}}$
LSTMP-V0 - interactive scenario	
$\overline{v}^{\mathrm{sv}}$, $\underline{v}^{\mathrm{sv}}$, L	$\hat{v}^{\mathrm{sv}} + 1\frac{\mathrm{m}}{\mathrm{s}}$, $\hat{v}^{\mathrm{sv}} - 1\frac{\mathrm{m}}{\mathrm{s}}$, 5
LSTMP-V1 - interactive scenario	
$\overline{v}^{\mathrm{sv}}$, $\underline{v}^{\mathrm{sv}}$, L	$\hat{v}^{\mathrm{sv}} + 1\frac{\mathrm{m}}{\mathrm{s}}$, $\hat{v}^{\mathrm{sv}} - 1\frac{\mathrm{m}}{\mathrm{s}}$, 3
LSTMP-V2 - interactive scenario	
$\overline{v}^{\mathrm{sv}}$, $\underline{v}^{\mathrm{sv}}$, L	$\hat{v}^{\mathrm{sv}} + 3\frac{\mathrm{m}}{\mathrm{s}}$, $\hat{v}^{\mathrm{sv}} - 3\frac{\mathrm{m}}{\mathrm{s}}$, 5

distance of 15m are merged by setting the corresponding upper and lower velocity bounds and increasing the occupied space. Second, a maximum number of $M = 7$ SVs per lane are considered, which are the 7 closest vehicles at the current time step on each lane.

Benchmark MIP-DM. The first benchmark is based on the MIP-DM formulation of [213]. It uses a fixed discrete-time trajectory, similar to the STF, however, with four binary variables per obstacle and per time step to account for the rectangular obstacle shape. One further binary variable per time step indicates a lane change. The number of binary variables for the MIP-DM is therefore $N_{\text{bin}} = 4NML + N$. The MIP-DM is adapted to be comparable to the LSTMP. First, only one lane change direction is allowed, which reduces the number of binary variables. Second, obstacle shapes are inflated to occupy the whole lane, equally to the LSTMP. Finally, the interactive braking behavior of succeeding SVs on the same lane is implemented by deactivating corresponding obstacles on the current lane at the current time step. For the MIP-DM, a total number of $M = 3$ vehicles are considered on $L = 5$ consecutive lanes, while the horizon length N is $10, 15$ or 20 steps.

Benchmark hybrid A*. The second benchmark is based on the hybrid A* of [8]. This planner considers lateral motion only at the discrete lane indices, with a search space of (t, s, l). In order to be comparable to the other planners, we modify the search space to (s, l, v_s), which includes the velocity v_s instead of time. Since the hybrid A* of [8] does not consider lateral states between lane centers, we use a sampling time of $7t_{\text{d}}$ to allow full lane changes in one expansion, i.e., it is guaranteed that the final planning vertex is always located on the center of a lane. We use the same planner model (6.36) for vertex expansions. Note that hybrid A* could use nonlinear models without increasing the computation time, which, in contrast, would be challenging for the LSTMP and MIP-DM. As an admissible heuristic, the relaxed solution of (6.54) without obstacle constraints is computed for each lane. The longitudinal acceleration control is discretized into 11 intervals, and the lateral acceleration is computed by using 11 lane change primitives. The lateral states correspond to the number of lanes. The longitudinal position is discretized with 100 intervals and the velocity with 20 intervals. The number of node expansions is varied in experiments between 5 and 500.

Low-level NMPC. The low-level NMPC is formulated as shown in [228], using a nonlinear single-track vehicle model, a sampling time of 10ms and a horizon of 1.5s. The controls comprise the acceleration a and the steering rate $\dot{\delta}$.

Table 6.5: Different scenario settings for SVs used in evaluations.

Scenario Name	tr.-flow	tr.-density	L	V^{sv}	$\tilde{v}$
	$\frac{\text{SVs}}{\text{lane}\cdot\text{min}}$	$\frac{\text{SVs}}{\text{lane}\cdot\text{km}}$		$\frac{\text{m}}{\text{s}}$	$\frac{\text{m}}{\text{s}}$
Deterministic					
custom	14.6	12.2	9	$[15, 35]$	25
Closed-loop interactive					
USA_US101-22_1_I-1	11.8	14.0	6	$[0, 22.2]$	15
DEU_Col.-63_5_I-1	22.3	26.9	3	$[11.0, 16.5]$	11
DEU_Col.-63_5_I-1_s	8.9	10.5	3	$[11.1, 16.5]$	15

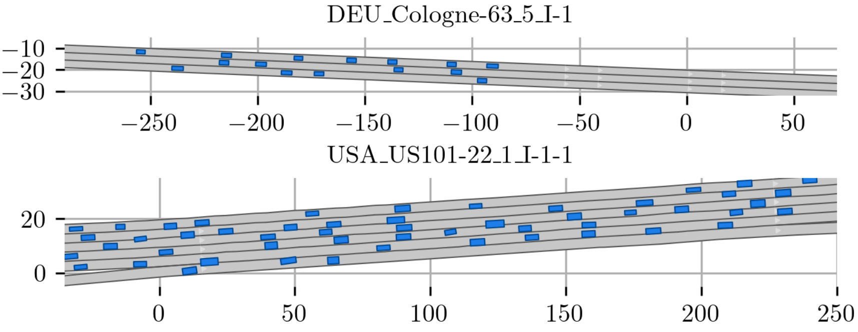

Figure 6.16: Different tracks from the `CommonRoad` scenario database used for the closed-loop simulation.

Scenarios. For deterministic comparisons in Sect. 6.2.7 and interactive closed-loop comparisons in Sect. 6.2.7, the scenarios are chosen according to Tab. 6.5. Due to traffic congestion, the velocity can be zero. Traffic flow and density are averaged over the simulation. The velocity range V^{sv} corresponds to the measured SVs velocities during all simulations. The scenarios are simulated for 40 seconds or until the end of the road is reached. Snapshots of the `CommonRoad` scenarios for interactive simulations are shown in Fig. 6.16.

Computations and Numerical Solvers. The MIQPs of the LSTMP and the MIP-DM are solved with `Gurobi` [114]. The NLP, arising in the low-level NMPC, is solved by the open-source solver `acados` [291]. Simulations are executed on a `LENOVO ThinkPad L15 Gen 1` Laptop with an `Intel(R) Core(TM) i7-10510U @ 1.80GHz` CPU.

Evaluation for Deterministic Traffic and Exact Tracking

In order to compare the performance of the planner without interference from the traffic prediction error, simulation modeling error, and controller performance, the planner is simulated with deterministic SVs and exact tracking. Deterministic SVs are simulated with constant speed if they are at a minimum distance to a slower leading SVs and with the speed of the leading SV if they are below the threshold distance. The planned trajectory X^* is assumed to be tracked exactly. This setup resembles the model of the traffic used in the LSTMP, where tight bounds for the obstacle-free sets (6.33) and (6.35) can easily be found. In Fig. 6.17, snapshots of a randomized simulation with five lanes are shown, where the vehicle starts at the bottom lane and has to reach the top lane. Red areas indicate the SVs after pre-processing. The STF trajectory X^* of the LSTMP is shown in black, whereas the transition gaps are shown in blue. The evaluated closed-loop cost and computation time of 100 randomized *custom* scenarios according to Tab. 6.5 for the deterministic setup are shown in Fig. 6.18 on the Pareto front.

The comparisons include evaluations for different parameter settings of the algorithms, i.e., the number of considered consecutive lanes in the LSTMP, the maximum node expansions in hybrid A*, and the horizon length of the MIP-DM.

The MIP-DM with the longest horizon outperforms the LSTMP in the average closed-loop cost over the full simulations, however, at a high computational expense which violates the real-time requirement. In fact, the computation time of the MIP-DM is an order of magnitude higher than the computation time of the LSTMP.

The hybrid A* can be faster to execute compared to the LSTMP, but it yields a higher closed-loop cost. In our experiments, increasing the iterations of hybrid A* could not yield a better performance. This may be due to the longer duration of motion primitives to allow lane changes and the resulting coarser time discretization. Further relevant properties related to the lane change multi-objective (6.28) are shown in Tab. 6.6. This includes the mean deviation from the reference speed $\Delta\tilde{v}$, mean and maximum values for the lateral and longitudinal accelerations, and the maximum reached lane $l_{\max}$ at the end of the simulation. It shows that the LSTMP with a longer horizon better keeps the reference speed and also changes lanes more often. By utilizing large acceleration values, the MIP-DM achieves the highest number of lane changes and the overall lowest closed-loop cost, cf., Fig. 6.18.

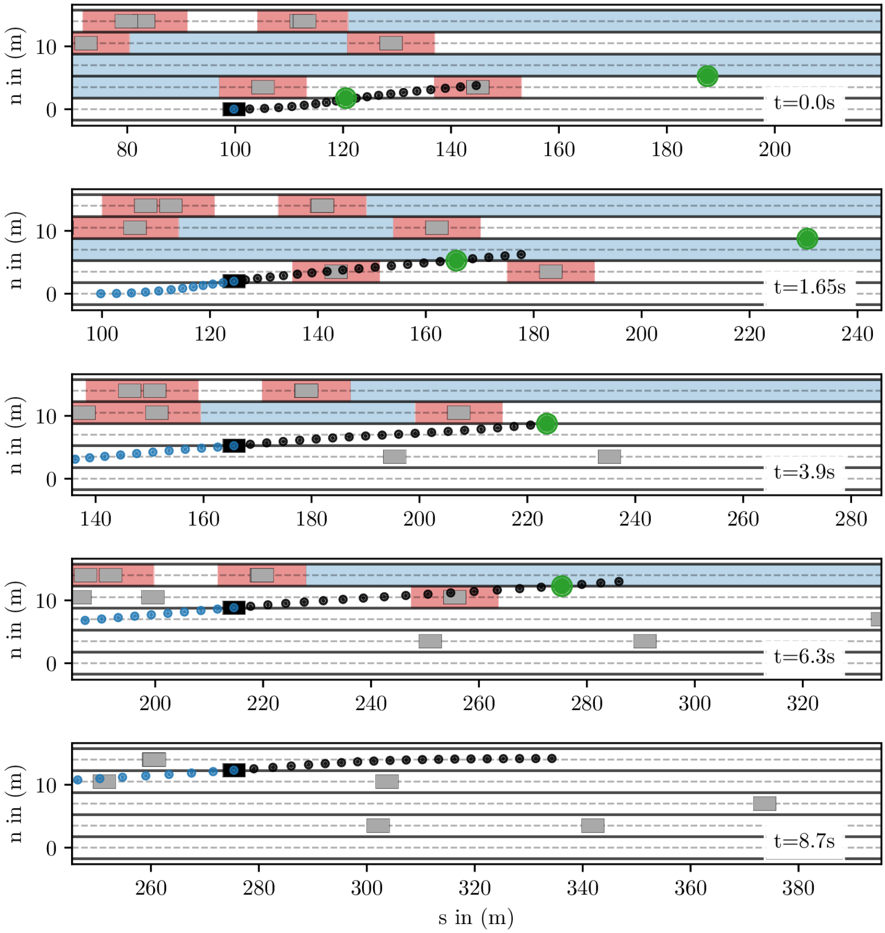

Figure 6.17: Snapshots during lane-changes on a five-lane deterministic environment with randomized SV (gray) initial speeds and lanes. Blue regions indicate computed gaps of the LSTMP, with green points corresponding to the expected transition position σ_l and the black STF trajectory X^*. Red areas correspond to occupied sets $\mathcal{O}(t_{\text{sim}})$, where t_{sim} is the current simulation time of the snapshot.

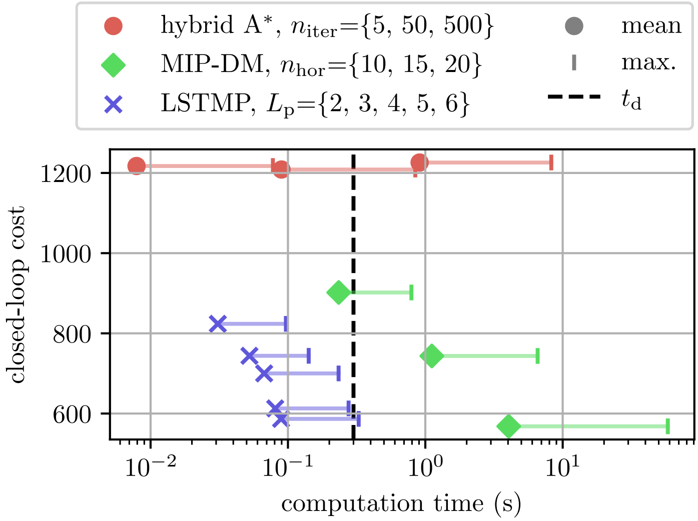

Figure 6.18: Pareto comparison of planners in randomized deterministic traffic scenarios. The hybrid A* planner can be parameterized to have the fastest computation time, and the MIP-DM achieves the lowest costs. However, the novel LSTMP formulation performs best when both a low computation time as well as low costs are required.

Notably, the number of binary variables required in the MIP-DM is much larger than in the LSTMP, which leads to a significantly longer computation time. For a prediction horizon of $N = 10$ and settings of Tab. 6.4, the MIP-DM requires 610 binary variables and for a prediction horizon of $N = 20$ a total of 1220 binary variables. The LSTMP that considers in total $L_{\text{p}} = 2$ lanes, requires only 22 binary variables, whereas considering $L_{\text{p}} = 5$ lanes requires only 50 binary variables.

Evaluation for Interactive Traffic and Closed-Loop Control

For different randomized scenarios according to Tab. 6.5 and Fig 6.16, the lane changing problem is simulated with interactive SVs, using a software architecture corresponding to Fig. 6.15. The ego vehicle starts at random free positions and has to reach the leftmost lane, according to cost (6.28), with parameters of Tab. 6.4. The LSTMP, hybrid A*, and MIP-DM are compared with a low-level tracking controller in closed-loop simulations. States are assumed to be estimated exactly. However, the velocity range of SVs is unknown. Different

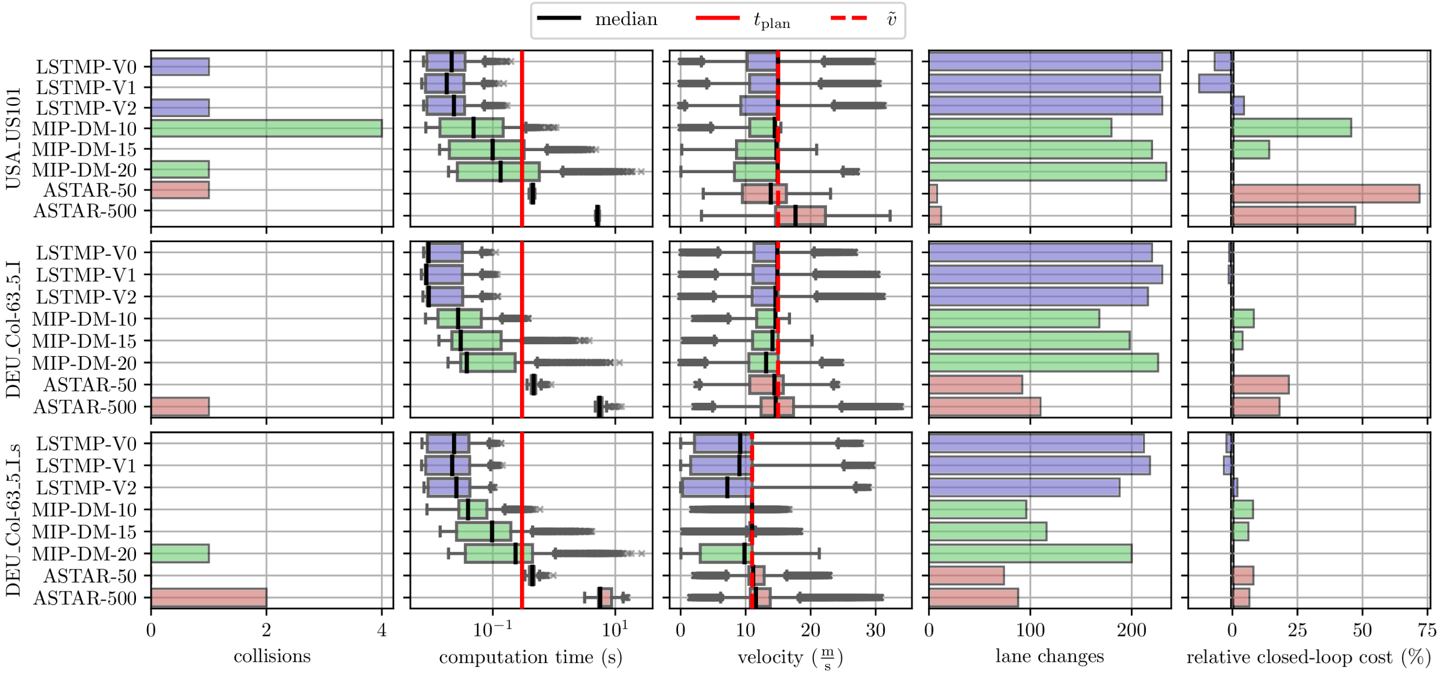

Figure 6.19: Closed-loop evaluation of variants of the proposed LSTMP planner (blue) compared to hybrid A* (red) and MIP-DM (green). The total number of collisions, the computation time, velocities, the total number of lane changes, and the closed-loop cost compared to the most exact MIP-DM formulation are compared for different randomized scenarios. The proposed LSTMP has the lowest computation time below the planning time t_{plan} and a high number of lane changes for all scenarios. Moreover, it can reduce the closed-loop cost significantly. Occasional collisions are observed for all variants, irrespective of their parameters.

Table 6.6: Comparison of planners in randomized deterministic traffic scenarios for different mean (maximum) quantities.

Planner			$\Delta\tilde{v}$	a_{lat}	a_{lon}	l_{max}
Par.	Val.	N_{bin}	$\frac{\text{m}}{\text{s}}$	$10^{-1}\,\frac{\text{m}}{\text{s}^2}$	$10^{-1}\,\frac{\text{m}}{\text{s}^2}$	
	LSTMP					
L_{p}	2	22	1.75	0.71 (4.58)	1.40 (5.30)	3.92
	3	29	0.42	0.63 (3.82)	1.46 (6.35)	4.20
	4	36	0.03	0.66 (3.70)	1.48 (6.45)	4.34
	5	43	0.07	0.64 (3.62)	1.37 (4.96)	4.62
	6	50	0.03	0.67 (3.62)	1.55 (6.15)	4.66
	MIP-DM					
N	10	610	2.50	0.48 (2.68)	0.69 (5.14)	3.62
	15	915	2.10	0.52 (2.67)	1.06 (8.16)	4.17
	20	1220	1.98	0.65 (2.67)	1.83 (8.70)	4.75
	hybrid A*					
iter.	5	N/A	1.78	0.47 (2.40)	0.14 (1.00)	2.61
	50	N/A	1.73	0.47 (2.40)	0.17 (1.14)	2.63
	500	N/A	1.70	0.43 (2.40)	0.22 (1.43)	2.58

settings of the planners are used according to Tab. 6.4 to create the statistical evaluation of performance measures as shown in Fig. 6.19.

The performance evaluations show rare collisions of all planners due to prediction errors. For the conservative LSTMP-V1 configuration, no collisions were recorded. Computation times are lowest for the LSTMP planner and well below the planning time threshold t_{plan}. The computation times for the hybrid A* are nearly constant since the planning nodes are expanded with a fixed number of iterations. Notably, the computations for hybrid A* were not performed on a runtime-optimized code. The velocity varies the most for the LSTMP, which promotes acceleration and deceleration to reach certain gaps. This can also be verified by the high number of lane transitions of the LSTMP. Particularly, in the DEU_Col.-63_5_I-1_s scenario, long-term decisions significantly raised the number of lane transitions in the LSTMP. The closed-loop cost for LSTMP configurations are below the benchmark comparisons, particularly below the MIP-DM-20 with the longest horizon of 20 steps, which we define as *expert*.

Costs are relatively expressed to the cost of MIP-DM-20 and outperformed by LSTMP-V0 and LSTMP-V1.

6.2.8 Conclusion and Discussion

Under the variety of different planning methods for AD, the proposed LSTMP for lane change planning achieves a good trade-off between performance and computational costs, thanks to the use of state-of-the-art MIQP solvers. The considered problem has relevant combinatorial and continuous parts, which makes MIQP formulations particularly suited to solve the proposed motion planning problem. Building on previous work to minimize the number of combinatorial variables, we introduced a novel long-horizon approximation. Together with a discrete-time trajectory, a single MIQP, which is computationally very efficient, was formulated consistently.

We compared our approach to the MIP-DM [213], which uses more integer variables to model rectangular obstacle shapes that are not required to be aligned with the lane boundaries and to model lane transitions in both directions. This makes the MIP-DM a more versatile approach, i.e., lane changing is only a subset of problems that can be addressed with it.

The fundamental modeling approach of the LSTMP is the decomposition into convex cells together with a simplification due to the road alignment. The authors assume that it is possible to add integer variables to achieve lane transitioning in both directions for a fixed maximum number of transitions and additional convex decompositions to resemble nonconvexities in the ST-space, as, for instance, traffic lights.

In the future, we will evaluate whether more flexible mixed-integer nonlinear programming can achieve better performance under real-time requirements.

6.3 Equivariant Deep Learning of Mixed-Integer Optimal Control Solutions for Vehicle Decision Making and Motion Planning

---

In this section, the paper published in [229] is reprinted verbatim. Note that the formatting of some formulas, terms, and numbers has been slightly adjusted for consistency without changing their meaning or content.

The contributions of each author are listed in the following.

Rudolf Reiter:	*idea for main contribution (network architecture, feasibility projector), programming and simulation validation, proposing mathematical formulations, algorithm design, creation of the document, creation of the rebuttal and the revised version*
Rien Quirynen:	*idea (preliminaries, binary variable prediction), proposing mathematical formulations, algorithm design, mathematical corrections, supporting document creation, grammar corrections, spelling style improvements, coherence review and improvement, simplification suggestions, rebuttal proof reading*
Moritz Diehl:	*mathematical corrections, algorithmic corrections for feasablility projection, grammar corrections, spelling style improvements, coherence review and improvement, simplification suggestions*
Stefano Di Cairano:	*idea (preliminaries, binary variable prediction), algorithmic corrections, mathematical corrections, grammar corrections, spelling style improvements, coherence review and improvement, simplification suggestions, rebuttal proof reading and answer suggestions*

Copyright ©2024 IEEE. Reprinted, with permission from Rien Quirynen, Moritz Diehl and Stefano Di Cairano. A Long-Short-Term Mixed-Integer Formulation for Highway Lane Change Planning. June/2024. DOI: 10.1109/TCST.2024.3400571.

Abstract. Mixed-integer quadratic programs (MIQPs) are a versatile way of formulating vehicle decision making and motion planning problems, where the prediction model is a hybrid dynamical system that involves both discrete and continuous decision variables. However, even the most advanced MIQP solvers can hardly account for the challenging requirements of automotive embedded platforms. Thus, we use machine learning to simplify and hence

speed up optimization. Our work builds on recent ideas for solving MIQPs in real-time by training a neural network to predict the optimal values of integer variables and solving the remaining problem by online quadratic programming. Specifically, we propose a recurrent permutation equivariant deep set that is particularly suited for imitating MIQPs that involve many obstacles, which is often the major source of computational burden in motion planning problems. Our framework also comprises a feasibility projector that corrects infeasible predictions of integer variables and considerably increases the likelihood of computing a collision-free trajectory. We evaluate the performance, safety, and real-time feasibility of decision-making for autonomous driving using the proposed approach on realistic multi-lane traffic scenarios with interactive agents in `SUMO` simulations.

6.3.1 Introduction

Decision-making and motion planning for automated driving are challenging due to several reasons [199]. First, in general, even formulations of deterministic, two-dimensional motion planning problems are PSPACE-hard [219, 159]. Second, (semi-)autonomous vehicles operate in highly dynamic environments, thus requiring a relatively high control update rate. Finally, there is always uncertainty that stems from model mismatch, inaccurate measurements as well as other drivers' unknown intentions. The complexity of motion planning and decision making (DM) for automated driving and its real-time requirements in resource-limited automotive platforms [78] requires the implementation of a multi-layer guidance and control architecture [199, 112].

Based on a route given by a navigation system, a decision-making module decides what maneuvers to perform, such as lane changing, stopping, waiting, and intersection crossing. Given the outcome of such decisions, a motion planning system generates a trajectory to execute the maneuvers, and a vehicle control system computes the input signals to track it. Recent work [213] presented a MIP-DM, which simultaneously performs maneuver selection and trajectory generation by solving a MIQP at each time instant. In this paper, we present an algorithm to implement MIP-DM based on supervised learning and sequential quadratic programming (SQP) to compute a collision-free and close-to-optimal solution with a considerably reduced online computation time compared to advanced MIQP solvers.

The presented approach consists of an *offline* supervised learning procedure and an *online* evaluation step that includes a feasibility projector (FP). In the *offline* procedure, expert data is collected by computing the exact solutions of the MIQP for a large number of samples from a distribution of parameter values in the MIP-DM, including, for example, states of the autonomous vehicle, its surrounding traffic environment, and speed limits. Along the paradigm of [63, 39], a NN is trained with the collected expert data to predict the binary variables that occur within the MIQP, which are the main source of the computational complexity of MIQPs. A novel NN architecture, referred to as recurrent equivariant deep set (REDS), is proposed that exploits key structural domain properties, such as permutation equivariance related to obstacles and recurrence of the time series.

In the *online* evaluation step, the NN predicts the optimal values of the binary variables in the MIQP. After fixing these, the resulting problem becomes a convex QP that can be solved efficiently. To account for potentially wrong predictions, the QP is formulated with slack variables (soft-QP). The soft-QP solution is forwarded to a FP to correct any infeasibilities, implemented by a NLP with smooth but concave obstacle constraints. Such convex-concave NLPs can be solved efficiently using an SQP algorithm [291, 284]. To further increase the likelihood of finding a feasible and possibly optimal solution, an ensemble of NNs is trained and evaluated, and the "best" solution is selected at each time step.

The overall performance of the proposed method is compared against the advanced MIQP solver `Gurobi` [114], alternative neural network architectures [62] and evaluated in high fidelity closed-loop simulations using `SUMO` [170] and `CommonRoad` [13].

Related Work

This work lies at the intersection of three research areas, i.e., geometric deep learning, mixed-integer program (MIP), and motion planning for AD (cf., Fig. 6.20). Motion planning problems are solved by algorithms that can handle combinatorial complexity and dynamic feasibility under real-time computation limits [67, 218].

Motion primitives are appealing due to their simplicity and alignment with the road geometry [257]. However, the motion plans are usually sub-optimal or conservative.

In several works, graph search is performed on a discretized state space [199]. Probabilistic road maps [139] and rapidly exploring random trees [23] are common in highway motion planning. Nonetheless, they suffer from the curse of dimensionality, a poor connectivity graph, and no repeatability [67]. By using heuristics, A* [8] aims to avoid the problem of high-dimensional discretization. However, choosing an appropriate heuristic is challenging, and graph generation in each iteration is time-consuming [199].

For some conservative simplifications related to highway driving, the state space can be decomposed into spatio-temporal driving corridors [30, 181, 164]. Such non-convex regions can be further decomposed into convex cells [77] or used in

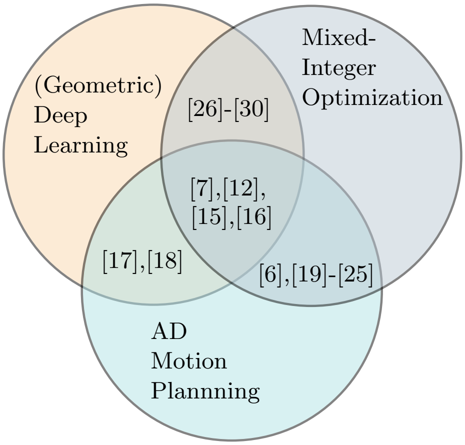

Figure 6.20: Categorical overview of related work. The research domain of the proposed approach is located at the intersection of three areas.

a sampling-based planner [164]. For these cases, the performance is limited due to the required over-approximations.

Derivative-based numerical optimization methods can successfully solve problems in high-dimensional state spaces in real-time [80]. However, these approaches are often restricted to convex problem structures or sufficiently good initial guesses. While highway motion planning and DM is highly non-convex, by introducing integer variables, the problem can be formulated as an MIQP [201, 209, 91, 140, 87, 164, 141, 213].

A common problem is the significant computation time of MIQP solvers. Authors mitigate the problem by either scaling down the problem size [209, 225, 87, 213], neglecting real-time requirements [91] or accepting the high computation time for non-real-time simulations while leaving real-time feasibility open for future work [201, 140, 141].

Several recent works use deep learning to accelerate MIQP solutions. One strategy is to improve algorithmic components within an online MIQP solver [175, 189, 144]. The authors in [175] use deep learning to warm-start MIQP solvers, which, however, cannot lower the worst-case computation time. The authors in [189, 144] propose a learning strategy to guide a branch-and-bound (BnB) algorithm. For solving MILPs, the authors in [242] use NNs for a custom solver to achieve similar computation times as commercial solvers, which, however, may still be large. Another strategy is based on supervised or imitation learning, using MIQP solutions as expert data to train function approximators offline [40, 309, 267]. The authors in [40] show how the classification of binary variables in MIQPs can produce high-quality solutions with a low computation time. Recent work [62] extends supervised learning for MIQP-based motion planning [63] using a recurrent neural network (RNN) and presolve techniques to increase the likelihood of predicting a feasible solution. Due to the RNN, the results in [62] scale well with the horizon length but not yet with the number of obstacles, as shown later.

None of the works that use deep learning to accelerate MIQP-based motion planning consider or leverage geometric deep learning, particularly permutation equivariance or invariance, to decrease the network size and increase the performance. As one consequence, they do not allow variable input and variable output dimensions, e.g., corresponding to a variable number of obstacles. However, some recent works successfully use geometric deep learning for related tasks in AD, e.g., the prediction of other road participants [324, 131], or within RL [126].

Other techniques for DM exclusively use NNs without solving optimization problems online, e.g., using deep RL [276]. These approaches usually require more data, are less interpretable, and may lack safety without additional safety layers due to the approximate nature of the NN output [67].

Contributions

Our main contribution is a REDS for the NN predicting integer variables of MIQPs, which is particularly suited for learning time-series and obstacle-related binary variables in motion planning problems. In particular, the REDS enables the NN to predict binary variables for collision avoidance concerning a varying number of obstacles, and the predicted solution is the same regardless of the order in which the obstacles are provided. Our proposed framework includes an ensemble of NNs in combination with a feasibility projector (FP) that increases the likelihood of computing a collision-free trajectory. Compared to the state-of-the-art, we show that our method improves the prediction accuracy, adds permutation equivariance, allows for a variable number of obstacles and horizon length, and generalizes well to unseen data, such as several obstacles not present in the training data. As a final contribution, we demonstrate the performance of a novel integrated planning system, which is further referred to as REDS planner. The REDS planner uses an ensemble of REDSs, a selection of the best soft-QP solution, and the FP for real-time vehicle decision-making and motion planning. We present closed-loop simulation results with reference tracking by a NMPC for realistic traffic scenarios, using interactive agents in `SUMO` [170], and a high-fidelity vehicle model and the problem setup provided by `CommonRoad` [13].

Preliminaries and Notation

The notation $I(n) = \{z \in \mathbb{N} | 0 \leq z \leq n\}$ is used for index sets and $\mathbb{B} = \{0, 1\}$. Throughout this paper, the attributes of *equivariance (invariance)* exclusively refer to *permutation equivariance (invariance)* with the following definition.

Definition 6.3.1. *Let $f(\zeta^{\text{us}}) : \mathbb{X}^M \to \mathbb{Y}$ be a function on a set of variables $\zeta^{\text{us}} = \{\zeta_1^{\text{us}}, \ldots, \zeta_M^{\text{us}}\} \in \mathbb{X}^M$ and let $\mathcal{G}$ be the permutation group on $\{1, \ldots, M\}$. The function f is* ***permutation invariant****, if $f(g \cdot \zeta^{\text{us}}) = f(\zeta^{\text{us}})$ for all $g \in \mathcal{G}, \zeta^{\text{us}} \in \mathbb{X}^M$.*

Definition 6.3.2. *Let $f(\zeta^{\text{eq}}) : \mathbb{X}^N \to \mathbb{Y}^N$ be a function on a set of variables $\zeta^{\text{eq}} = \{\zeta_1^{\text{eq}}, \ldots, \zeta_N^{\text{eq}}\} \in \mathbb{X}^N$ and let $\mathcal{G}$ be the permutation group on $\{1, \ldots, N\}$. The function f is* ***permutation equivariant****, if $f(g \cdot \zeta^{\text{eq}}) = g \cdot f(\zeta^{\text{eq}})$ for all $g \in \mathcal{G}, \zeta^{\text{eq}} \in \mathbb{X}^N$.*

Features that are modeled without structural symmetries are referred to as *unstructured features.* Lower and upper bounds on decision variables x are denoted by $\underline{x}$ and $\overline{x}$, respectively. The all-one vector $[1, \ldots, 1]^\top$ of size n is $\mathbf{1}_n^\top$. The notation $f(x; y)$ in the context of optimization problems indicates that function $f(\cdot)$ depends on decision variables x and parameters y. Lower case letters $x \in \mathbb{R}^n$ refer to scalars or vectors of size n and their upper case version $X \in \mathbb{R}^{n \times N}$ refer to the matrix associated with a sequence of those vectors along a time horizon N. *Obstacles* normally refer to surrounding vehicles.

6.3.2 Problem Setup and Formulation

In this work, an autonomous vehicle shall drive *safely* along a multi-lane road while obeying the traffic rules. We consider decision-making based on MIP that determines the driving action and a reference trajectory for the vehicle control to follow. The definition of *safety* can be ambiguous [55]. We use the term *safe* to refer to satisfying hard collision avoidance constraints concerning known obstacles' trajectories. Our problem setup relies on the following assumptions.

Assumption 6.3.1. *There exists a prediction time window along which the following are known:*

1. *the predicted position and orientation for each of the obstacles in a neighborhood of the ego vehicle up to a sufficient precision,*
2. *the high-definition map information, including center lines, road curvature, lane widths, and speed limits.*

Assumption 6.3.1 requires the vehicle to be equipped with on-board sensors and perception systems [54], an obstacle prediction module [199, 112], the high definition map database, either on-board or obtained through vehicle-to-infrastructure (V2I) communication [89]. The effect of prediction inaccuracies and uncertainty on vehicle safety is outside the scope of the present paper, but relevant work can be found in [199, 112] and references therein.

Assumption 6.3.2. *We assume a localization with cm-level accuracy at each sampling time.*

Assumption 6.3.2 is possible thanks to recent advances in GNSS that achieve cm-level accuracy at a limited cost [34, 108].

We propose a DM that satisfies the following requirements.

Requirement 6.3.1. *The DM must plan a sequence of maneuvers, possibly including one or multiple lane changes and a corresponding collision-free trajectory to make progress along the vehicle's future route with a desired nominal velocity.*

Requirement 6.3.2. *The DM yields kinematically feasible trajectories, satisfying vehicle limitations and traffic rules (e.g., speed limits).*

Requirement 6.3.3. *The DM must be agnostic to permutations of surrounding vehicles, i.e., the plan must be consistent, irrespective of the order in which the vehicles are processed (cf. Def. 6.3.2).*

Requirement 6.3.4. *The DM must hold a worst-case computation time lower than the real-time planning period* t_p*, with a target value of* $t_\mathrm{p} \leq 0.2\mathrm{s}$*.*

Requirement 6.3.5. *At any time, the DM must satisfy all requirements, regardless of the number of surrounding vehicles.*

Based on Assumption 6.3.1, Req. 6.3.1-6.3.3 can be met by the MIP-DM from [213]. However, the crucial Req. 6.3.4-6.3.5 may be difficult to meet by the MIP-DM when executing on embedded control hardware with limited computational resources and memory [78]. The main focus of this work is approximating the MIP-DM with a suitable framework that satisfies all Req. 6.3.1-6.3.5. Notably, Req. 6.3.3 is trivially true for the MIP-DM [213]. However, this is not the case for NN-based planners unless the architecture is invariant to permutations [312].

Nominal and Learning-based Architecture

A hierarchical control architecture (cf., Fig 6.21) of a DM is proposed, followed by a reference tracking NMPC. Within the architecture of Fig. 6.21, the expert MIP-DM is used as a benchmark. We refer to the module aiming to imitate the expert MIP-DM as the REDS planner. It comprises an ensemble of NNs that predict the binary variables of the MIQP, as used in the expert MIP-DM. For each NN, a soft-QP is solved with the binary variables fixed to the predictions and softened obstacle avoidance constraints. This yields a set of candidate trajectories, and a selector evaluates their costs and picks the least sub-optimal one. Finally, the FP projects the solution guess to the feasible set to ensure that the constraints are satisfied. The FP solves an NLP, minimizing the error with respect to the best candidate solution and subject to ellipsoidal collision avoidance constraints. The reference provided by the REDS planner or the expert MIP-DM is tracked by the reference tracking NMPC at a high sampling rate, which includes collision avoidance constraints. This adds additional safety and leverages the requirements on the REDS planner on conservative constraints related to uncertain predictions.

The individual modules differ in motion prediction models, their objective, obstacle avoidance constraints, the approximate computation time t_{comp}, the final horizon t_{f} and the discretization time t_{Δ}, cf., Tab. 6.7.

6.3.3 Expert Motion Planner

We model the vehicle state in a curvilinear coordinate frame, which is defined by the curvature $\kappa(s)$ along the reference path [222]. The state vector $x = [s, n, v_s, v_n]^\top \in \mathbb{R}^{n_{\mathrm{x}}}$ includes the position $p = [s, n]^\top \in \mathbb{R}^2$, with the longitudinal position s, the lateral position n, the longitudinal velocity v_s and the lateral velocity v_n, where $n_{\mathrm{x}} = 4$. The control vector $u = [a_{\mathrm{s}}, a_{\mathrm{n}}]^\top \in \mathbb{R}^{n_u}$ comprises the longitudinal and lateral acceleration in the curvilinear coordinate frame, where $n_u = 2$. The discrete-time double integrator dynamics are written as $x_{i+1} = Ax_i + Bu_i$, using the discretization time t_{d}, where $A \in \mathbb{R}^{n_{\mathrm{x}} \times n_{\mathrm{x}}}$ and $B \in \mathbb{R}^{n_{\mathrm{x}} \times n_u}$. By considering N prediction steps, the prediction horizon can be computed by $t_{\mathrm{f}} = Nt_{\mathrm{d}}$. Constraints $v_n \leq \overline{\alpha}\, v_s$ and $v_n \geq \underline{\alpha}\, v_s$ account for limited lateral movement of the vehicle with bounds $\underline{\alpha}, \overline{\alpha}$ that can be computed based on a maximum steering angle $\overline{\delta}$ and the maximum absolute signed curvature $\overline{\kappa} = \kappa(s^*)$, with $s^* = \arg\max_{0 \leq s \leq \overline{s}} |\kappa(s)|$ along a lookahead distance $\overline{s}$ [213]. An acceleration limit $\overline{a}_{\mathrm{fric}}$ of the point-mass model formulated as box constraints $||u||_\infty \leq \overline{a}_{\mathrm{fric}}$ inner-approximate tire friction constraints related to Kamm's circle [13]. Since we formulate our model in the Frenet coordinate frame, the lateral acceleration bounds $\overline{a}_n, \underline{a}_n$ are modified, based on the centrifugal acceleration, resulting in $\overline{a}_n = \overline{a}_{\mathrm{fric}} + \overline{\kappa} v_{s,0}^2$ and $\underline{a}_n = -\overline{a}_{\mathrm{fric}} + \overline{\kappa} v_{s,0}^2$. The fixed parameter d_{bnd} is used as the safety distance to the road boundary, bounds $\overline{u}, \underline{u}$ for acceleration limits, and $\overline{v}_s, \overline{v}_n$ for the maximum velocity. In order to account for the road width, bounds $\underline{n} \leq n \leq \overline{n}$ on the lateral position are used. The model used within expert MIP-DM is able to approximate the dynamics in a variety of situations [213]. However, certain maneuvers, such as sharp turns, may require additional modeling concepts [140].

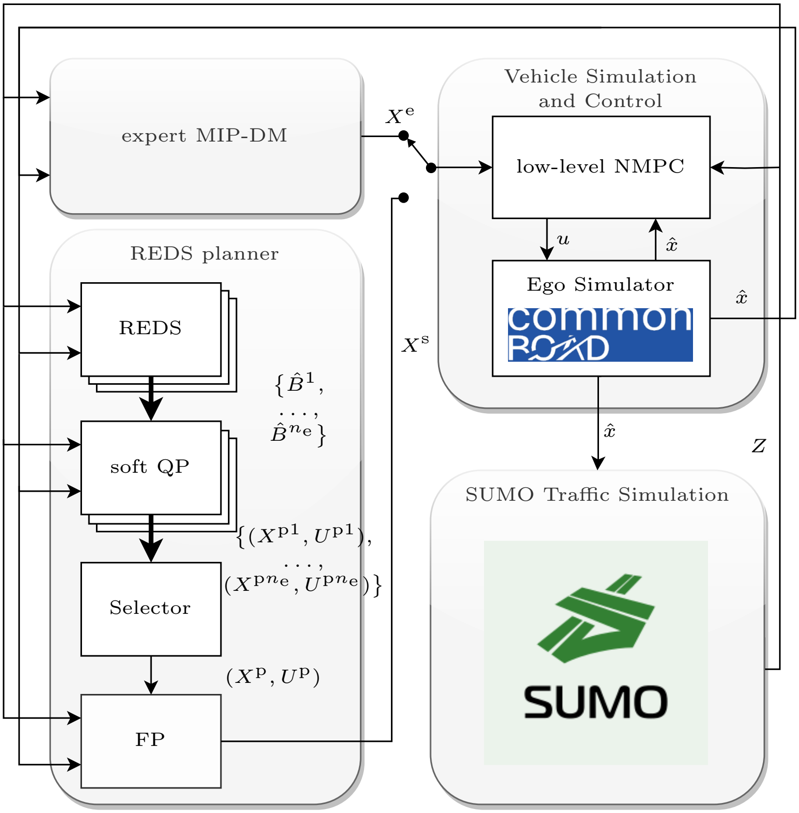

Figure 6.21: Planning and closed-loop simulation architecture. The expert MIP-DM is imitated by the REDS planner, which uses an ensemble of n_e NNs to predict values of the binary variables $\{\hat{B}^1, \dots, \hat{B}^{n_\mathrm{e}}\}$. A soft-QP solves a formulation of the expert MIP-DM with binary variables fixed to the prediction. The lowest cost solution X^{p} is chosen by a Selector and corrected by the FP. A reference tracking NMPC with obstacle avoidance tracks the corrected solution X^{s}.

Property / Module	expert MIP-DM	soft-QP	FP	tracking NMPC
Opt. Class	MIQP	QP	NLP	NLP
Model	point-mass	point-mass	point-mass	kinematic
Objective	global economic	global economic	tracking	tracking
Obstacle Avoidance	hyper-planes $\mathcal{O}^{\text{out}}$	fixed hyper-planes $\mathcal{O}^{\text{out}}$	ellipsoids $\mathcal{O}^{\text{safe}}$	ellipsoids $\mathcal{O}^{\text{safe}}$
t_{comp}	$\sim$ 2s	$\sim$ 3ms	$\sim$ 10ms	$\sim$ 2ms
t_{f}	10s	10s	10s	1s
t_{Δ}	0.2s	0.2s	0.2s	0.05s
Comment	high computation time	possibly unsafe	safe projection	

Table 6.7: Overview of optimization problems solved within the individual modules.

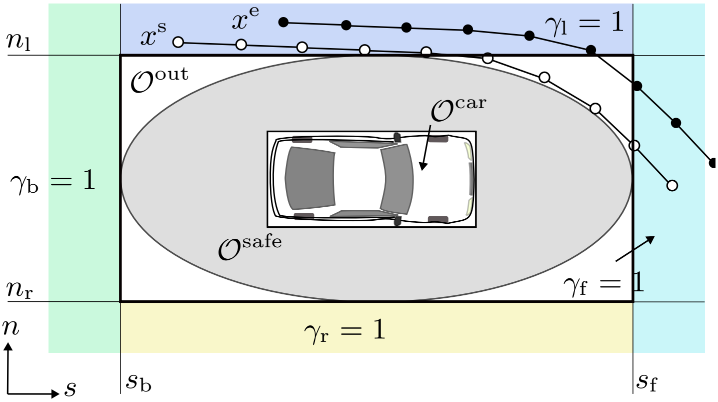
x^{e}
x^{s}
$\gamma_{\mathrm{l}} = 1$
n_{l}
$\mathcal{O}^{\mathrm{out}}$
$\mathcal{O}^{\mathrm{car}}$
$\gamma_{\mathrm{b}} = 1$
$\mathcal{O}^{\mathrm{safe}}$
$\gamma_{\mathrm{f}} = 1$
n_{r}
n
$\gamma_{\mathrm{r}} = 1$
s
s_{b}
s_{f}

Figure 6.22: Vehicle over-approximations. All trajectories outside of $\mathcal{O}^{\mathrm{safe}}$ are considered free of collision. The expert MIP-DM plans with the most conservative obstacle set $\mathcal{O}^{\mathrm{out}}$. The ellipsoidal smooth over-approximation $\mathcal{O}^{\mathrm{safe}}$ is used within the FP and the NMPC. The four colored regions are uniquely determined by the binary variables γ, where in each region, exactly one binary variable is equal to one.

Collision Avoidance Constraints

Collision avoidance constraints for n_{obs} obstacles are formulated by considering the ego vehicle as a point mass and using a selection matrix $P \in \mathbb{R}^{2\times n_x}$, with $p = Px$ that selects the position states p from the states x. The true occupied obstacle space $\mathcal{O}_j^{\mathrm{car}} \subseteq \mathbb{R}^2$, for $j \in I(n_{\mathrm{obs}}-1)$, is expanded to include all possible configurations where the ego and obstacle vehicle are in a collision in the curvilinear coordinate frame, resulting in an ellipsoid $\mathcal{O}_j^{\mathrm{safe}}$, cf. Fig. 6.22. The expert MIP-DM uses a road-frame-aligned rectangular constraint $Px_i \notin \mathcal{O}_j^{\mathrm{out}}$, for all $i \geq 0, j \in I(n_{\mathrm{obs}} - 1)$, which over-approximates the ellipsoidal set $\mathcal{O}_j^{\mathrm{safe}}$, leading to additional robustness of the multi-layer control architecture, with $\mathcal{O}_j^{\mathrm{car}} \subseteq \mathcal{O}_j^{\mathrm{safe}} \subseteq \mathcal{O}_j^{\mathrm{out}} \subseteq \mathbb{R}^2$ (cf., Fig. 6.21 and Tab. 6.7). For each obstacle j, the rectangular shape of $\mathcal{O}_j^{\mathrm{out}}$ can be characterized by the boundaries $d^\top = [s_{\mathrm{f}}, s_{\mathrm{b}}, n_{\mathrm{l}}, n_{\mathrm{r}}]^\top$. Four collision-free regions $k \in \{\mathrm{f}, \mathrm{b}, \mathrm{l}, \mathrm{r}\}$ around an obstacle are defined, where each region can be expressed by a convex set $A^k(d)\,p \leq b^k(d)$, with either one for $k \in \{f, b\}$ or three half-space constraints for $k \in \{l, r\}$, cf., Fig. 6.22. Four binary variables $\gamma = [\gamma_{\mathrm{f}}, \gamma_{\mathrm{b}}, \gamma_{\mathrm{l}}, \gamma_{\mathrm{r}}]^\top \in \mathbb{B}^4$ are used with a big-M formulation to ensure that the vehicle is inside one of the four regions. Therefore, with $\mathbf{1}$ in the according dimension of either $\mathbf{1}_1$ for $k \in \{f, b\}$ or $\mathbf{1}_3$

for $k \in \{l, r\}$, the collision-free set is

$$\begin{aligned} \mathcal{F}^{\text{out}}(d, \overline{\sigma}) = \Big\{ (p, \gamma, \sigma) \in (\mathbb{R}^2, \mathbb{B}^4, \mathbb{R}) \Big| \forall k \in \{\text{f}, \text{b}, \text{l}, \text{r}\} : \\ A^k(d)\, p \leq b^k(d) + \mathbf{1}(1 - \gamma_k) M + \mathbf{1}(\sigma - 1)\overline{\sigma}_k,\ \mathbf{1}_4^\top \gamma = 1 \Big\}. \end{aligned} \tag{6.55}$$

The values $\overline{\sigma}_k$ define additional safety margins for each of the four collision-free regions. The slack variable $\sigma \in \mathbb{R}$ is bounded $0 \leq \sigma \leq 1$, such that only points in the exterior of $\mathcal{O}_j^{\text{out}}$ satisfy the condition in (6.55). A model predicts the future positions of the obstacle boundaries d_i^j for each obstacle $j \in I(n_{\text{obs}} - 1)$ at time steps $i \in I(N)$ along the horizon.

MIQP Cost Function

The MIQP cost comprises quadratic penalties for the state vector $x \in \mathbb{R}^{n_{\text{x}}}$ with weight $Q \in \mathbb{R}^{n_{\text{x}} \times n_{\text{x}}}$, and the control vector $u \in \mathbb{R}^{n_u}$ with weight $R \in \mathbb{R}^{n_u \times n_u}$, for tracking of their references $\tilde{x}$ and $\tilde{u}$, respectively. The reference $\tilde{x}_i = [\tilde{s}_i, \tilde{n}_i, \tilde{v}_s, 0]^\top$ at time step i is determined by the desired velocity $\tilde{v}_s$, as well as by the binary control vector $\lambda = [\lambda^{\text{up}}, \lambda^{\text{down}}]^\top \in \mathbb{B}^2$. These binary variables are used to determine lane changes at time step i, resulting in the road-aligned lateral reference $\tilde{X}_n = [\tilde{n}_0, \ldots, \tilde{n}_N]^\top \in \mathbb{R}^{N+1}$ always being the center of the target lane. The longitudinal position $\tilde{s}_i$ follows from the velocity $\tilde{v}_s$. The MIQP cost function reads

$$\begin{aligned} &\sum_{i=0}^{N} \|x_i - \tilde{x}_i\|_Q^2 + \sum_{i=0}^{N-1} \|u_i - \tilde{u}_i\|_R^2 + \\ &w_{\text{lc}} \sum_{i=0}^{N-1} \mathbf{1}_2^\top \lambda_i + w_{\text{rght}} \sum_{i=0}^{N} n_i + w_{\text{dst}} \sum_{j=0}^{n_{\text{obs}}-1} \sum_{i=0}^{N} (\sigma_i^j)^2, \end{aligned} \tag{6.56}$$

including a penalty with weight $w_{\text{rght}} > 0$ to minimize deviations from the right-most lane, a weight $w_{\text{lc}} > 0$ to penalize lane changes, and a weight $w_{\text{dst}} > 0$ penalizing slack variables to avoid being too close to any obstacle.

Parametric MIQP Formulation

For a horizon of N steps, the binary MIQP decision variables are $\Lambda = [\lambda_0, \ldots, \lambda_{N-1}] \in \mathbb{B}^{2 \times N}$ and $\Gamma = [\gamma_0^0, \gamma_1^0, \ldots, \gamma_N^{n_{\text{obs}}-1}] \in \mathbb{B}^{4 \times n_{\text{obs}}(N+1)}$, the real-valued states are $X = [x_0, \ldots, x_N] \in \mathbb{R}^{4 \times (N+1)}$, the control inputs are $U = [u_0, \ldots, u_{N-1}] \in \mathbb{R}^{2 \times N}$ and the slack variables are $\Sigma = [\sigma_0^0, \sigma_1^0, \ldots, \sigma_N^{n_{\text{obs}}-1}] \in$

$\mathbb{R}^{n_{\text{obs}}(N+1)}$. Hard linear inequality constraints are

$$\begin{aligned}
&\{H(X,U) \geq 0\} \;\Leftrightarrow \\
&\{X,U|\; \underline{u} \leq u_i \leq \overline{u}, \quad i \in I(N-1), \\
&\quad \underline{x} \leq x_i \leq \overline{x}, \quad \underline{\alpha}\, v_{s,i} \leq v_{\text{n},i} \leq \overline{\alpha}\, v_{\text{s},i}, \quad i \in I(N)\}.
\end{aligned}$$

The parameters $\Pi = (\hat{x}, \tilde{v}_s, n_{\text{lanes}}, D)$ include the initial ego state $\hat{x}$, the desired velocity $\tilde{v}_s$, the number of lanes n_{lanes} and the time-dependent obstacle bounds D, where

$$D = \left\{d_i^j \,\middle|\, \forall j \in I(n_{\text{obs}}-1), \forall i \in I(N)\right\}.$$

The parametric MIQP solved within each iteration of the MIP-DM is

$$\min_{X,U,\tilde{X}_n,\Lambda,\Gamma,\Sigma} \quad J^{\text{e}}(X,U,\tilde{X}_n,\Lambda,\Sigma) \tag{6.57a}$$

$$\text{s.t.}$$

$$x_0 = \hat{x}, \qquad H(X,U) \geq 0, \qquad 0 \leq \Sigma \leq 1,, \tag{6.57b}$$

$$\tilde{n}_{i+1} = \tilde{n}_i + d_{\text{lane}}\lambda_i^{\text{up}} - d_{\text{lane}}\lambda_i^{\text{down}},, \tag{6.57c}$$

$$x_{i+1} = Ax_i + Bu_i, \qquad i \in I(N-1),, \tag{6.57d}$$

$$(Px_i, \gamma_i^j, \sigma_i^j) \in \mathcal{F}^{\text{out}}(d_i^j, \overline{\sigma}_i^j), \qquad i \in I(N),, \quad j \in I(n_{\text{obs}}-1), \tag{6.57e}$$

where the cost $J^{\text{e}}(\cdot)$ is defined in (6.56), and $\tilde{n}_0$ is the lateral position of the center of the desired lane. An MIQP solving (6.57) is used as an “expert” to collect supervisory data, i.e., feature-label pairs (Π, B^*), where B^* is the optimal value of binary variables, cf., Fig. 6.21. For the closed-loop evaluation of the expert MIP-DM, the optimizer X^* is used as the output of the expert MIP-DM X^{e}.

6.3.4 Scalable Equivariant Deep Neural Network

Because the MIQP (6.57) is computationally demanding to solve in real-time, especially for long prediction horizons and a large number of obstacles, we propose a novel variant of the combinatorial offline convex online (COCO) algorithm [63] to accelerate MIQP solutions using supervised learning. We train a NN to predict binary variables and then solve the remaining convex QP online after fixing the binary variables. The MIQP (6.57) comprises $4n_{\text{obs}}N$ structured binary variables related to obstacles and $2N$ lane change variables.

We refer to the latter as *unstructured* binary variables because, differently from the others, there is no specific relation besides recurrence among them.

In the following, we first describe the desired properties of the NN prediction in Sec. 6.3.4 and review the classification of binary variables in Sec. 6.3.4. Then, in Sec. 6.3.4, we introduce one of our main contributions, the REDS, to achieve the desired properties. Finally, we show in Sec. 6.3.4 how to use an ensemble of NNs to generate multiple predictions.

Desired Predictor Properties for Motion Planning

The desired properties of the predictor may be divided into performance, i.e., general metrics that define the NN prediction performance, and structural properties, i.e., structure-exploiting properties related to Requirements 6.3.1-6.3.5.

Performance. The prediction performance is quantified by the likelihood of predicting a feasible solution μ, and a measure of optimality ρ. However, supervised learning optimizes accuracy, i.e., cross-entropy loss, which was shown to correlate well with feasibility [39, 63, 62]. In fact, we evaluated the Pearson correlation coefficient (PCC) for our experiments and obtained a PCC of 0.81 for the correlation between the training loss and the infeasibility and a PCC of 0.88 between accuracy and feasibility. In addition, the computation time t_{comp} to evaluate the NN is important for real-time feasibility. We aim for t_{comp} to be very small compared to the MIQP solution time, and $t_{\text{comp}} < t_{\text{p}} \leq 0.2\text{s}$ (see Req. 6.3.4). Finally, the memory footprint of the NN should be small for implementation on embedded microprocessors [78].

Structural. First, the REDS planner needs to operate on a variable number of obstacles, which is required in real traffic scenarios, see Req. 6.3.5. Second, to comply with Req. 6.3.3, obstacle-related predictions need to be equivariant to permutations on the input, see Def. 6.3.2. For unstructured binary variables, the predictions should be permutation invariant, see Def. 6.3.1. Third, again, relating to Req. 6.3.5, the NN architecture is expected to generalize to unseen data. In particular, it should provide accurate predictions for several obstacles that may not be present in the training data. Finally, the NN should predict multiple guesses to increase the likelihood of feasibility and/or optimality. The proposed REDS planner provides the desired structural properties, and it improves the performance properties.

Since we consider a highly structured problem domain, we propose to directly include the known structure into the NN architecture. Alternatively, one could learn the structure for general problems by, e.g., attention mechanisms [288].

However, the related attention-based architectures usually have a high inference time, which may clash with the desired fast online evaluation.

Prediction of Binary Variables by Classification

As the authors in [40, 62] show, solving the prediction of binary variables as a multi-class classification problem yields superior results than solving it as a regression problem. Since the naive enumeration of binary assignments grows exponentially, i.e., the number of assignments is $2^{|B|}$, where $|B|$ is the number of binary variables, an effective strategy is to enumerate only combinations of binary assignments that actually appear in the data set. While it cannot guarantee to predict all combinations, this leads to a much smaller number of possible classes, and it has been observed that the resulting classifications still significantly outperform regression (cf., [62]).

Recurrent Equivariant Deep Set Architecture

The REDS architecture, shown in Fig. 6.23, achieves the structural properties and improves the prediction performance. We use training data that consists of feature-label pairs (Π, B). The features Π are split into obstacle-related features $\zeta^{\mathrm{eq}} = \{\zeta_0^{\mathrm{eq}}, \dots, \zeta_{n_{\mathrm{obs}}-1}^{\mathrm{eq}}\}$, where $\zeta_i^{\mathrm{eq}} \in \mathbb{R}^{m_{\mathrm{eq}}}$, and unstructured features $\zeta^{\mathrm{us}} \in \mathbb{R}^{m_{\mathrm{us}}}$. The equivariant features $\zeta^{\mathrm{eq}} = (z_j, d_j)$ are the *initial* obstacle state z_j and its spatial dimension d_j since all states along the horizon are predicted based on the initial state. The unstructured features contain all other parameters of Π, i.e., $\zeta^{\mathrm{us}} = (\hat{x}, \tilde{v}_s, n_{\mathrm{lanes}})$.

The work in [312] proposes a simple but effective NN architecture that provides either permutation equivariance or invariance. The blocks are combined in our tailored *encoder layer* that maintains permutation equivariance for the equivariant outputs and invariance for the unstructured outputs, see Fig. 6.23. A hidden state $h^{\mathrm{us}} \in \mathbb{R}^{m_{\mathrm{h}}}$ is propagated for unstructured features and hidden states $h_j^{\mathrm{eq}} \in \mathbb{R}^{m_{\mathrm{h}}}$, with $j \in I(n_{\mathrm{obs}} - 1)$, for the equivariant features, where $h^{\mathrm{eq}} = [h_0^{\mathrm{eq}}, \dots, h_{n_{\mathrm{obs}}-1}^{\mathrm{eq}}]^\top \in \mathbb{R}^{n_{\mathrm{obs}} \times m_{\mathrm{h}}}$. The encoder layer has four directions of information passing between the fixed-size unstructured and the variable-size equivariant hidden states with input dimension m_{h} and output dimension m_{h}':

1. Equivariant to Equivariant: Equivariant deep sets [312] are used as layers with ReLU activation functions $\sigma(\cdot)$ and parameters $\Theta^{\mathrm{ee}}, \Gamma^{\mathrm{ee}} \in \mathbb{R}^{m_{\mathrm{h}} \times m_{\mathrm{h}}'}$:

$$f^{\mathrm{ee}}(h^{\mathrm{eq}}) = \sigma(h^{\mathrm{eq}}\Theta^{\mathrm{ee}} + \mathbf{1}\mathbf{1}^\top h^{\mathrm{eq}}\Gamma^{\mathrm{ee}}).$$

2. Equivariant to Unstructured: To achieve invariance from the set elements h_j^{eq} to the unstructured hidden state h^{us}, the invariant layer of [312] is

added that sums up over the set elements with parameters $\Theta^{\text{eu}} \in \mathbb{R}^{m_\text{h} \times m'_\text{h}}$,

$$f^{\text{eu}}(h^{\text{eq}}) = \sigma\left(\Big(\sum_{j=0}^{n_\text{obs}-1} h_j^{\text{eq}}\Big)\Theta^{\text{eu}}\right).$$

3. Unstructured to Equivariant: To implement a dependency of the equivariant elements on the unstructured hidden state while maintaining equivariance, this layer equally influences each set element by

$$f^{\text{ue}}(h^{\text{us}}) = \sigma(\mathbf{1}_{n_\text{obs}} \otimes h^{\text{us}}\Theta^{\text{ue}}),$$

with parameters $\Theta^{\text{ue}} \in \mathbb{R}^{m_\text{h} \times m'_\text{h}}$.

4. Unstructured to Unstructured: We use a standard feed-forward layer with parameters $\Theta^{\text{uu}} \in \mathbb{R}^{m_\text{h} \times m'_\text{h}}$

$$f^{\text{uu}}(h^{\text{us}}) = \sigma(h^{\text{us}}\Theta^{\text{uu}}).$$

Feed forwards (FFs) act as encoders to the input features, which allows to matching the dimensions of the equivariant and unstructured hidden states. For each layer of the REDS in Fig. 6.23, the contributions are summed up to obtain the new hidden states

$$h^{\text{eq}\prime} = f^{\text{ee}}(h^{\text{eq}}) + f^{\text{ue}}(h^{\text{us}}),$$

$$h^{\text{us}\prime} = f^{\text{uu}}(h^{\text{us}}) + f^{\text{eu}}(h^{\text{eq}}).$$

A long short term memory (LSTM) is used as *decoder* for each equivariant hidden state, transforming the hidden state into a time series of binary predictions, see Fig. 6.23. Another LSTM is used as a *decoder* for unstructured hidden states. For the REDS, the classification problem per time step (Sec. 6.3.4) needs to consider only four classes per obstacle (one for each collision-free region), cf. Fig. 6.22, and three classes for lane changes (change to the left or right, stay in lane).

Neural Network Ensembles for Multiple Predictions

As suggested in early works [118, 262], using an ensemble of n_e stochastically trained NNs is a simple approach to obtain multiple guesses and improve classification accuracy. Producing multiple predictions, the lowest cost maneuver among different candidate solutions can be selected, e.g., staying behind a vehicle or overtaking. For typical classification tasks, no a-posteriori oracle exists that identifies the *best* guess [118]. However, in our problem setup, the soft-QP solution can directly evaluate the feasibility and optimality and, therefore, identify the best guess, as discussed next.

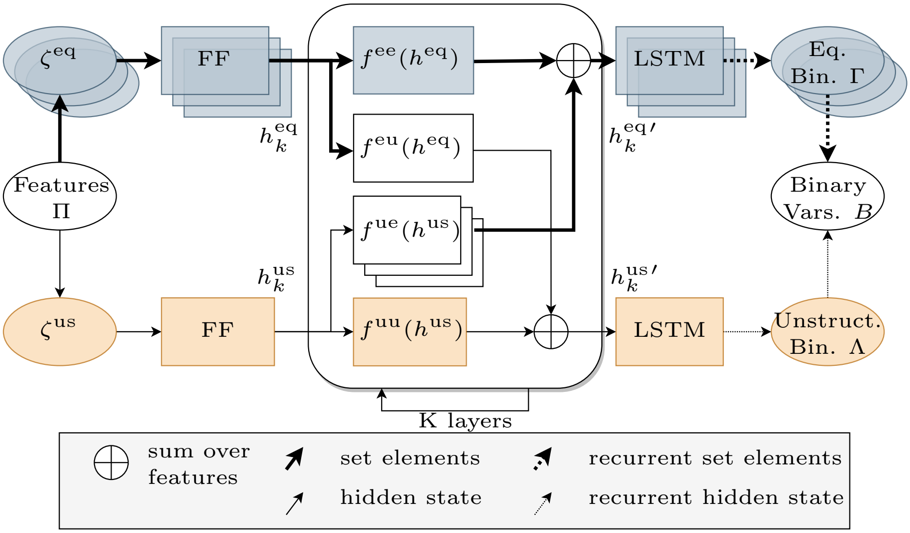

Figure 6.23: REDS network. The blue blocks show the propagation of equivariant features, whereas the orange blocks show the propagation of unstructured features. An invariant connection couples both hidden states.

6.3.5 Soft QP Solution and Selection Method

For each candidate solution from the ensemble of NNs, the soft-QP is constructed based on the MIQP (6.57) with fixed binary variables and unbounded slack variables for the obstacle constraints. Each candidate solution consists of the binary values $\hat{\Lambda} = [\hat{\lambda}_0, \dots, \hat{\lambda}_{N-1}]$ and $\hat{\Gamma} = [\hat{\gamma}_0^0, \hat{\gamma}_1^0, \dots, \hat{\gamma}_N^{n_{\text{obs}}-1}]$ that are predicted by the NN. The reference $\tilde{X}_n$ in (6.57c) is also fixed when the binary variables are fixed. The resulting soft-QP is convex, and a feasible solution always exists due to removing the upper bound for each slack variable σ_i^j, with

$j \in I(n_{\text{obs}} - 1)$, $i \in I(N)$ and

$$J^{\text{s}*} = \min_{X, U, \Sigma} \quad J^{\text{s}}(X, U, \Sigma) \tag{6.58a}$$

$$\text{s.t.}$$

$$x_0 = \hat{x}, \quad H(X, U) \geq 0, \qquad \Sigma \geq 0, \tag{6.58b}$$

$$x_{i+1} = Ax_i + Bu_i, \qquad i \in I(N-1), \tag{6.58c}$$

$$(Px_i, \sigma_i^j) \in \mathcal{F}^{\text{out}}(d_i^j, \hat{\gamma}_i^j), \quad i \in I(N), \quad j \in I(n_{\text{obs}} - 1), \tag{6.58d}$$

where the cost $J^{\text{s}}(\cdot)$ is (6.56), see Tab. 6.7. We construct (6.58d) from (6.55) by removing the safety margins $\overline{\sigma}_i^j$ and fixing the binary variables to the predicted solution guess $\hat{\Gamma}$. Therefore, the soft-QP (6.58) is a relaxation of MIQP (6.57) with fixed integers. Problem (6.58) is solved for each prediction of the NN ensemble, and the solution leading to the lowest cost for (6.58a) is selected as output X^{p} of the module, see Fig. 6.21. The soft-QP is convex and can be solved efficiently using a structure exploiting QP solver [291, 97]. Despite solving multiple QPs for multiple candidate solutions, the computational burden is much lower than solving an MIQP that typically requires solving a combinatorial amount of convex relaxations.

6.3.6 Feasibility Projection and SQP Algorithm

The soft-QP solution $(X^{\text{p}}, U^{\text{p}})$ may not be collision-free due to binary classification errors from the ensemble of NNs, i.e., some of the slack variables may be nonzero in the soft-QP solution. In order to project the soft-QP solution to a collision-free trajectory, a smooth convex-concave NLP, referred to as FP, is solved in each iteration. The FP solves an optimization problem that is similar to (6.58) (see Tab. 6.7), but the reference trajectory is equal to the soft-QP solution, i.e., $\tilde{X} := X^{\text{p}}$ and $\tilde{U} := U^{\text{p}}$. In addition, the obstacle constraints in (6.58d) are replaced by smooth concave constraints based on the ellipsoidal collision region, cf. Fig. 6.22, which allows the use of an efficient SQP algorithm, e.g., [74].

Optimization Problem Formulation

With the geometry parameter d of an obstacle, the inner ellipse $\mathcal{O}^{\text{safe}}$ within $\mathcal{O}^{\text{out}}$ is used for defining the safe-set $\mathcal{F}^{\text{safe}}$. The ellipse center and axis matrix

$$t(d) = \frac{1}{2}\big[(s_{\text{f}} + s_{\text{b}}), (n_{\text{l}} + n_{\text{r}})\big]^{\top}, \quad \Upsilon(d) = \frac{1}{\sqrt{2}}\text{diag}([s_{\text{f}} - s_{\text{b}}, n_{\text{l}} - n_{\text{r}}]),$$

are used to formulate the smooth obstacle constraint

$$\mathcal{F}^{\text{safe}}(d) = \Big\{(p, \xi)\Big| \, \|p - t(d)\|^2_{\Upsilon^{-1}(d)} \geq 1 - \xi\Big\}, \tag{6.59}$$

with slack variables $\xi \geq 0$. A tracking cost

$$J^{\text{f}}_{\text{tr}}(X, U) = \sum_{i=0}^{N} \|x_i - \tilde{x}_i\|^2_Q + \sum_{i=0}^{N-1} \|u_i - \tilde{u}_i\|^2_R \,, \tag{6.60}$$

and a slack violation cost

$$J^{\text{f}}_{\text{slack}}(\Xi) = w_{\text{h}} \sum_{j=0}^{n_{\text{obs}}-1} \sum_{i=0}^{N} \xi^j_i \,, \tag{6.61}$$

are defined, where $\Xi = \{\xi^j_i | i \in I(N), j \in I(n_{\text{obs}} - 1)\}$. The penalty $w_{\text{h}} \gg 0$ is sufficiently large such that a feasible solution with $\xi^j_i = 0$ can be found when it exists. The resulting NLP can be written as

$$\min_{X, U, \Xi} \quad J^{\text{f}}_{\text{tr}}(X, U) + J^{\text{f}}_{\text{slack}}(\Xi) \tag{6.62a}$$

$$\text{s.t.}$$

$$x_0 = \hat{x}, \quad H(X, U) \geq 0, \quad \Xi \geq 0, \tag{6.62b}$$

$$x_{i+1} = Ax_i + Bu_i, \qquad i \in I(N-1), \tag{6.62c}$$

$$(Px_i, \xi^j_i) \in \mathcal{F}^{\text{safe}}(d^j_i), \qquad i \in I(N), \quad j \in I(n_{\text{obs}} - 1), \tag{6.62d}$$

using the least-squares tracking cost (6.60) and smooth obstacle avoidance constraints (6.59). The optimal trajectory X^* of (6.62) is the output of the FP X^{s} and also of the REDS planner and tracked by the NMPC, see Fig. 6.21. Since $\mathcal{F}^{\text{out}} \subseteq \mathcal{F}^{\text{safe}}$, the NLP (6.62) is a smooth relaxation of the MIQP (6.57).

Application of Sequential Quadratic Programming

NLP (6.62) has a convex-concave structure. Except for the concave constraints (6.62d), the problem can be formulated as a convex QP. When solving NLP (6.62) using the Gauss-Newton SQP method [111], this guarantees a bound on the constraint violation for the solution guess at each SQP iteration [284].

Proposition 6.3.1. *Consider NLP* (6.62)*, let* X_i, U_i, Ξ_i *be the primal variables after an SQP iteration with Gauss-Newton Hessian approximation, and let* X_0, U_0, Ξ_0 *be an initial guess equal to the reference, i.e.,* $X_0 = \tilde{X}$*,* $U_0 = \tilde{U}$*. Then, the decrease in the slack cost reads*

$$J^{\mathrm{f}}_{\mathrm{slack}}(\Xi_i) \leq J^{\mathrm{f}}_{\mathrm{slack}}(\Xi_0) - J^{\mathrm{f}}_{\mathrm{tr}}(X_i, U_i). \tag{6.63}$$

Proof. According to Lemma 4.2 of [284], the cost of (6.62) decreases after each iteration for our problem structure, i.e.,

$$J^{\mathrm{f}}_{\mathrm{tr}}(X_{i+1}, U_{i+1}) + J^{\mathrm{f}}_{\mathrm{slack}}(\Xi_{i+1}) \leq J^{\mathrm{f}}_{\mathrm{tr}}(X_i, U_i) + J^{\mathrm{f}}_{\mathrm{slack}}(\Xi_i).$$

Consequently, Eq. (6.63) can be verified, since it holds that $J^{\mathrm{f}}_{\mathrm{tr}}(X_0, U_0) = 0$ due to the initialization equal to the reference and $J^{\mathrm{f}}_{\mathrm{tr}}(X, U) \geq 0, \forall X, U$, such that

$$J^{\mathrm{f}}_{\mathrm{tr}}(X_i, U_i) + J^{\mathrm{f}}_{\mathrm{slack}}(\Xi_i) \leq J^{\mathrm{f}}_{\mathrm{slack}}(\Xi_0).$$

□

Additionally, it can be guaranteed that the SQP iterations remain feasible, i.e., collision-free, once a feasible solution is found, if the slack weights $w_{\mathrm{h}} \gg 0$ are chosen *sufficiently large.* The latter requires to select a weight value such that the gradient of the *exact* L1 penalty (6.61) is larger than any gradient of the quadratic cost (6.60) in the bounded feasible domain. This property is due to the exact penalty formulation [194] and the inner approximations of the concave constraints (6.62d), since a feasible linearization point in iteration j guarantees feasibility also in the next iterate $j+1$ [74]. Notably, choosing sufficiently large weights, yet not too large to avoid ill-conditioned QPs, may be challenging in practice. Therefore, the weights could be increased in each iteration if convergence issues were encountered [194]. However, we used constant weights without encountering numerical problems.

Under mild assumptions [284], the SQP method converges to a stationary point of (6.62). The FP provides a certificate of feasibility if the slack variables are zero, i.e., $\Xi_j = 0$. We show that the FP effectively increases the likelihood of computing a collision-free trajectory in numerical simulations.

6.3.7 Implementation Details

In this section, we list the most important details of implementation. Further information on the structure of the NN model, training parameters, and the FP are in the Appendix.

Numerical solvers. For solving the MIQP of the expert MIP-DM and the QP of the soft-QP, we use `Gurobi` [114] and formulate both problems in `cvxpy` [79]. The NLPs of the FP, as well as the reference tracking NMPC, are solved by using the open-source solver `acados` [291] and the algorithmic differentiation framework `CasADi` [14]. We use Gauss-Newton Hessian approximations, full steps, an explicit RK4 integrator, and non-condensed QPs that are solved by `HPIPM` [97]. We use real-time iterations [80] within the reference tracking NMPC and limit the maximum number of SQP iterations to 10 within the FP.

Training of the NN. In order to formulate and train the NNs, we use `PyTorch`. We train on datasets of 10^5 expert trajectories that we generate by solving the expert MIP-DM with randomized initial parameters, sampled from an uniform distribution within the problem bounds, see Appendix 6.3.11. We use a learning rate of 10^{-4} and a batch size of 1024. The performance is evaluated on a test dataset of $2 \cdot 10^3$ samples using a *cross-entropy loss* and the `adam` optimizer [147].

Computations. Simulations are executed on a `LENOVO ThinkPad L15 Gen 1` Laptop with an `Intel(R) Core(TM) i7-10510U @ 1.80GHz` CPU. The training and GPU evaluations of the NNs are performed on an Ubuntu workstation with two `GeForce RTX 2080 Ti PCI-E 3.0 11264MB GPUs`. Parts of the REDS planner, namely the n_e NN ensemble and the soft-QP, can be parallelized, which speeds up our approach by approximately the number of NNs used. Therefore, the *serial* computation time

$$t_\mathrm{s} = \sum_{i=1}^{n_\mathrm{e}} (t_{\mathrm{NN},i} + t_{\mathrm{QP},i}) + t_\mathrm{FP},$$

includes each individual NN inference time $t_{\mathrm{NN},i}$, each soft-QP evaluation time $t_{\mathrm{QP},i}$ and the FP evaluation time t_FP. The *parallel* computation time is

$$t_\mathrm{p} = \max_{i=1,\dots,n_\mathrm{e}} (t_{\mathrm{NN},i} + t_{\mathrm{QP},i}) + t_\mathrm{FP}.$$

Nonlinear Model Predictive Control. The lower-level reference tracking NMPC is formulated as shown in [222], which is similar to (6.62), but using a more detailed nonlinear kinematic vehicle model, a shorter sampling period and a shorter horizon (see Tab. 6.7). The reference tracking NMPC can be solved efficiently using the real-time iterations [80], based on a Gauss-Newton SQP method [111] in combination with a structure exploiting QP solver [291].

6.3.8 In-Distribution Evaluations

First, we show the performance of the REDS architecture, compared to the state-of-the-art architectures of [62], and we demonstrate its *structural properties*, see Sec. 6.3.4. Next, we show how the soft-QP and the feasibility projection increase the prediction performance and the influence on the overall computation time. In this section, we use the same distribution over the input parameters for training and testing of the NNs.

Evaluation of the REDS

We compare two variants of the proposed architecture. First, the REDS is evaluated, see Fig. 6.23. Second, as an ablation study, the same architecture but without the LSTM, referred to as equivariant deep set (EDS), is evaluated. In the EDS, a FF is used to predict the binary variables along the full prediction horizon (see Fig. 6.23). The performance is compared against the state-of-the-art architectures for similar tasks [62], i.e., an LSTM and a FF network with a comparable amount of parameters.

The evaluation metrics are the infeasibility rate, i.e., the share of infeasible soft-QPs violating constraints (with nonzero slack variables), and the misclassification rate, i.e., the share of wrong classifications concerning the prediction of any binary variable. If at least one binary variable in the prediction is wrongly classified, the whole prediction is counted as misclassified, even though the soft-QP computes a feasible low-cost solution. Furthermore, we consider the suboptimality ρ, i.e., the objective of the feasible soft-QP solutions $J^{\mathrm{s}*}$ compared to the expert MIP-DM cost $J^{\mathrm{e}*}$,

$$\rho = \frac{J^{\mathrm{s}*} - J^{\mathrm{e}*}}{J^{\mathrm{e}*}} \geq 0. \tag{6.64}$$

A suboptimality of $\rho = 0$ means the cost of the prediction is equal to the optimal cost of the expert MIP-DM. Fig. 6.24 shows how the performance scales with the number of obstacles n_{obs} and with the horizon length N, without the FP from Sec. 6.3.6. The REDS and EDS yield superior results to the LSTM as soon as $n_{\mathrm{obs}} \geq 2$, and they vastly outperform the FF network for an increasing number of obstacles and horizon length.

Besides improving the *performance* metrics, the REDS also provides the desired structural properties. A REDS network can be trained and evaluated with a variable number of obstacles and prediction horizon. In Tab. 6.8, the generalization performance of REDS network is shown, i.e., it is evaluated

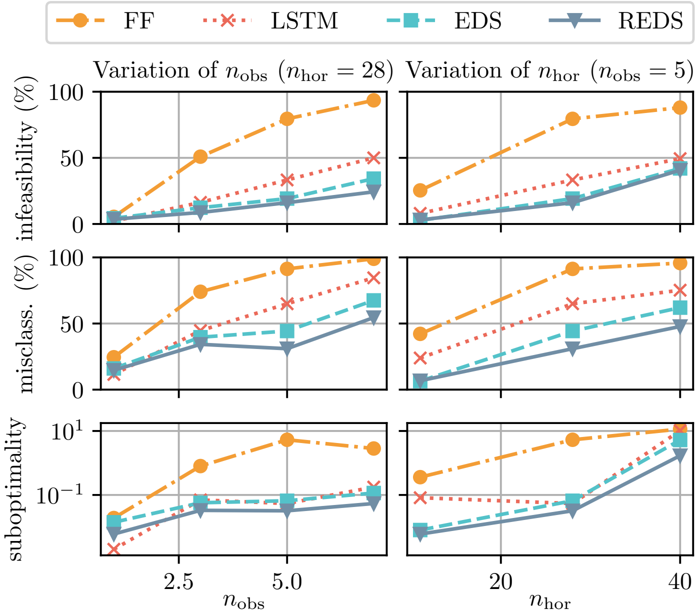

Figure 6.24: Performance evaluation for infeasibility rate, misclassification rate, and suboptimality of different network architectures, depending on the number of obstacles and horizon length. The REDS network outperforms the other architectures, particularly for a larger number of obstacles and a longer horizon. Suboptimality is shown in a logarithmic scale.

for the number of obstacles that were not present in the training data. Tab. 6.8 compares generalization for an *interpolation* and *extrapolation* of the number of obstacles and the prediction horizon. According to Tab. 6.8, REDS generalizes well for samples out of the training data distribution for the number of obstacles and the horizon length.

Evaluation for Ensemble of REDS Networks

In Fig. 6.25, we show the achieved feasibility rate on the test data using an ensemble with $n_{\text{e}} = 10$ REDS networks. In total, $2 \cdot 10^3$ samples are evaluated for each NN individually and cumulatively. The performance improves by adding more NNs. However, also the total computation time increases. The parallel computation time results in the maximum time over the individual networks. Tab. 6.9 shows the memory usage and the number of parameters for the different architectures. The sizes of the networks are feasible for embedded devices [78], i.e., the REDS and EDS provide improved performance with a comparable or

	Training		Testing		Performance (%)		
	n_{obs}	N	n_{obs}	N	misclass.	infeas.	subopt.
	Generalization for the number of obstacles: *interpolation*						
No general.	1-5	28	3	28	41.7	19.7	17.4
General.	1,2,4,5	28	3	28	**42.0**	**19.8**	**11.6**
	Generalization for the number of obstacles: *extrapolation*						
No general.	1-5	28	5	28	30.3	24.4	3.9
General.	1-4	28	5	28	**34.4**	**26.5**	**3.4**
	Generalization for the horizon length						
No general.	1-3	16	1-3	16	20.4	8.9	38.1
General.	1-3	12	1-3	16	**26.0**	**10.8**	**20.9**

Table 6.8: Evaluation of the REDS network generalization performance (highlighted in bold).

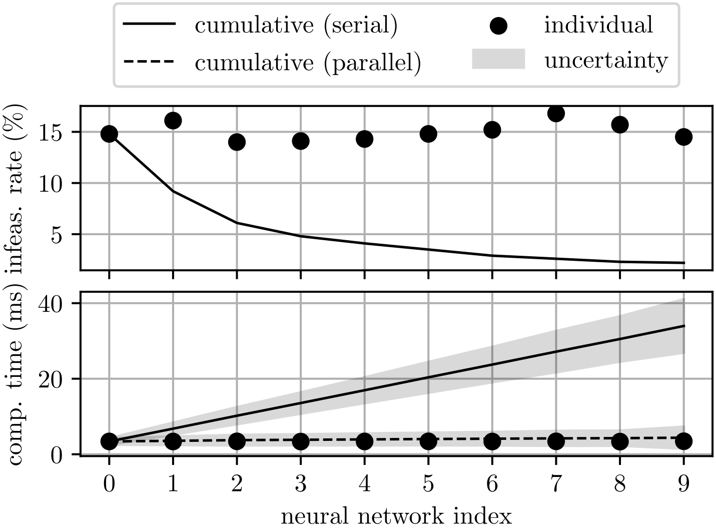

Figure 6.25: Individual and cumulative REDS prediction performance (infeasibility rate) and computation time concerning the number of NNs used in the ensemble. The computation time differs for parallel and serial evaluation. For the parallel evaluation, the slowest network determines the computation time.

	FF	LSTM	EDS	REDS
Memory usage (MB)	2.40	0.97	0.51	1.40
Number of parameters (10^5)	5.98	2.41	1.26	3.43

Table 6.9: Memory usage and number of parameters in NN architectures involving five obstacles and a horizon of 28 steps.

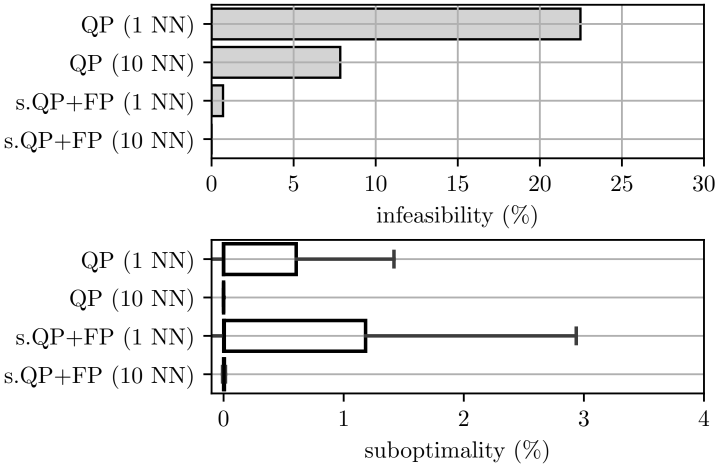

Figure 6.26: Open-loop comparison of infeasibility rate and suboptimality of the REDS planner, using the QP without slack variables and the soft-QP followed by the FP. Infeasible problems are not considered in the suboptimality.

even reduced memory footprint than FFs and LSTMs. This may be due to exploiting the structure of equivariances and invariances inherently occurring in the application. Similar observations were made in other applications where deep sets have been applied, e.g., [126].

Evaluation of REDS Planner with Feasibility Projection

The REDS planner is validated on random samples of the test data, i.e., samples of the same distribution as the NN training data. Fig. 6.26 shows the infeasibility rate and suboptimality (6.64) of the REDS planner, using either the QP without slack variables, or the soft-QP followed by the FP. For an ensemble of ten NNs, the infeasibility rate of the QP without slack variables is below 10% and decreased to almost 0% by using the soft-QP followed by the FP, and the

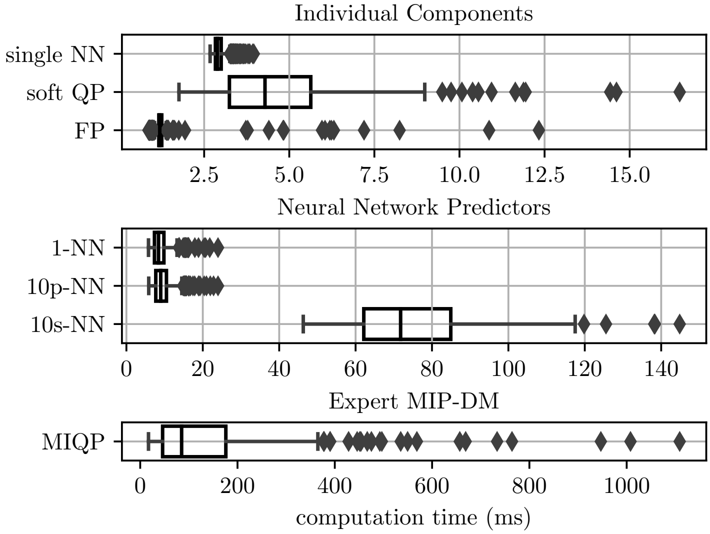

Figure 6.27: Box plots for open-loop comparison of the computation times of 10^3 samples. REDS planner with an ensemble of one (1-NN) or ten NNs is parallelized (10p) or serial (10s).

suboptimality is also negligible. The suboptimality of the soft-QP followed by the FP is higher than the suboptimality of the QP since also infeasible problems are rendered feasible, yet with increased suboptimality values. Fig. 6.27 shows box plots of the computation times related to the different components. The main contributions to the total computation time of the REDS planner stem from the soft-QP (median of $\sim$ 4.2ms) and the NNs (median of $\sim$ 3.0ms per network). While the parallel architecture with ten NNs, as well as a single NN, decrease the worst case MIQP computation time by a factor of approximately 50 and the median by a factor of 10, the serial approach decreases the maximum by a factor of 8 and the median by a factor of 2. Besides computation time, the REDS planner is suitable for embedded system implementation as opposed to high-performance commercial solvers like the one used here as an expert, i.e., `Gurobi` [114].

6.3.9 Closed-loop Validations with SUMO Simulator

The following closed-loop evaluations of the REDS planner on a multi-lane highway scenario yield a more realistic performance measure. This involves challenges such as the distribution shift of the input parameters, wrong predictions, and errors of the reference tracking NMPC.

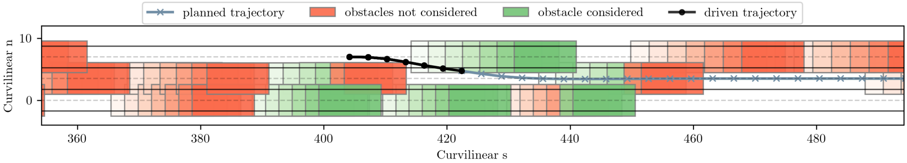

Figure 6.28: Obstacles considered at each planning step in the Frenet coordinate frame. The plot shows the ego point-mass trajectory and the inflated obstacles in proximity to the ego vehicle for seven time steps. The obstacle color indicates whether it was considered in planning or not.

Setup for Closed-loop Simulations

Several choices need to be made for parameters in the REDS planner and in the simulation environment.

Collision avoidance. For an environment with a large number of obstacles, we implement the following heuristic to select up to $n_{\text{obs}} = 5$ obstacles in proximity to the ego vehicle. We consider the closest successive obstacles in each lane, which are not the lane of the ego vehicle and the leading vehicle on the ego lane. In Fig. 6.28, the selection of obstacles in proximity to the ego vehicle is shown in the Frenet coordinate frame. Obstacles are plotted in consecutive planner time steps, and the color indicates whether they are considered at each time step. Overtaking is allowed on both sides of a leading vehicle.

Sampling frequency. The planning frequency is set to 5Hz, the control and ego vehicle simulation rate is 50Hz, and the `SUMO` simulation is 10Hz. The traffic simulator in `SUMO` is slower than the ego vehicle simulation frequency. Therefore, the motion of the vehicles is linearly extrapolated between `SUMO` updates. Indeed, the REDS planner is real-time capable for selected frequencies, see Req. 6.3.4.

Vehicle models. We use parameters of a *BMW 320i* for the ego vehicle, which is a medium-sized passenger vehicle whose parameters are provided in `CommonRoad` [13] for models of different fidelity. An *odeint* integrator of `scipy` simulates the 29-state *multi-body model* [13] with a 20ms time step. The traffic simulator `SUMO` [170] simulates interactive driving behaviors with the *Krauss model* [155] for car following and the *LC2013* [88] model for lane changing. From zero, up to five surrounding vehicles are selected for our comparison.

Road layout. Standardized scenarios on German roads, provided by the *scenario database* of `CommonRoad`, are fully randomized before each closed-loop simulation, including the start configuration of the ego vehicle and all other vehicles. This initial randomization, in addition to the interactive and stochastic behavior simulated in `SUMO`, covers a wide range of traffic situations, including traffic jams, blocked lanes, and irrational driver decisions such as half-completed lane changes. The basis of our evaluations is the three-lane scenario `DEU_Cologne-63_5_I-1` with all (*dense*) or only a fourth (*sparse*) of the vehicles in the database.

Distribution Shift

The generally unknown state distribution encountered during closed-loop simulations, referred to as simulation distribution (SD), differs from the training distribution (TD). We aim to generalize the proposed approach to a wide range of scenarios. Hence, we use the uniform distribution given in Tab. 6.3.11 to train the NNs and in consecutive closed-loop evaluations. However, if the encountered SD is known better, we propose to include NNs trained on an *a priori* known SD and include it in the ensemble of NNs. In Fig. 6.29, the worse performance of an ensemble of REDS, purely trained on the uniform TD, followed by the soft-QP is shown when evaluated for samples taken from closed-loop simulations of the `DEU_Cologne-63_5_I-1` scenario using the expert MIP-DM. Additionally, Fig. 6.29 shows NNs trained on this SD and how they can improve the prediction performance as single networks by $\sim 10\%$ and in an ensemble by $\sim 3\%$.

Closed-loop Results with `SUMO` Simulator

We compare the closed-loop performance of the REDS planner with varying numbers of NNs and of the expert MIP-DM for the dense and sparse traffic in scenario `DEU_Cologne-63_5_I-1`, cf., Fig. 6.30. The closed-loop cost is computed by evaluating the objective (6.56) for the closed-loop trajectory and is separated into its components. The computation times are shown for the serial and parallel evaluation of the NNs. Similar to the open-loop evaluation in Fig. 6.25, the closed-loop results in Fig. 6.30 show a considerable performance gain when more NNs are added to the ensemble. Evaluating 10 NNs in parallel within the REDS planner leads to nearly the same closed-loop performance as the expert MIP-DM. Remarkably, a parallel computation achieves a tremendous speed-up of the worst-case computation time of approximately 100 times. A serial evaluation of 10 NNs could still be computed around 25 times faster for the maximum computation time compared to the expert MIP-DM. All computation times of the REDS planner variations are below the threshold of 200ms, while the expert MIP-DM computation time exceeds the threshold in more than 50%,

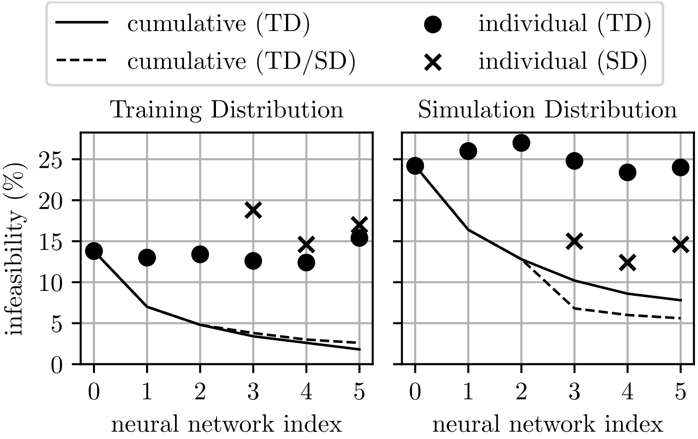

Figure 6.29: Individual and cumulative REDS prediction performance for the training distribution (TD) and samples encountered by an simulation distribution (SD). Two variants of the REDS ensemble are compared: one purely trained on the uniform training distribution (TD) and one ensemble with three NNs trained on the SD.

taking up to 4s for one iteration. Tab. 6.10 shows further performance metrics. Using more NNs within an ensemble increases the average velocity and decreases the closed-loop cost towards the expert MIP-DM performance. Using six or ten NNs, a single situation occurred where a leading vehicle started to change lanes but halfway decided to change back towards the original lane, resulting in a collision, which in a real situation will be avoided by an emergency (braking) maneuver. Fig. 6.31 shows snapshots of a randomized closed-loop `SUMO` simulation of a dense and sparse traffic scenario `DEU_Cologne-63_5_I-1`. The *green* colored ego vehicle successfully plans lane changes and overtaking maneuvers to avoid collisions with other vehicles (blue), using the proposed REDS planner feeding a reference tracking NMPC.

6.3.10 Conclusions and Discussion

We proposed a supervised learning approach for achieving real-time feasibility for mixed-integer motion planning problems. Several concepts are introduced to achieve a nearly optimal closed-loop performance when compared to an expert MIQP planner. First, it was shown that inducing structural problem properties such as invariance, equivariance, and recurrence into the NN architecture improves the prediction performance among other useful properties such as generalization to unseen data. Secondly, the soft-QP can correct wrong predictions, inevitably linked to the NN predictions, and are able to evaluate

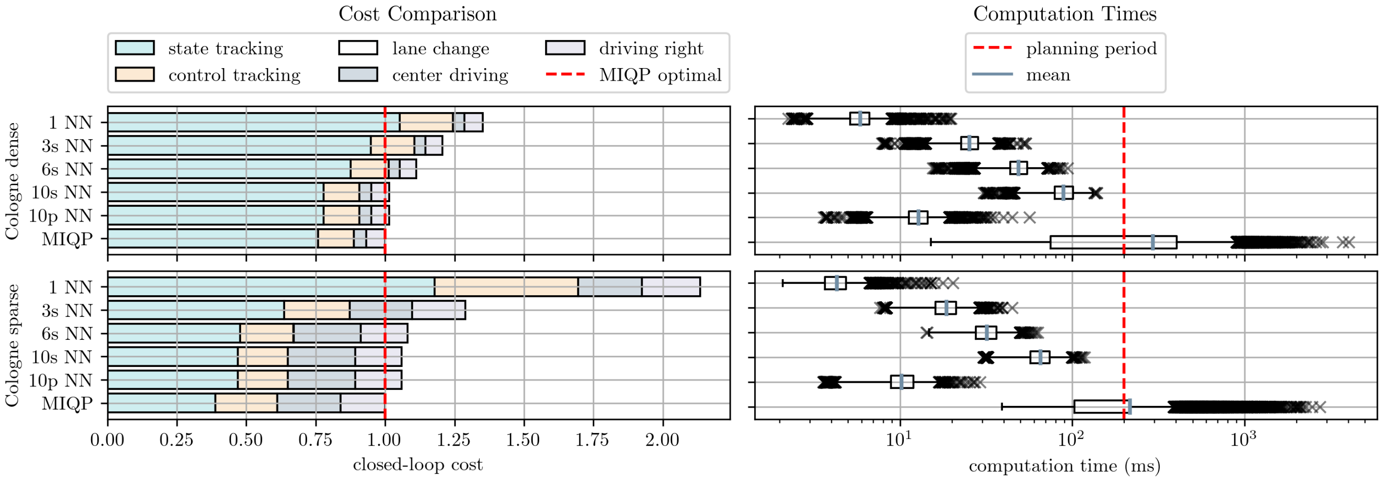

Figure 6.30: Performance comparison for different numbers of serial (s) and parallel (p) NNs, for dense and sparse traffic in scenario `DEU_Cologne-63_5_I-1`. The closed-loop cost was computed by evaluating (6.56) for the closed-loop ego trajectory and normalized against the closed-loop cost of expert MIP-DM. The state tracking cost, including the lateral position and the desired velocity tracking error, contributes the most to the chosen weights.

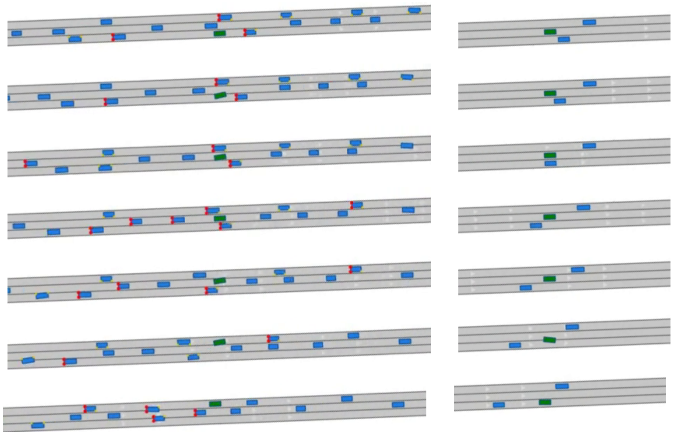

Figure 6.31: Snapshots of simulated traffic scenario `DEU_Cologne-63_5_I-1` dense (left) and sparse (right) with `SUMO` and `CommonRoad`, showing the ego vehicle (green) and other vehicles (blue). The REDS planner is real-time feasible and used in combination with reference tracking NMPC, see Fig. 6.21. A red light at the rear of the vehicle indicates braking.

		`DEU_Cologne-63_5_I-1`-dense				
Property	Unit	MIQP	NN-1	NN-3	NN-6	NN-10
collisions		0	0	0	1	1
vel. mean	$\frac{\mathrm{m}}{\mathrm{s}}$	13.10	12.71	12.91	13.00	13.07
vel. min	$\frac{\mathrm{m}}{\mathrm{s}}$	0.41	1.17	0.00	0.00	0.00
lane changes		457	431	469	471	462
cost	$\frac{1}{\mathrm{s}}$	49.48	66.89	59.68	55.02	50.22
		`DEU_Cologne-63_5_I-1`-sparse				
Property	Unit	MIQP	NN-1	NN-3	NN-6	NN-10
collisions		0	0	0	0	0
vel. mean	$\frac{\mathrm{m}}{\mathrm{s}}$	13.94	13.69	13.83	13.87	13.89
vel. min	$\frac{\mathrm{m}}{\mathrm{s}}$	7.10	7.10	7.10	7.10	7.10
lane changes		354	337	337	353	353
cost	$\frac{1}{\mathrm{s}}$	5.81	12.40	7.49	6.27	6.16

Table 6.10: Closed-loop evaluation for scenario `DEU_Cologne-63_5_I-1` with dense and sparse traffic.

an open-loop cost. This favors a parallel architecture of an ensemble of predictions and soft-QP computations to choose the lowest-cost trajectory. In our experiments, adding NN to the ensemble improved the performance considerably and monotonously. This leads to the conclusion that NNs may be added as long as the computation time is below the planning threshold and as long as parallel resources are available. To further promote safety, an NLP, i.e., the FP, is used to plan a collision-free trajectory. The computational burden of the NLP is small compared to the expert MIP-DM since it optimizes the trajectory only locally and, therefore, omits the combinatorial variables.

Although we have evaluated the proposed approach for multi-lane traffic, the application to other urban driving scenarios is expected to perform similarly for a similar number of problem parameters due to the following. Many works, e.g., [140, 141, 213], use similar MIQP formulations for a variety of AD scenarios, including traffic lights, blocked lanes and merging. The formulations mainly differ in the specific environment. Many scenario specifics can be modeled by using obstacles and constraints related to the current lane [213], both considered in the presented approach. However, with an increasing number of problem parameters and rare events, the prediction of binary variables may become more challenging.

6.3.11 Appendix

In the following, we define the most important parameters used in the numerical simulations of this paper. For the closed-loop simulations in `SUMO`, we used the parameter values in Tab. 6.3.11. The reference velocity was set higher than the average nominal velocity to cause more overtaking maneuvers. For the optimization problems, we used the parameters shown in Tab. 6.3.11, in addition to the values presented in Tab. 6.7. In Tab. 6.3.11, neural network hyperparameters are shown for architectures FF, LSTM, EDS and REDS. The same LSTM layer is used for each set element (i.e., for each obstacle in our case) to achieve equivariance in the REDS network.

Category	Parameter	Value
General	episode length	30s
	nominal road velocity	$13.9\,\frac{\mathrm{m}}{\mathrm{s}}$
	maximum number of lanes	3
	lanes width d_{lane}	3.5m
	vehicle lengths	5.39m
	vehicle widths	2.07m
	ego desired velocity	$15\,\frac{\mathrm{m}}{\mathrm{s}}$
	maximum obstacle velocity	$23\,\frac{\mathrm{m}}{\mathrm{s}}$
	minimum obstacle velocity	$0.2\,\frac{\mathrm{m}}{\mathrm{s}}$
Dense scenario	traffic flow	$0.56\,\frac{\mathrm{vehicles}}{\mathrm{lane}\cdot\mathrm{s}}$
	traffic density	$0.04\,\frac{\mathrm{vehicles}}{\mathrm{lane}\cdot\mathrm{m}}$
Sparse scenario	traffic flow	$0.13\,\frac{\mathrm{vehicles}}{\mathrm{lane}\cdot\mathrm{s}}$
	traffic density	$0.01\,\frac{\mathrm{vehicles}}{\mathrm{lane}\cdot\mathrm{m}}$
FP	$\mathrm{diag}(Q)$	$[1,1,1,1]^\top$
	$\mathrm{diag}(R)$	$[1,1]^\top$
	slack weight w_{h}	10^6

Table 6.11: Parameters for closed-loop simulations in `SUMO`.

Category	Parameter	Value
expert MIP-DM	$\mathrm{diag}(Q)$	$[0,14,10,1]^\top$
	$\mathrm{diag}(R)$	$[4,0.5]^\top$
	lane change weight w_{lc}	$3\cdot 10^3$
	side preference weight w_{rght}	3
	safe distances $[\sigma_{\mathrm{f}},\sigma_{\mathrm{b}},\sigma_{\mathrm{l}},\sigma_{\mathrm{r}}]^\top$	$[0.5,12,0.5,0.5]^\top\mathrm{m}$
	minimum controls $\underline{u}$	$[-10,-5]^\top\,\frac{\mathrm{m}}{\mathrm{s}}$
	maximum controls $\overline{u}$	$[3,5]^\top\,\frac{\mathrm{m}}{\mathrm{s}}$
	velocity ratio constraint α	0.3

Table 6.12: Parameters in MIQP formulation (6.57) for expert MIP-DM.

Par.	Range	Par.	Range
lat. pos.	$[0,n_{\mathrm{lanes}}d_{\mathrm{lanes}}]-\frac{d_{\mathrm{lanes}}}{2}$	lanes	[1,3]
obs. lon. pos.	$[-120,200]\mathrm{m}$	lon. vel.	$[0,30]\,\frac{\mathrm{m}}{\mathrm{s}}$
lat. vel.	$[-1,1]\,\frac{\mathrm{m}}{\mathrm{s}}$	obstacles	[1,5]

Table 6.13: Ranges of uniform training data distributions.

Category	Parameter	Value
General	activation function	ReLU
	batch size	128
	step size	$5 \cdot 10^{-5}$
	epochs	1500
	optimizer	`adam`
	loss function	cross-entropy
	weight decay	10^{-5}
	training samples	10^5
	test samples	10^3
FF	hidden layers	7
	neurons per layer	128
LSTM	layers	2
	hidden network size	128
	input network layers	2
	output network layers	2
EDS	layers	7
	equivariant hidden size	64
	unstructured hidden size	64
	input network layers	2
	input network hidden size	128
	output network layers	2
	output network hidden size	128
REDS	layers	7
	hidden network sizes	64
	input network layers	1
	output network layers	1
	eq. LSTM output network layers	1
	unstr. LSTM output network layers	1

Table 6.14: Hyperparameters for the NN architectures.

6.4 Critical Discussion

Motion planning with multiple obstacles is demanding due to the high number of combinatorial choices related to the highly nonconvex planning space. Pure nonlinear optimization requires initial guesses to converge to acceptable local minima, which are generally unknown. Mixed-integer programming is a powerful approach for solving such problems to global optimality. However, mixed-integer solvers are often burdened by the computational complexity. In Sect. 6.1 and Sect. 6.2, efficient mixed-integer optimization-based problem formulations were proposed for particular motion planning domains. Sect. 6.1 and the related publication [225] focus on static obstacles, while Sect. 6.2 and the related publication [227] focus on structured highway driving. In Sect. 6.3 and the related publication [229], a combined machine learning and online optimization approach was presented that replaced the combinatorial part of a mixed-integer solver with a predictor trained on simulated highway driving data. The underlying objective was to achieve real-time feasibility under the highest possible closed-loop performance, which included prioritizing safety requirements.

The main contributions of Sect. 6.1 and Sect. 6.2 are novel formulations that significantly reduce the number of integer variables compared to other state-of-the-art formulations such as the formulations proposed in [181, 213]. While [213] requires $4NN_{\text{obs}}$ binary variables for collision-avoidance, where N is the discrete-time horizon length and N_{obs} is the number of considered obstacles, the approach in Sect 6.1 requires only N_{obs} binary variables, independent of the horizon length. The low number of binary variables makes the approach applicable for scenarios with static obstacles that are sophisticated to traverse. An open-source MILP solver acquired an average online computation time of 1.9 seconds on the NVIDIA DRIVE PX2 embedded hardware. The proposed homotopy within SQP iterations was considerably faster than the MILP solver with 72 ms total online computation time per iteration on the embedded hardware.

The long-horizon formulation in Sect. 6.2 requires binary variables only in the order of N_{obs}. This reduction of binary variables decreases the computation time 2 to 100 times, compared to alternative planners of [213] and [8], which makes the approach real-time applicable with the requested planner iteration time. A novel integrated long-horizon planner was proposed, where the computational complexity is independent of the horizon length. This long-term prediction decreased the closed-loop cost up to 10% in highway traffic simulations involving the proposed planner, a low-level controller, and interactive agents.

Sect. 6.3 instead contributes to mixed-integer-based planners following the paradigm of replacing the combinatorial part with machine learning predictors [39]. Moreover, the section utilizes powerful deep neural network architectures [312] that capture relevant symmetries of the motion planning domain. The novel neural network architecture increased the accuracy of binary variable predictions compared to the LSTM architecture of [62] from 15% to 48% for a horizon of 28 steps and seven simultaneously present obstacles. Additionally, the feasibility of the consecutive QP resembling the expert MIQP, where the binary variables were fixed, was increased from 50% to 76% for the same setting. In addition to the performance increase, the novel neural network architecture provided the equivariance to obstacle permutations, an exceptional generalization capability to an unseen number of obstacles, and the possibility to change the horizon length or the number of obstacles posterior to the training. By utilizing parallel ensembles of neural networks and QP solvers, the share of infeasible problems could be decreased from 15% to 2%. After adding a novel feasibility projector, formulated as an NLP, the share of infeasible solutions is nearly 0%. After adding all of the novel safety-relevant modules, the worst-case online computation time could be reduced by up to two orders of magnitude, i.e., from 4000 ms to 60 ms and from 3000 ms to 30 ms for two different randomly simulated highway traffic environments. The closed-loop cost is slightly increased compared to an expert MIQP solver by 1.5% and 6.4%, without sacrificing safety.

The applicability of the algorithms of Sect. 6.1 and Sect. 6.2 is specific to certain environments, i.e., static obstacles and rewards or highway driving, respectively. While the approach of Sect. 6.3 was also applied to highway driving, this technique is general enough to be applied to comparable urban motion planning problems.

A primary advantage of mixed-integer-based solvers compared to graph-based planners, such as planners surveyed in [199, 159], is their appealing way of solving the continuous problem parts by derivative-based optimization and the combinatorial part by graph-search techniques, cf., Sect. 2.1.3, without discretizing the continuous variables. Using derivatives to solve continuous optimization problems is particularly efficient for high-dimensional state spaces, where the curse of dimensionality limits discretization-based approaches. Nonetheless, solving mixed-integer problems has some disadvantages, which legitimizes alternative approaches.

A major drawback of the methods in this thesis is the requirement of MIQP formulations. MIQPs require less expressive linear models and linear or possibly convex quadratic constraints. MIQPs can be solved efficiently by specialized state-of-the-art solvers [114]. However, the online computation times are already close to real-time feasibility. Thus, it is assumed that using more expressive formulations and related solvers, e.g., mixed-integer nonlinear programmings (MINLPs), would violate the real-time feasibility excessively. Nonetheless, no MINLP benchmarking results for motion planning are known to the author of this thesis, leaving the potential for future research. Notably, the combinatorial variables learning approach of Sect 6.3 could be directly applied to learning binary variables of MINLPs in future work. An intrinsic challenge with replacing the MIQP formulation of Sect 6.3 with a more expressive MINLP formulation would be the vastly increased time to create machine learning training data.

When following the paradigm of constraining the motion planning problem formulation to MIQPs, which is motivated by the outstanding high-performance solvers of this problem class, further modeling details could be added by introducing additional integer variables to model nonconvexities or nonlinearities in the model equations, such as shown in [86].

The author believes mixed-integer formulations are the most natural way of formulating the inherently continuous but highly nonconvex planning problems arising in automotive applications. This research provides algorithms tested on embedded hardware or extensive simulations to illustrate their potential for future planning systems. Presumably, on the long way to significantly advance autonomous driving, some fundamental work is still required for the proposed algorithms. For instance, commercial high-performance embedded MIQP solvers are unavailable. Moreover, dedicated hardware to solve such problems could increase the performance considerably.

Chapter 7

Collision Avoidance for Autonomous Racing

This chapter considers two challenges for collision avoidance in autonomous racing. The knowledge of the opponents' racing intention is exploited to predict their trajectory. The prediction is then used to formulate the collision avoidance constraints in the ego optimization problem. It is reasonable to assume that opponents maximize their progress similarly to the ego racing objective. Even if the opponent's vehicle model is assumed to be known, the exact weighting and acceleration limits it will allow may vary based on the driving style. In Sect. 7.1, a novel approach estimates the weighting and constraint parameters, including the assumption that the opponent optimizes a particular objective. A parameterized model predictive control (MPC) optimization problem is used to predict the opponents' trajectory and is referred to as a low-level program in this context. In order to estimate its' parameters, a moving horizon estimator utilizing observed data is proposed, which involves the optimality conditions of the low-level program as constraints.

Besides predicting opponents in autonomous racing, a further challenge is formulating an optimization problem that yields strategic driving policies. Sect. 7.2 uses the technique of reinforcement learning (RL) to develop strategic behavior in simulations by only specifying the racing goal as the overall rank among several competitors. Since RL requires a large number of samples and often struggles to provide safety guarantees, the RL policy is not used directly to specify controls but rather parameterizes the objective function of an MPC. The MPC uses a simple model, obstacle predictions, and constraints to provide the required safety guarantees. The optimization layer is used during offline learning and within the online policy. The mapping from the RL parameters via the MPC to actual actions makes the learning more sample efficient.

7.1 An Inverse Optimal Control Approach for Trajectory Prediction of Autonomous Race Cars

---

In this section, the paper published in [226] is reprinted with permission of Florian Messerer, Markus Schratter, Daniel Watzenig and Moritz Diehl. Note that the formatting of some formulas, terms, and numbers has been slightly adjusted for consistency without changing their meaning or content.

The contributions of each author are listed in the following.

Rudolf Reiter:	*Idea, programming of the published algorithm and the interface to the overall system, programming on the overall system (various program modules of the autonomous driving stack), design of the experiments, writing of the document*
Florian Messerer:	*Mathematical corrections, stylistic corrections, linguistic improvements*
Markus Schratter:	*Programming of the overall system (embedded system, Autonomous Racing Graz software stack)*
Daniel Watzenig:	*head of the "Autonomous Racing Graz" team. Design of the overall system. Operational management and organization of the competitions.*
Moritz Diehl:	*Mathematical corrections, stylistic corrections, linguistic improvements*

©2022 European Control Association. DOI: 10.23919/ECC55457.2022.9838100.

Abstract. This paper proposes an optimization-based approach to predict trajectories of autonomous race cars. We assume that the observed trajectory is the result of an optimization problem that trades off path progress against acceleration and jerk smoothness and is restricted by constraints. The algorithm predicts a trajectory by solving a parameterized nonlinear program (NLP), which contains path progress and smoothness in cost terms. By observing the actual motion of a vehicle, the parameters of prediction are updated by means of solving an inverse optimal control problem that contains the parameters of the predicting NLP as optimization variables. The algorithm, therefore, learns to predict the observed vehicle trajectory in a least-squares relation to measurement data and to the presumed structure of the predicting NLP. This work contributes with an algorithm that allows for accurate and interpretable predictions with sparse data. The algorithm is implemented on embedded hardware in an autonomous

real-world race car that is competing in the challenge `Roborace` and analyzed with respect to recorded data.

7.1.1 Introduction

In real-world autonomous driving scenarios, a core challenge is the prediction of other agents in the environment. The prediction algorithms differ in relation to the scenario and to the availability of data. For instance, in urban driving, a large amount of data might be available due to the massive data collection of the vehicle industry. For autonomous racing tasks, there is a lack of extensive data sets, thus supervised learning of data-driven predictions is not feasible. In our research, we focus on a racing setting related to a competition called `Roborace`. As part of this racing series, the participating teams develop software for the fully autonomous operation of electric race cars and are confronted with increasingly demanding challenges from one event to the other. Whereas the ego vehicle moves on a real racetrack, the state observations of the (currently) purely virtual opponent race cars are provided by the mixed-reality simulator to the car's software about 200 meters in advance. The up to six virtually present opponent cars are set up by the organizers with different racing algorithms that are supposed to race with different performance and driving styles. The opponent cars are currently not considered as strategic decision makers, i.e., they are not performing game theoretic actions such as blocking. Thus, the race cars can be seen as non-interactive agents. Generally, there is no a priori knowledge available about the opponents, except for their racing intention. Therefore, it is impossible to use an a priori fully parameterized vehicle model as a basis for prediction. Furthermore, extensive system identification is impossible due to the short time it takes for the vehicle to be observed before it needs to be overtaken. The goal of this paper is to present a method that predicts the behavior of other race cars even with sparse data. A typical scenario is shown in Fig. 7.1, where the ego race car and three other opponent cars are on a racetrack, and trajectories of our presented predictor are shown with bars corresponding to the predicted velocity.

Our work starts with framing the basic and limited knowledge about the opponents as a sparsely parameterized predictor whose parameters can be estimated by a limited amount of data. Since it is known that the intention of the other opponents is time-optimal driving, the predictor is stated as a parameterized optimization problem for progress maximization, referred to as low-level nonlinear program (LLNLP), which is assumed to be solved by the other agent. The estimation of the parameters is performed by solving an inverse optimal control (IOC) problem, which enforces the optimality conditions for the LLNLP as constraints and performs least-squares optimization on the deviation of the resulting LLNLP-trajectory to the collected observed data of the particular opponent vehicle. This results in a set of parameters for the LLNLP,

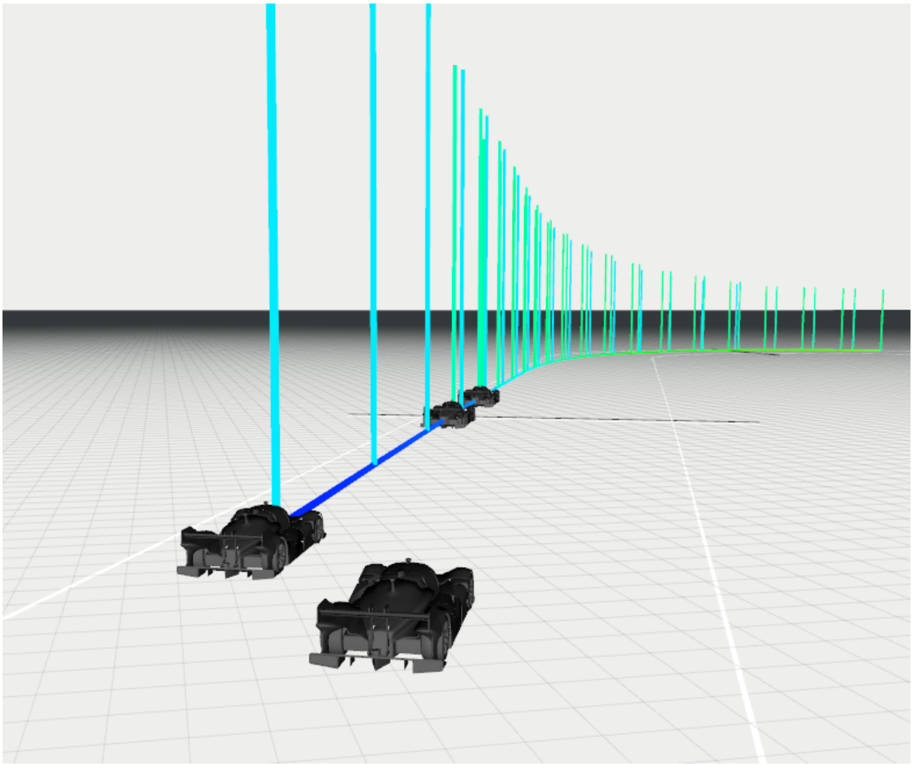

Figure 7.1: Presented trajectory prediction in a simulation. The height and color of the bars correspond to the predicted speed.

which are locally optimal with respect to the chosen structure of the LLNLP and the observed data. In fact, the chosen formulation only finds a stationary point as opposed to an optimal point and is dependent on the initialization due to its non-convex structure, but in practice, both were observed to not have a significant influence on the performance. The LLNLP is solved in real-time for each opponent, starting with an initial set of parameters, which are updated as soon as enough data is available.

The LLNLP is used to predict the velocity along a curve, which is obtained by blending the current motion into a previously computed minimum curvature path. The parameters related to LLNLP are the constraint limits and the square penalties on the input (jerk) and the acceleration states. The estimation of the acceleration constraints is separated from the bi-level optimization problem into a separate constraint estimation QP (CQP) whose constraint estimates update both the bi-level program for the weight parameter estimation and the final LLNLP for predicting the opponent trajectories in real-time.

The performance of the described algorithm is shown with recorded data from differently driving opponent race cars in a *Hardware-In-The-Loop* setting.

Related Work

Trajectory prediction in the domain of autonomous vehicles is dominated by data-driven approaches, which are based on regression and pattern matching. This is applicable if the availability of sufficient data related to human driven vehicles on public streets is given. If interaction and sequential decision making are considered, often inverse optimal control (IOC) or inverse reinforcement learning (IRL) are used. Often deep neural networks (DNNs) are used as function approximators [162], and the time dependency suggests the use of recurrent neural architectures as seen in [61, 314, 127]. Also, various other DNN architectures are used, such as convolutional neural networks [193]. If statistically qualitative data is available, these approaches work well, and even their application to real-time systems as trained networks is favorable due to the high evaluation speed of DNNs. Using an optimization problem as a function approximator or even within a neural network is a field with many related research areas, ranging from reinforcement learning with an embedded MPC structure [110] to generic optimization layers [7]. Using bi-level optimization to estimate the parameters of a low-level problem is used more rarely. Related to vehicle predictions, it was used in a similar approach, which focuses on urban driving scenarios and the game theoretic interaction between agents [68, 160]. Furthermore, for robotic predictions [177] or even human motion predictions [182], a bi-level problem was used. For unconstrained linear systems, [178], the authors show that the IOC can even be stated as a convex semidefinite program. A detailed survey of vehicle prediction approaches is given in [162], although IOC appears only in the context of IRL. A general survey on bi-level optimization is given in [260], which mentions the presented approach of solving the lower-level program by restricting it to a stationary point, especially for convex problems.

Contribution

In the domain of autonomous racing, to the best knowledge of the authors, this work is the first that uses bi-level optimization together with the LLNLP for real-time trajectory prediction. Since bi-level problems are hard to solve, this work also addresses novel techniques that can be used in challenging real-world conditions such as racing. This paper follows previous work for solving motion planning problems for autonomous racing [225, 222].

7.1.2 Prediction Architecture

In Fig. 7.2, the architecture of the proposed algorithm is shown. The algorithm consists of an offline and an online part. The precomputations in the offline part account for the optimal racing path along the known racetrack. The online part is constructed for each observed opponent vehicle and is split into a slower (0.5 Hz) estimation part and a faster (10 Hz) prediction part. In the path prediction (PP), a curve is blended from the current opponents vehicle position to the precomputed racing line. The main prediction component is the LLNLP, which computes the trajectory with respect to the parameterized constraints and the parameterized weights, starting at the observed current opponent value. The constraint estimator (CQP) passes its estimated constraint parameters to the high-level NLP (HLNLP), and both the CQP and the high-level nonlinear program (HLNLP) estimate the parameters of the LLNLP. The online part is executed for each of the M observed vehicles.

7.1.3 Prediction Algorithm

In the following, the prediction algorithm is described by each component. In Sections 7.1.3 to 7.1.3, the main blocks of Fig. 7.2 are described, and in the final part, the pseudocode (2) are stated.

Path Prediction (PP)

Given the racetrack layout, a time-optimal path $p_{\text{topt}}(s)$ is computed by curvature minimization related to [222]. The path variable s is related to the position on a reference center track line. Given the current opponent vehicle state, a linear extended constant motion path $p_{\text{c}}(s)$ is blended into the precomputed path for $s < s_f$ with

$$p_{\text{p}}(s) = \frac{s}{s_f} p_{\text{topt}}(s) + \frac{s - s_f}{s_f} p_{\text{c}}(s). \tag{7.1}$$

For $s \geq s_f$, the prediction path is set equal to the racing path.

Low-level Program for the Trajectory Prediction (LLNLP)

The path predictor predicts the curve that is described by its path length s and the associated curvature $\kappa(s) = \frac{d\phi}{ds}$. The curvature is described by a piece-wise

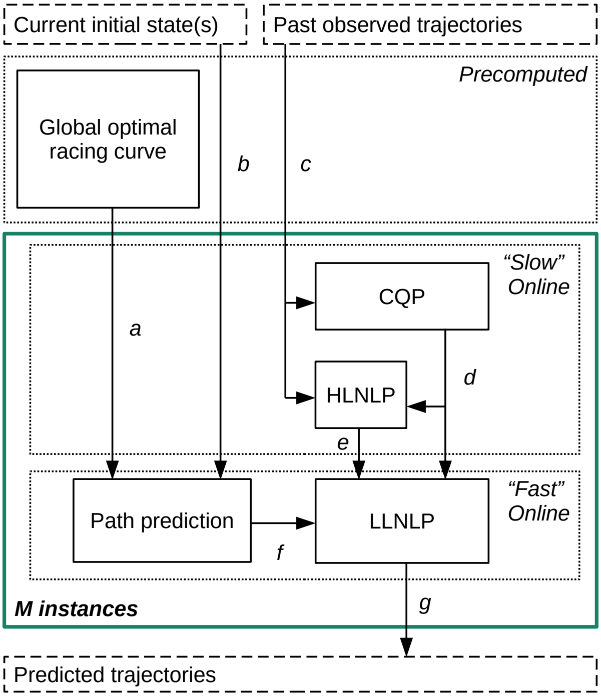

Figure 7.2: Algorithm architecture. (a: global racing path, b: initial state $\bar{x}_0$, c: trajectory data samples, d: constraints a_{max}, e: weights w, f: Cartesian coordinates and curvature parameters of blended path segment $\bar{\kappa}$, g: predicted trajectory)

linear polynomial and parameterized to interpolate N_κ precomputed values κ_i for the curvature along the path segment. The values are summarized as $\bar{\kappa} = \begin{bmatrix} \kappa_i & \dots & \kappa_{N_\kappa - 1} \end{bmatrix}$. Details on the computation can be found in [222]. Note that the *linear* interpolation leads to discontinuous derivatives in the inequality constraints and violates the condition of twice continuously differentiable functions required for second-order NLP algorithms. Nevertheless, we empirically found a speedup of a factor of 100 to 1000 compared to *bsplines* as interpolating polynomials, with practically no convergence problems.

The LLNLP predicts the estimated velocity along this curve by solving an optimal control problem which consists of a linear discrete model $F(x_k, u_k, \Delta t)$, acceleration constraints $h_a(x_k, \bar{\kappa}, a_{\text{max}})$ and state constraints $\underline{x}$ and $\overline{x}$. Since the path is given, the predicted motion along the curve is described by means of three chained integrators, where the input u is the jerk. The state vector consequently consists of the path progress s, the velocity v and the acceleration a with $x = \begin{bmatrix} s & v & a \end{bmatrix}^\top \in \mathbb{R}^3$. Since the integrator chain is a linear system, the discretization (zero-order-hold controls) of the dynamics can be computed exactly by matrix exponentials and leads to the affine function $F(x_k, u_k, \Delta t) = A(\Delta t)x_k + B(\Delta t)u_k$. The only constraint captured in the box constraints $\underline{x} \leq x_k \leq \overline{x}$ is the limitation of the speed v to $v_{\max}$ and to positive values. The optimal control problem is discretized in $N-1$ intervals using discrete multiple shooting and solved by sequential quadratic programming using the real-time NMPC solver `acados` [291]. To account for the progress maximizing requirement for the resulting trajectory, a linear negative cost $q_n = \begin{bmatrix} -1 & 0 & 0 \end{bmatrix}^\top$ for the last discrete position is used. The matrix $W = \operatorname{diag}(\begin{bmatrix} 0 & 0 & w_{\text{acc}} \end{bmatrix})$ and the scalar $R = w_{\text{jerk}}$ are the weights that describe the motion of the predicted opponent vehicle in the presented structure, if no constraints are active. Finding the values of w_{acc} and w_{jerk} is the objective of the HLNLP component. Slack variables $s_{\text{LL}} = \begin{bmatrix} s_{\text{LL},0}, & \ldots, & s_{\text{LL},N} \end{bmatrix} \in \mathbb{R}^{8\times N}$ with weights α_1, α_2 are added for the online forward implementation to account for the robustness of the SQP algorithm. We can then state the lower-level problem $P_{\text{LL}}(w, \bar{x}_0, \bar{\kappa}, a_{\max})$ as

$$\begin{aligned}
\min_{\substack{x_0,\ldots,x_N,\\ U_0,\ldots,U_{N-1},\\ s_0,\ldots,s_N}} \quad & \sum_{k=0}^{N-1} \|x_k\|_{2,W}^2 + \|U_k\|_{2,R}^2 + q_N^\top x_N + \sum_{k=0}^{N} \alpha_1 \mathbf{1}^\top s_{\text{LL},k} + \alpha_2 \|s_{\text{LL},k}\|_2^2 \\
\text{s.t.} \quad & x_0 = \bar{x}_0, \\
& x_{k+1} = F(x_k, U_k, \Delta t), \qquad k = 0, \ldots, N-1, \\
& \underline{x} \leq x_k \leq \overline{x}, \\
& 0 \leq h_a(x_k, \bar{\kappa} a_{\max}) + s_{\text{LL},k}, \\
& 0 \leq s_{\text{LL},k}, \qquad k = 0, \ldots, N,
\end{aligned} \tag{7.2}$$

where $\mathbf{1}$ is a vector of all 1's of appropriate size. The acceleration constraints $h_a(x_k, \bar{\kappa}, a_{\max})$ approximate the friction, throttle, and breaking boundaries of the vehicle by means of a polytope in the space of the two-dimensional acceleration vector $a(x_k, \bar{\kappa}) = \begin{bmatrix} a_{\text{lat}}(x_k, \bar{\kappa}) & a_{\text{lon}}(x_k) \end{bmatrix}$ which are often related to the “Kamm's circle”. The polytope is typically symmetric to the longitudinal axis. It is chosen such that it consists of box constraints along the axes and diagonal constraints that are parallel to the lines described by the connection of the axis-aligned maximum values. Consequently, the diagonal constraints depend on the values of the axis-aligned constraints. The presented approach computes the axis aligned constraints first and uses those values as inputs to the diagonal constraints. An example of the fitted acceleration constraints can be seen in Fig. 7.3. Therefore, 8 linear constraints arise,

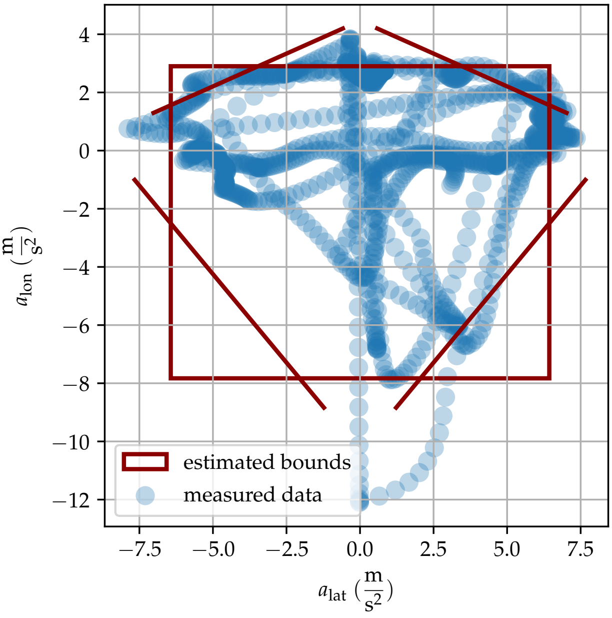

Figure 7.3: Acceleration constraint estimation. In total, eight constraints are fitted as a convex polytope to measurement data.

where 6 of them are pair-wise symmetric. The only non convexity in (7.2) emerges from the dependency of $a_{\text{lat}}(x_k) = -v_k^2\kappa(s_k, \bar{\kappa})$. The acceleration constraints $h_a(x_k, \bar{\kappa}, a_{\max})$ can be stated as

$$h_a(x_k, \bar{\kappa}, a_{\max}) = a_{\max} - \operatorname{diag}(d_{\text{len}}) D a(x_k, \bar{\kappa}), \tag{7.3a}$$

$$\bar{a} = \sqrt{a_{\text{lat,max}}^2 + a_{\text{lon,max}}^2}, \tag{7.3b}$$

$$d_{\text{len}}^\top = \begin{bmatrix} 1 & 1 & 1 & 1 & \bar{a} & \bar{a} & \bar{a} & \bar{a} \end{bmatrix}, \tag{7.3c}$$

$$D = \begin{bmatrix} 1 & 0 \\ -1 & 0 \\ 0 & -1 \\ 0 & -1 \\ a_{\text{lon,max}} & a_{\text{lat,max}} \\ -a_{\text{lon,max}} & a_{\text{lat,max}} \\ a_{\text{lon,min}} & -a_{\text{lat,max}} \\ -a_{\text{lon,min}} & -a_{\text{lat,max}} \end{bmatrix}, \quad a_{\max} = \begin{bmatrix} a_{\text{lat,max}} \\ a_{\text{lat,max}} \\ a_{\text{lon,max}} \\ a_{\text{lon,min}} \\ a_{\text{q,north}} \\ a_{\text{q,north}} \\ a_{\text{q,south}} \\ a_{\text{q,south}} \end{bmatrix}. \tag{7.3d}$$

The term $\operatorname{diag}(d_{\text{len}})D$ collects the row vectors with unit length that represent direction vectors that are used to project the acceleration vector and to constrain to the consecutive projected scalar value. Obviously, the second part of the vector d_{len} and the lower part of the matrix D shows the dependency on the axis aligned constraints. The maximum acceleration values are collected in the vector $a_{\max}$, the initial observed state $\bar{x}_0$, together with the weights $w^\top = \begin{bmatrix} w_{\text{jerk}} & w_{\text{acc}} \end{bmatrix}$ are the input parameters of the LLNLP. The parameters that are not subject to be changed by the estimation within the HLNLP are summarized as $p^\top = \begin{bmatrix} \bar{x}_0 & a_{\max}^\top & \bar{\kappa} \end{bmatrix}$.

Quadratic Program for Constraint Estimation (CQP)

In order to remove computational complexity from the HLNLP, the constraint estimation was separated. Even with fixed constraints, the structure of the HLNLP is highly non-convex and challenging to solve. By means of a Kalman-filter-based vehicle state estimation, the observed accelerations a_i are computed and stored as a data set of N_a data samples in $\mathbb{R}^2$. Those acceleration data are used to fit linear constraints. The projection $e_k^\top D a_i$ for the estimation of the linear constraint c_k is performed for the 8 constraints. The vector e_k represents the k-th unit vector in $\mathbb{R}^8$. Since the measured data is noisy, a robust estimation of the constraints that account for outliers is required. This is achieved by the quadratic program (7.4), where the estimated constraint violation is penalized linearly and realized by means of a hinge loss $h(x) = \max(0, x)$. This function adds no costs if the measured value is lower than the constraint and penalizes linearly otherwise. The deflection of the constraint is penalized quadratically with a weight ω and a minimum at the prior estimated value $\hat{c}_k$. Since the prior estimated value acts as a lower bound and would not decrease during iterations, the value is lowered by a factor $r < 1$ in each iteration for the axis aligned constraints with $\hat{c}_k \leftarrow r\hat{c}_k$. For the diagonal constraints, $\hat{c}_k$ is chosen as the distance of the diagonal line of the origin. The problem formulation for estimating constraint k in the bounding polytope with 8 linear constraints is stated as

$$\min_{c_k \in \mathbb{R}} \quad \frac{1}{N_a} \sum_{i=0}^{N_a - 1} \max(0, e_k^\top D a_i - c_k) + \omega (c_k - \hat{c}_k)^2. \tag{7.4}$$

The problem can be formulated into a smooth quadratic program using slack variables ζ for implementing the hinge function, which leads to the formulation

$$\begin{aligned} \min_{\substack{c_k \in \mathbb{R}, \\ \zeta \in \mathbb{R}^{N_a}}} \quad & \omega (c_k - \hat{c}_k)^2 + \frac{1}{N_a} \sum_{i=0}^{N_a - 1} \zeta_i \\ \text{s.t.} \quad & 0 \leq \zeta_i, \quad e_k^\top D a_i - c_k \leq \zeta_i, \quad i = 0, \dots, N_a - 1. \end{aligned} \tag{7.5}$$

The solutions of the CQP are directly used as acceleration constraints $a_{\max}$ in (7.3d).

Bi-level Program for the LLNLP Parameter Estimation (HLNLP)

The HLNLP fits the LLNLP with the parameters derived from the CQP to observed measurement data $\bar{x}$ with a least-squares error measure. The observed trajectory might differ at the last points from the predicted trajectory, even if the true parameters were used, since the controller of the observed vehicle most likely had adapted to even further distant constraints like a sharp curve. To account for this structural uncertainty, the weight matrix Q_k is linearly reduced to zero for the final N_R points. Problem (7.6) shows the basic structure of the problem. The optimization variables are the estimated trajectory x of the LLNLP and the weighting parameters w. To account for the iterative estimation of the parameter w, the previously estimated parameter $\hat{w}$ together with the associated weight matrix P is used as an arrival cost, as shown with MHE in [217]. To simplify the algorithm, the weight matrix is set constant. The basic structure of the problem can be written as

$$\begin{aligned} \min_{x, U, w} \quad & \sum_{k=1}^{N_T-1} \|x_k - \bar{x}_k\|_{2,Q_k}^2 + \|w - \hat{w}\|_{2,P^{-1}}^2 \\ \text{s.t.} \quad & x, u \in \operatorname{argmin} P_{\mathrm{LL}}(w, \bar{x}_0, \bar{\kappa}, a_{\max}), \quad w \geq 0 \end{aligned} \tag{7.6}$$

where $x \in \mathbb{R}^{N_x \times N_T}, U \in \mathbb{R}^{N_u \times (N_T-1)}$ and $w \in \mathbb{R}^2$. The optimization variables are the estimated trajectory x of the LLNLP and the weighting parameters w.

The low-level program $P_{\mathrm{LL}}(w, \bar{x}_0, \bar{\kappa}, a_{\max})$ in (7.2) can be written as

$$\begin{aligned} \min_{z \in \mathbb{R}^{N_z}} \quad & f_{\mathrm{LL}}(z, w) \\ \text{s.t.} \quad & g_{\mathrm{LL}}(z) = 0, \quad h_{\mathrm{LL}}(z, \bar{x}_0, \bar{\kappa}, a_{\max}) \geq 0 \end{aligned} \tag{7.7}$$

with $z = \begin{bmatrix} \operatorname{vec}(x)^\top & \operatorname{vec}(u)^\top \end{bmatrix}^\top$ and $N_z = N_x N_T + N_u(N_T - 1)$. The domains and co-domains of the functions are $f_{\mathrm{LL}} : \mathbb{R}^{N_z \times N_w} \to \mathbb{R}$, $g_{\mathrm{LL}} : \mathbb{R}^{N_z} \to \mathbb{R}^{N_T N_x}$ and $h_{\mathrm{LL}} : \mathbb{R}^{N_z} \to \mathbb{R}^{N_T N_{h,\mathrm{LL}}}$, where $N_{h,\mathrm{LL}} = 10$ in this case, with two bounds on the velocity state and 8 acceleration constraints. The number of weights corresponding to the smoothness is $N_w = 2$. The constraints $a_{\max}$ are parameters and updated by means of the estimation of the CQP.

To solve the problem, the bi-level problem can be formulated as an NLP, which is summarized as

$$\min_{\substack{x,U,w,\\ \tau,\lambda,\mu,s}} \quad \sum_{k=0}^{N_T-1} \|x_k - \bar{x}_k\|_{2,Q_k}^2 + q_\tau \tau + \beta_1 \mathbf{1}^\top s + \beta_2 \|s\|_2^2 + \|w - \hat{w}\|_{2,P^{-1}}^2 \tag{7.8a}$$

$$\text{s.t.} \quad 0 = \nabla_z f(z,w) - \nabla_z g_{\mathrm{LL}}(z)\lambda - \nabla_z h_{\mathrm{LL}}(z,p)\mu, \tag{7.8b}$$

$$0 \leq w, \tag{7.8c}$$

$$0 = g_{\mathrm{LL}}(z), \tag{7.8d}$$

$$0 \leq \tau, \tag{7.8e}$$

$$0 \leq \mu, \tag{7.8f}$$

$$0 \leq h_{\mathrm{LL}}(z,p) + s, \tag{7.8g}$$

$$\tau \geq \mu_i h_{\mathrm{LL},i}(z,p), \quad i = 0,\dots,N_{h,\mathrm{LL}} - 1, \tag{7.8h}$$

$$s \geq 0, \tag{7.8i}$$

where $x \in \mathbb{R}^{N_x \times N_T}, U \in \mathbb{R}^{N_u \times (N_T-1)}, w \in \mathbb{R}^2, s \in \mathbb{R}^{N_T N_{h,\mathrm{LL}}}, \tau \in \mathbb{R}, \lambda \in \mathbb{R}^{N_T N_x}$, and $\mu \in \mathbb{R}^{N_T N_{h,\mathrm{LL}}}$.

We enforce a stationary point in the LLNLP as a constraint in the high-level problem by enforcing the KKT conditions by means of constraints which are stated in (7.8b-7.8h). For this aim, additional optimization variables arise that are the dual variables λ and μ. A major challenge here is to account for the highly non-convex complementarity conditions arising from the inequalities of the LLNLP. Therefore, a relaxed problem is stated within the constraints, which is lower bounded by the actual complementarity condition and upper bounded by its relaxed version related to the interior point approach as seen in (7.8e-7.8h). If the complimentarity is relaxed too much, the estimation of the weight parameters can become wrong. Consequently, the relaxing parameter $\tau \in \mathbb{R}$ is also integrated as an optimization variable into the HLNLP and initialized with a "high" value (e.g., 1.0). A high value for q_τ together with the linear penalty of τ is used to achieve the exact complementarity constraints. Slack variables s account for infeasibilities.

The number of primal variables in the high-level program, which are $N_{\mathrm{var,HL}} = 2N_xN_T + N_u(N_x - 1) + 2N_hN_T$ rises notably compared to the low-level program, which is $N_{\mathrm{var,LL}} = N_xN_T + N_u(N_x - 1)$, but is of the same complexity w.r.t. N_x, N_T and N_h. This program is solved using the interior point solver `IPOPT` [307] formulated in `CasADi` [14], which again uses a relaxation of the problem in order to account for the inequality constraints. By using the presented formulation, we can explicitly account for the accuracy of the complementarity constraint in the stationary point of the low-level program. Note that the

Hessian of the HLNLP actually contains third-order derivatives of the original LLNLP, thus posing the condition of three times differentiable smooth functions in the LLNLP.

Algorithm

Algorithm (2) describes the sequential interaction of the components with respect to the architecture in Fig. 7.2. The solvers CQP and HLNLP are executed as threads that update the estimation values in a lower frequency than the main predicting solver LLNLP, together with the path prediction PP.

Algorithm 2: IOC Prediction

```
input  : Initial weights and constraints $\hat{w}$, $c_k$,
         Observed state measurements $\bar{x}_0$
output : Predicted trajectory $x_{\text{pred}}$
HLNLPsolved←True;
CQPsolved←True;
while True do
    if CQPsolved then
        CQPsolved←False;
        $\bar{x}$ ←last $N_a$ state samples;
        $\bar{\kappa}$ ←curv($\bar{x}$);
        $\hat{c}_k \leftarrow rc_k \quad k = 0, \dots 3$;
        $\hat{c}_k \leftarrow$ dist($\hat{c}$)  $k = 4, \dots 7$;
        Set CQP parameters $\hat{c_k}$, $\omega$, $a(\bar{x}, \bar{\kappa})$;
        Start CQP solver (Updates: CQPsolved, $c_k$);
    end
    if HLNLPsolved then
        HLNLPsolved←False;
        $\bar{x}$ ← last $N_T$ state samples;
        $\bar{\kappa}$ ← curv($\bar{x}$);
        $\hat{w} \leftarrow w$;
        $a_{\text{lat},k} \leftarrow c_k \quad k = 0, \dots 7$;
        Set HLNLP parameters $a_{\text{lat}}$, $\bar{x}$, $\bar{\kappa}$, $\hat{w}$;
        Start HLNLP solver (Updates: HLNLPsolved, $w$);
    end
    $\bar{x}_0$ ← State measurement input;
    $a_{\text{lat},k} \leftarrow c_k \quad k = 0, \dots 7$;
    $\bar{\kappa} \leftarrow PP(\bar{x}_0)$;
    $x_{\text{pred}}$ ←solve LLNLP($\bar{x}_0, \bar{\kappa}, w, a_{\text{lat}}$);
end
```

Table 7.1: Parameter Settings

Parameter	Value	Parameter	Value
$c_{k,0} \dots c_{k,3}$	5, 5, 2.5, -5 m/s	s_f	300m
w_0^T	[0.5 0.2]	N	111
ω	12.5	ΔT (LL)	0.1s
N_a	10^3	ΔT (HL)	1s
N_κ	700	α_1, α_2	10^4, 10^8
N_T	25s	β_1, β_2	10^5, 10^6
P	$\mathrm{diag}([2 \cdot 10^{-7}\ 9 \cdot 10^7])$	q_τ	10^7

7.1.4 Results

The algorithm was tested with recorded data. Qualitatively, these tests fully describe the performance of the algorithm. Nevertheless, the embedded performance, especially the real-time performance of the LLNLP was proven in several real racing events. This shows that the proposed algorithm can work in embedded real-world systems.

Hardware and Software Setup

The proposed LLNLP was tested on race car hardware (Section 7.1.4), including the NVIDIA DrivePX 2 in a Docker environment with Ubuntu 20.04. This electronic control unit (ECU) provides two CPUs (4x ARM Denver, 8x ARM Cortex A57) and two GPUs (2x Tegra X2, 2x Pascal GPU). The open-source `OSQP` solver [268] was used in a mixed Python/C++ `ROS`-framework for solving the problem (7.1.3) using `CasADi` [14] as an interface. `CasADi` was also used as an interface together with `IPOPT` [307] to solve the HLNLP of Section 7.1.3. The time-critical real-time estimation related to the LLNLP (Section 7.1.3) was performed using `acados` [291] as an NLP solver. For each opponent car, a separate solver was created, which was executed as a thread, updating a data structure that contained the most recent prediction. The full algorithm was tested offline (Section 7.1.4) in simulations with an Alienware m-15 Notebook and an Intel Core i7-8550 CPU (1.8 GHz). The parameters used for the evaluation are shown in Table 7.1. In Table 7.3, the time statistics of the different optimization parts are shown, and in Table 7.2, the relevant settings are given. Notably, the LLNLP was failing in 2 out of 1000 randomly parameterized test runs, which was due to the linear interpolation of the curvature as described in Section 7.1.3. This failure rate is outweighed in practice by the enormous speed gain of a linear interpolation.

Table 7.2: Component settings

Component	Samples/Nodes	Notes	Runs
PP	150		1e3
CQP	500	Eval. per constraint (1/5)	1e2
HLNLP	35	Δt=1s	50
LLNLP	60	Δt=0.1s	1e3

Table 7.3: Solver timing statistics

Component	Solver	t_{max} (ms)	t_{ave} (ms)	fail rate (%)
PP	none	< 1	< 1	0
CQP	`OSQP`	15.5	8.1	0
HLNLP	`IPOPT`	6237	520	5
LLNLP	`acados` `HPIPM`	2748	91	0.2

Performance Analysis with Recorded Data

Validation of the CQP. Fig. 7.3 shows the estimation of constraints related to 1000 recorded acceleration data samples. The acceleration was computed out of the observed and estimated trajectory state. Obviously, the constraints of any observed race car acceleration data could be of any shape, but the representational capacity of the constraint assumptions have to be traded off for fast and reliable real-time execution in the LLNLP. According to our experience, the approximation with either 4 (box only) or 8 (adding diagonals) constraints achieved the best performance.

Validation of the velocity profile estimation. Using the same recorded real-world trajectory as in 7.1.4 and also its estimated constraints $a_{\max}$ as seen in Fig. 7.3, we use the HLNLP to estimate the parameters w. All estimated parameters together are then forwarded to the LLNLP, which predicts the velocity and the progress along the given curve by solving the nonlinear program. The results are compared to the standard constant velocity predictor, which is often used in robotic applications [254], and that assumes a vehicle progression with the measured constant velocity. In Fig. 7.4, the mean position error of the presented algorithm $\bar{e}_s$ is compared to the mean position error of the constant velocity predictor $\bar{e}_{s,\text{const}}$. Furthermore, the standard deviations σ_s and $\sigma_{s,\text{const}}$ are compared respectively. In Fig. 7.5 the prediction velocity error $\bar{e}_v$ and its standard deviation σ_v of the presented algorithm are compared to the mean

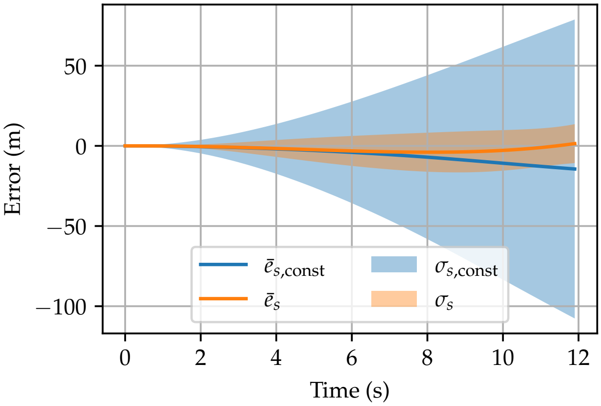

Figure 7.4: Mean and standard deviation of the position estimation errors for the constant velocity estimator $\bar{e}_{s,\text{const}}/\sigma_{s,\text{const}}$ and the presented algorithm $\bar{e}_s/\sigma_s$ along the path obtained from the PP component evaluated on recorded data.

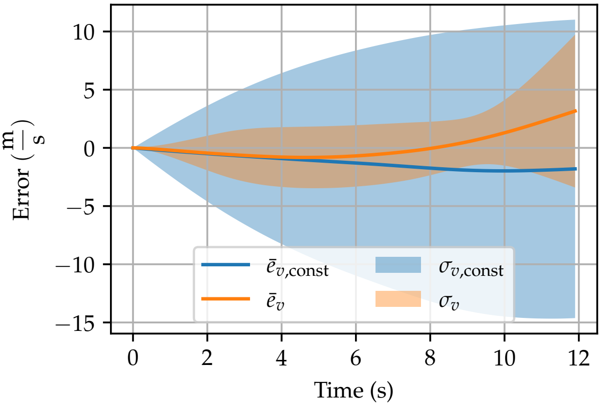

Figure 7.5: Mean and standard deviation of the velocity estimation errors for the constant velocity estimator $\bar{e}_{v,\text{const}}/\sigma_{v,\text{const}}$ and the presented algorithm $\bar{e}_v/\sigma_v$ along the path obtained from the PP component evaluated on recorded data.

error and standard deviation of the velocity of the constant velocity estimator, that is $\bar{e}_{v,\text{const}}$ and $\sigma_{v,\text{const}}$ The presented algorithm outperforms the constant velocity predictor significantly, although in a short prediction horizon, the errors are similar.

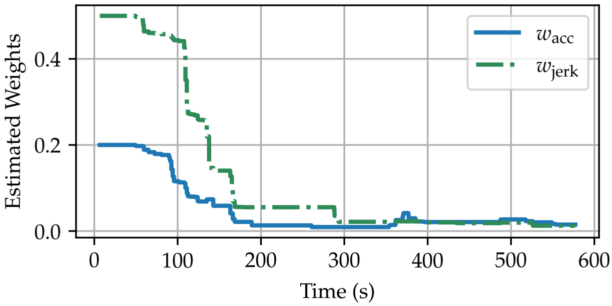

Figure 7.6: Weight estimates for the jerk and acceleration weighting obtained by the HLNLP component. Low weights correspond to aggressive driving that is only limited by the velocity and acceleration constraints.

Validation of the Full Algorithm

The algorithm was evaluated with two opponent vehicles in a simulation environment, as shown in Fig. 7.1. The two opponent race cars follow a racing line that was computed by a semi-analytic velocity profile computation as shown in [289] together with differently parameterized racing paths according to [222]. Therefore, the resulting trajectories are not in the solution space of the LLNLP and consequently can not be approximated exactly, which is similar to real observations. The HLNLP estimates the weight parameters and keeps converging to a semi-stationary solution after approximately 200 seconds as shown in Fig. 7.6. The convergence behavior depends heavily on the choice of hyperparameters, particularly on the arrival weight P in (7.8). After the weights converged, the predictions of all active components (*all components*) were compared to other estimation algorithms. First, the initial parameter setting was simulated, where the weights and constraints were kept constant, and only the LLNLP was active (referred to as *LLNLP*). Secondly, a constant velocity estimation was used, where the path was computed by means of the PP component, but the velocity was set constant to the observed velocity (referred to as *constant velocity*). Fig. 7.7 shows the comparison of the three settings by evaluating the Euclidean position error after certain prediction horizons. It can be seen that for increasing prediction horizons, the differences in the error measures become large due to the acceleration constraints related to curves and the integrating errors. For short prediction horizons, the constant velocity estimator is performing similarly in our test cases, which was also observed in [254]. The prediction performance with respect to the Euclidean position error was further compared by deactivating either the CQP or the HLNLP part. Fig. 7.8 shows the comparison with either component active (*CQP active* or *HLNLP active*), with all parameters fixed (*fixed parameters*) or with the

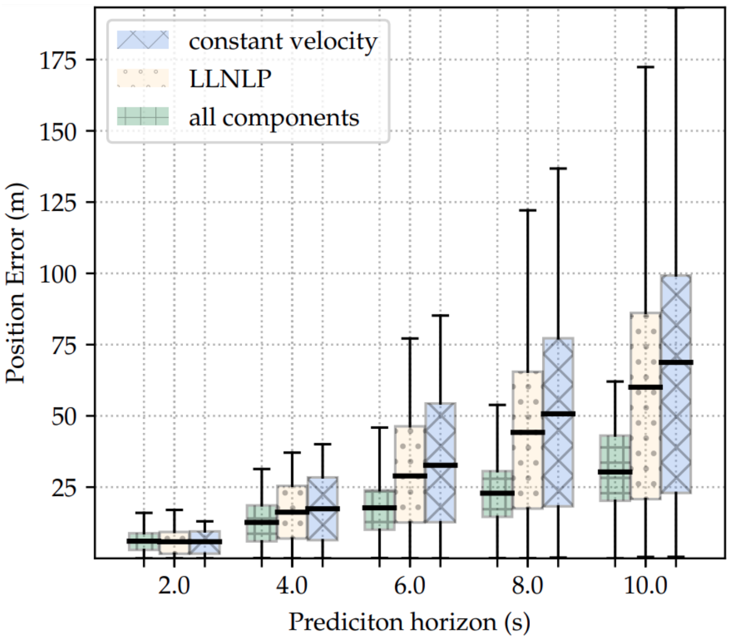

Figure 7.7: Box plot statistics (mean, standard deviation, maximum and minimum values) of the Euclidean position error of the prediction compared to the ground truth for different prediction algorithms and at particular prediction horizons

full algorithm (*all active*) at a prediction horizon of 6 and 8 seconds. The results looked similar for all observed race cars. In our simulation, the biggest improvement originated from the HLNLP part, which can be seen in Fig. 7.8 and which is due to the rather aggressive driving behavior of the observed vehicles and the moderate initialization of the corresponding weight parameters of the LLNLP.

7.1.5 Conclusions

The paper presents a novel approach for predicting race car trajectories in real-time and with sparse observation data. It is shown that the algorithm works in an embedded setting and yields satisfying predictions. The key advantage of using an optimization problem as a predictor is the natural integration of constraints. Nevertheless, the quality of the solution is restricted by the assumptions related to the low-level problem, e.g., which norms are used as penalties and what quantities are supposed to be penalized. The expressiveness

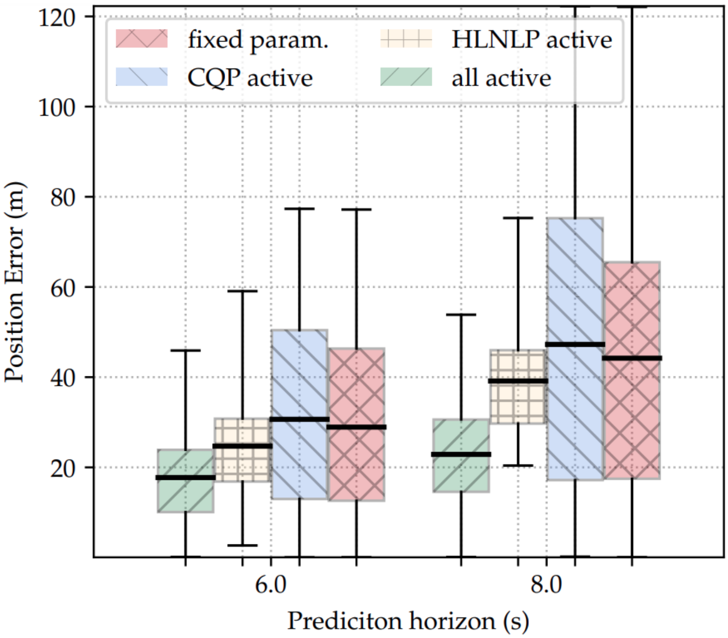

Figure 7.8: Box plot statistics (mean, standard deviation, maximum and minimum values) of the Euclidean position error of the prediction compared to the ground truth for different active components at a prediction horizon of 6 and 8 seconds.

of the low-level problem is limited due to its KKT conditions arising in a high-level optimization problem, which poses a non-smooth optimization problem with no guaranteed solution. Yet, in practice, the problem, as stated, is posed well enough to be solvable by means of a robust solver like `IPOPT`. Future investigations might include an algorithm that also estimates an uncertainty measure and updates the arrival cost correspondingly, as well as investigating rich function approximators in various parts of the algorithm to achieve a vanishing error as the number of samples increases.

7.2 A Hierarchical Approach for Strategic Motion Planning in Autonomous Racing

---

In this section, the paper published in [224] is reprinted with permission of Jasper Hoffmann, Joschka Boedecker and Moritz Diehl. Note that the formatting of some formulas, terms, and numbers has been slightly adjusted for consistency without changing their meaning or content.

The contributions of each author are listed in the following.

Rudolf Reiter:	*idea, programming of the overall system and the published algorithm, design of the experiments, programming and training for machine learning (reinforcement learning), writing of the document (all sections)*
Jasper Hoffmann:	*programming the training for machine learning (reinforcement learning), transcript of the document ("reinforcement learning" section)*
Joschka Boedecker:	*corrections on the topic of "reinforcement learning"*
Moritz Diehl:	*mathematical corrections, stylistic corrections, linguistic improvements*

©2023 European Control Association. *DOI: 10.23919/ECC57647.2023.10178143.*

Abstract. We present an approach for safe trajectory planning, where a strategic task related to autonomous racing is learned sample efficiently within a simulation environment. A high-level policy, represented as a neural network, outputs a reward specification that is used within the function of a parametric nonlinear model predictive controller. By including constraints and vehicle kinematics in the nonlinear program, we can guarantee safe and feasible trajectories related to the used model. Compared to classical reinforcement learning, our approach restricts the exploration to safe trajectories, starts with an excellent prior performance and yields complete trajectories that can be passed to a tracking lowest-level controller. We do not address the lowest-level controller in this work and assume perfect tracking of feasible trajectories. We show the superior performance of our algorithm on simulated racing tasks that include high-level decision-making. The vehicle learns to efficiently overtake slower vehicles and avoids getting overtaken by blocking faster ones.

7.2.1 Introduction

Motion planning for autonomous racing is challenging due to the fact that vehicles operate at performance limits and planning requires interactive yet safe behavior. This work focuses on strategic planning for fixed opponent policies with safety guarantees. Current research is usually based on either graph-based, sampling-based, learning-based, or optimization-based planners [41, 199]. We propose a combination of model-predictive control (MPC) and a neural network (NN) trained by a reinforcement learning (RL) algorithm within simulations. MPC is a powerful optimization-based technique commonly used to solve trajectory planning and control problems. Using efficient numerical solvers and the possibility to incorporate constraints directly makes MPC attractive in terms of safety, explainability, and performance [217]. Nevertheless, in problems like interactive driving, it is difficult to model the behavior of other vehicles. In contrast to MPC, RL is an exploration-driven approach for solving optimal control problems. Instead of an optimization-friendly model, RL only requires samples of the dynamics and can, in theory, optimize over arbitrary cost functions. The flexibility of RL comes at the cost of a high sample inefficiency that is often unfavorable for real-world applications, where data is expensive and rare. Furthermore, RL, in the general setting, lacks safety guarantees. However, once the amount and quality of data are sufficient, the learned policies can show impressive results [306]. In this paper, we combine MPC and RL by using an MPC-inspired low-level trajectory planner to yield kinematic feasible and safe trajectories and use the high-level RL policy for strategic decision-making. We use the expression reference tracking nonlinear model predictive control (NMPC) (parameterized model predictive planner) to refer to an MPC-based planner, which outputs feasible reference trajectories that we assume to be tracked by a *lowest*-level control systems. This hierarchical approach is common in automotive software stacks [293, 199]. We use the reference tracking NMPC to formulate safety-critical constraints and basic time-optimal behavior but let the cost function be subject to changes by the high-level RL policy. Particularly, we propose an interface where the high-level RL policy outputs a reference in the Frenet coordinate frame. With this approach, we start with an excellent prior strategy for known model parts. We can guarantee safe behavior concerning the chosen vehicle model and the prediction of opponents.

The structure of this paper is as follows. In Sec. 7.2.2, we motivate our approach by a similar formulation named *safety filter* [294], in Sec. 7.2.4, we describe the MPC-based planner, and in Sec. 7.2.5, we explain the implementation of the high-level RL policy and how we train it. Finally, in Sec. 7.2.6, we evaluate the algorithm, which we refer to as HILEPP (*hierarchical learning-based predictive planner*), in a multi-agent simulation that involves strategic decision-making.

Contribution: We contribute by deriving and evaluating a sample efficient and safe motion planning algorithm for autonomous race cars. It includes a novel cost function formulation for the interaction of MPC and RL with a strong prior performance, real-time applicability, and high interpretability.

Related work: Several works consider RL as a set-point generator for MPC for autonomous agents [107, 48]. As opposed to our approach, they focus on final target points. Another research branch focuses on safety verification with a so-called "safety filter" [55]. For instance, in [294], a rudimentary MPC variant is proposed that considers constraints using MPC as a verification module. Similarly, the authors of [171] use MPC to correct an RL policy if a collision check fails. RL is also used for MPC weight tuning, such as in [265] for UAVs and in [313] for adaptive control in autonomous driving. Related research for motion planning of autonomous racing was recently surveyed in [41]. Several works focus on local planning without strategic considerations [293, 196], thus can not directly be used in multi-agent settings. Other works use a game-theoretic framework [167], which often limits the applicability due to its complexity.

An algorithm for obtaining Nash equilibria is iterated best response (IBR), as shown for drone racing in [266] or for vehicle racing in [299]. However, IBR has high computation times. An algorithm aiming at the necessary KKT conditions of the generalized Nash equilibrium problem is presented in [161]. However, the resulting optimization problem is hard to solve. In [252], long-term strategic behavior is learned through simulation without safety considerations.

7.2.2 Background and Motivation

A trained neural network (NN) used as a function approximator for the policy $\pi^\theta(s)$, where $\theta \in \mathbb{R}^{n_\theta}$ is the learned parameter vector and $s \in \mathbb{R}^{n_s}$ is the RL environment state, can generally not guarantee safety. Safety is related to constraints for states and controls that must be satisfied at all times. Therefore, the authors in [294] propose an MPC-based policy $\pi^{\mathrm{S}} : \mathbb{R}^{n_a} \to \mathbb{R}^{n_a}$ that projects the NN output $a \in \mathbb{R}^{n_a}$ to a safe control $u^{\mathrm{S}} = \pi^{\mathrm{S}}(x, a)$, where it is guaranteed that $u^{\mathrm{S}} \in \mathcal{U}^{\mathrm{S}} \subseteq \mathbb{R}^{n_a}$. The safe set $\mathcal{U}^{\mathrm{S}}$ is defined for a known (simple) system model $\dot{x} = f(x, u)$ with states x and controls u and corresponding, often tightened, constraints. In this formulation, the input u has the same interpretation as the action a and the state x relates to the model inside the filter. Constraint satisfaction for states is expressed via the set membership $x \in \mathcal{X}$ and for controls via $u \in \mathcal{U}$. The system model is usually transformed to discrete-time via an integration function $x_{i+1} = F(x_i, u_i)$ with step size Δt. When using direct multiple shooting [45] one obtains decision variables for the state $X = [x_0, \dots, x_N] \in \mathbb{R}^{n_x \times (N+1)}$ and for the controls $u = [u_0, \dots, u_{N-1}] \in \mathbb{R}^{n_u \times N}$. Since the optimization problem can only be formulated for a finite horizon, a control invariant terminal set S^t needs to be included. The safety filter solves the following optimization problem

$$\begin{aligned} \min_{X,U} \quad & \|u_0 - \bar{a}\|_R^2 \\ \text{s.t.} \quad & x_0 = \bar{x}_0, \quad x_N \in \mathcal{S}^{\mathrm{t}}, \\ & x_{i+1} = F(x_i, u_i), \quad i = 0, \dots, N-1, \\ & x_i \in \mathcal{X}, u_i \in \mathcal{U}, \quad i = 0, \dots, N-1 \end{aligned} \tag{7.9}$$

and takes the first control u_0^* of the solution (X^*, U^*) as output $u^{\mathrm{S}} := u_0^*$. The authors in [294] use the filter as a post-processing safety adaption. However, we propose to use this formulation as a basis for an online filter, even during learning, which makes it applicable to safety-relevant environments. We do not require the same physical inputs to our filter, but rather modifications to a parametric optimization problem, similar to [110]. We propose a general interface between the high-level RL policy and MPC, namely a cost function $L(X, U, a)$, modified by action a. Our version of the reference tracking NMPC as a fundamental part of the algorithm solves the optimization problem

$$\begin{aligned} \min_{X,U} \quad & L(X, U, a) \\ \text{s.t.} \quad & x_0 = \hat{x}_0, \quad x_N \in \mathcal{S}^{\mathrm{t}}, \\ & x_{i+1} = F(x_i, u_i), \quad i = 0, \dots, N-1, \\ & x_i \in \mathcal{X}, u_i \in \mathcal{U}, \quad i = 0, \dots, N-1, \end{aligned} \tag{7.10}$$

and takes the optimal state trajectory of the solution (X^*, U^*) as output $X_{\mathrm{ref}} := X^*$ of the reference tracking NMPC algorithm. Due to the pruning of infeasible, i.e., unsafe, trajectories of the actual control, the algorithm becomes sample efficient.

7.2.3 General Method

We apply our algorithm to a multi-agent vehicle competition on a race track. We aim to obtain a sample efficient planner that performs time-optimal trajectory planning, avoids interactive opponents, and learns strategic behavior, such as blocking other vehicles in simulation. We assume fixed policies of a fixed number of N_{ob} opponents and, therefore, do not consider the interaction as a game-theoretical problem [315]. We use an obstacle avoidance rule, according to the autonomous racing competitions `Roborace` [233] and `F1TENTH` [196], where in a dueling situation, the following vehicle (FV) is mainly responsible avoiding a crash. However, the leading vehicle (LV) must not provoke a crash.

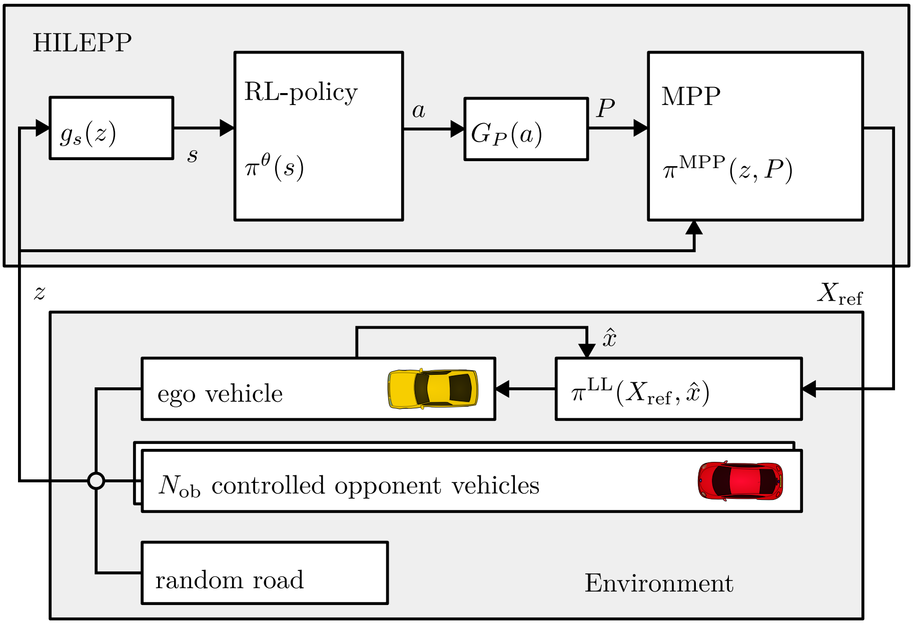

Figure 7.9: Proposed control structure. The multi-vehicle environment constitutes a trajectory tracking ego agent (lowest-level controller $\pi^{\text{LL}}(\cdot)$). A state z concatenates all $N_{\text{ob}} + 1$ vehicle states and road curvature information. A function $g_s(z)$ projects the state to a lower dimensional state space. A high-level RL policy $\pi^\theta(s)$ and an expanding function $G_P(a)$ modify the cost function of parameterized model predictive planner (reference tracking NMPC) $\pi^{\text{MPC}}(z, P)$ by action a. The reference tracking NMPC outputs a feasible and safe trajectory X_{ref} to the ego lowest-level controller.

Unfortunately, to the best of the author's knowledge, there is no rigorous rule for determining the allowed actions of dueling vehicles. However, we formalize the competition rules of `F1TENTH` similar to [166], where the LV only avoids an inevitable crash, which we state detailed in Sec. 7.2.4. A block diagram of our proposed algorithm is shown in Fig. 7.9, where we assume a multi-agent environment with a measured state $z \in \mathbb{R}^{n_z}$, which concatenates the ego agent states x, the obstacle/opponent vehicle states x^{ob} and the road curvature $\kappa(\zeta_i)$ on evaluation points ζ_i. We include prior domain knowledge to get the high-level RL policy state $s \in \mathbb{R}^{n_s}$ with the pre-processing function $s = g_s(z)$. For instance, we use relative distances of the opponents to the ego vehicle instead of absolute values. An expansion function $P = G_P(a)$, with the high-level RL policy $a = \pi^\theta(s)$, is used to increase the dimension of the NN output to obtain a parametric cost function. The expansion function is used to include prior knowledge and to obtain an optimization-friendly cost function in the reference tracking NMPC.

7.2.4 Parameterized Model Predictive Planner

Our core component reference tracking NMPC constitutes an MPC formulation that accounts for safety and strong initial racing performance. It comprises a vehicle model, safety constraints, and a parameterized cost function, which we will explain in the following section.

Vehicle Model

We use rear-wheel-centered kinematic single-track vehicle models in the Frenet coordinate frame, as motivated in previous work [222]. The models are governed by the longitudinal force F_d that accounts for accelerating and braking, and the steering rate r, which is the first derivative of the steering angle δ. The most prominent resistance forces $F_\mathrm{res}(v) = c_\mathrm{air} v^2 + c_\mathrm{roll}\mathrm{sign}(v)$ are included. The air drag depends on the vehicle speed v with the constant c_air. The rolling resistance is proportional to $\mathrm{sign}(v)$ by the constant c_roll. We drop the sign function since we only consider positive speed. As shown in previous work [222, 225], the Frenet transformation $\mathcal{F}(\cdot)$ relates Cartesian states $x^\mathrm{C} = [x_\mathrm{e} \quad y_\mathrm{e} \quad \varphi]^\top$, where x_e and y_e are Cartesian positions and φ is the heading angle, to the curvilinear states

$$x^\mathrm{F} = \mathcal{F}(x^\mathrm{C}) = [\zeta \quad n \quad \alpha]^\top. \tag{7.11}$$

The Frenet states are related to the center lane $\gamma(\zeta) = [\gamma_x(\zeta) \quad \gamma_y(\zeta)]$, with signed curvature $\kappa(\zeta)$ and tangent angle $\varphi^\gamma(\zeta)$, where ζ is the 1d-position of the closest point of the center lane, n is the lateral normal distance and α is the difference of the vehicle heading angle to the tangent of the reference curve. Under mild assumptions [222], the Frenet transformation and its inverse

$$x^\mathrm{C} = \mathcal{F}^{-1}(x^\mathrm{F}) = \begin{bmatrix} \gamma_x(\zeta) - n\sin(\varphi^\gamma(\zeta)) \\ \gamma_y(\zeta) + n\cos(\varphi^\gamma(\zeta)) \\ \varphi^\gamma(\zeta) - \alpha \end{bmatrix} \tag{7.12}$$

are well-defined. We summarize the states with $x = \begin{bmatrix}\zeta & n & \alpha & v & \delta\end{bmatrix}^\top$ and controls with $u = \begin{bmatrix}F_\mathrm{d} & r\end{bmatrix}^\top$. The Frenet frame vehicle model is parameterized by the mass m and length l and given as

$$\dot{x} = f(x,u) = \begin{bmatrix} \frac{v\cos(\alpha)}{1-n\kappa(\zeta)} \\ v\sin(\alpha) \\ \frac{v}{l}\tan(\delta) - \frac{\kappa(\zeta)v\cos(\alpha)}{1-n\kappa(\zeta)} \\ \frac{1}{m}(F_\mathrm{d} - F_\mathrm{res}(v)) \\ r \end{bmatrix}. \tag{7.13}$$

The discrete states x_k at sampling time $k\Delta t$ are obtained by an RK4 integration function $x_{k+1} = F(x_k, u_k, \Delta t)$.

Safety Constraints

As stated in Sec. 7.2.2, the reference tracking NMPC formulation should restrict trajectories X_{ref} to be within model constraints. Since we assume known vehicle parameters and no measurement noise, this can be guaranteed for most limitations in a straightforward way. Nevertheless, the interactive behavior of the opponent vehicles poses a severe challenge to the formulation. On one extreme, we could model the other vehicles robustly, which means we account for all possible maneuvers, which yields quite conservative constraints. On the other extreme, with known parameters of all vehicles, one could model the opponent by "leaving space" for at least one possible motion of the opponent without a crash, thus not forcing a collision. The latter leads to a hard bi-level optimization problem since the feasibility problem, which is an optimization problem itself, is needed as a constraint of the reference tracking NMPC. In this work, we aim at a heuristic explained in Sec. 7.2.4.

Vehicle Limitations. Slack variables $\sigma = [\sigma_v, \sigma_\alpha, \sigma_n, \sigma_\delta, \sigma_a, \sigma_{\text{o}}]^\top \in \mathbb{R}^6$, for state (7.14), acceleration (7.16) and obstacles constraints (7.18) are used to achieve numerically robust behavior. We use box constraints for states

$$B_x(\sigma) := \Big\{x \;\Big|\; -\sigma_n + \underline{n} \leq n \leq \overline{n} + \sigma_n, \tag{7.14a}$$

$$-\sigma_\alpha + \underline{\alpha} \leq \alpha \leq \overline{\alpha} + \sigma_\alpha, \tag{7.14b}$$

$$0 \leq v \leq \overline{v} + \sigma_v, \tag{7.14c}$$

$$-\sigma_\delta + \underline{\delta} \leq \delta \leq \overline{\delta} + \sigma_\delta\Big\}, \tag{7.14d}$$

and controls

$$B_u := \big\{u \mid \underline{F}_{\text{d}} \leq F_{\text{d}} \leq \overline{F}_{\text{d}}, \quad \underline{r} \leq r \leq \overline{r}\big\}. \tag{7.15}$$

Further, we use a lateral acceleration constraints set

$$B_{\text{lat}}(\sigma) := \left\{x \;\middle|\; \left|\frac{v^2 \tan(\delta)}{l}\right| \leq \overline{a}_{\text{lat}} + \sigma_a \right\}, \tag{7.16}$$

to account for friction limits.

Obstacle Constraints. We approximate the rectangular shape of obstacles (referenced by "ob") in the Cartesian coordinate frame by an ellipse and the ego vehicle by a circle, which yields superior computational properties compared to other approaches, cf. [228]. We assume a predictor of an obstacle vehicle i that outputs the expected Cartesian positions of the vehicle center $p_k^{\mathrm{ob}i} = [x_{\mathrm{e},k}^{\mathrm{ob}i} \quad y_{\mathrm{e},k}^{\mathrm{ob}i}]^\top \in \mathbb{R}^2$ with a constraint ellipse shape matrix $\hat{\Sigma}_k^{\mathrm{ob}i}(x) \in \mathbb{R}^{2\times 2}$ at time step k that depends on the (Frenet) vehicle state in x. The ellipse area is increased by $\Sigma^{\mathrm{ob}i}(x) = \hat{\Sigma}^{\mathrm{ob}i}(x) + \mathbb{I}(r + \Delta r)^2$ with radii of the ego covering circle r and a safety distance Δr. Since the ego vehicle position $p^\top = [x_\mathrm{e} \quad y_\mathrm{e}]$ is measured at the rear axis and in order to have a centered covering circle, we project the rear position to the ego vehicle center p_mid by

$$p_\mathrm{mid} = \begin{bmatrix} x_{\mathrm{e,mid}} \\ y_{\mathrm{e,mid}} \end{bmatrix} = P(x^\mathrm{C}) = \begin{bmatrix} x_\mathrm{e} + \frac{l}{2}\cos\varphi \\ y_\mathrm{e} + \frac{l}{2}\sin\varphi \end{bmatrix} \tag{7.17}$$

For obstacle avoidance with respect to the ellipse matrix, we use the constraint set in compact notation

$$B_\mathrm{O}(x^\mathrm{ob}, \Sigma^\mathrm{ob}, \sigma) = \left\{ x \in \mathbb{R}^2 \middle| \left\| P(\mathcal{F}^{-1}(x)) - p^\mathrm{ob} \right\|^2_{(\Sigma^\mathrm{ob}(x))^{-1}} \geq 1 - \sigma_\mathrm{o} \right\}. \tag{7.18}$$

Obstacle Prediction. The opponent prediction uses a simplified model with states $x^\mathrm{ob} = [\zeta^\mathrm{ob}, n^\mathrm{ob}, v^\mathrm{ob}]^\top$ and assumes curvilinear motion depending on the initial estimated state $\hat{x}^\mathrm{ob}$. With the constant acceleration force F_d^ob, the ODE of the opponent estimator can be written as

$$\dot{\zeta}^\mathrm{ob} = \frac{v^\mathrm{ob}(t)\cos(\hat{\alpha}^\mathrm{ob})}{1 - n^\mathrm{ob}\kappa(\zeta^\mathrm{ob})}, \quad \dot{n}^\mathrm{ob} = v^\mathrm{ob}(t)\sin(\hat{\alpha}^\mathrm{ob}), \quad \dot{v}^\mathrm{ob} = \frac{1}{m^\mathrm{ob}} F_\mathrm{d}^\mathrm{ob}. \tag{7.19a}$$

Since the FV is responsible for a crash, it *generously* predicts the LV by assuming constant velocity motion, where F_d^ob is set to 0. The LV predicts the FV most *evasively*, which we realize by assuming an FV full stop with its maximum braking force $F_\mathrm{d}^\mathrm{ob} = \underline{F}_\mathrm{d}^\mathrm{ob}$. In any situation, this allows the FV to plan for at least one safe trajectory (i.e., a full stop) Thus, the LV does not "provoke" a crash, as required in racing competition rules [233, 196]. Besides these minimum safety restrictions, interaction should be learned by the high-level RL policy. We simulate the system forward with a function $\Phi()$, using steps of the RK4 integration function to obtain the predicted states $[x_0^\mathrm{ob}, \ldots, x_N^\mathrm{ob}] = \Phi(\hat{x}^\mathrm{ob}, \hat{\alpha}^\mathrm{ob}, F_\mathrm{d}^\mathrm{ob})$.

Recursive Feasibility. In order to guarantee safety for a finite horizon and constraints (7.14), (7.15) and (7.16), we refer to the concept of recursive feasibility and control invariant sets (CIS) [217]. A straightforward CIS is the trivial set of zero velocity $\{x \mid v = 0\}$. An approximation to the CIS, which is theoretically not a CIS but which has shown good performance in practice, is the limited-velocity terminal set $\mathcal{S}^{\mathrm{t}} := \{x \mid \alpha = 0, v \leq \overline{v}_{\max}\}$. For long horizons, the influence of the terminal set vanishes.

Objective

For the parameterized cost function $L(X, U, a)$, we propose a formulation with the following properties:

1. Simple structure for reliable and fast NLP iterations
2. Expressive behavior related to strategic driving
3. Low dimensional action space
4. Good initial performance

The first property is achieved by restricting the cost function to a quadratic form. The second property is achieved by formulating the state reference in the Frenet coordinate frame. The final properties of a low dimensional action space and an excellent initial performance are achieved by interpreting the actions as reference lateral position n_{ref} and reference speed v_{ref}. By setting the reference speed, also the corresponding longitudinal state $\zeta_{\mathrm{ref},k}$ of a curvilinear trajectory is defined by $\zeta_{\mathrm{ref},k} = \hat{\zeta} + k\Delta t v_{\mathrm{ref}}$. The reference heading angle miss-match α_{ref} and the steering angle δ_{ref} are set to zero, with fixed weights w_α and w_δ, since these weights are tuned for smooth driving behavior. Setting the reference speed v_{ref} above maximum speed approximates time-optimal driving [151]. We compare the influence using references with their associated weights w_v, w_n (HILEPP-II with $a_{\mathrm{II}} = [v_{\mathrm{ref}} \quad n_{\mathrm{ref}} \quad w_v \quad w_n]^\top$) to fixed weights without using them in the action space (HILEPP-I with $a_{\mathrm{I}} = [v_{\mathrm{ref}} \quad n_{\mathrm{ref}}]^\top$).

NLP Formulation

We use the action-dependent stage cost matrix $Q_{\mathrm{w}}(a)$ with $Q_{\mathrm{w}} : \mathbb{R}^{n_a} \to \mathbb{R}^{n_x \times n_x}$ and a cost independent terminal cost $Q^t \in \mathbb{R}^{n_x \times n_x}$. We set the values of R, Q_0 and Q^{t} to values corresponding to driving smoothly and time-optimally. With constant action inputs $\bar{a}$, this leads to a strong initial performance at the beginning of training the high-level RL policy. With the constant time action-dependent reference values $\xi_{\mathrm{ref},k}(a) = [0 \quad n \quad 0 \quad v_x \quad 0]^\top \in \mathbb{R}^{n_x}$ for HILEPP-I/II and constant time reference weights $Q_{\mathrm{w}}(a) = \mathrm{diag}([0 \quad w_n \quad 0 \quad w_v \quad 0])$ for HILEPP-II, we can write the expanding function as

$$G_P(a) : a \to \Big(\xi_{\mathrm{ref},0}(a), \ldots, \xi_{\mathrm{ref},N}(a), Q_{\mathrm{w}}(a)\Big), \tag{7.20}$$

which maps n_a to $n_x^2(N+1)+n_x(N+1)$ dimensions for cost matrices and reference values. We state the final NLP, using the vehicle model (7.13), the MPC path constraints for obstacle avoidance (7.18), vehicle constraints (7.14), (7.15) and (7.16) and the parametric cost functions of (7.10). The full objective, including slack variables $\Xi = [\sigma_0, \dots, \sigma_N] \in \mathbb{R}^{6\times N}$ for each stage, associated L2 weights $Q_{\sigma,2} = \operatorname{diag}(q_{\sigma,2}) \in \mathbb{R}^{6\times 6}$ and L1 weights $q_{\sigma,1} \in \mathbb{R}^6$, reads as

$$\begin{aligned} L(X,U,a,\Xi) = &\sum_{k=0}^{N-1} \|x_k - \xi_{\mathrm{ref},k}(a)\|^2_{Q_{\mathrm{w}}(a)} + \|u_k\|^2_R \\ &+ \|x_N - \xi_{\mathrm{ref},N}(a)\|^2_{Q^t} + \sum_{k=0}^{N} \|\sigma_k\|^2_{Q_{\sigma,2}} + |q_{\sigma,1}^\top \sigma_k|. \end{aligned} \tag{7.21}$$

Together with the predictor for time step k of the j-th future opponent vehicle states, represented as bounding ellipses with the parameters $p_i^{\mathrm{ob},j}, \Sigma_i^{\mathrm{ob},j}$, the parametric NLP can be written as

$$\begin{aligned} \min_{X,U,\Xi} \quad & L(X,U,a,\Xi) \\ \text{s.t.} \quad & x_0 = \hat{x}, \quad \Xi \geq 0, \quad x_N \in \mathcal{S}^{\mathrm{t}}, \\ & x_{i+1} = F(x_i,u_i) && i = 0,\dots,N-1, \\ & U_i \in B_u, && i = 0,\dots,N-1, \\ & x_i \in B_x(\sigma_k) \cap B_{\mathrm{lat}}(\sigma_k) && i = 0,\dots,N, \\ & x_i \in B_{\mathrm{ob}}(p_i^{\mathrm{ob},j}, \Sigma_i^{\mathrm{ob},j}, \sigma_k) && i = 0,\dots,N, \\ & && j = 0,\dots,N_{\mathrm{ob}}. \end{aligned} \tag{7.22}$$

The final reference tracking NMPC algorithm is stated in Alg. (3).

7.2.5 Hierarchical Learning-based Predictive Planner

The reference tracking NMPC of Sec. 7.2.4 plans safely and time-optimally, but not strategically. Therefore, we learn a policy π^θ with RL that decides how to parameterize the reference tracking NMPC to achieve a strategic goal at each time step. Since we assume stationary opponent policies, we can apply standard, i.e., single-agent RL algorithms [315] and solve for the best response. In the following, we give a brief theoretical background to policy gradient methods and then describe the training procedure in detail.

Algorithm 3: reference tracking NMPC

```
input  : action $a$, ego states $\hat{x}$, $N_{\mathrm{ob}}$ obstacle states $\hat{x}^{\mathrm{ob}}$
output : planned trajectory $X_{\mathrm{ref}}$
for $j$ in range($N_{\mathrm{ob}}$) do
    if $\hat{\zeta}^{\mathrm{ob}} \leq \hat{\zeta}$ then
        Consider opp. as FV: $F_{\mathrm{d}}^{\mathrm{ob}} \leftarrow \underline{F}_{\mathrm{d}}^{\mathrm{ob}}$
    end
    else
        Consider opp. as LV $F_{\mathrm{d}}^{\mathrm{ob}} \leftarrow 0$
    end
    Predict $[x_0^{\mathrm{ob}}, \dots, x_N^{\mathrm{ob}}] = \Phi(\hat{x}^{\mathrm{ob}}, \hat{\alpha}^{\mathrm{ob}}, F_{\mathrm{d}}^{\mathrm{ob}})$;
    Compute constraint ellipses $\Sigma_k^{\mathrm{ob},j} = \Sigma_0(\varphi^{\mathrm{ob},j})$;
end
Compute weights $(\zeta_{\mathrm{ref},k}, Q_{\mathrm{w}}) \leftarrow G_P(a)$;
$X_{\mathrm{ref}} \leftarrow$Solve NLP (7.10) with $(\zeta_{\mathrm{ref},k}, Q_{\mathrm{w}})$;
```

Policy Gradient

RL requires a Markov Decision Process (MDP) framework. A MDP consists of a state space $\mathcal{S}$, an action space $\mathcal{A}$, a transition kernel $P(s_{k+1} \mid s_k, a_k)$, a reward function $R : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$ that describes how desirable a state is (equal to the negative cost) and a discount factor $\gamma \in [0, 1)$. The goal for a given MDP is finding a policy $\pi^\theta : \mathcal{S} \mapsto \mathcal{A}$ that maximizes the expected discounted return

$$J(\pi^\theta) = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k R(s_k, a_k) \mid s_0 = s\right], \; s_k \sim P(s_{k+1} \mid s_k, a_k), \; a_k \sim \pi^\theta(s_k). \tag{7.23}$$

where s_k is the state and a_k the action taken by the policy π^θ at time step k. An important additional concept is the state-action value function

$$Q^{\pi^\theta}(s, a) = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k R(s_k, a_k) \mid s_0 = s, a_0 = a\right] \tag{7.24}$$

that is the expected value of the policy π^θ starting in state s and taking an action a. In general, finding an optimal policy π^θ by directly optimizing θ in (7.23) is impossible. The expectation in (7.23) might be computationally intractable, or the exact transition probabilities P may be unknown. Thus, the policy gradient $\nabla J(\pi^\theta)$ is approximated using only transition samples from the environment. We sample these transitions from a simulator. However, they could also come from real-world experiments. A particularly successful branch of policy gradient methods is *actor-critic* methods [274], where we train two neural networks, an actor π^θ, and a critic Q^ϕ. The critic estimates the value of a chosen action and is trained by minimizing the temporal difference loss

$$J_Q(\phi) = \mathbb{E}_{s,a,r,s' \sim \mathcal{D}} \left[\left(r + \gamma Q_{\phi'} \left(s', \pi^\theta(s') \right) - Q_\phi(s,a) \right)^2 \right]. \tag{7.25}$$

The trained critic is used to train the actor with the objective

$$J_\pi(\theta) = \mathbb{E}_{s \sim \mathcal{D}} \left[Q^\phi(s, \pi^\theta(s)) \right]. \tag{7.26}$$

To derive the gradient for the policy π^θ from (7.26) we can use the chain rule [258]

$$\nabla_\theta J_\pi(\theta) = \mathbb{E}_{s \sim \mathcal{D}} \left[\nabla_\theta \pi^\theta(s) \; \nabla_a Q^\phi(s,a) \big|_{a=\pi^\theta(x)} \right]. \tag{7.27}$$

The soft-actor critic method, introduced by [117] enhances the actor-critic method by adding an entropy term into (7.26) and using the *reparameterization trick* to calculate the gradient. For a complete description, we refer to [117]. Note that, with a slight abuse of notation, in the equations from above, we sample transitions tuple (s,a,r,s') and states s from the same distribution $\mathcal{D}$. In our context, $\mathcal{D}$ is a buffer storing all transitions and states that have occurred so far by interacting with the environment.

Training Environment

We reduce the RL state space based on domain knowledge, which we put into the function $g_s(z_k)$. The race track layout is approximated by finite curvature evaluations $\kappa(\zeta + d_i)$ at different longitudinal distances d_i relative to the ego vehicle position ζ, for $i = 1, \dots, N_\kappa$. For the RL ego state $s(z_k) = [n, v, \alpha]^\top$, we include the lateral position n, the velocity v and the heading angle miss-match α. For opponent i, we additionally add the opponent longitudinal distance $\zeta_{\mathrm{ob}} - \zeta$ to the ego vehicle to state $s_{\mathrm{ob}_i} = [\zeta_{\mathrm{ob}_i} - \zeta, n_{\mathrm{ob}_i}, v_{\mathrm{ob}_i}, \alpha_{\mathrm{ob}_i}]^\top$. Combined, we get the following state definition for the RL agent

$$s_k = g_s(z_k) = [\kappa(\zeta + d_i), \dots, \kappa(\zeta + d_N), s^\top, s_{\mathrm{ob}_1}^\top, \dots, s_{\mathrm{ob}_{N_{\mathrm{ob}}}}^\top]^\top. \tag{7.28}$$

We propose a simple reward that encourages time-optimal and strategic driving: For driving time-optimally, we reward the progress on the race track by measuring the velocity of the ego vehicle projected point on the center line $\dot{s}_k$. For driving strategically, we reward ego vehicle overall rank by adding 1 for being in front of every opponent. Combined, we get the reward function

$$R(s,a) = \frac{\dot{s}}{200} + \sum_{i=1}^{N_{\mathrm{ob}}} 1_{\zeta_k > \zeta_k^{\mathrm{ob}_i}}. \tag{7.29}$$

At each time step, the high-level RL policy chooses a parameter for the reference tracking NMPC; thus, the action space is the parameter space of the reference tracking NMPC. For training the high-level RL policy, an essential part is the simulation function of the environment $z_{\text{next}} = \text{sim}(z, \kappa(\cdot))$, which we simulate for n_{epi} episodes and a maximum of n_{scene} steps. The road layout defined by $\kappa(\zeta)$ is randomized within an interval $[\text{-}0.04, 0.04]\text{m}^{-1}$ before each training episode. The curvature is set together with initial random vehicle states z by a reset function $(z, \kappa(\zeta)) = Z()$. We use Alg. 4 for training and Alg. 5 for the final deployment of HILEPP.

Algorithm 4: HILEPP training

```
input  : number of episodes $n_{\text{epi}}$, maximum scenario steps $n_{\text{scene}}$, reset
         function $(z, \kappa(\zeta)) = Z()$, reward function $r(z)$
output : learned policy $\pi^{\theta}(\zeta)$
for $j$ in range($n_{\text{epi}}$) do
    reset+randomize environment $(z, \kappa(\zeta)) \leftarrow Z()$;
    for $i$ in range($n_{\text{scene}}$) do
        get NN input state $s \leftarrow g_s(z)$;
        get high-level action $a \leftarrow \pi^{\theta}(s)$;
        evaluate planner $X_{\text{ref}} \leftarrow$reference tracking NMPC$(a, z)$;
        simulate environment $z_{\text{next}} = \text{sim}(X_{\text{ref}})$;
        get reward $(r, \text{done}) \leftarrow R(z_{\text{next}}, a)$;
        RL update $\theta \leftarrow$train$(z, z_{\text{next}}, r, a)$;
        if done then
            exit loop
        end
        z$\leftarrow z_{\text{next}}$
    end
end
return $\pi^{\theta}(s)$;
```

Algorithm 5: HILEPP deployment

```
input  : environment state $z$, trained policy $\pi^{\theta}(s)$
output : reference trajectory $X_{\text{ref}}$
compute NN input state $s \leftarrow g_s(z)$;
compute high-level RL policy output $a \leftarrow \pi^{\theta}(s)$;
return reference tracking NMPC output $X_{\text{ref}} \leftarrow$reference tracking
  NMPC$(z, a)$;
```

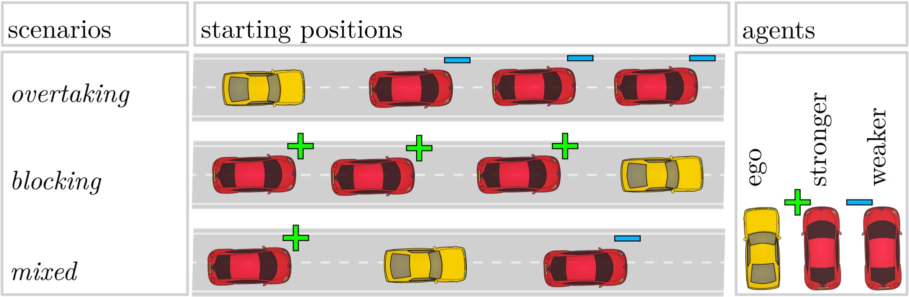

Figure 7.10: Scenarios differ in the initial rank and performance of vehicles.

7.2.6 Simulated Experiments

We evaluate (Alg. 5) and train (Alg. 4) HILEPP on three different scenarios that resemble racing situations (cf. Fig. 7.10). The first scenario *overtaking* constitutes three "weaker", initially leading opponent agents, where "weaker" relates to the parameters of maximum accelerations, maximum torques, and vehicle mass (cf. Tab. 7.4). The second scenario *blocking* constitutes three "stronger", initially subsequent opponents. The ego agent starts between a stronger and a weaker opponent in a third *mixed* scenario. Each scenario is simulated for one minute, where the ego agent has to perform best related to the reward (7.29). We train the HILEPP agent with the different proposed action interfaces (I: $\mathcal{A} = \{n_{\text{ref}}, v_{\text{ref}}\}$, II: $\mathcal{A} = \{n_{\text{ref}}, v_{\text{ref}}, w_n, w_v\}$). Opponent agents, as well as the ego agent baseline, are simulated with the state-of-the-art reference tracking NMPC (Alg. 3) with a fixed action a that accounts for non-strategic time-optimal driving with obstacle avoidance. We perform a hyper-parameter (HP) search for the RL parameters with *Optuna* [9]. The search space was defined by $[10^{-5}, 10^{-3}]$ for the learning rate, $\tau \in [10^{-5}, 10^{-2}]$ for the polyak averaging of the target networks, $\{64, 128, 256\}$ for the width of the hidden layers, $\{1, 2, 3\}$ for the number of hidden layers and $\{128, 256\}$ for the batch size. We used the average return of 30 evaluation episodes after training for 10^5 steps as the search metric. We trained on randomized scenarios for $10 \cdot 10^5$ steps with 10 different seeds on each scenario. For estimating the performance of the final policy, we evaluated the episode return (sum of rewards) on 100 episodes. We further compare our trained HILEPP against a pure RL policy that directly outputs the controls u. The final experiments were run on a computing cluster were all 30 runs for one method where run on 8 GeForce RTX 2080 Ti with a AMD EPYC 7502 32-Core Processor with a training time of around 6 hours. We use the NLP solver `acados` [291] with `HPIPM` [97], RTI iterations and a partial condensing horizon of $\frac{N}{2}$.

Name	Variable	Ego Agent	“Weak” Agent	“Strong” Agent
wheelbase	$l_{\text{r}}, l_{\text{f}}$	1.7	1.7	1.7
chassis lengths	$l_{\text{r,ch}}, l_{\text{f,ch}}$	2	2	2
chassis width	w_{ch}	1.9	1.9	1.9
mass	m	1160	2000	600
max. lateral acc.	$\underline{a}_{\text{lat}}, \overline{a}_{\text{lat}}$	±8	±5	±13
max. acc. force	$\overline{F}_{\text{d}}$	10kN	8kN	12kN
max. brake force	$\underline{F}_{\text{d}}$	20kN	20kN	20kN
max. steering rate	$\underline{r}, \overline{r}$	±0.39	±0.39	±0.39
velocity bound	$\overline{v}$	60	60	60
steering angle bound	$\underline{\delta}, \overline{\delta}$	±0.3	±0.3	±0.3
road bounds	$\underline{n}, \overline{n}$	±7	±7	±7

Table 7.4: Vehicle model parameters. SI-units, if not stated explicitly.

Name	Variable	Value
nodes / disc. time	$N/\ \Delta t$	50/ 0.1
terminal velocity	$\overline{v}_N$	15
state weights	q	$[1, 500, 10^3, 10^3, 10^4]\Delta t$
terminal state weights	q_N	$[10, 90, 100, 10, 10]$
L2 slack weights	$q_{\sigma,2}$	$[10^2, 10^3, 10^6, 10^3, 10^6, 10^6]$
L1 slack weights	$q_{\sigma,1}$	$[0, 0, 10^6, 10^4, 10^7, 10^6]$
control weights	R	$\text{diag}([10^{-3}, 2 \cdot 10^6])\Delta t$

Table 7.5: Parameters for reference tracking NMPC in SI units

Results

In Fig. 7.11, we compare the training performance related to the reward (7.29), and in Fig. 7.12, we show the final performance of the two HILEPP formulations. HILEPP quickly outperforms the base-line reference tracking NMPC as well as the pure RL formulation, showing its high sample efficiency. With a smaller action space, HILEPP-I seems to learn faster. However, with more samples, HILEPP-II outperforms the smaller action space in all three scenarios on the evaluation runs in terms of median performance, see Fig. 7.12. The training was stopped after 10^6 steps due to the already high training time and the slow return increase, as shown in Fig. 7.11. Despite using state-of-the-art RL learning algorithms and an extensive HP search on GPU clusters, the pure RL

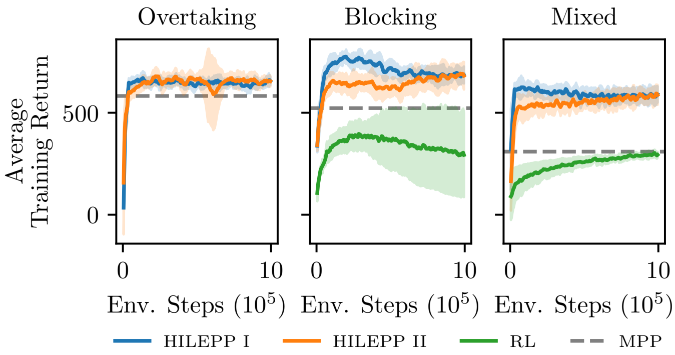

Figure 7.11: Training performance of average episode return (sum of rewards) of HILEPP with different action interfaces (I: actions $v_{\mathrm{ref}}, n_{\mathrm{ref}}$, II: actions $v_{\mathrm{ref}}, n_{\mathrm{ref}}, w_v, w_n$), pure RL and the reference tracking NMPC baseline. We used a moving average of over 1000 steps. Remarkably, we could not train a successful pure RL agent in the overtaking scenario.

Module	Mean± Std.	Max
reference tracking NMPC	5.45 ± 2.73	8.62
RL policy	0.13 ± 0.01	0.26
HILEPP-I	6.90 ± 3.17	9.56
HILEPP-II	7.41 ± 2.28	9.21

Table 7.6: Computation times (ms) of modules.

agent could not, in general, outperform the reference tracking NMPC baseline. Furthermore, the pure RL policy could not prevent crashes, whereas reference tracking NMPC successfully filters the actions within HILEPP to safe actions that do not cause safety violations. Notably, due to the struggle of the pure RL agent with lateral acceleration constraints, it has learned a less efficient strategy to drive slowly and just focus on blocking subsequent opponents in scenarios *blocking* and *mixed*. Therefore, pure RL could not perform efficient overtaking maneuvers in the *overtaking* scenario and yields evasive returns (consequently excluded in Fig. 7.12). In Tab. 7.6, we show that HILEPP is capable of planning trajectories with approximately 100Hz, which is sufficient and competitive for automotive motion planning [41]. A rendered plot of learned blocking is shown in Fig. 7.13, where also the time signals are shown of how the high-level RL policy sets the references of HILEPP-I. A rendered simulation for all three scenarios can be found on the website `https://rudolfreiter.github.io/hilepp_vis/`.

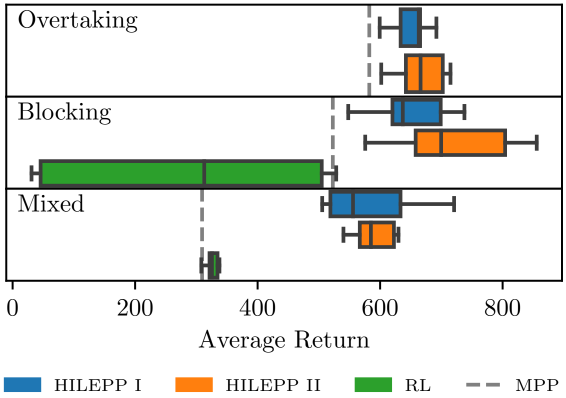

Figure 7.12: Final episode return of 100 evaluation runs of the proposed interfaces for different scenarios (Fig. 7.10).

7.2.7 Conclusions

We have shown a hierarchical planning algorithm for strategic racing. We use RL to train for strategies in simulated environments and have shown to outperform a basic time-optimal and obstacle-avoiding approach, as well as pure deep-learning-based RL in several scenarios. The major drawbacks of our approach are the restrictive prediction and the stationary policy of opponents. Further work could consider multi-agent RL (MARL) algorithms based on Markov games, which, however, is still a challenging open research area [315].

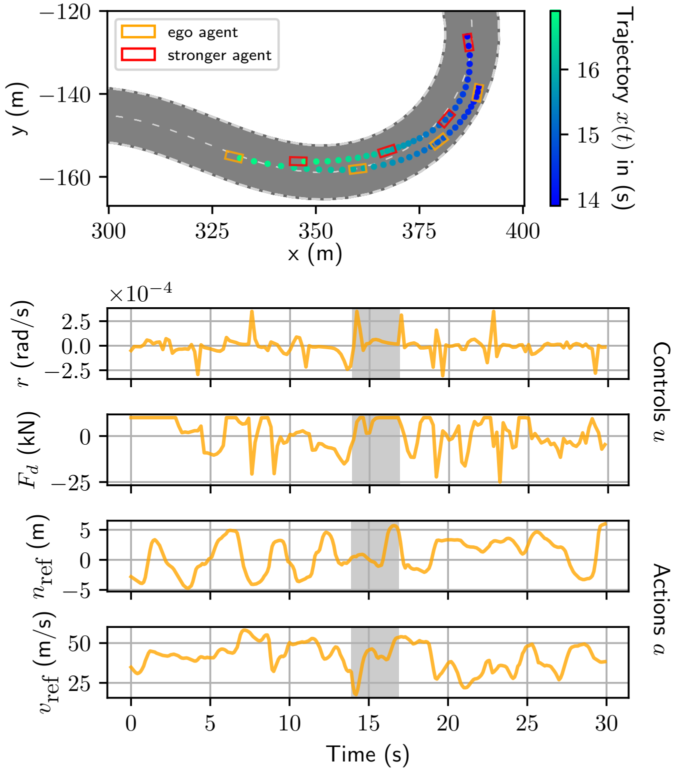

Figure 7.13: Exemplary evaluation episode of the HILEPP-I planner in the mixed scenario. On the bottom, the controls of the reference tracking NMPC and the actions of the high-level RL policy are shown. The grey box indicates a time window where snapshots of a blocking maneuver are shown in the top plot. The vehicles move from right to left.

7.3 Critical Discussion

In this chapter, two concepts for collision avoidance within autonomous racing were presented. Sect. 7.1 and the related publication [226] focused on estimating other future vehicle trajectories in order to include those within the ego vehicle motion planner for collision avoidance. In Sect 7.2 and the related publication [224], the aim was to learn strategic behavior from simulation by reinforcement learning. A low-level model predictive control (MPC) provided guarantees for safety to a known model, which is essential for autonomous racing tasks. As opposed to Sect. 7.1, the low-level MPC in Sect 7.2 used constant-velocity predictions for other racing vehicles. However, it could directly be combined with the inverse optimal control prediction approach from Sect. 7.1.

To the best of the author's knowledge, the prediction approach of Sect. 7.1 is the first method in the domain of autonomous racing to employ derivative-based bi-level optimization for real-time trajectory prediction. Given the inherent complexity of bi-level problems, our research introduces a specific smooth formulation of the Karush-Kuhn-Tucker conditions to solve the high-level problems. Using the proposed optimization-based approach to predict trajectories, the signed longitudinal position error after 12 seconds was decreased from 10 meters mean and 200 meters standard deviation to zero meters mean and 25 meters standard deviation compared to the constant velocity predictor [253]. The online computation time of the optimization-based low-level predictor was real-time capable on embedded hardware with an average computation time of 91 ms. The bi-level estimation was performed asynchronously with an average computation time of 520 ms, making it independent on sampling time limits.

In Sect 7.2, significant contributions to the field of autonomous race car motion planning are shown by deriving and evaluating a sample efficient and safe motion planning algorithm. The approach includes a novel cost function formulation that uses an reinforcement learning (RL) policy to parameterize an MPC. This hierarchical architecture ensures strong prior performance, real-time applicability with online computation times of less than 10 ms, and high interpretability. On three scenarios, i.e., overtaking, blocking, and performing both, the average reward for driving fast and having a good rank was increased by 8%, 37%, and 96% compared to standard MPC. Compared to pure RL, the reward was increased by 133% and 84% for blocking and mixed overtaking and blocking. The pure RL did not perform reasonably on the overtaking scenario, even after more than one million training steps.

Using a low-level MPC to predict other traffic participants turned out to be highly efficient and was used by the Autonomous Racing Graz (ARG) team [3] as one of the most successful predictors in further races held by `Roborace` [233]. Notably, the computational requirements for solving multiple nonlinear programs (NLPs) in real-time for motion prediction were compared to other components high but within the limits of the embedded computing platform. Our reported mean computation time of 91 ms is comparable to a timing analysis reported by [12] of the popular `Apollo` open source driving stack [2], where a mean computation time of 130 ms is reported for the prediction module.

The need for online adaption of the high-level nonlinear program (HLNLP) declined in this particular racing series because the simulated opponent race cars followed a similar behavior throughout the races. Moreover, the HLNLP exhibited the risk of converging to local solutions that could even degrade the performance on parts of the race track. The ARG team generally assumes that the online adaptation of safety critical components is challenging. Remarkably, the parameterization of the low-level nonlinear program (LLNLP) could be restricted to a reasonable set of parameters by sacrificing the LLNLP expressiveness. A further limitation of the approach is rooted in the assumption of non-interactive agents, which was the setting in the particular Roborace [233] competition. In an autonomous racing event where several “intelligent” agents would compete, the interactions would become essential and imply the connection of the ego planner with the prediction. The final algorithm would be more sophisticated since game-theoretic considerations based on the particular racing rules would also apply. Up to the publication of this work, no clearly defined rules for competitive interactive racing were available. The approach of Sect. 7.2 aims to separate interactive decision-making from collision avoidance based on a simple predictor. The presented method is both limiting but also provides safety guarantees. In fact, the high-level policy aims to learn interactions that are restricted by MPC plans that evade constant-velocity obstacle predictions for leading vehicles. The following vehicles are predicted by assuming full braking, which accounts for the leader’s assumed rule-book advantage. The switching behavior of the opponents is not considered within the planning horizon, which is a conservative approximation. Nevertheless, a striking advantage of this architecture is that for every control of the ego vehicle, a safe trajectory of the opponent exists, i.e., the ego agent never forces an inevitable crash, assuming exact vehicle models.

Noteworthy, the environment setting assumes constant policies of other agents. Policies could be changed in a game-like race with multiple “intelligent” agents, which, again, requires game-theoretic considerations or multi-agent reinforcement learning techniques [109].

The combination of RL with MPC is an active field of research due to their appealing “orthogonal” advantages [115], where the weaknesses of one approach are the strengths of the other. RL and MPC can be combined in various configurations and overall desiderata. Our architecture conceptually uses MPC as part of the environment during learning, i.e., the RL exploration and the critic evaluation are performed on the actions that parameterize the MPC. This has the advantage that when using RL experience replay, the forward path of the MPC does not need to be reevaluated by solving an NLP, and gradients in the backward path do not require differentiating through the NLP solver. A similar approach was used in, e.g., [203, 210, 48]. An alternative would use the MPC as part of the RL policy, add exploration noise on either the MPC parameters or the MPC controls, and evaluate the critic based on the MPC actions instead of the parameters. This configuration profits from the computation of gradients through the NLP, possibly allowing a higher number of MPC parameters set by the RL policy. This configuration was used in, e.g., [235, 279]. Besides these two particular configurations, many other architectures were proposed, such as using MPC only after training as a safety-filter [280], using an RL policy to initialize the primal variables of an MPC [105, 223] or providing expert trajectories in the initial RL training phase [163]. The benchmarking among different approaches is open for future research.

Chapter 8

Conclusion

This thesis presents significant contributions to the field of motion planning and control for autonomous vehicles, focusing on mixed-integer programming formulations, reinforcement learning (RL) and model predictive control (MPC)-based hierarchical motion planning, and Frenet coordinate frame (FCF) vehicle models. Critical challenges are addressed, and novel solutions are proposed to advance state-of-the-art autonomous vehicle motion planning and control. This thesis comprises several publications in peer-reviewed journals and conferences. The following summarizes the contributions, elaborates on the strengths and limitations, and provides an outlook for further research.

Summary of Contributions

The contributions of this thesis can be clustered into the three main optimization-based automotive motion planning topics. First, model formulations for FCF-based MPC are proposed for local nonlinear optimization algorithms. Second, mixed-integer formulations and an improved learning-based solution strategy are proposed for the global optimization of obstacle avoidance problems. Finally, novel approaches for optimization-based obstacle prediction and interactive driving in autonomous racing are contributed. The three main topic clusters are aligned with the core Chapters 5 to 7.

Chapter 5: Model Formulations for Optimization-Based Motion Planning. The first main chapter introduces novel FCF vehicle model formulations used within MPC that improve the numerical robustness, safety, and online computation speed in real-time applications. This section focuses on improving problem formulations for local derivative-based nonlinear optimization. The proposed preprocessing algorithm of Sect. 5.1 guarantees a singularity-free state space and smoothens nonlinearities, which results in improved closed-loop performance, empirically evaluated in simulations. For a sharp curve where the singularity originating from the Frenet model representation is close to the feasible state space, the quadratic program (QP) solver `HPIPM` [97] failed without the proposed preprocessing method in 40% of the simulated scenarios. No QP errors occurred when the track was preprocessed in the same scenarios with the algorithm in Sect. 5.1. Moreover, the preprocessing reduced the online computation time by 23% in the particular scenarios. The novel model formulation in Sect. 5.2 guarantees safe and tight over-approximations of obstacles using sequential quadratic programming (SQP). The tighter over-approximations result in a 30% increased maximum progress in the simulated environment, which involved overtaking three truck-sized obstacles. Additionally, the online computation time was decreased by 6.6% for truck-sized obstacles in the simulated scenarios. The core idea of Sect. 5.2 is to lift the model formulation into two coordinate frames, the FCF and the Cartesian coordinate frame (CCF), which leads to redundant configuration states. Constraints and costs are formulated using the more favorable configuration states of either coordinate frame to achieve superior numerical properties when solving optimization problems.

Chapter 6: Mixed-Integer Optimization for Collision Avoidance. The second main chapter addresses mixed-integer quadratic program (MIQP)-based optimization for motion planning with multiple obstacles by leveraging novel problem formulations and machine learning techniques. The proposed formulations significantly reduce the number of integer variables required for collision-avoidance problems from $\mathcal{O}(NN_{\text{obs}})$ [213], where N is the number of prediction steps and N_{obs} is the number of obstacles, to $\mathcal{O}(N_{\text{obs}})$ for static obstacles and long-term prediction in highway scenarios. The formulations improve the computational efficiency to achieve real-time feasibility. For static obstacles, this is achieved by combining a mixed-integer linear program (MILP) with an SQP homotopy, cf., Sect. 6.1, and for highway predictions by an MIQP formulation in the position-time-lane space, cf., Sect. 6.2. The approach of Sect. 6.1 was evaluated in a real-world competition (`Roborace` [233], Bedford UK, 2021) on embedded hardware of an autonomous race car. The long short term motion planner (LSTMP) of Sect. 6.2 was evaluated in traffic simulations using `CommonRoad` [13] where it achieved a speedup between 2% and 100% compared to A* [8] and an mixed-integer programming-based decision maker (MIP-DM) [213] for different hyper-parameters. Additionally, the LSTMP achieved a closed-loop cost reduction of up to 10% and is Pareto optimal for the trade-off between online-computation time and closed-loop cost compared to A* and the MIP-DM.

Moreover, a combined machine learning and online optimization approach was presented, cf. Sect. 6.3. The trained learning-based predictor replaces the combinatorial part of a mixed-integer solver and drastically reduces the worst-case computation time from 4000 ms to 60 ms and 3000 ms to 30 ms compared to the MIP-DM in two randomized closed-loop traffic scenarios provided by `CommonRoad` [13]. A slight increase of the closed-loop cost of 1.5% and 6.4% is accepted since collisions can still be prevented by deploying a so-called feasibility projector and parallel ensemble networks. Key to the approach is a novel recurrent equivariant deep set (REDS) architecture that provides a powerful inductive bias aligned with the obstacle avoidance problem. In fact, the novel neural network (NN) architecture improves the accuracy of predicting all integer variables, e.g., from 15% to 48% when using seven simultaneous obstacles and 28 prediction steps compared to [62]. The applicability of the planning framework is demonstrated via simulation in interactive environments for highway driving. The framework is general enough to be extended to urban motion planning.

Chapter 7: Collision Avoidance for Autonomous Racing. The final main chapter focuses on obstacle prediction and strategic behavior learning in autonomous racing. The primary contributions are trajectory prediction utilizing bi-level and online optimization and a novel hierarchical RL and MPC architecture that learns strategic maneuvers while maintaining safety. The use of derivative-based bi-level optimization for real-time trajectory prediction is a novel contribution to the field of autonomous racing. It provides superior prediction performance for non-interactive predictions. For example, the standard deviation of signed longitudinal position prediction error after 12 seconds was decreased from 200 meters to 25 meters in simulations utilizing embedded hardware and the Autonomous Racing Graz (ARG) driving stack, when compared to the constant velocity predictor [253]. The computation time of the low-level optimization-based predictor was, on average, below 100 ms with a maximum of 2.7 s on the embedded hardware NVIDIA Drive PX2. When the computation time overshot the real-time sampling time, the prediction of the previous iteration was used, making computational outliers acceptable. The integration of trajectory optimization and RL into a hierarchical motion planning framework in Sect. 7.2 ensures real-time applicability with online computation times of less than 10 ms and high performance in learning strategic behavior in autonomous racing scenarios. Notably, in three different scenarios, the average reward was increased by $8\%, 37\%$ and 96% compared to standard MPC and by 133% and 84% compared to plain RL. In one scenario, the RL agent could not achieve any meaningful performance even after more than 10^6 training steps.

Strengths and Limitations

The strengths of this thesis lie in its computationally efficient formulations for optimization-based motion planning and their real-time applicability, making these methods suitable for real-time motion planning. The empirical performance was evaluated in simulations and, partly, in real-world experiments.

The major limitations of the proposed thesis are listed in the following.

- **Assumptions:** The methods were tested based on particular assumptions, such as known road geometry, deterministic obstacles, perfectly estimated obstacle or ego states, a racing objective of surrounding vehicles (SVs) or stationary policies. The assumptions are reasonably motivated in the specific sections and appropriate for the proposed contribution. However, in real-world integrated systems, the assumptions may not always hold. In this case, the proposed algorithms may need to be adapted.
- **Dependencies on the integrated software stack:** Most algorithms were tested as part of an autonomous driving (AD) software stack, cf., Sect. 4. The different modules within the software stack all have certain characteristics and influence each other during performance measurements. For instance, the performance of the lowest-level controller significantly influences the performance of the motion planner. Particularly, if any module of the AD software stack performs poorly, the performance evaluation of one individual component, such as the planner, is challenging. In order to omit any cross-dependencies between modules, the AD stacks were kept as simple as possible but as realistic as necessary. Moreover, a major effort was made to achieve high performance in each necessary module.
- **Specific Environments:** In this thesis, a great effort was taken to verify the proposed contributions on a wide range of environments. In case simulations were used to evaluate the performance of an algorithm, the environment was randomized to a great extent. For instance, the curvature or the road boundaries of reference tracks, SV parameters, or initial states were randomized to create highly random road environments. However, the generated scenarios do not cover all possible scenarios, and it cannot be excluded that there exists a particular environment in which the proposed algorithms do not perform superiorly.
- **Solving hard optimization problems:** Most algorithms rely on nonlinear program (NLP) or MIQP problem formulations, utilizing solvers such as `acados` [291] or `Gurobi` [114]. Due to the complex problem classes these solvers can treat, they have few guarantees of solving the problem correctly. Even though, in practice, robust empirical convergence is observed. It remains a more fundamental question whether less expressive

convex formulations and related solvers are inevitable in guaranteeing reliable convergence and a limited worst-case computation time or if the empirically evaluated performance of more sophisticated solvers is sufficient for real-world systems.

- **Linear and convex mixed-integer formulations:** The reliance on MIQP formulations for combinatorial obstacle avoidance of Chapter 6 limits the expressiveness of the models. The equality constraints within the optimization problem related to the model are required to be linear, and constraints need to be convex quadratic. Notably, it is possible to add model complexity to a linear model within the proposed MIQP-based formulation. Specifically, nonlinear functions and convex sets can be approximated by adding continuous and integer variables [304], which is also used in several formulations of Sect. 6.2. Even though nonlinear functions and nonconvex sets can be approximated by MIQP formulations, they cannot be formulated exactly. Alternatively, nonlinear functions can be directly used within mixed-integer nonlinear programmings (MINLPs) formulations. MINLP formulations treat nonlinearities differently and, therefore, exhibit the possibility of improving the overall performance.

- **Mixed-integer solvers for embedded systems:** Up to this point, only few commercial high-performance MIQP solvers comparable to, e.g., `Gurobi` [114], for embedded real-time critical hardware are available. Even though research is performed in this direction, cf. [18, 212, 269] and high-performance open-source solvers exist, such as `HiGHS` [125].

Future Directions

The proposed methods show outstanding performance in the particular tested environments. Nonetheless, the research gave rise to several possible extensions and new research questions.

- **Comparison of CCF and FCF model formulations:** As mentioned in Sect. 5.2 and the introduction Sect. 3.7, an alternative to the FCF model formulation uses the CCF and an auxiliary path variable. Up to this point, no thorough comparison is known to the author of this thesis, making the choice of the coordinate frame in many publications rather based on intuitive arguments. A theoretical and practical comparison of real-world embedded systems would make the decision of the coordinate frame more informed for future researchers and practitioners.

- **Evaluation of MINLP formulations:** The methods of Chapter 6 are all based on MIQP formulations, which limit the expressiveness of the model. It remains an interesting research question if state-of-the-art MINLP solvers can solve nonlinear formulations more efficiently.

- **Advancing motion planning algorithms to game-theoretic foundations:** The proposed methods do not consider game-theoretic considerations due to their sophisticated implications for motion planning algorithms. However, ignoring game-theoretic considerations limits the performance due to, e.g., an increased conservativeness. Particularly for autonomous racing, even a rigorous mathematical description of the "racing game" is missing due to diverse rules in real-world competitions. A formulation of autonomous racing as a game-theoretic problem would allow the development of more rigorous algorithms.

- **Multi-agent reinforcement learning and MPC:** This direction is conditioned on the previous point of a game-theoretic formulation of the autonomous racing problem. The hierarchical RL and MPC approach of Sect. 7.2 only considers static policies of the opponents, making it a single-agent RL problem. Assuming the other agents are rational "players", the setting becomes a multi-agent RL problem. Combining multi-agent RL in the hierarchical setting of Sect. 7.2 would be an exciting direction for future research.

- **Unifying MPC and RL:** Both, the implicit online-optimization approach of MPC and the explicit model-free RL method achieve outstanding performance in many real-world tasks. Often, the underlying goal is equal, such as in time-optimal racing. Interestingly, the advantages of both methods are "orthogonal", meaning the weaknesses of one approach are the strengths of the other. Unifying both frameworks into a partly model-based and model-free learning approach is a fascinating direction for future research.

Bibliography

[1] DARPA Urban Challenge, https://www.darpa.mil/about-us/timeline/darpa-urban-challenge, 2007.

[2] Baidu Apollo team, Apollo: Open Source Autonomous Driving, https://github.com/ApolloAuto/apollo, 2017.

[3] ARG - Autonomous Racing Graz, https://autonomousracing.ai/, 2024.

[4] Eclipse Foundation. Eclipse Cyclone DDS, https://cyclonedds.io/, 2024.

[5] Eclipse Foundation. Eclipse iceoryx - An inter-process communication middleware, https://iceoryx.io/v2.0.2/, 2024.

[6] IAC - Indy Autonomous Challenge, https://www.indyautonomouschallenge.com/, 2024.

[7] Agrawal, A., Amos, B., Barratt, S. T., Boyd, S. P., Diamond, S., and Kolter, J. Z. Differentiable Convex Optimization Layers. *CoRR abs/1910.12430* (2019).

[8] Ajanovic, Z., Lacevic, B., Shyrokau, B., Stolz, M., and Horn, M. Search-Based Optimal Motion Planning for Automated Driving. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (2018), pp. 4523–4530.

[9] Akiba, T., Sano, S., Yanase, T., Ohta, T., and Koyama, M. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining* (2019), pp. 2623–2631.

[10] Albersmeyer, J., and Diehl, M. The Lifted Newton Method and Its Application in Optimization. *SIAM Journal on Optimization 20*, 3 (Jan. 2010), 1655–1684.

[11] Alcalá, E., Puig, V., and Quevedo, J. LPV-MP planning for autonomous racing vehicles considering obstacles. *Robotics and Autonomous Systems 124* (2020), 103392.

[12] Alcon, M., Tabani, H., Kosmidis, L., Mezzetti, E., Abella, J., and Cazorla, F. J. Timing of Autonomous Driving Software: Problem Analysis and Prospects for Future Solutions. *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)* (2020), 267–280.

[13] Althoff, M., Koschi, M., and Manzinger, S. CommonRoad: Composable benchmarks for motion planning on roads. In *IEEE Intelligent Vehicles Symposium (IV)* (2017), pp. 719–726.

[14] Andersson, J. A. E., Gillis, J., Horn, G., Rawlings, J. B., and Diehl, M. CasADi – A software framework for nonlinear optimization and optimal control. *Mathematical Programming Computation 11*, 1 (2019), 1–36.

[15] Anstreicher, K. M. Linear programming: interior point methodsLinear Programming: Interior Point Methods. In *Encyclopedia of Optimization*, C. A. Floudas and P. M. Pardalos, Eds. Springer US, Boston, MA, 2001, pp. 1279–1281.

[16] Appa, G., Pitsoulis, L., Williams, H. P., and Hillier, F. S., Eds. *Handbook on Modelling for Discrete Optimization*, vol. 88 of *International Series in Operations Research & Management Science*. Kluwer Academic Publishers, Boston, MA, 2006.

[17] Aradi, S. Survey of Deep Reinforcement Learning for Motion Planning of Autonomous Vehicles. *IEEE Transactions on Intelligent Transportation Systems 23*, 2 (Feb. 2022), 740–759.

[18] Arnström, D., and Axehill, D. BnB-DAQP: A Mixed-Integer QP Solver for Embedded Applications. *IFAC-PapersOnLine 56*, 2 (Jan. 2023), 7420–7427.

[19] Arrizabalaga, J., and Ryll, M. Spatial motion planning with Pythagorean Hodograph curves. In *IEEE 61st Conference on Decision and Control (CDC)* (Dec. 2022), pp. 2047–2053. ISSN: 2576-2370.

[20] Arrizabalaga, J., and Ryll, M. Towards Time-Optimal Tunnel-Following for Quadrotors. In *International Conference on Robotics and Automation (ICRA)* (May 2022), pp. 4044–4050.

[21] Arrizabalaga, J., and Ryll, M. Pose-Following with Dual Quaternions. In *62nd IEEE Conference on Decision and Control (CDC)* (2023), pp. 5959–5966.

[22] Arrizabalaga, J., and Ryll, M. SCTOMP: Spatially Constrained Time-Optimal Motion Planning. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (Detroit, MI, USA, Oct. 2023), IEEE, pp. 4827–4834.

[23] Arslan, O., Berntorp, K., and Tsiotras, P. Sampling-based algorithms for optimal motion planning using closed-loop prediction. In *IEEE International Conference on Robotics and Automation (ICRA)* (2017), pp. 4991–4996.

[24] Aspvall, B., and Stone, R. E. Khachiyan's linear programming algorithm. *Journal of Algorithms 1*, 1 (Mar. 1980), 1–13.

[25] Bast, H., Delling, D., Goldberg, A., Müller-Hannemann, M., Pajor, T., Sanders, P., Wagner, D., and Werneck, R. Route Planning in Transportation Networks. vol. 9220. Nov. 2016, pp. 19–80.

[26] Batkovic, I., Rosolia, U., Zanon, M., and Falcone, P. A Robust Scenario MPC Approach for Uncertain Multi-Modal Obstacles. *IEEE Control Systems Letters 5*, 3 (July 2021), 947–952.

[27] Baumann, N., Ghignone, E., Kühne, J., Bastuck, N., Becker, J., Imholz, N., Kränzlin, T., Lim, T. Y., Lötscher, M., Schwarzenbach, L., Tognoni, L., Vogt, C., Carron, A., and Magno, M. ForzaETH Race Stack – Scaled Autonomous Head-to-Head Racing on Fully Commercial off-the-Shelf Hardware, Mar. 2024. arXiv:2403.11784 [cs, eess].

[28] Bellman, R. Dynamic programming. *Science 153*, 3731 (1966), 34–37. Publisher: American Association for the Advancement of Science.

[29] Belotti, P., Kirches, C., Leyffer, S., Linderoth, J., Luedtke, J., and Mahajan, A. Mixed-integer nonlinear optimization. *Acta Numerica 22* (May 2013), 1–131.

[30] Bender, P., Tas, O. S., Ziegler, J., and Stiller, C. The combinatorial aspect of motion planning: Maneuver variants in structured environments. In *IEEE Intelligent Vehicles Symposium (IV)* (2015), pp. 1386–1392.

[31] Bergman, K., and Axehill, D. Combining Homotopy Methods and Numerical Optimal Control to Solve Motion Planning Problems. In *IEEE Intelligent Vehicles Symposium (IV)* (June 2018), pp. 347–354.

[32] Bergman, K., Ljungqvist, O., and Axehill, D. Improved Path Planning by Tightly Combining Lattice-Based Path Planning and Optimal Control. *IEEE Transactions on Intelligent Vehicles* (2020), 1–1.

[33] Bergman, K., Ljungqvist, O., Glad, T., and Axehill, D. An Optimization-Based Receding Horizon Trajectory Planning Algorithm. *IFAC-PapersOnLine 53*, 2 (Jan. 2020), 15550–15557.

[34] Berntorp, K., Weiss, A., and Di Cairano, S. Integer ambiguity resolution by mixture Kalman filter for improved GNSS precision. *IEEE Transactions on Aerospace and Electronic Systems 56*, 4 (2020), 3170–3181.

[35] Bertsekas, D. *Reinforcement Learning and Optimal Control*, first edition ed. Athena Scientific, Belmont, Massachusetts, July 2019.

[36] Bertsekas, D., and Castanon, D. Adaptive aggregation methods for infinite horizon dynamic programming. *IEEE Transactions on Automatic Control 34*, 6 (1989), 589–598.

[37] Bertsekas, D. P. Dynamic Programming and Suboptimal Control: A Survey from ADP to MPC*. *European Journal of Control 11*, 4 (Jan. 2005), 310–334.

[38] Bertsekas, D. P., and Tsitsiklis, J. N. *Neuro-dynamic programming.* Optimization and neural computation series. Athena Scientific, Belmont, Mass, 1996.

[39] Bertsimas, D., and Stellato, B. The voice of optimization. *Machine Learning 110*, 2 (Feb. 2021), 249–277.

[40] Bertsimas, D., and Stellato, B. Online Mixed-Integer Optimization in Milliseconds. *INFORMS J. on Computing 34*, 4 (July 2022), 2229–2248.

[41] Betz, J., Zheng, H., Liniger, A., Rosolia, U., Karle, P., Behl, M., Krovi, V., and Mangharam, R. Autonomous Vehicles on the Edge: A Survey on Autonomous Vehicle Racing. *IEEE Open Journal of Intelligent Transportation Systems 3* (2022), 458–488.

[42] Bishop, R. L. There is More than One Way to Frame a Curve. *The American Mathematical Monthly 82*, 3 (1975), 246–251.

[43] Black, K., Janner, M., Du, Y., Kostrikov, I., and Levine, S. Training Diffusion Models with Reinforcement Learning. *CoRR 2305.13301* (2023).

[44] Block, A., Jadbabaie, A., Pfrommer, D., Simchowitz, M., and Tedrake, R. Provable Guarantees for Generative Behavior Cloning: Bridging Low-Level Stability and High-Level Behavior. In *Advances in Neural Information Processing Systems* (2023), A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, Eds., vol. 36, Curran Associates, Inc., pp. 48534–48547.

[45] Bock, H. G., and Plitt, K. J. A Multiple Shooting Algorithm for Direct Solution of Optimal Control Problems. In *IFAC* (1984), Pergamon Press, pp. 242–247.

[46] Boyd, S., and Vandenberghe, L. *Convex Optimization.* Cambridge University Press, USA, 2004.

[47] Braghin, F., Cheli, F., Melzi, S., and Sabbioni, E. Race driver model. *Computers & Structures 86*, 13-14 (July 2008), 1503–1516.

[48] Brito, B., Everett, M., How, J. P., and Alonso-Mora, J. Where to go Next: Learning a Subgoal Recommendation Policy for Navigation in Dynamic Environments. *IEEE Robotics and Automation Letters 6* (2021), 4616–4623.

[49] Brito, B., Floor, B., Ferranti, L., and Alonso-Mora, J. Model Predictive Contouring Control for Collision Avoidance in Unstructured Dynamic Environments, 2020.

[50] Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. OpenAI Gym, 2016.

[51] Bronstein, E., Palatucci, M., Notz, D., White, B., Kuefler, A., Lu, Y., Paul, S., Nikdel, P., Mougin, P., Chen, H., Fu, J., Abrams, A., Shah, P., Racah, E., Frenkel, B., Whiteson, S., and Anguelov, D. Hierarchical Model-Based Imitation Learning for Planning in Autonomous Driving. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (Oct. 2022), pp. 8652–8659.

[52] Brossette, S., and Wieber, P.-B. Collision avoidance based on separating planes for feet trajectory generation. In *IEEE-RAS 17th International Conference on Humanoid Robotics (Humanoids)* (2017), pp. 509–514.

[53] BROYDEN, C. G. The Convergence of a Class of Double-rank Minimization Algorithms 1. General Considerations. *IMA Journal of Applied Mathematics 6*, 1 (Mar. 1970), 76–90.

[54] Brummelen, J. V., O'Brien, M., Gruyer, D., and Najjaran, H. Autonomous vehicle perception: The technology of today and tomorrow. *Transportation Research Part C: Emerging Technologies 89* (2018), 384–406.

[55] Brunke, L., Greeff, M., Hall, A. W., Yuan, Z., Zhou, S., Panerati, J., and Schoellig, A. P. Safe Learning in Robotics: From Learning-Based Control to Safe Reinforcement Learning. *Annual Review of Control, Robotics, and Autonomous Systems 5*, 1 (2022), 411–444.

[56] Buckman, N., Pierson, A., Schwarting, W., Karaman, S., and Rus, D. Sharing is Caring: Socially-Compliant Autonomous Intersection Negotiation. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (2019), pp. 6136–6143.

[57] Buckman, N., Schwarting, W., Karaman, S., and Rus, D. Semi-Cooperative Control for Autonomous Emergency Vehicles. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (2021), pp. 7052–7059.

[58] Burger, C., and Lauer, M. Cooperative Multiple Vehicle Trajectory Planning using MIQP. In *21st International Conference on Intelligent Transportation Systems (ITSC)* (2018), pp. 602–607.

[59] Burger, C., Yan, S., Burgard, W., and Stiller, C. Interaction-Aware Motion Planning as a Game. In *Cooperatively Interacting Vehicles: Methods and Effects of Automated Cooperation in Traffic*, C. Stiller, M. Althoff, C. Burger, B. Deml, L. Eckstein, and F. Flemisch, Eds. Springer International Publishing, Cham, 2024, pp. 203–229.

[60] Buyval, A., Gabdulin, A., Mustafin, R., and Shimchik, I. Deriving overtaking strategy from nonlinear model predictive control for a race car. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (2017), pp. 2623–2628.

[61] Capobianco, S., Millefiori, L. M., Forti, N., Braca, P., and Willett, P. Deep Learning Methods for Vessel Trajectory Prediction based on Recurrent Neural Networks. *CoRR abs/2101.02486* (2021).

[62] Cauligi, A., Chakrabarty, A., Cairano, S. D., and Quirynen, R. PRISM: Recurrent Neural Networks and Presolve Methods for Fast Mixed-integer Optimal Control. In *Proceedings of The 4th Annual Learning for Dynamics and Control Conference* (June 2022), R. Firoozi, N. Mehr, E. Yel, R. Antonova, J. Bohg, M. Schwager, and M. Kochenderfer, Eds., vol. 168 of *Proceedings of Machine Learning Research*, PMLR, pp. 34–46.

[63] Cauligi, A., Culbertson, P., Schmerling, E., Schwager, M., Stellato, B., and Pavone, M. CoCo: Online Mixed-Integer Control Via Supervised Learning. *IEEE Robotics and Automation Letters 7*, 2 (2022), 1447–1454.

[64] Chan, C. C., and Cheng, M. Vehicle Traction Motorsvehicletraction motors. In *Encyclopedia of Sustainability Science and Technology*, R. A. Meyers, Ed. Springer New York, New York, NY, 2012, pp. 11522–11552.

[65] Chen, S., and Chen, H. MPC-based path tracking with PID speed control for autonomous vehicles. *IOP Conference Series: Materials Science and Engineering 892*, 1 (2020).

[66] Chougule, A., Chamola, V., Sam, A., Yu, F. R., and Sikdar, B. A Comprehensive Review on Limitations of Autonomous Driving and Its Impact on Accidents and Collisions. *IEEE Open Journal of Vehicular Technology 5* (2024), 142–161.

[67] Claussmann, L., Revilloud, M., Gruyer, D., and Glaser, S. A Review of Motion Planning for Highway Autonomous Driving. *IEEE Transactions on Intelligent Transportation Systems 21*, 5 (2020), 1826–1848.

[68] Cleac'h, S. L., Schwager, M., and Manchester, Z. LUCIDGames: Online Unscented Inverse Dynamic Games for Adaptive Trajectory Prediction and Planning. *CoRR abs/2011.08152* (2020).

[69] Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. *Introduction to Algorithms, third edition*, third edition ed. The MIT Press, Cambridge, Mass, July 2009.

[70] Curiel, I. *Cooperative Game Theory and Applications: Cooperative Games Arising from Combinatorial Optimization Problems*, softcover reprint of hardcover 1st ed. 1997 edition ed. Springer US, Boston, Feb. 2010.

[71] Dakin, R. J. A tree-search algorithm for mixed integer programming problems. *The Computer Journal 8*, 3 (Jan. 1965), 250–255.

[72] Dantzig, G. B. *The Simplex Method.* RAND Corporation, Santa Monica, CA, 1956.

[73] de Klerk, E., and Pasechnik, D. V. Approximation of the Stability Number of a Graph via Copositive Programming. *SIAM Journal on Optimization 12*, 4 (2002), 875–892.

[74] Debrouwere, F., Van Loock, W., Pipeleers, G., Dinh, Q. T., Diehl, M., De Schutter, J., and Swevers, J. Time-Optimal Path Following for Robots With Convex–Concave Constraints Using Sequential Convex Programming. *IEEE Transactions on Robotics 29*, 6 (2013), 1485–1495.

[75] Deichmann, J., Ebel, E., Heineke, K., Heuss, R., Kellner, M., and Steiner, F. The future of autonomous vehicles (AV) \textbar McKinsey, Jan. 2023.

[76] Delling, D., Goldberg, A. V., Nowatzyk, A., and Werneck, R. F. PHAST: Hardware-accelerated shortest path trees. *Journal of Parallel and Distributed Computing 73*, 7 (July 2013), 940–952.

[77] Deolasee, S., Lin, Q., Li, J., and Dolan, J. M. Spatio-temporal Motion Planning for Autonomous Vehicles with Trapezoidal Prism Corridors and Bézier Curves. In *American Control Conference, San Diego, CA, USA* (2023), IEEE, pp. 3207–3214.

[78] Di Cairano, S., and Kolmanovsky, I. V. Real-time optimization and model predictive control for aerospace and automotive applications. Amer. Control Conf., pp. 2392–2409.

[79] Diamond, S., and Boyd, S. CVXPY: A Python-embedded modeling language for convex optimization. *Journal of Machine Learning Research 17*, 83 (2016), 1–5.

[80] Diehl, M., Bock, H. G., and Schlöder, J. P. A Real-Time Iteration Scheme for Nonlinear Optimization in Optimal Feedback Control. *SIAM Journal on Control and Optimization 43*, 5 (2005), 1714–1736.

[81] Dijkstra, E. W. A note on two problems in connexion with graphs. *Numerische Mathematik 1*, 1 (Dec. 1959), 269–271.

[82] Domahidi, A., Chu, E., and Boyd, S. ECOS: An SOCP solver for embedded systems. In *European Control Conference (ECC)* (2013), pp. 3071–3076.

[83] Domahidi, A., and Jerez, J. FORCES Professional, 2014.

[84] Dugoff, H., Fancher, P. S., and Segel, L. An Analysis of Tire Traction Properties and Their Influence on Vehicle Dynamic Performance. *SAE Transactions 79* (1970), 1219–1243.

[85] Duijkeren, N. v., Keviczky, T., Nilsson, P., and Laine, L. Real-Time NMPC for Semi-Automated Highway Driving of Long Heavy Vehicle Combinations. *IFAC-PapersOnLine 48*, 23 (Jan. 2015), 39–46.

[86] Eilbrecht, J., and Stursberg, O. Challenges of Trajectory Planning with Integrator Models on Curved Roads. *IFAC-PapersOnLine 53*, 2 (2020), 15588–15595.

[87] Eiras, F., Hawasly, M., Albrecht, S. V., and Ramamoorthy, S. A Two-Stage Optimization-Based Motion Planner for Safe Urban Driving. *IEEE Transactions on Robotics 38*, 2 (2022), 822–834.

[88] Erdmann, J. SUMO's Lane-Changing Model. In *Modeling Mobility with Open Data* (Cham, 2015), M. Behrisch and M. Weber, Eds., Springer International Publishing, pp. 105–123.

[89] Ersal, T., Kolmanovsky, I., Masoud, N., Ozay, N., Scruggs, J., Vasudevan, R., and Orosz, G. Connected and automated road vehicles: state of the art and future challenges. *Vehicle system dynamics 58*, 5 (2020), 672–704.

[90] Espinoza, J. L. V., Liniger, A., Schwarting, W., Rus, D., and Gool, L. V. Deep Interactive Motion Prediction and Planning: Playing Games with Motion Prediction Models. In *Proceedings of The 4th Annual Learning for Dynamics and Control Conference* (June 2022), R. Firoozi, N. Mehr, E. Yel, R. Antonova, J. Bohg, M. Schwager, and M. Kochenderfer, Eds., vol. 168 of *Proceedings of Machine Learning Research*, PMLR, pp. 1006–1019.

[91] Esterle, K., Kessler, T., and Knoll, A. Optimal Behavior Planning for Autonomous Driving: A Generic Mixed-Integer Formulation. In *IEEE Intelligent Vehicles Symposium (IV)* (2020), pp. 1914–1921.

[92] Evens, B., Schuurmans, M., and Patrinos, P. Learning MPC for Interaction-Aware Autonomous Driving: A Game-Theoretic Approach. In *European Control Conference (ECC)* (2022), IEEE, pp. 34–39.

[93] Evestedt, N., Ward, E., Folkesson, J., and Axehill, D. Interaction aware trajectory planning for merge scenarios in congested traffic situations. In *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)* (Nov. 2016), pp. 465–472. ISSN: 2153-0017.

[94] Ferreau, H. J., Kirches, C., Potschka, A., Bock, H. G., and Diehl, M. qpOASES: A parametric active-set algorithm for quadratic programming. *Mathematical Programming Computation 6*, 4 (2014), 327–363.

[95] Fletcher, R. A new approach to variable metric algorithms. *The Computer Journal 13*, 3 (Jan. 1970), 317–322.

[96] Frasch, J. V., Gray, A., Zanon, M., Ferreau, H. J., Sager, S., Borrelli, F., and Diehl, M. An auto-generated nonlinear MPC algorithm for real-time obstacle avoidance of ground vehicles. In *European Control Conference (ECC)* (Zurich, July 2013), IEEE, pp. 4136–4141.

[97] Frison, G., and Diehl, M. HPIPM: a high-performance quadratic programming framework for model predictive control. *IFAC-PapersOnLine 53*, 2 (Jan. 2020), 6563–6569.

[98] Fujimoto, S., van Hoof, H., and Meger, D. Addressing Function Approximation Error in Actor-Critic Methods. In *Proceedings of the 35th International Conference on Machine Learning* (July 2018), J. Dy and A. Krause, Eds., vol. 80 of *Proceedings of Machine Learning Research*, PMLR, pp. 1587–1596.

[99] Garey, M. R., and Johnson, D. S. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman, 1979.

[100] Goerzen, C., Kong, Z., and Mettler, B. A Survey of Motion Planning Algorithms from the Perspective of Autonomous UAV Guidance. *Journal of Intelligent and Robotic Systems 57* (Nov. 2010), 65–100.

[101] Goldstein, H. *Classical Mechanics.* Addison-Wesley, 1980.

[102] Goli, S. A., Far, B. H., and Fapojuwo, A. O. Vehicle Trajectory Prediction with Gaussian Process Regression in Connected Vehicle Environment. In *IEEE Intelligent Vehicles Symposium (IV)* (June 2018), pp. 550–555.

[103] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. Generative Adversarial Nets. In *Advances in Neural Information Processing Systems* (2014), vol. 27, Curran Associates, Inc.

[104] Graf, M. *Methode zur Erstellung und Absicherung einer modellbasierten Sollvorgabe für Fahrdynamikregelsysteme.* Ph.D. Thesis, Technische Universität München, Munich, Nov. 2014.

[105] Grandesso, G., Alboni, E., Papini, G. P. R., Wensing, P. M., and Prete, A. D. CACTO: Continuous Actor-Critic With Trajectory Optimization—Towards Global Optimality. *IEEE Robotics and Automation Letters 8*, 6 (June 2023), 3318–3325.

[106] Grant, M., and Boyd, S. CVX: Matlab Software for Disciplined Convex Programming, version 2.1, Mar. 2014.

[107] Greatwood, C., and Richards, A. G. Reinforcement learning and model predictive control for robust embedded quadrotor guidance and control. *Autonomous Robots 43*, 7 (Oct. 2019), 1681–1693.

[108] Greiff, M., Di Cairano, S., Kim, K. J., and Berntorp, K. A System-Level Cooperative Multiagent GNSS Positioning Solution. *IEEE Transactions on Control Systems Technology* (2023).

[109] Gronauer, S., and Diepold, K. Multi-agent deep reinforcement learning: a survey. *Artificial Intelligence Review 55*, 2 (Feb. 2022), 895–943.

[110] Gros, S., and Zanon, M. Data-Driven Economic NMPC Using Reinforcement Learning. *IEEE Transactions on Automatic Control 65*, 2 (Feb. 2020), 636–648.

[111] Gros, S., Zanon, M., Quirynen, R., Bemporad, A., and Diehl, M. From Linear to Nonlinear MPC: bridging the gap via the Real-Time Iteration. *International Journal of Control 93*, 1 (2020), 62–80.

[112] Guanetti, J., Kim, Y., and Borrelli, F. Control of connected and automated vehicles: State of the art and future challenges. *Annual Reviews in Control 45* (2018), 18–40.

[113] Guiggiani, M. *The Science of Vehicle Dynamics*, 3 ed. Springer Cham, Nov. 2023.

[114] Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2023.

[115] Görges, D. Relations between Model Predictive Control and Reinforcement Learning. *IFAC-PapersOnLine 50*, 1 (July 2017), 4920–4928.

[116] Haarnoja, T., Tang, H., Abbeel, P., and Levine, S. Reinforcement learning with deep energy-based policies. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70* (2017), ICML'17, JMLR.org, pp. 1352–1361.

[117] Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning* (2018), PMLR, pp. 1861–1870.

[118] Hansen, L., and Salamon, P. Neural network ensembles. *IEEE Transactions on Pattern Analysis and Machine Intelligence 12*, 10 (1990), 993–1001.

[119] Hasselt, H. Double Q-learning. In *Advances in Neural Information Processing Systems* (2010), J. Lafferty, C. Williams, J. Shawe-Taylor, R. Zemel, and A. Culotta, Eds., vol. 23, Curran Associates, Inc.

[120] Heilmeier, A., Wischnewski, A., Hermansdorfer, L., Betz, J., Lienkamp, M., and Lohmann, B. Minimum curvature trajectory planning and control for an autonomous race car. *Vehicle System Dynamics 58*, 10 (Oct. 2020), 1497–1527.

[121] Hermans, B., Themelis, A., and Patrinos, P. QPALM: A Newton-type Proximal Augmented Lagrangian Method for Quadratic Programs. In *IEEE 58th Conference on Decision and Control (CDC)* (2019), pp. 4325–4330.

[122] Hess, D., Lattarulo, R., Pérez, J., Schindler, J., Hesse, T., and Köster, F. Fast Maneuver Planning for Cooperative Automated Vehicles. In *21st International Conference on Intelligent Transportation Systems (ITSC)* (2018), pp. 1625–1632.

[123] Hicks, G. A., and Ray, W. H. Approximation methods for optimal control synthesis. *The Canadian Journal of Chemical Engineering 49*, 4 (1971), 522–528.

[124] Hoschek, J., and Lasser, D. *Fundamentals of Computer-aided Geometric Design.* Jones and Bartlett, 1993.

[125] Huangfu, Q., and Hall, J. A. J. Parallelizing the dual revised simplex method. *Mathematical Programming Computation 10*, 1 (Mar. 2018), 119–142.

[126] Huegle, M., Kalweit, G., Mirchevska, B., Werling, M., and Boedecker, J. Dynamic Input for Deep Reinforcement Learning in Autonomous Driving. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (2019), pp. 7566–7573.

[127] Ip, A., Irio, L., and Oliveira, R. Vehicle Trajectory Prediction based on LSTM Recurrent Neural Networks. In *IEEE 93rd Vehicular Technology Conference (VTC2021-Spring)* (2021), pp. 1–5.

[128] Janeček, F., Klaučo, M., Kalúz, M., and Kvasnica, M. OPTIPLAN: A Matlab Toolbox for Model Predictive Control with Obstacle Avoidance. *IFAC-PapersOnLine 50*, 1 (2017), 531 – 536.

[129] Jaynes, E. T. Information Theory and Statistical Mechanics. *Phys. Rev. 106*, 4 (May 1957), 620–630. Publisher: American Physical Society.

[130] Jeroslow, R. C. There Cannot be any Algorithm for Integer Programming with Quadratic Constraints. *Operations Research 21*, 1 (1973), 221–224.

[131] Jo, E., Sunwoo, M., and Lee, M. Vehicle Trajectory Prediction Using Hierarchical Graph Neural Network for Considering Interaction among Multimodal Maneuvers. *Sensors 21*, 16 (2021).

[132] Johansen, T. A. Introduction to Nonlinear Model Predictive Control and Moving Horizon Estimation. In *Selected Topics on Constrained and Nonlinear Control*, M. Huba, S. Skogestad, M. Fikar, M. Hovd, T. A. Johansen, and B. Rohal'-Ilkiv, Eds. 2011.

[133] Johnson, J., and Hauser, K. Optimal longitudinal control planning with moving obstacles. In *IEEE Intelligent Vehicles Symposium (IV)* (2013), pp. 605–611.

[134] Kammel, S., Ziegler, J., Pitzer, B., Werling, M., Gindele, T., Jagzent, D., Schöder, J., Thuy, M., Goebl, M., von Hundelshausen, F., Pink, O., Frese, C., and Stiller, C. Team AnnieWAY's Autonomous System for the DARPA Urban Challenge 2007. In *The DARPA Urban Challenge: Autonomous Vehicles in City Traffic*, M. Buehler, K. Iagnemma, and S. Singh, Eds. Springer, Berlin, Heidelberg, 2009, pp. 359–391.

[135] Kapania, N. R., Subosits, J., and Christian Gerdes, J. A Sequential Two-Step Algorithm for Fast Generation of Vehicle Racing Trajectories. *Journal of Dynamic Systems, Measurement, and Control 138*, 9 (Sept. 2016), 091005.

[136] Kappen, H. J. Linear Theory for Control of Nonlinear Stochastic Systems. *Physical Review Letters 95*, 20 (Nov. 2005), 200201.

[137] Karaman, S., and Frazzoli, E. Sampling-Based Algorithms for Optimal Motion Planning. *Int. J. Rob. Res. 30*, 7 (June 2011), 846–894.

[138] Kato, S., Tokunaga, S., Maruyama, Y., Maeda, S., Hirabayashi, M., Kitsukawa, Y., Monrroy, A., Ando, T., Fujii, Y., and Azumi, T. Autoware on Board: Enabling Autonomous Vehicles with Embedded Systems. In *ACM/IEEE 9th International Conference on Cyber-Physical Systems (ICCPS)* (Porto, Apr. 2018), IEEE, pp. 287–296.

[139] Kavraki, L., Svestka, P., Latombe, J.-C., and Overmars, M. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation 12*, 4 (1996), 566–580.

[140] Kessler, T., Esterle, K., and Knoll, A. Linear Differential Games for Cooperative Behavior Planning of Autonomous Vehicles Using Mixed-Integer Programming. In *59th IEEE Conference on Decision and Control (CDC)* (2020), pp. 4060–4066.

[141] Kessler, T., Esterle, K., and Knoll, A. Mixed-Integer Motion Planning on German Roads Within the Apollo Driving Stack. *IEEE Transactions on Intelligent Vehicles 8*, 1 (2023), 851–867.

[142] Kesting, A., Treiber, M., and Helbing, D. General Lane-Changing Model MOBIL for Car-Following Models. *Transportation Research Record 1999*, 1 (Jan. 2007), 86–94. Publisher: SAGE Publications Inc.

[143] Khachiyan, L. G. Polynomial algorithms in linear programming. *USSR Computational Mathematics and Mathematical Physics 20*, 1 (1980), 53–72.

[144] Khalil, E. B., Morris, C., and Lodi, A. MIP-GNN: A Data-Driven Framework for Guiding Combinatorial Solvers. In *Proceedings of the 36th AAAI Conference on Artificial Intelligence* (2022).

[145] Khorkov, A., and Galiev, S. Optimization of a k-covering of a bounded set with circles of two given radii. *Open Computer Science 11*, 1 (2021), 232–240.

[146] Kim, D., Kim, G., Kim, H., and Huh, K. A Hierarchical Motion Planning Framework for Autonomous Driving in Structured Highway Environments. *IEEE Access 10* (2022), 20102–20117.

[147] Kingma, D. P., and Ba, J. Adam: A Method for Stochastic Optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings* (2015), Y. Bengio and Y. LeCun, Eds.

[148] Kiran, B. R., Sobh, I., Talpaert, V., Mannion, P., Sallab, A. A. A., Yogamani, S., and Pérez, P. Deep Reinforcement Learning for Autonomous Driving: A Survey. *IEEE Transactions on Intelligent Transportation Systems 23*, 6 (June 2022), 4909–4926.

[149] Klančar, G., Seder, M., Blažič, S., Škrjanc, I., and Petrović, I. Drivable Path Planning Using Hybrid Search Algorithm Based on E* and Bernstein–Bézier Motion Primitives. *IEEE Transactions on Systems, Man, and Cybernetics: Systems 51*, 8 (2021), 4868–4882.

[150] Klischat, M., Dragoi, O., Eissa, M., and Althoff, M. Coupling SUMO with a Motion Planning Framework for Automated Vehicles. In *SUMO User Conference* (2019), pp. 1–9.

[151] Klöser, D., Schöls, T., Sartor, T., Zanelli, A., Prison, G., and Diehl, M. NMPC for Racing Using a Singularity-Free Path-Parametric Model with Obstacle Avoidance. *IFAC-PapersOnLine 53*, 2 (Jan. 2020), 14324–14329.

[152] Kobilarov, M. Cross-entropy motion planning. *The International Journal of Robotics Research 31*, 7 (2012), 855–871.

[153] Kong, J., Pfeiffer, M., Schildbach, G., and Borrelli, F. Kinematic and dynamic vehicle models for autonomous driving control design. In *IEEE Intelligent Vehicles Symposium (IV)* (2015), pp. 1094–1099.

[154] Kozlov, M. K., Tarasov, S. P., and Khachiyan, L. The polynomial solvability of convex quadratic programming. *{USSR} Computational Mathematics and Mathematical Physics 20* (1980), 223–228.

[155] Krauss, S. *Microscopic Modeling of Traffic Flow: Investigation of Collision Free Vehicle Dynamics.* PhD thesis, University of Cologne, Apr. 1998.

[156] Kronqvist, J., Bernal, D. E., Lundell, A., and Grossmann, I. E. A review and comparison of solvers for convex MINLP. *Optimization and Engineering 20*, 2 (June 2019), 397–455.

[157] Lam, D., Manzie, C., and Good, M. Model Predictive Contouring Control. In *IEEE Conference on Decision and Control (CDC)* (2010).

[158] Laurense, V. A., and Gerdes, J. C. Long-Horizon Vehicle Motion Planning and Control Through Serially Cascaded Model Complexity. *IEEE Transactions on Control Systems Technology 30*, 1 (2022), 166–179.

[159] LaValle, S. M. *Planning Algorithms.* Cambridge University Press, USA, 2006.

[160] Le Cleac'h, S., Schwager, M., and Manchester, Z. ALGAMES: A Fast Solver for Constrained Dynamic Games. *Robotics: Science and Systems XVI* (July 2020).

[161] Le Cleac'h, S., Schwager, M., and Manchester, Z. ALGAMES: A Fast Augmented Lagrangian Solver for Constrained Dynamic Games. *Auton. Robots 46*, 1 (Jan. 2022), 201–215.

[162] Leon, F., and Gavrilescu, M. A Review of Tracking and Trajectory Prediction Methods for Autonomous Driving. *Mathematics 9*, 6 (2021).

[163] Levine, S., and Koltun, V. Guided Policy Search. In *Proceedings of the 30th International Conference on Machine Learning* (May 2013), PMLR, pp. 1–9. ISSN: 1938-7228.

[164] Li, J., Xie, X., Lin, Q., He, J., and Dolan, J. M. Motion Planning by Search in Derivative Space and Convex Optimization with Enlarged Solution Space. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (2022), pp. 13500–13507.

[165] Li, L., Yang, M., Bing Wang, and Wang, C. An overview on sensor map based localization for automated driving. In *2017 Joint Urban Remote Sensing Event (JURSE)* (Dubai, United Arab Emirates, Mar. 2017), IEEE, pp. 1–4.

[166] Li, N., Goubault, E., Pautet, L., and Putot, S. Autonomous racecar control in head-to-head competition using Mixed-Integer Quadratic Programming. In *Opportunities and challenges with autonomous racing, 2021 ICRA workshop* (Online, United States, May 2021).

[167] Liniger, A., Domahidi, A., and Morari, M. Optimization-based autonomous racing of 1:43 scale RC cars. *Optimal Control Applications and Methods 36*, 5 (2015), 628–647.

[168] Liniger, A., and Lygeros, J. A Noncooperative Game Approach to Autonomous Racing. *IEEE Transactions on Control Systems Technology 28*, 3 (2020), 884–897.

[169] Livint, G., Horga, V., Ratoi, M., and Albu, M. Control of Hybrid Electrical Vehicles. In *Electric Vehicles*, S. Soylu, Ed. IntechOpen, Rijeka, 2011.

[170] Lopez, P. A., Behrisch, M., Bieker-Walz, L., Erdmann, J., Flötteröd, Y.-P., Hilbrich, R., Lücken, L., Rummel, J., Wagner, P., and Wiessner, E. Microscopic Traffic Simulation using SUMO. In *21st IEEE International Conference on Intelligent Transportation Systems* (2018), IEEE.

[171] Lubars, J., Gupta, H., Chinchali, S., Li, L., Raja, A., Srikant, R., and Wu, X. Combining Reinforcement Learning with Model Predictive Control for On-Ramp Merging, 2021.

[172] Mamedov, S., Reiter, R., Azad, S. M. B., Boedecker, J., Diehl, M., and Swevers, J. Safe Imitation Learning of Nonlinear Model Predictive Control for Flexible Robots. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), (accepted for publication)* (2024).

[173] Marcucci, T., Petersen, M., Wrangel, D. v., and Tedrake, R. Motion planning around obstacles with convex optimization. *Science Robotics 8*, 84 (2023).

[174] Maruyama, Y., Kato, S., and Azumi, T. Exploring the performance of ROS2. In *Proceedings of the 13th ACM SIGBED International Conference on Embedded Software (EMSOFT)* (2016), pp. 1–10.

[175] Masti, D., and Bemporad, A. Learning binary warm starts for multiparametric mixed-integer quadratic programming. In *18th European Control Conference (ECC)* (2019), pp. 1494–1499.

[176] Matthaei, R., and Maurer, M. Autonomous driving – a top-down-approach. *at - Automatisierungstechnik 63*, 3 (Mar. 2015), 155–167.

[177] Menner, M., Worsnop, P., and Zeilinger, M. N. Constrained Inverse Optimal Control With Application to a Human Manipulation Task. *IEEE Transactions on Control Systems Technology 29*, 2 (2021), 826–834.

[178] Menner, M., and Zeilinger, M. N. Convex Formulations and Algebraic Solutions for Linear Quadratic Inverse Optimal Control Problems. *European Control Conference (ECC)* (2018), 2107–2112.

[179] Meshginqalam, A., and Bauman, J. Two-Level MPC Speed Profile Optimization of Autonomous Electric Vehicles Considering Detailed Internal and External Losses. *IEEE Access 8* (2020), 206559–206570.

[180] Meshginqalam, A., and Bauman, J. Integrated Convex Speed Planning and Energy Management for Autonomous Fuel Cell Hybrid Electric Vehicles. *IEEE Transactions on Transportation Electrification 9*, 1 (2023), 1072–1086.

[181] Miller, C., Pek, C., and Althoff, M. Efficient Mixed-Integer Programming for Longitudinal and Lateral Motion Planning of Autonomous Vehicles. In *IEEE Intelligent Vehicles Symposium (IV)* (2018), pp. 1954–1961.

[182] Mombaur, K., Truong, A., and Laumond, J.-P. From human to humanoid locomotion-An inverse optimal control approach. *Auton. Robots 28* (Apr. 2010), 369–383.

[183] Montemerlo, M., Becker, J., Bhat, S., Dahlkamp, H., Dolgov, D., Ettinger, S., Haehnel, D., Hilden, T., Hoffmann, G., Huhnke, B., Johnston, D., Klumpp, S., Langer, D., Levandowski, A., Levinson, J., Marcil, J., Orenstein, D., Paefgen, J., Penny, I., Petrovskaya, A., Pflueger, M., Stanek, G., Stavens, D., Vogt, A., and Thrun, S. Junior: The Stanford entry in the Urban Challenge. *Journal of Field Robotics 25*, 9 (2008), 569–597.

[184] MOSEK ApS. *MOSEK Optimization Toolbox for MATLAB Manual.* 2024.

[185] Mouhagir, H., Talj, R., Cherfaoui, V., Aioun, F., and Guillemard, F. Integrating safety distances with trajectory planning by modifying the occupancy grid for autonomous vehicle navigation. In *IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)* (2016), pp. 1114–1119.

[186] Murty, K. G., and Kabadi, S. N. Some NP-complete problems in quadratic and nonlinear programming. *Mathematical Programming 39*, 2 (June 1987), 117–129.

[187] Nagabandi, A., Kahn, G., Fearing, R. S., and Levine, S. Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. In *IEEE international conference on robotics and automation (ICRA)* (2018), IEEE, pp. 7559–7566.

[188] Nair, S. H., Tseng, E. H., and Borrelli, F. Collision Avoidance for Dynamic Obstacles with Uncertain Predictions using Model Predictive Control. In *IEEE 61st Conference on Decision and Control (CDC)* (2022), pp. 5267–5272.

[189] Nair, V., Bartunov, S., Gimeno, F., Glehn, I. v., Lichocki, P., Lobov, I., O'Donoghue, B., Sonnerat, N., Tjandraatmadja, C., Wang, P., Addanki, R., Hapuarachchi, T., Keck, T., Keeling, J., Kohli, P., Ktena, I., Li, Y., Vinyals, O., and Zwols, Y. Solving Mixed Integer Programs Using Neural Networks. *ArXiv abs/2012.13349* (2020).

[190] Nemhauser, G., and Wolsey, L. Computational Complexity. In *Integer and Combinatorial Optimization.* John Wiley & Sons, Ltd, 1988, pp. 114–145.

[191] Nemhauser, G., and Wolsey, L. The Scope of Integer and Combinatorial Optimization. In *Integer and Combinatorial Optimization.* John Wiley & Sons, Ltd, 1988, pp. 1–26.

[192] Ng, A. Y., Harada, D., and Russell, S. J. Policy Invariance Under Reward Transformations: Theory and Application to Reward Shaping.

In *Proceedings of the Sixteenth International Conference on Machine Learning* (San Francisco, CA, USA, 1999), ICML '99, Morgan Kaufmann Publishers Inc., pp. 278–287.

[193] Nikhil, N., and Morris, B. T. Convolutional Neural Network for Trajectory Prediction. In *Computer Vision – ECCV 2018 Workshops*, vol. 11131. Springer International Publishing, Cham, 2019, pp. 186–196.

[194] Nocedal, J., and Wright, S. J. *Numerical Optimization*, second ed. Springer, New York, NY, USA, 2006.

[195] Novi, T., Liniger, A., Capitani, R., and Annicchiarico, C. Real-time control for at-limit handling driving on a predefined path. *Vehicle System Dynamics 58*, 7 (2020), 1007–1036.

[196] O'Kelly, M., Zheng, H., Karthik, D., and Mangharam, R. F1TENTH: An Open-source Evaluation Environment for Continuous Control and Reinforcement Learning. *Proceedings of Machine Learning Research 123* (2020).

[197] Ort, T., Paull, L., and Rus, D. Autonomous Vehicle Navigation in Rural Environments Without Detailed Prior Maps. In *IEEE International Conference on Robotics and Automation (ICRA)* (Brisbane, QLD, May 2018), IEEE, pp. 2040–2047.

[198] Pacejka, H. B., and Bakker, E. THE MAGIC FORMULA TYRE MODEL. *Vehicle System Dynamics 21* (1991), 1–18.

[199] Paden, B., Čáp, M., Yong, S. Z., Yershov, D., and Frazzoli, E. A Survey of Motion Planning and Control Techniques for Self-Driving Urban Vehicles. *IEEE Transactions on Intelligent Vehicles 1*, 1 (2016), 33–55.

[200] Pardalos, P. M., and Vavasis, S. A. Quadratic programming with one negative eigenvalue is NP-hard. *Journal of Global Optimization 1*, 1 (Mar. 1991), 15–22.

[201] Park, J., Karumanchi, S., and Iagnemma, K. Homotopy-Based Divide-and-Conquer Strategy for Optimal Trajectory Planning via Mixed-Integer Programming. *IEEE Transactions on Robotics 31*, 5 (Oct. 2015), 1101–1115.

[202] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, pp. 8024–8035.

[203] Pfrommer, S., Gautam, T., Zhou, A., and Sojoudi, S. Safe Reinforcement Learning with Chance-constrained Model Predictive Control. In *Proceedings of The 4th Annual Learning for Dynamics and Control Conference* (May 2022), PMLR, pp. 291–303. ISSN: 2640-3498.

[204] Pham, H. *Continuous-time Stochastic Control and Optimization with Financial Applications*, vol. 61 of *Stochastic Modelling and Applied Probability*. Springer, Berlin, Heidelberg, 2009.

[205] Piovesan, J. L., and Tanner, H. G. Randomized model predictive control for robot navigation. In *IEEE International Conference on Robotics and Automation* (2009), IEEE, pp. 94–99.

[206] Polack, P., Altché, F., d'Andréa Novel, B., and de La Fortelle, A. The kinematic bicycle model: A consistent model for planning feasible trajectories for autonomous vehicles? In *IEEE Intelligent Vehicles Symposium (IV)* (June 2017), pp. 812–818.

[207] Pressley, A. Curves in the plane and in space. In *Elementary Differential Geometry*, A. Pressley, Ed. Springer, London, 2010, pp. 1–27.

[208] Puterman, M. L., and Shin, M. C. Modified Policy Iteration Algorithms for Discounted Markov Decision Problems. *Management Science 24*, 11 (1978), 1127–1137. Publisher: INFORMS.

[209] Qian, X., Altché, F., Bender, P., Stiller, C., and de La Fortelle, A. Optimal trajectory planning for autonomous driving integrating logical constraints: An MIQP perspective. In *IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)* (2016), pp. 205–210.

[210] Qu, Y., Chu, H., Gao, S., Guan, J., Yan, H., Xiao, L., Li, S. E., and Duan, J. RL-Driven MPPI: Accelerating Online Control Laws Calculation With Offline Policy. *IEEE Transactions on Intelligent Vehicles* (2023), 1–12. Conference Name: IEEE Transactions on Intelligent Vehicles.

[211] Quirynen, R. *Numerical Simulation Methods for Embedded Optimization.* PhD Thesis, Jan. 2017.

[212] Quirynen, R., and Cairano, S. D. Sequential Quadratic Programming Algorithm for Real-Time Mixed-Integer Nonlinear MPC. In *60th IEEE Conference on Decision and Control (CDC)* (2021), pp. 993–999.

[213] Quirynen, R., Safaoui, S., and Di Cairano, S. Real-time Mixed-Integer Quadratic Programming for Vehicle Decision Making and Motion Planning. *ArXiv abs/2308.10069* (2023).

[214] Raffin, A., Hill, A., Ernestus, M., Gleave, A., Kanervisto, A., and Dormann, N. Stable baselines3, 2019.

[215] Raji, A., Liniger, A., Giove, A., Toschi, A., Musiu, N., Morra, D., Verucchi, M., Caporale, D., and Bertogna, M. Motion Planning and Control for Multi Vehicle Autonomous Racing at High Speeds. In *IEEE 25th International Conference on Intelligent Transportation Systems (ITSC)* (2022), pp. 2775–2782.

[216] Rasekhipour, Y., Khajepour, A., Chen, S.-K., and Litkouhi, B. A Potential Field-Based Model Predictive Path-Planning Controller for Autonomous Road Vehicles. *IEEE Transactions on Intelligent Transportation Systems 18*, 5 (2017), 1255–1267.

[217] Rawlings, J. B., Mayne, D. Q., and Diehl, M. M. *Model Predictive Control: Theory, Computation, and Design*, 2nd edition ed. Nob Hill, 2017.

[218] Reda, M., Onsy, A., Haikal, A. Y., and Ghanbari, A. Path planning algorithms in the autonomous driving system: A comprehensive review. *Robotics and Autonomous Systems 174* (2024), 104630.

[219] Reif, J. H. Complexity of the mover's problem and generalizations. In *20th Annual Symposium on Foundations of Computer Science* (1979), pp. 421–427.

[220] Reimpell, J., and Burckhardt, M. *Fahrwerktechnik, Radschlupf-Regelsysteme*, 1. aufl ed. Vogel Communications Group GmbH & Co. KG, Würzburg, Jan. 1993.

[221] Reiter, R., Baumgärtner, K., Quirynen, R., and Diehl, M. Progressive Smoothing for Motion Planning in Real-Time NMPC. In *European Control Conference (ECC)* (2024), pp. 1816–1823.

[222] Reiter, R., and Diehl, M. Parameterization Approach of the Frenet Transformation for Model Predictive Control of Autonomous Vehicles. In *European Control Conference (ECC)* (2021), pp. 2414–2419.

[223] Reiter, R., Ghezzi, A., Baumgärtner, K., Hoffmann, J., McAllister, R. D., and Diehl, M. AC4MPC: Actor-Critic Reinforcement Learning for Nonlinear Model Predictive Control. *CoRR* (June 2024).

[224] Reiter, R., Hoffmann, J., Boedecker, J., and Diehl, M. A Hierarchical Approach for Strategic Motion Planning in Autonomous Racing. In *European Control Conference (ECC)* (June 2023), pp. 1–8.

[225] Reiter, R., Kirchengast, M., Watzenig, D., and Diehl, M. Mixed-integer optimization-based planning for autonomous racing with obstacles and rewards. *IFAC-PapersOnLine 54*, 6 (2021), 99–106.

[226] Reiter, R., Messerer, F., Schratter, M., Watzenig, D., and Diehl, M. An Inverse Optimal Control Approach for Trajectory Prediction of Autonomous Race Cars. In *European Control Conference (ECC)* (2022), pp. 146–153.

[227] Reiter, R., Nurkanović, A., Bernardini, D., Diehl, M., and Bemporad, A. A Long-Short-Term Mixed-Integer Formulation for Highway Lane Change Planning. *IEEE Transactions on Intelligent Vehicles* (2024), 1–15.

[228] Reiter, R., Nurkanović, A., Frey, J., and Diehl, M. Frenet-Cartesian model representations for automotive obstacle avoidance within nonlinear MPC. *European Journal of Control* (2023), 100847.

[229] Reiter, R., Quirynen, R., Diehl, M., and Di Cairano, S. Equivariant Deep Learning of Mixed-Integer Optimal Control Solutions for Vehicle Decision Making and Motion Planning. *IEEE Transactions on Control Systems Technology* (2024), 1–15.

[230] Richards, A., Schouwenaars, T., How, J. P., and Feron, E. Spacecraft Trajectory Planning with Avoidance Constraints Using Mixed-Integer Linear Programming. *Journal of Guidance, Control, and Dynamics 25*, 4 (July 2002), 755–764.

[231] Rizano, T., Fontanelli, D., Palopoli, L., Pallottino, L., and Salaris, P. Global path planning for competitive robotic cars. In *52nd IEEE Conference on Decision and Control* (2013), pp. 4510–4516.

[232] Robinson, S. M. Local structure of feasible sets in nonlinear programming, Part III: Stability and sensitivity. In *Nonlinear Analysis and Optimization*, B. Cornet, V. H. Nguyen, and J. P. Vial, Eds. Springer, Berlin, Heidelberg, 1987, pp. 45–66.

[233] Roborace. Roborace Season Beta, 2020.

[234] Roffel, B., and Betlem, B. H. Internal Model Control. In *Advanced Practical Process Control*, B. Roffel and B. H. Betlem, Eds. Springer, Berlin, Heidelberg, 2004, pp. 161–169.

[235] Romero, A., Song, Y., and Scaramuzza, D. Actor-Critic Model Predictive Control. In *2024 IEEE International Conference on Robotics and Automation (ICRA)* (Yokohama, Japan, May 2024), IEEE, pp. 14777–14784.

[236] Romero, A., Sun, S., Foehn, P., and Scaramuzza, D. Model Predictive Contouring Control for Time-Optimal Quadrotor Flight. *IEEE Transactions on Robotics 38*, 6 (Dec. 2022), 3340–3356.

[237] Rong, J., Arrigoni, S., Luan, N., and Braghin, F. Attention-based Sampling Distribution for Motion Planning in Autonomous Driving. In *39th Chinese Control Conference (CCC)* (2020), pp. 5671–5676.

[238] Rosolia, U., De Bruyne, S., and Alleyne, A. G. Autonomous Vehicle Control: A Nonconvex Approach for Obstacle Avoidance. *IEEE Transactions on Control Systems Technology 25*, 2 (2017), 469–484.

[239] Ross, S., and Bagnell, D. Efficient Reductions for Imitation Learning. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics* (Chia Laguna Resort, Sardinia, Italy, May 2010), Y. W. Teh and M. Titterington, Eds., vol. 9 of *Proceedings of Machine Learning Research*, PMLR, pp. 661–668.

[240] Ross, S., Gordon, G., and Bagnell, D. A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics* (June 2011), JMLR Workshop and Conference Proceedings, pp. 627–635.

[241] Russell, S., and Norvig, P. *Artificial Intelligence: A Modern Approach (4th Edition)*. Pearson, 2020.

[242] Russo, L., Nair, S. H., Glielmo, L., and Borrelli, F. Learning for Online Mixed-Integer Model Predictive Control With Parametric Optimality Certificates. *IEEE Control Systems Letters 7* (2023), 2215–2220.

[243] Rüde, U. *Mathematical and Computational Techniques for Multilevel Adaptive Methods*. Society for Industrial and Applied Mathematics, 1993.

[244] Sager, S. *Numerical methods for mixed–integer optimal control problems*. PhD Thesis, Universitaet Heidelberg, Interdisciplinary Center for Scientific Computing, Jan. 2006.

[245] Sathya, A., Sopasakis, P., Van Parys, R., Themelis, A., Pipeleers, G., and Patrinos, P. Embedded nonlinear model predictive control for obstacle avoidance using PANOC. In *European Control Conference (ECC)* (2018), pp. 1523–1528.

[246] Schouwenaars, T., De Moor, B., Feron, E., and How, J. Mixed integer programming for multi-vehicle path planning. In *European Control Conference (ECC)* (Sept. 2001), pp. 2603–2608.

[247] Schratter, M., Kirchengast, M., Ronecker, M., Riepl, S., Renzler, T., and Watzenig, D. From Simulation to the Race Track: Development, Testing, and Deployment of Autonomous Racing Software. In *IEEE International Automated Vehicle Validation Conference (IAVVC)* (Austin, TX, USA, Oct. 2023), IEEE, pp. 1–8.

[248] Schulman, J., Duan, Y., Ho, J., Lee, A., Awwal, I., Bradlow, H., Pan, J., Patil, S., Goldberg, K., and Abbeel, P. Motion planning with sequential convex optimization and convex collision checking. *International Journal of Robotics Research 33*, 9 (Aug. 2014), 1251–1270.

[249] Schulman, J., Levine, S., Abbeel, P., Jordan, M., and Moritz, P. Trust Region Policy Optimization. In *Proceedings of the 32nd International Conference on Machine Learning* (Lille, France, July 2015), F. Bach and D. Blei, Eds., vol. 37 of *Proceedings of Machine Learning Research*, PMLR, pp. 1889–1897.

[250] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. *CoRR abs/1707.06347* (2017).

[251] Schwarting, W., Pierson, A., Alonso-Mora, J., Karaman, S., and Rus, D. Social behavior for autonomous vehicles. *Proceedings of the National Academy of Sciences 116*, 50 (Dec. 2019), 24972–24978.

[252] Schwarting, W., Seyde, T., Gilitschenski, I., Liebenwein, L., Sander, R., Karaman, S., and Rus, D. Deep Latent Competition: Learning to Race Using Visual Control Policies in Latent Space, 2021.

[253] Schöller, C., Aravantinos, V., Lay, F., and Knoll, A. What the Constant Velocity Model Can Teach Us About Pedestrian Motion Prediction. *IEEE Robotics and Automation Letters 5*, 2 (Apr. 2020), 1696–1703. Conference Name: IEEE Robotics and Automation Letters.

[254] Schöller, C., Aravantinos, V., Lay, F., and Knoll, A. C. The Simpler the Better: Constant Velocity for Pedestrian Motion Prediction. *CoRR abs/1903.07933* (2019).

[255] Schürmann, B., Hess, D., Eilbrecht, J., Stursberg, O., Köster, F., and Althoff, M. Ensuring drivability of planned motions using formal methods. In *IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)* (2017), pp. 1–8.

[256] Sharma, O., Sahoo, N. C., and Puhan, N. B. Recent advances in motion and behavior planning techniques for software architecture of autonomous vehicles: A state-of-the-art survey. *Engineering Applications of Artificial Intelligence 101* (May 2021), 104211.

[257] Sheckells, M., Caldwell, T. M., and Kobilarov, M. Fast approximate path coordinate motion primitives for autonomous driving. In *IEEE 56th Annual Conference on Decision and Control (CDC)* (2017), pp. 837–842.

[258] Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., and Riedmiller, M. Deterministic policy gradient algorithms. In *International conference on machine learning* (2014), PMLR, pp. 387–395.

[259] Singh, A. K., and Bhadauria, B. S. Finite difference formulae for unequal sub-intervals using Lagrange's interpolation formula. *International Journal of Mathematical Analysis 3*, 17-20 (2009), 815–827.

[260] Sinha, A., Malo, P., and Deb, K. A Review on Bilevel Optimization: From Classical to Evolutionary Approaches and Applications. *IEEE Transactions on Evolutionary Computation 22*, 2 (2018), 276–295.

[261] Sipser, M. *Introduction to the theory of computation*, third edition, international edition ed. Cengage Learning, 2013.

[262] Sollich, P., and Krogh, A. Learning with ensembles: How overfitting can be useful. In *Advances in Neural Information Processing Systems* (1995), D. Touretzky, M. C. Mozer, and M. Hasselmo, Eds., vol. 8, MIT Press.

[263] Soloperto, R., Köhler, J., Allgöwer, F., and Müller, M. A. Collision avoidance for uncertain nonlinear systems with moving obstacles using robust Model Predictive Control. In *18th European Control Conference (ECC)* (Naples, Italy, June 2019), IEEE, pp. 811–817.

[264] Song, J., Meng, C., and Ermon, S. Denoising Diffusion Implicit Models. In *International Conference on Learning Representations* (2021).

[265] Song, Y., and Scaramuzza, D. Learning High-Level Policies for Model Predictive Control. *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (2020), 7629–7636.

[266] Spica, R., Cristofalo, E., Wang, Z., Montijano, E., and Schwager, M. A Real-Time Game Theoretic Planner for Autonomous Two-Player Drone Racing. *IEEE Transactions on Robotics 36*, 5 (2020), 1389–1403.

[267] Srinivasan, M., Chakrabarty, A., Quirynen, R., Yoshikawa, N., Mariyama, T., and Di Cairano, S. Fast Multi-Robot Motion Planning via Imitation Learning of Mixed-Integer Programs. *IFAC-PapersOnLine 54*, 20 (2021), 598–604.

[268] Stellato, B., Banjac, G., Goulart, P., Bemporad, A., and Boyd, S. OSQP: an operator splitting solver for quadratic programs. *Mathematical Programming Computation 12*, 4 (2020), 637–672.

[269] Stellato, B., Naik, V. V., Bemporad, A., Goulart, P., and Boyd, S. Embedded Mixed-Integer Quadratic optimization Using the OSQP Solver. In *European Control Conference (ECC)* (Limassol, June 2018), IEEE, pp. 1536–1541.

[270] Sterman, J. *Business Dynamics: Systems Thinking and Modeling for a Complex World*, ed edition ed. McGraw Hill Higher Education, Boston Burr Ridge, IL Dubuque, IA Madison, WI New York San Francisco St. Louis Bangkok Bogotá Caracas Lisbon London Madrid Mexico City Milan New Delhi Seoul Singapore Sydney Taipei Toronto, Dec. 2000.

[271] Subash, A. J., Klöser, D., Frey, J., Reiter, R., Diehl, M., and Bohlmann, K. Model Predictive Control for Frenet-Cartesian Trajectory Tracking of a Tricycle Kinematic Automated Guided Vehicle. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), (accepted for publication)* (Abu Dhabi, 2024).

[272] Subosits, J., and Gerdes, J. C. Autonomous vehicle control for emergency maneuvers: The effect of topography. In *2015 American Control Conference (ACC)* (Chicago, IL, USA, July 2015), IEEE, pp. 1405–1410.

[273] Sutton, R. S. *Temporal credit assignment in reinforcement learning.* PhD Thesis, University of Massachusetts Amherst, 1984.

[274] Sutton, R. S., and Barto, A. G. *Reinforcement learning: An introduction.* MIT press, 2018.

[275] Sutton, R. S., McAllester, D., Singh, S., and Mansour, Y. Policy Gradient Methods for Reinforcement Learning with Function Approximation. In *Advances in Neural Information Processing Systems* (1999), vol. 12, MIT Press.

[276] Szalay, Z., Tettamanti, T., Esztergár-Kiss, D., Varga, I., and Bartolini, C. Development of a Test Track for Driverless Cars: Vehicle Design, Track Configuration, and Liability Considerations. *Periodica Polytechnica Transportation Engineering 46*, 1 (2018), 29–35.

[277] Szepesvari, C., Brachman, R., and Dietterich, T. *Algorithms for Reinforcement Learning.* Morgan and Claypool Publishers, San Rafael, Calif., June 2010.

[278] Tampuu, A., Matiisen, T., Semikin, M., Fishman, D., and Muhammad, N. A Survey of End-to-End Driving: Architectures and Training Methods. *IEEE Transactions on Neural Networks and Learning Systems 33*, 4 (Apr. 2022), 1364–1384.

[279] Tao, R., Cheng, S., Wang, X., Wang, S., and Hovakimyan, N. DiffTune-MPC: Closed-Loop Learning for Model Predictive Control. *IEEE Robotics and Automation Letters* (2024). Publisher: IEEE.

[280] Tearle, B., Wabersich, K. P., Carron, A., and Zeilinger, M. N. A Predictive Safety Filter for Learning-Based Racing Control. *IEEE Robotics and Automation Letters 6*, 4 (Oct. 2021), 7635–7642.

[281] Theodorou, E. A. Nonlinear Stochastic Control and Information Theoretic Dualities: Connections, Interdependencies and Thermodynamic Interpretations. *Entropy 17*, 5 (May 2015), 3352–3375.

[282] Theodorou, E. A., and Todorov, E. Relative entropy and free energy dualities: Connections to Path Integral and KL control. In *IEEE 51st IEEE Conference on Decision and Control (CDC)* (Dec. 2012), pp. 1466–1473.

[283] Torrisi, F., and Bemporad, A. HYSDEL-a tool for generating computational hybrid models for analysis and synthesis problems. *IEEE Transactions on Control Systems Technology 12*, 2 (2004), 235–249.

[284] Tran Dinh, Q., Gumussoy, S., Michiels, W., and Diehl, M. Combining Convex–Concave Decompositions and Linearization Approaches for Solving BMIs, With Application to Static Output Feedback. *IEEE Transactions on Automatic Control 57*, 6 (2012), 1377–1390.

[285] Trautman, P., and Krause, A. Unfreezing the robot: Navigation in dense, interacting crowds. In *IEEE/RSJ International Conference on Intelligent Robots and Systems* (2010), pp. 797–803.

[286] Treiber, M., Hennecke, A., and Helbing, D. Congested traffic states in empirical observations and microscopic simulations. *Physical Review E 62*, 2 (Aug. 2000), 1805–1824. Publisher: American Physical Society.

[287] Van Duijkeren, N., Verschueren, R., Pipeleers, G., Diehl, M., and Swevers, J. Path-following NMPC for serial-link robot manipulators using a path-parametric system reformulation. In *European Control Conference (ECC)* (Aalborg, Denmark, June 2016), IEEE, pp. 477–482.

[288] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. Attention is all you need. In *Advances in neural information processing systems* (2017), pp. 5998–6008.

[289] Velenis, E., and Tsiotras, P. Optimal velocity profile generation for given acceleration limits: theoretical analysis. In *Proceedings of the American Control Conference* (2005), pp. 1478–1483 vol. 2.

[290] Verschueren, R. *Convex approximation methods for nonlinear model predictive control.* PhD Thesis, University of Freiburg, 2018.

[291] Verschueren, R., Frison, G., Kouzoupis, D., Frey, J., Duijkeren, N. v., Zanelli, A., Novoselnik, B., Albin, T., Quirynen, R., and Diehl, M. acados – a modular open-source framework for fast embedded optimal control. *Mathematical Programming Computation* (Oct. 2021).

[292] Verschueren, R., Zanon, M., Quirynen, R., and Diehl, M. Time-optimal race car driving using an online exact hessian based nonlinear MPC algorithm. *European Control Conference* (2017), 141–147.

[293] Vázquez, J. L., Brühlmeier, M., Liniger, A., Rupenyan, A., and Lygeros, J. Optimization-Based Hierarchical Motion Planning for Autonomous Racing. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (Oct. 2020), pp. 2397–2403.

[294] Wabersich, K. P., and Zeilinger, M. N. A predictive safety filter for learning-based control of constrained nonlinear dynamical systems. *Automatica 129* (2021), 109597.

[295] Wang, J., Chi, W., Li, C., Wang, C., and Meng, M. Q.-H. Neural RRT*: Learning-Based Optimal Path Planning. *IEEE Transactions on Automation Science and Engineering 17*, 4 (2020), 1748–1758.

[296] Wang, J., Li, B., and Meng, M. Q.-H. Kinematic Constrained Bi-directional RRT with Efficient Branch Pruning for robot path planning. *Expert Systems with Applications 170* (2021), 114541.

[297] Wang, J., Yan, Y., Zhang, K., Chen, Y., Mingcong, C., and Yin, G. Path Planning on Large Curvature Roads Using Driver-Vehicle-Road System Based on the Kinematic Vehicle Model. *IEEE Transactions on Vehicular Technology PP* (Nov. 2021), 1–1.

[298] Wang, M., Mehr, N., Gaidon, A., and Schwager, M. Game-Theoretic Planning for Risk-Aware Interactive Agents. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (2020), pp. 6998–7005.

[299] Wang, M., Wang, Z., Talbot, J., Gerdes, J. C., and Schwager, M. Game-Theoretic Planning for Self-Driving Cars in Multivehicle Competitive Scenarios. *IEEE Transactions on Robotics 37*, 4 (2021), 1313–1325.

[300] Wang, P., Liu, D., Chen, J., Li, H., and Chan, C.-Y. Decision Making for Autonomous Driving via Augmented Adversarial Inverse Reinforcement Learning. In *IEEE International Conference on Robotics and Automation (ICRA)* (2021), pp. 1036–1042.

[301] Wang, Q., Weiskircher, T., and Ayalew, B. Hierarchical Hybrid Predictive Control of an Autonomous Road Vehicle. In *ASME 2015 Dynamic Systems and Control Conference* (Oct. 2015).

[302] Werling, M., Ziegler, J., Kammel, S., and Thrun, S. Optimal Trajectory Generation for Dynamic Street Scenarios in a Frenet Frame. In *IEEE International Conference on Robotics and Automation* (June 2010), pp. 987 – 993.

[303] Williams, G., Aldrich, A., and Theodorou, E. A. Model Predictive Path Integral Control: From Theory to Parallel Computation. *Journal of Guidance, Control, and Dynamics 40*, 2 (2017), 344–357.

[304] Williams, H. P. *Model Building in Mathematical Programming.* Wiley, Hoboken, N.J., 2013.

[305] Williams, R. J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning 8*, 3 (May 1992), 229–256.

[306] Wurman, P., Barrett, S., Kawamoto, K., MacGlashan, J., Subramanian, K., Walsh, T., Capobianco, R., Devlic, A., Eckert, F., Fuchs, F., Gilpin, L., Khandelwal, P., Kompella, V., Lin, H., MacAlpine, P., Oller, D., Seno, T., Sherstan, C., Thomure, M., and Kitano, H. Outracing champion Gran Turismo drivers with deep reinforcement learning. *Nature 602* (Feb. 2022), 223–228.

[307] Wächter, A., and Biegler, L. T. *On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming*, vol. 106. 2006.

[308] Würsching, G., and Althoff, M. Robust and Efficient Curvilinear Coordinate Transformation with Guaranteed Map Coverage for Motion Planning. In *2024 IEEE Intelligent Vehicles Symposium (IV)* (2024).

[309] Xi, C., Shi, T., Wu, Y., and Sun, L. Efficient Motion Planning for Automated Lane Change based on Imitation Learning and Mixed-Integer Optimization. In *IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)* (2020), pp. 1–6.

[310] Xing, X., Zhao, B., Han, C., Ren, D., and Xia, H. Vehicle Motion Planning With Joint Cartesian-Frenet MPC. *IEEE Robotics and Automation Letters 7*, 4 (2022), 10738–10745.

[311] Ye, B.-L., Niu, S., Li, L., and Wu, W. A Comparison Study of Kinematic and Dynamic Models for Trajectory Tracking of Autonomous Vehicles Using Model Predictive Control. *International Journal of Control, Automation and Systems 21*, 9 (Sept. 2023), 3006–3021.

[312] Zaheer, M., Kottur, S., Ravanbakhsh, S., Poczos, B., Salakhutdinov, R. R., and Smola, A. J. Deep Sets. In *Advances in Neural Information Processing Systems* (2017), vol. 30, Curran Associates, Inc.

[313] Zarrouki, B., Klös, V., Heppner, N., Schwan, S., Ritschel, R., and Vosswinkel, R. Weights-varying MPC for Autonomous Vehicle Guidance: a Deep Reinforcement Learning Approach. In *European Control Conference (ECC)* (2021), pp. 119–125.

[314] Zhang, J., Liu, H., Chang, Q., Wang, L., and Gao, R. X. Recurrent neural network for motion trajectory prediction in human-robot collaborative assembly. *CIRP Annals 69*, 1 (2020), 9–12.

[315] Zhang, K., Yang, Z., and Başar, T. Multi-Agent Reinforcement Learning: A Selective Overview of Theories and Algorithms. In *Handbook of Reinforcement Learning and Control*, K. G. Vamvoudakis, Y. Wan, F. L. Lewis, and D. Cansever, Eds. Springer International Publishing, Cham, 2021, pp. 321–384.

[316] Zhang, Q., Langari, R., Tseng, H. E., Filev, D., Szwabowski, S., and Coskun, S. A Game Theoretic Model Predictive Controller With Aggressiveness Estimation for Mandatory Lane Change. *IEEE Transactions on Intelligent Vehicles 5*, 1 (Mar. 2020), 75–89.

[317] Zhang, T., Sun, Y., Wang, Y., Li, B., Tian, Y., and Wang, F.-Y. A Survey of Vehicle Dynamics Modeling Methods for Autonomous Racing: Theoretical Models, Physical/Virtual Platforms, and Perspectives. *IEEE Transactions on Intelligent Vehicles 9*, 3 (Mar. 2024), 4312–4334.

[318] Zhang, W., Drugge, L., Nybacka, M., Jerrelind, J., Wang, Z., and Zhu, J. Exploring Model Complexity for Trajectory Planning of Autonomous Vehicles in Critical Driving Scenarios. In *Advances in Dynamics of Vehicles on Roads and Tracks II* (Cham, 2022), A. Orlova and D. Cole, Eds., Springer International Publishing, pp. 1154–1165.

[319] Zhang, W., Wang, Z., Drugge, L., and Nybacka, M. Evaluating Model Predictive Path Following and Yaw Stability Controllers for Over-Actuated Autonomous Electric Vehicles. *IEEE Transactions on Vehicular Technology 69*, 11 (2020), 12807–12821.

[320] Zhang, X., Liniger, A., and Borrelli, F. Optimization-Based Collision Avoidance. *IEEE Transactions on Control Systems Technology 29*, 3 (May 2021), 972–983.

[321] Zhang, Y., Sun, H., Zhou, J., Pan, J., Hu, J., and Miao, J. Optimal Vehicle Path Planning Using Quadratic Optimization for Baidu Apollo Open Platform. In *IEEE Intelligent Vehicles Symposium (IV)* (Oct. 2020), IEEE.

[322] Zhong, S., Liu, A., Jiang, Y., Hu, S., Xiao, F., Huang, H.-J., and Song, Y. Energy and environmental impacts of shared autonomous vehicles under different pricing strategies. *npj Urban Sustainability 3*, 1 (Feb. 2023), 1–10.

[323] Zhou, B., Schwarting, W., Rus, D., and Alonso-Mora, J. Joint Multi-Policy Behavior Estimation and Receding-Horizon Trajectory Planning for Automated Urban Driving. In *IEEE international conference on robotics and automation (ICRA)* (May 2018), pp. 2388–2394.

[324] Zhou, H., Ren, D., Xia, H., Fan, M., Yang, X., and Huang, H. AST-GNN: An attention-based spatio-temporal graph neural network for Interaction-aware pedestrian trajectory prediction. *Neurocomputing 445* (2021), 298–308.

[325] Zhou, J. *Interaction and Uncertainty-Aware Motion Planning for Autonomous Vehicles Using Model Predictive Control.* PhD Thesis, Linköping University Electronic Press, 2023.

[326] Zhu, E. L., Busch, F. L., Johnson, J., and Borrelli, F. A Gaussian Process Model for Opponent Prediction in Autonomous Racing. In *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (Oct. 2023), pp. 8186–8191.

[327] Ziebart, B. D., Maas, A., Bagnell, J. A., and Dey, A. K. Maximum Entropy Inverse Reinforcement Learning. In *Proc. AAAI* (2008), pp. 1433–1438.

[328] Ziegler, J., Bender, P., Dang, T., and Stiller, C. Trajectory planning for Bertha - A local, continuous method. In *IEEE Intelligent Vehicles Symposium, Proceedings* (June 2014), pp. 450–457.

[329] Ziegler, J., Bender, P., Schreiber, M., Lategahn, H., Strauss, T., Stiller, C., Dang, T., Franke, U., Appenrodt, N., Keller, C. G., Kaus, E., Herrtwich, R. G., Rabe, C., Pfeiffer, D., Lindner, F., Stein, F., Erbs, F., Enzweiler, M., Knöppel, C., Hipp, J., Haueis, M., Trepte, M., Brenk, C., Tamke, A., Ghanaat, M., Braun, M., Joos, A., Fritz, H., Mock, H., Hein, M., and Zeeb, E. Making Bertha Drive—An Autonomous Journey on a Historic Route. *IEEE Intelligent Transportation Systems Magazine 6*, 2 (2014), 8–20.

[330] Zilberstein, S. Using Anytime Algorithms in Intelligent Systems. *AI Magazine 17*, 3 (1996), 73–83.

Curriculum Vitae

Rudolf Reiter was born on July 8, 1989 in Hallein, Austria.

Education:

Ph.D. Candidate, University of Freiburg, Department of Microsystems Engineering, Prof. Dr. Moritz Diehl	Freiburg, Germany	2020 - 2024
Research Visit, ETH Zürich, Department of Mechanical and Process Engineering, Prof. Dr. Melanie Zeilinger	Zürich, Switzerland	(Jan.-May) 2024
Research Visit, IMT Lucca, Prof. Dr. Alberto Bemporad	Lucca, Italy	May 2022 Sep. 2023
Master of Science, Graz University of Technology, Electrical Engineering: Control Systems and Mechatronics	Graz, Austria	2009 - 2016
Community Service, Paramedic at Red Cross	Salzburg, Austria	2008 - 2009
Secondary Technical College, HTL Salzburg, Department of Electronics and Computer Science	Salzburg, Austria	2004 - 2008

Industrial Experience:

Software Developer, Autonomous Racing Graz	Remote, Graz, Austria	2019 - 2024
Robotics Intern, Mitsubishi Electric Research Laboratories	Boston, MA, USA	(Jan.-May) 2023
Research Visit, ODYS S.r.l.	Milano, Italy	(Apr.-May) 2022
Researcher, Virtual Vehicle Research Center	Graz, Austria	2018 - 2021
Control Systems Specialist, Anton Paar GmbH	Graz, Austria	2016 - 2018
Master Thesis Intern, Virtual Vehicle Research Center	Graz, Austria	2015 - 2016
Control Systems Intern, B&R Industrial Automation GmbH	Salzburg, Austria	(Sep.-Dec.) 2012
Software Development Intern, B&R Industrial Automation GmbH	Salzburg, Austria	(Jul.-Aug.) 2010
Software Development Intern, Step Four GmbH	Salzburg, Austria	(Apr.-Jul.) 2009

List of Publications

Journal Publications

1. Reiter R., Nurkanović A., Frey J., Diehl M. (2023). Frenet-Cartesian Model Representations for Automotive Obstacle Avoidance within Nonlinear MPC. European Journal of Control, p. 100847, ISSN: 0947-3580.
2. Reiter R., Nurkanović A., Bernardini D., Diehl M., Bemporad A. (2024). A Long-Short-Term Mixed-Integer Formulation for Highway Lane Change Planning. IEEE Transactions on Intelligent Vehicles, pp. 1-15, ISSN: 2379-8858.
3. Reiter R., Quirynen R., Diehl M., Di Cairano S. (2024). Equivariant Deep Learning of Mixed-Integer Optimal Control Solutions for Vehicle Decision Making and Motion Planning. IEEE Transactions on Control Systems Technology, ISSN: 1558-0865.

Conference Publications (as the Main Author)

1. Reiter R., Diehl M. (2021). Parameterization Approach of the Frenet Transformation for Model Predictive Control of Autonomous Vehicles. In 2021 European Control Conference (ECC) (pp. 2414-2419).
2. Reiter R., Kirchengast M., Watzenig D., Diehl M. (2021). Mixed-integer optimization-based planning for autonomous racing with obstacles and rewards. IFAC-PapersOnLine 54, 6, (pp. 99-106).
3. Reiter R., Messerer F., Schratter M., Watzenig D., Diehl M. (2022). An Inverse Optimal Control Approach for Trajectory Prediction of Autonomous Race Cars. In 2022 European Control Conference (ECC) (pp. 146-153).
4. Reiter R., Hoffmann J., Boedecker J., Diehl M. (2023). A Hierarchical Approach for Strategic Motion Planning in Autonomous Racing. In 2023 European Control Conference (ECC) (pp. 1-8).

5. Reiter R., Baumgärtner K., Quirynen R., Diehl M. (2024). Progressive Smoothing for Motion Planning in Real-Time NMPC. In 2024 European Control Conference (ECC).

Conference Publications (as Coauthor)

6. Baumgärtner K., Reiter R., Diehl, M. (2022). Moving Horizon Estimation with Adaptive Regularization for Ill-Posed State and Parameter Estimation Problems. IEEE 61st Conference on Decision and Control (CDC), 2165-2171.
7. Mamedov S., Reiter R., Azad S. M. B., Viljoen R., Boedecker J., Diehl M., Swevers J. (2024). Safe Imitation Learning of Nonlinear Model Predictive Control for Flexible Robots. IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS).
8. Subash J. A., Kloeser D., Frey J., Reiter R., Bohlmann K., Diehl M. (2024). Model Predictive Control for Frenet-Cartesian Trajectory Tracking of a Tricycle Kinematic Automated Guided Vehicle. IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS).

Workshop Papers

1. Reiter R., Hoffmann J., Boedecker J., Diehl M. (2024). Hierarchical Reinforcement Learning and Model Predictive Control for Strategic Motion Planning in Autonomous Racing. 2024 International Conference on Machine Learning (ICML), Workshop on Foundations of Reinforcement Learning and Control.

FACULTY OF ENGINEERING
DEPARTMENT OF MICROSYSTEMS ENGINEERING
SYSTEMS CONTROL AND OPTIMIZATION LABORATORY
Georges-Köhler-Allee 102
DE-79110 Freiburg i. Br.