TECHNISCHE UNIVERSITÄT MÜNCHEN
Fakultät für Informatik

Using Reachability Analysis in Controller Synthesis for Safety-Critical Systems

Bastian Schürmann

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung eines

Doktors der Ingenieurwissenschaften (Dr.-Ing.)

genehmigten Dissertation.

Vorsitz:	Prof. Dr.-Ing. Alin Albu-Schäffer
Prüfer der Dissertation:	1. Prof. Dr.-Ing. Matthias Althoff
	2. Prof. Dr.-Ing. habil. Boris Lohmann

Die Dissertation wurde am 09.11.2021 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 15.02.2022 angenommen.

Abstract

With the rise of autonomous systems acting in human environments, such as autonomous vehicles, robots collaborating with humans, and robotic surgeries, we continually face the question of how to ensure safety. In this thesis, we tackle this problem by combining control theory, formal verification, and optimization to develop novel control algorithms which provide safety guarantees for constrained nonlinear systems despite the presence of disturbances and sensor noise. We use these techniques to solve reach-avoid problems, where we ensure that any trajectory starting from a set of possible initial states reaches a desired final set at a given time while avoiding unsafe sets, both in the state and input spaces. By including the computations of reachable sets into the optimization, we are able to optimize over a whole set of possible trajectories at once, thereby formally ensuring safety and constraint satisfaction for all of them.

We explore this idea in various directions, which leads to a number of different algorithms. Some interpolate between finitely many trajectories, either for extreme states or for zonotope representations. Other algorithms use continuous feedback and directly optimize over the resulting reachable set of the closed-loop dynamics in a nonlinear optimization problem. We extend these ideas to both hybrid dynamics through the example of piecewise affine systems and to backward reachable set computation. The majority of the presented algorithms precompute controllers and reachable sets for short motion primitives offline and store them in an automaton. The resulting maneuver automaton can be used for safe and fast online planning, as no optimization or verification problems have to be solved online. We also show how our formally correct controllers can be used as safety nets to safeguard unverified, classical controllers. In addition to the offline algorithms, we present a robust model predictive control approach which computes reachable sets online to verify safety.

The presented algorithms have a number of advantages: we obtain simple online control laws which allow easy implementation and fast application. The inclusion of formal verification in the controller synthesis allows us to directly optimize over the reachable sets, using this information to improve performance while still formally guaranteeing safety and constraint satisfaction. The resulting algorithms are push-button approaches, which enable controller synthesis without requiring deep control knowledge or finding Lyapunov functions. The applicability of the presented approaches is shown through an implementation on a robotic manipulator and simulations based on real data of an autonomous vehicle, as well as through several numerical examples.

Zusammenfassung

Mit dem Aufkommen autonomer Systeme, die in der unmittelbaren Umgebung von Menschen agieren, stellt sich die Frage, wie man für diese Systeme Sicherheit garantieren kann. Beispiele für solche Systeme sind autonome Fahrzeuge, Roboter, die mit Menschen kollaborieren, oder Operationsroboter. In der vorliegenden Dissertation werden zu diesem Zweck neuartige Regelungsverfahren vorgestellt, die Methoden aus der Regelungstheorie, formaler Verifikation und Optimierung miteinander kombinieren. Das Ziel dieser Verfahren ist es, formale Sicherheitsgarantien für beschränkte, nichtlineare Systeme bereitzustellen die durch externe Störungen und Messrauschen beeinflusst werden. Die Methoden werden genutzt um Reach-Avoid-Probleme zu lösen, bei denen sichergestellt werden muss, dass alle Trajektorien, die aus einer gegebenen Anfangsmenge starten, nach einer festen Zeit in einer gewünschten Endmenge enden. Währenddessen müssen sie unsichere Zustandsmengen vermeiden sowie Eingangsbeschränkungen einhalten. Durch die Berechnung von erreichbaren Mengen innerhalb der Regleroptimierung ist es möglich, eine Menge von Trajektorien auf einmal zu optimieren und dadurch das Einhalten von Zustands- und Eingangsbeschränkungen für alle formal zu garantieren.

Diese Grundidee wird in verschiedene Richtungen weiterentwickelt und führt zu einer Reihe von Regelansätzen. Einige interpolieren zwischen endlich-vielen Trajektorien und nutzen dafür entweder Extrempunkte oder Zonotopdarstellungen. Andere verwenden eine kontinuierliche Rückführung und optimieren direkt über die dazugehörige erreichbare Menge des geschlossenen Regelkreises in einem nichtlinearen Optimierungsproblem. Im Laufe der Arbeit werden diese Ideen erweitert, um sowohl hybride Systeme am Beispiel von stückweise affinen Systemen als auch rückwärts erreichbare Mengen zu berücksichtigen. Die meisten der vorgestellten Algorithmen eignen sich zur Offline-Berechnung von kurzen Bewegungsprimitiven, die in einem Automaten gespeichert werden können. Der resultierende Manöverautomat kann dann zur Laufzeit genutzt werden, um schnell und sicher Trajektorien zu planen, ohne dass Regleroptimierungs- oder Verifikationsprobleme in Echtzeit gelöst werden müssen. Neben der direkten Anwendung der neu entwickelten Regelungsverfahren, können diese auch als Sicherheitsnetz für klassische, unverifizierte Regler genutzt werden und deren Einsatz absichern. Zusätzlich zu den Offline-Ansätzen, wird auch die direkte Online-Optimierung von Reglern in Form einer robusten modellprädiktiven Regelung auf Basis erreichbarer Mengen vorgestellt.

Die entwickelten Ansätze bieten eine Reihe von Vorteilen: Die resultierende Regelgesetze haben eine einfache Form, was die Implementierung erleichtert und schnelle

Taktzeiten ermöglicht. Durch die Einbindung von formaler Verifikation in die Reglersynthese kann direkt über erreichbare Mengen optimiert werden. Die resultierenden Informationen können zur Verbesserung des Reglerverhaltens genutzt werden, während gleichzeitig die Sicherheit und die Einhaltung der Beschränkungen formal garantiert werden. Darüber hinaus werden die Regler durch die entwickelten Algorithmen automatisch berechnet, so dass für ihre Anwendung kein tiefes Regelungstechnikverständnis oder die Suche nach einer Ljapunow-Funktion notwendig ist.

Die praktische Anwendbarkeit der vorgestellten Ansätze wird durch die Implementierung auf einem Roboterarm und durch Simulationen auf Basis echter Fahrdaten eines autonomen Fahrzeugs sowie in einer Reihe von numerischen Beispielen demonstriert.

Acknowledgements

During my work on this dissertation, I had the chance to collaborate with many exceptional people. This work would not have been possible without their support and the numerous stimulating discussions and conversations. First and foremost, I would therefore like to thank my supervisor, Professor Matthias Althoff, whose guidance during my four years at the Technical University of Munich was immeasurable. He always took the time for meaningful discussions about my research and to give detailed feedback on my papers. He also enthusiastically supported my various collaborations with other universities, including a four-month research visit at the California Institute of Technology. There, I had the fortune of working with Professor Richard Murray, who welcomed me into his group and provided many valuable ideas and insight that allowed me to broaden my research. I am very grateful to him for this opportunity.

My deepest thanks to all of the people I met during my PhD. I was very fortunate to have had wonderful colleagues at the Cyber-Physical Systems Group at TU Munich, where we had a great atmosphere and participated in fun events; to have been part of the exciting research project UnCoVerCPS; to have supervised excellent students, many of whom later became colleagues; and to have had a very rewarding stay at the Control & Dynamical Systems Group at Caltech. These experiences led to many interesting and fruitful collaborations, which gave me the chance to apply the ideas from this dissertation to new fields. For this I would like to thank specifically from TU Munich Ahmed El-Guindy, Albert Rizaldi, Anna-Katharina Rettinger, Anna-Kathrin Kopetzki, Davide Calzolari, Felix Gruber, Moritz Klischat, Niklas Kochdumper, Stefan Liu, and Victor Gaßmann; from the UnCoVerCPS Project Professor Maria Prandini, Professor Olaf Stursberg, Daniel Heß, Jan Eilbrecht, and Riccardo Vignali; and from Caltech Jin Ge and Tung Phan.

I am particularly grateful to my family, whose love and support I could always count on. Thank you to my parents, who sparked my curiosity in science and engineering and whose continuous encouragement led me to where I am now. Finally, and most importantly, I want to thank my wife Carmella. Not only did she provide immense help while creating figures, but she also proofread all of my papers and this thesis. Most of all, she was always supportive, understanding, and patient when I needed to forgo free time in order to work. Thank you so much for everything.

Contents

List of Figures **vii**

List of Tables **ix**

1 Introduction **1**
1.1 Motivation . . . 1
1.2 Integration in Autonomous Systems . . . 4
1.3 Related Work . . . 8
1.4 Organization of the Thesis and Contributions . . . 9

2 Background **15**
2.1 Notation . . . 15
2.2 Convexity . . . 15
2.3 Set Operations . . . 16
2.4 Set Representations . . . 17
2.5 Nonlinear Systems and their Analysis . . . 20
2.6 Reachability Analysis . . . 21
2.7 Maneuver Automata . . . 25
2.8 Conformance Checking . . . 25
2.8.1 Trace Conformance . . . 26
2.8.2 Reachset Conformance . . . 27

3 Offline Controller Synthesis **29**
3.1 Introduction . . . 29
3.2 Problem Statement . . . 29
3.3 Convex Control . . . 31
3.3.1 Linear Systems . . . 32
3.3.2 Nonlinear Systems with Disturbances . . . 34
3.3.3 Closed-Form Expression of Convex Combinations . . . 39
3.3.4 Linear Approximation of the Convex Control Approach . . . 40
3.3.5 Possible Extensions . . . 42
3.3.6 Implementation . . . 43

3.3.7 Numerical Example . . . 44
3.4 Generator Interpolation Controller . . . 49
3.4.1 Linear Systems . . . 49
3.4.2 Nonlinear Dynamics and Disturbances . . . 54
3.4.3 Numerical Example . . . 58
3.5 Continuous Feedback Controller . . . 61
3.5.1 Closed-Loop Controller Optimization . . . 61
3.5.2 Implementation . . . 64
3.5.3 Cost Function . . . 66
3.5.4 Constraint Function . . . 66
3.5.5 Extension to Other Control Laws . . . 67
3.5.6 Numerical Example . . . 67
3.6 Combined Control . . . 71
3.6.1 Overview . . . 71
3.6.2 State-Dependent Feedforward Control . . . 74
3.6.3 Feedback Control . . . 75
3.6.4 Implementation . . . 76
3.6.5 Numerical Example . . . 79
3.7 Discussion . . . 84
3.7.1 Formal Guarantees . . . 84
3.7.2 Complexity . . . 84
3.7.3 Optimality . . . 87
3.7.4 Comparison of the Four Algorithms . . . 87
3.7.5 Extension to Include Measurement Noise . . . 88
3.8 Summary . . . 89

4 Offline Controller Synthesis for Piecewise Affine Systems 91
4.1 Introduction . . . 91
4.2 Problem Formulation . . . 93
4.3 Overview of the Approach . . . 95
4.4 Reference Trajectory Computation . . . 97
4.5 Set-Based Control Design . . . 100
4.6 Numerical Example . . . 108
4.7 Discussion . . . 113
4.8 Summary . . . 114

5 Offline Controller Synthesis Using Backward Reachable Sets 117
5.1 Introduction . . . 117
5.2 Problem Formulation . . . 119
5.3 Online Control with Safety Net . . . 119
5.4 Backward Controller Synthesis for Linear, Discrete-Time Systems . . . 122
5.4.1 Backward Reachable Set . . . 123
5.4.2 Resulting Online Control Law . . . 124

5.5 Backward Controller Synthesis for Nonlinear, Continuous-Time Systems 124
5.5.1 Overview 125
5.5.2 Reference Trajectory 125
5.5.3 Zonotope Order Reduction 127
5.5.4 Backward Reachable Set 127
5.5.5 Reachability Analysis 128
5.5.6 Optimization Problem 131
5.6 Numerical Example 134
5.7 Discussion 140
5.8 Summary 142

6 Online Controller Synthesis **145**
6.1 Introduction 145
6.2 Problem Formulation 146
6.3 Reachset Model Predictive Control 146
6.3.1 Overview 146
6.3.2 Dual-Mode MPC 149
6.3.3 Considering the Computation Time 150
6.3.4 Contraction Constraint 150
6.3.5 Optimal Control Problem 152
6.3.6 Tightened Constraints 152
6.3.7 Guarantees Through Reachability Analysis 153
6.3.8 Reachset MPC Theorem 153
6.3.9 Extension 155
6.4 Numerical Example 155
6.5 Discussion 158
6.6 Summary 159

7 Practical Application **161**
7.1 Autonomous Car 161
7.1.1 Obtaining the Model 162
7.1.2 Obtaining Real Vehicle Data 162
7.1.3 Results from Conformance Checking 163
7.1.4 Computation of the Maneuver Automaton 163
7.1.5 Planning with Motion Primitives 169
7.2 Robot Manipulator 176
7.3 Summary 186

8 Conclusion and Future Directions **187**
8.1 Conclusion 187
8.2 Future Directions 189

A Closed-Form Expressions of Convex Combinations **193**
A.1 Simplices 194
A.2 Parallelotopes 197
A.3 General Polytopes 200
A.4 Numerical Example 202
A.5 Summary 203

References **207**

Author's Publications **225**

List of Figures

1.1 Comparison testing, formal verification, and combined control and verification . . 2
1.2 Offline computation of maneuver automata 5
1.3 Online planning with maneuver automata using sampling 6
1.4 Online planning with maneuver automata using a reference trajectory 7

2.1 Polytope representations 17
2.2 Illustration construction zonotope 18
2.3 Illustration construction polynomial zonotope 19
2.4 Illustration of reachable set computation 23

3.1 Planning with non-convex obstacles 31
3.2 Basic idea of the convex control approach 33
3.3 Convex control for nonlinear systems 36
3.4 Reachable sets for the *turn left* motion primitive with the convex controller . . . 46
3.5 Initial and shifted final sets for motion primitives with the convex controller . . . 47
3.6 Resulting maneuver automaton for the car model 48
3.7 Comparison number of extreme points vs. number of generators 49
3.8 Optimization in generator space 52
3.9 Generator interpolation control for nonlinear systems 55
3.10 Reachable sets for the *turn left* motion primitive with the generator controller . . 58
3.11 Initial and shifted final sets for motion primitives with the generator controller . 60
3.12 Comparison trajectory vs. set-based optimization 62
3.13 Reachability analysis in nonlinear program 64
3.14 Illustration of the two time scales for control and for reachability analysis 65
3.15 Reachable sets at different times during the optimization of the controller 69
3.16 Initial and shifted final sets for platoon example 70
3.17 Overview combined control approach 72
3.18 Illustration input distribution of classical control approach 73
3.19 Illustration input distribution of combined control approach 73
3.20 Initial and final sets for the linear platooning example 79
3.21 Reachable sets for the nonlinear car example 80
3.22 Initial and final sets for the nonlinear car example 81

3.23 200 simulations for the nonlinear car example 81
3.24 Comparison of tube-based MPC with our combined control approach 83

4.1 Effects of different reference trajectories on mode intersection of reachable sets . 94
4.2 Overview set-based control approach for PWA systems 96
4.3 Optimization in generator space for PWA system 105
4.4 Reachable sets for two tank example 109
4.5 State and input trajectories of 5,000 simulation runs for two tank example 110

5.1 Overview online application of safety net controller 120
5.2 Illustration of the backward controller synthesis for nonlinear systems 126
5.3 Illustration of the input scaling due to the reference trajectory 129
5.4 Scaling of reachable sets 131
5.5 Reachable sets for the safety controller for a *turn left* motion primitive 135
5.6 Initial and shifted final sets for motion primitives with the safety controller . . . 136
5.7 Plot of 100 simulations of the LQR controller and MPC with safety net 138
5.8 Inputs of the 100 simulations of the LQR controllers and MPC with safety net . 139
5.9 Comparison backward controller with convex interpolation controller 140

6.1 Illustration of our reachset MPC approach 147
6.2 Results of our reachset MPC approach for first example 157
6.3 Results of MPC approach from literature for first example 157
6.4 Results of our reachset MPC approach for second example 158

7.1 Disturbance traces for six example cases from the conformance checking 163
7.2 Measurement error traces for six example cases from the conformance checking . 164
7.3 Reachable sets of motion primitives for the autonomous car application 166
7.4 Initial and final sets of example motion primitives 167
7.5 Illustration of the coordinate system relative to the reference trajectory 168
7.6 Over-approximation of the occupancy set and its use for collision checking 169
7.7 Results of our safe motion planning for the first scenario 171
7.8 Results of our safe motion planning for the second scenario 172
7.9 Results of our safe motion planning for the third scenario 173
7.10 Results of our safe motion planning for the fourth scenario 174
7.11 Schunk LWA 4P robot 176
7.12 Illustration of the reachable sets of our feedback controller for the six joints . . . 181
7.13 3-dimensional visualization of the test trajectory 182
7.14 Measurements of joint positions with reachable set 183
7.15 Measurements of joint velocities with reachable set 184
7.16 Measurements of the closed-loop system 185

A.1 Illustration closed-form expression for simplices 195
A.2 Illustration closed-form expression for parallelotopes 197
A.3 Illustration closed-form expression for general polytopes 201

List of Tables

3.1 Comparison of the four offline control algorithms 87

A.1 Average run times for simplices and parallelotopes 203
A.2 Average run times for general polytopes 204

Chapter 1

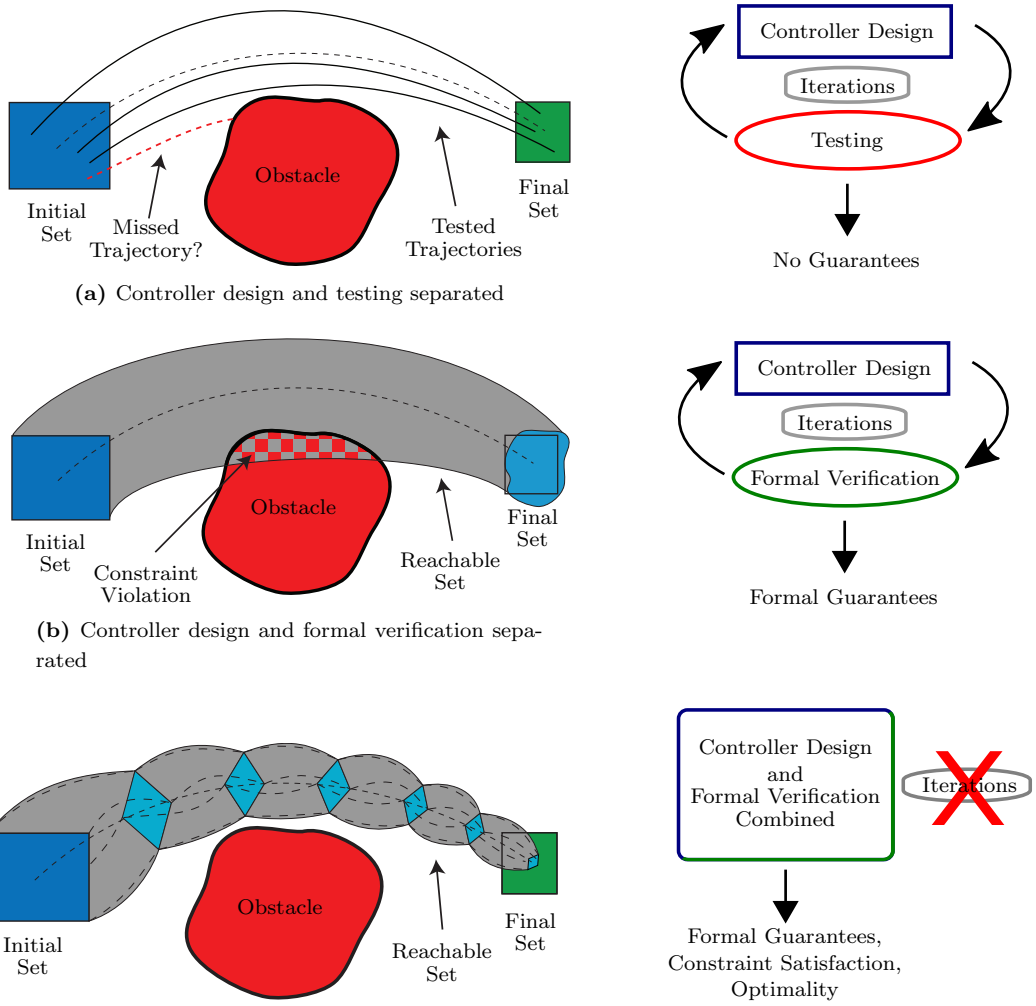
Introduction

1.1 Motivation

Autonomous systems working closely with humans have a huge potential to improve all of our lives. Recent progress in technology and computing capabilities is finally enabling their large-scale application. Autonomous vehicles, robots working in collaboration with or assisting humans, and robotic surgeries are just some of the many examples. All of these applications are key technologies, as they promise to make our life, work, and medical treatment easier and safer. Self-driving cars could change the entire traffic situation in cities, making driving safer, less stressful, and more efficient, as well as reducing the number of overall vehicles, thereby solving parking problems. Robots collaborating with humans on the same tasks can combine the strength and precision of robotic manipulators with the agility and flexibility of humans. Furthermore, medical robots might be able to perform surgeries more precisely than is possible for any human.

At the same time, all these systems are challenging to control: They exhibit complex, nonlinear dynamics and are limited in their actuation capabilities, e.g., limited steering angles or acceleration and braking forces for the self-driving vehicle or limited torques for robotic manipulators. And as all real-world systems, they are affected by external disturbances and uncertain measurements. While being hard to control, their correct operation and the satisfaction of safety constraints is absolutely crucial for their application in tight interaction with humans, as any failure could have severe and even fatal consequences. It must be ensured that a self-driving vehicle never causes any accidents, the robotic manipulator never accidentally hits a human worker, and the robotic surgeon does not damage surrounding tissue or organs. Thus, finding novel solutions to guarantee the safety of autonomous systems is crucial. The new control algorithms developed in this thesis aim at providing these solutions by directly incorporating formal verification.

Until now, there have been no techniques to obtain efficient controllers with formal guarantees for complex systems affected by uncertainties. In the vast majority of industries, the current procedure is to design a controller for a system and then extensively test it. For a set



(a) Controller design and testing separated

(b) Controller design and formal verification separated

(c) Controller design and formal verification combined

Figure 1.1: (a) Testing alone is never sufficient, as one cannot be sure that no unsafe trajectories were missed. (b) Formal verification can provide formal guarantees but does not use the full potential of the controller. (c) The full potential can be obtained by combining control and verification in one approach, which leads to formal guarantees and better control performance.

of possible initial states and a set of possible disturbances affecting the system, one chooses finitely many combinations and checks, for example, through simulation if the system's behavior is safe. If this is not the case for some combinations, one has to go back, design a new controller and test it again. This might lead to many design iterations, where the results from the failed tests do not necessarily help to find a better controller in the next iteration. A bigger problem, however, is that testing can never completely ensure the absence of errors, as one

cannot be certain whether one of the infinitely many non-tested scenarios might lead to an unsafe situation, see Fig. 1.1(a). This can only be guaranteed by using formal methods, which check all possible combinations.

For continuous systems, one of the most promising formal methods is reachability analysis [1, 2]. Here, formal guarantees are obtained by computing the (over-approximative) reachable set, i.e., the set which contains all possible solutions of a system for a whole set of initial states and all possible inputs and disturbances, see Fig. 1.1(b). While reachability analysis ensures that no combination is missed, this approach still has the drawback that every time the verification indicates unsafe behavior, a new controller has to be computed and reverified without being able to use much information from the previous verification steps.

The main idea of this work is to overcome these problems by directly incorporating the formal verification in the controller synthesis process. By applying numerical optimization methods, which have proven their efficiency in many classical applications, we are able to use the knowledge gained from reachability analysis in the controller design to obtain the optimal controller. While classical approaches often keep certain safety margins to the actual limits of the system to decrease the chance of violating safety constraints due to insufficient testing, our new method is able to safely operate the controller close to the limits of the system. This enables the controller to optimally react to difficult and dangerous situations, while ensuring that the constraints are satisfied at all times, see Fig. 1.1(c).

By integrating the optimization over reachable sets in the controller synthesis problem, the whole approach can be implemented as a simple "push-button" approach, where the user does not need a deep understanding of control methods, but can simply start the optimization. Similarly, through the use of reachable set computations, we do not require the user to know Lyapunov functions, for example, or perform other stability checks. This, together with the rather simple control structures used in this thesis, makes the presented approaches useful for practical application.

From a control theory point of view, most considered problems can be viewed as reach-avoid problems, where the system must be controlled close to a given final state at a fixed final time while avoiding unsafe regions, both in the state and input spaces. Examples include autonomous vehicles that should reach a desired position while avoiding other traffic participants and staying on the road, or robot manipulators whose end-effector must reach a desired position without colliding with surrounding persons or objects.

For a single initial state and undisturbed systems, these reach-avoid problems can be solved efficiently using numerical optimization tools such as multiple shooting [3]. However, if the exact initial state is not known beforehand or cannot be measured exactly due to noisy sensors, one has to consider a whole set of initial states, which leads to a similar problem as with testing: the optimal control problem has to be solved for each initial state. Since this cannot be done as there are uncountably many states, the optimization has to be performed online, often without guarantees that a solution exists for every possible state or if the optimization can be performed in time.

Our set-based approaches consider this problem while taking advantage of existing efficient optimal control and optimization approaches developed for single states. In different algorithms

presented in this thesis, we use the optimal control approaches to solve the problem for all possible states in the initial set. We do so by either iteratively solving optimal control problems for finitely many states and interpolating between them or by directly optimizing the reachable sets of the closed-loop dynamics. This allows us to obtain good control performance, while the included reachability analysis formally verifies the resulting controller.

In fact, this combination of heuristics with formal verification provides a significant advantage to our approaches: As we deal with complex nonlinear systems, many useful properties like convexity or the superposition principle no longer hold, and any exact and formal calculation might become prohibitively hard or conservative. Since we aim for efficient methods which can be actually applied in practice, we take advantage of the fact that efficient solutions often exist for simplified problems, e.g., by linearizing or time-discretizing the system dynamics or neglecting disturbances. By using solutions of the simplified dynamics to obtain controllers for the real dynamics, we find efficient heuristics to solve the problem. Due to the integrated reachability analysis, however, we are still able to obtain formal guarantees[1] for the actual nonlinear dynamics with disturbances and uncertain measurements.

1.2 Integration in Autonomous Systems

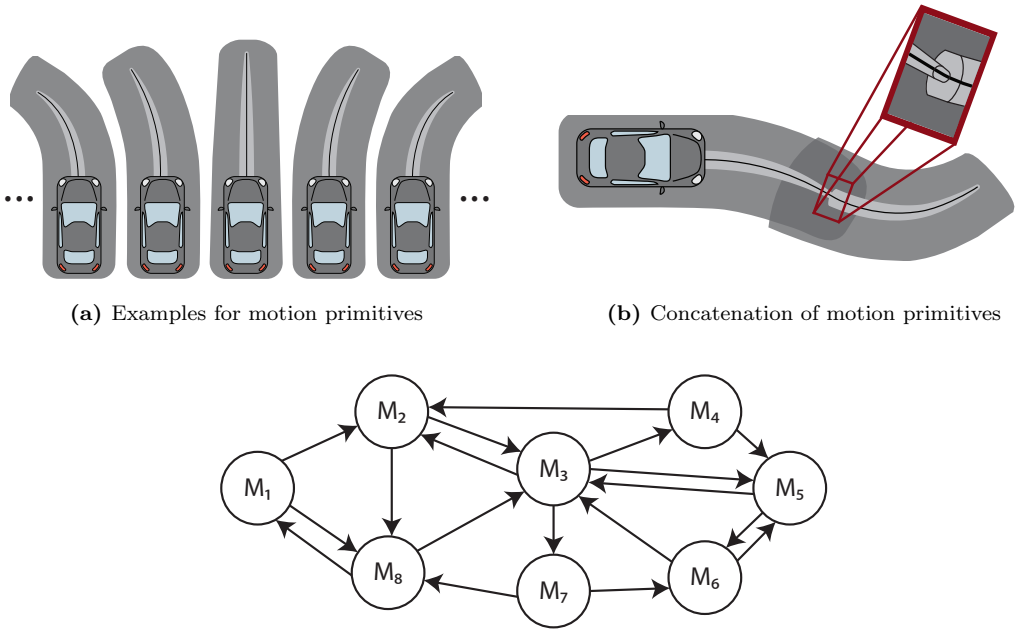
It is crucial for autonomous systems acting in human environments to be able to react to unexpected situations. Since there are infinitely many such situations, it is impossible to design and verify controllers offline for all possible scenarios. On the other hand, for fast and complex systems, it is also impossible to do the complete verification online, since it might take too long and we might end up in an unsafe situation.

Developing safe maneuver automata is a useful way to overcome this problem [4]. We present the idea for the example of autonomous driving as illustrated in Figs. 1.2 – 1.4[2]; however, maneuver automata can be used for many other systems as well. To obtain a maneuver automaton, we begin by computing controllers for a number of short trajectory pieces and verify them offline in advance (see Fig. 1.2(a)). Two of these so-called motion primitives can be concatenated if the initial set of a motion primitive is completely contained in the final set of the previous one (Fig. 1.2(b)). We store the motion primitives in a maneuver automaton, which contains the motion primitives as states and has transitions from one state to another, if the corresponding motion primitives can be connected (Fig. 1.2(c)).

After precomputing all of the motion primitives and constructing the maneuver automaton offline in advance, the online planning and control problem simplifies to a discrete search using those motion primitives according to the specific situation (Fig. 1.3(a) and Fig. 1.4(a)). Based on the current situation and prediction of other traffic participants, which can be done using tools such as SPOT [5], we can either use the motion primitives for a sampling-based search (Fig. 1.3(b)) or we can use a reference trajectory to guide the search (Fig. 1.4(b)). In the

[1] The approaches in this thesis are formally correct under the assumption of finite precision arithmetic; for our implementation, we neglect rounding errors, but the presented theory results in a formally correct approach. Interval arithmetic can be used to consider rounding errors.

[2] Figs. 1.2 and 1.3 will also be published in [214]



(a) Examples for motion primitives

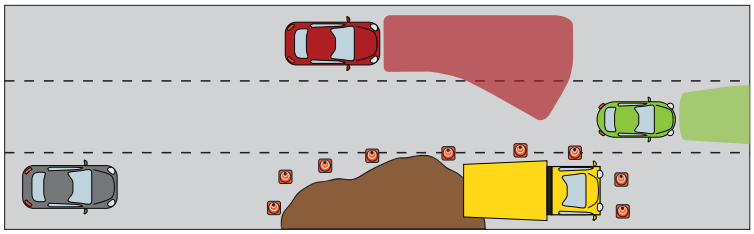
(b) Concatenation of motion primitives

(c) Resulting maneuver automaton

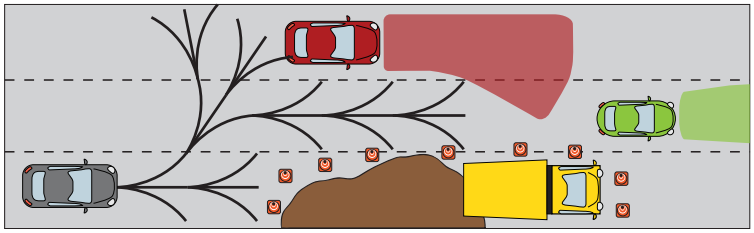
Figure 1.2: Overview of the offline computation of maneuver automata for autonomous driving: (a) First, a variety of motion primitives for different maneuvers is computed. By using reachability analysis, we obtain the uncertainty set (light gray) of the car around the reference trajectory (black line), as well as the resulting uncertain occupancy sets (dark gray). (b) It is checked which motion primitive can be concatenated. (c) The resulting motion primitives are stored as states in the maneuver automaton and transitions show which ones can be connected.

first case, we can use search trees and similar approaches [6] to find a good combination of motion primitives. In the second case, we compute a reference trajectory with existing tools for simplified dynamics and match it with our motion primitives [215]. In either case, we use the precomputed reachable sets of our motion primitives to check for a collision with the predicted possible reachable sets of other traffic participants and select the best choice (Fig. 1.3(c) and Fig. 1.4(c)) which is safe and follow it (Fig. 1.3(d) and Fig. 1.4(d)). The planning can then be performed in an iterative fashion with a moving horizon, where fail-safe trajectories can be used to ensure that a safe fallback solution exists, if we cannot find a new, feasible solution in time (see [7]). This ensures that a safe control strategy exists for any situation.

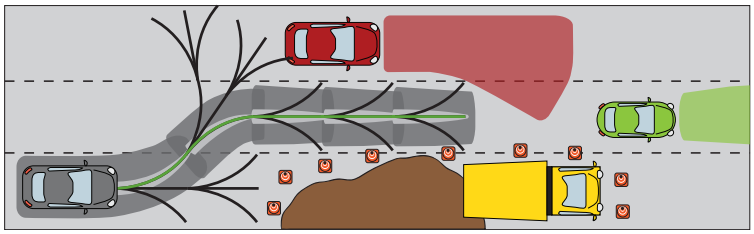
The quality of the outcome of the online planning directly depends on the size of the maneuver automaton and the number of transitions, which provide flexibility during the online planning. In order to have a maneuver automaton with as many transitions as possible, we need controllers which are able to steer all states from a large initial set into a small final set. So far, there is no efficient way to obtain controllers for disturbed nonlinear systems which guarantee



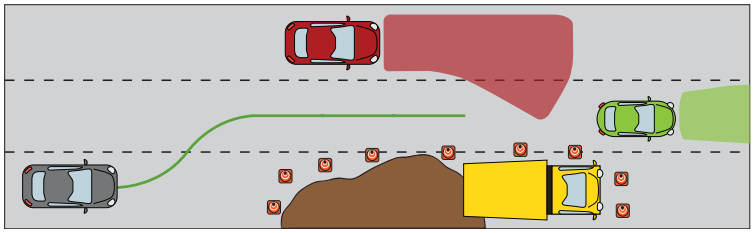
(a) Occupancy prediction of other traffic participants



(b) Discrete trajectory planning using motion primitives

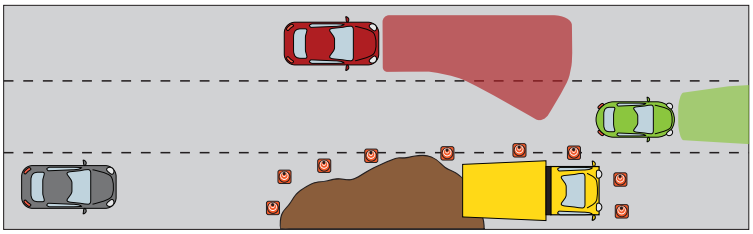


(c) Check for collisions using precomputed reachable sets

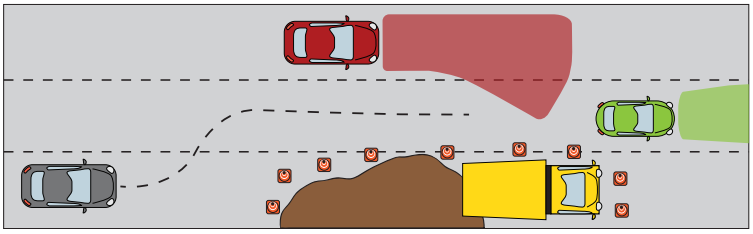


(d) Drive safe trajectory using precomputed controllers

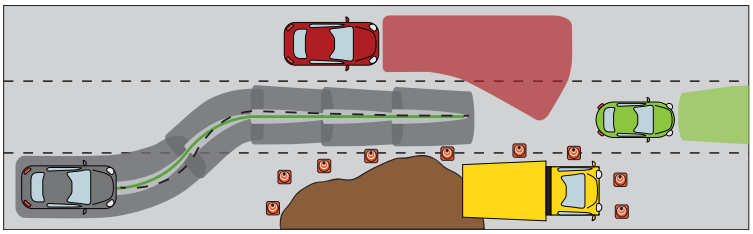
Figure 1.3: Overview of the online planning with maneuver automata for autonomous driving using sampling: (a) During online application, we know the actual situation and can predict the possible behaviors of other traffic participants. (b) We use the precomputed motion primitives from the maneuver automaton to sample different possible trajectories for the controlled vehicle. (c) We check for possible collisions using the precomputed reachable sets of our motion primitives and the predicted reachable sets of other traffic participants and choose the best solution (reachable sets are only depicted for the optimal solution). (d) We use the controllers corresponding to the chosen motion primitives to drive along the trajectory.



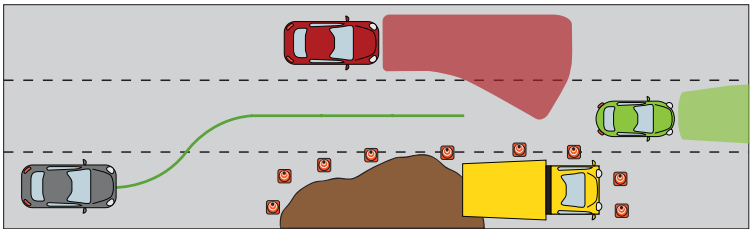
(a) Occupancy prediction of other traffic participants



(b) Compute reference trajectory with simplified dynamics



(c) Match reference trajectory with motion primitives



(d) Drive safe trajectory using precomputed controllers

Figure 1.4: Overview of the online planning with maneuver automata for autonomous driving using a reference trajectory: (a) During online application, we know the actual situation and can predict the possible behaviors of other traffic participants. (b) We compute a possible reference trajectory using existing planning approaches and simplified dynamics. (c) We match this reference trajectory without motion primitives, thereby ensuring the driveability and collision-freeness of the planned motion using the precomputed reachable sets of our motion primitives and the predicted reachable sets of other traffic participants. (d) We use the controllers corresponding to the chosen motion primitives to drive along the trajectory.

that all states from an initial set are inside a small final set after a fixed time while taking state and input constraints into account. The offline controller synthesis approaches developed in this thesis aim at solving this problem in order to obtain highly connected maneuver automata which enable efficient planning.

1.3 Related Work

There exist many methods in the literature which aim to control constrained nonlinear systems despite disturbances. However, providing guarantees without becoming too conservative or resulting in overly complex computations is still an open challenge. A direct way to obtain optimal control inputs, which satisfy state and input constraints, is by explicitly solving the corresponding Hamilton-Jacobi-Bellman (HJB) equation or by using dynamic programming [8–11]. While these approaches work well for single states and undisturbed systems, when considering disturbances or sets of states, they become prohibitively difficult for high-dimensional systems and their use is therefore limited to low-dimensional systems.

Another well-known method is to use model predictive control (MPC) [12, 13], which allows one to take state and input constraints into account. There are two different types: In the classical, implicit case, an open-loop optimal control problem is solved iteratively for a finite prediction horizon online. After each step, only the input for the first time interval is applied to the system and a new optimal control problem with a shifted prediction horizon is solved. Since all computations have to be performed online, solving the optimization problem in real time becomes challenging for complex and fast systems and is often infeasible when formal guarantees are required. An alternative to solving the optimization problem online is provided by explicit MPC, where the state space is divided into different regions, and for each region, an (sub-)optimal control law is computed offline which satisfies the constraints [14, 15]. This works for linear [16–18] and for nonlinear systems [19, 20], even with disturbances [21]. While this reduces the online computational effort, due to the division of the state space, this approach becomes easily computationally infeasible for higher-dimensional systems.

An established form of robust MPC is tube-based MPC, where an auxiliary controller is used to keep the system in a tube around the optimized trajectory [22–24]. For linear systems, this works well due to the superposition principle [22, 25]. For nonlinear systems, this is harder and more conservative due to the often fixed auxiliary controller and over-approximation of the invariant sets for the tubes. Different auxiliary controllers have been proposed, e.g., sliding mode controllers [26], dynamic games [27], or even a second MPC controller [28]. Recent approaches suggest the application of contraction theory to obtain a controller [29]. Other developments are parametrized [23] and homothetic MPC [24], which can adapt the tube size. Some approaches combine MPC with online reachability analysis using interval arithmetic [30] and reachable sets based on zonotopes [31]; however, they only consider fixed feedback controllers and discrete-time systems and use conservative approximations of the reachable sets.

Another method which controls all states around a single trajectory is presented in [32]. This method exploits so-called trajectory robustness, which is, however, restricted to feedback-linearizable and differentially flat systems, and it does not take disturbances into account. An

extended version using feedback control is shown in [33].

Controlling all states in so-called funnels, similar to the tubes in tube-based MPC, to a final set for disturbed nonlinear systems can be done using linear quadratic regulator (LQR) trees [34–36]. In these works, the authors use sums-of-squares programming to find LQR tracking controllers [37], which satisfy specified constraints. This method is also successfully applied to safe maneuver automata [38]; however, they are often restricted to polynomial dynamics. In addition, the computational effort of sums-of-squares techniques can grow very fast with the system dimension and considered polynomial degree. Similar approaches which can be used for motion planning are shown in [39], where the positive invariant sets of different controllers are concatenated in trees to obtain maneuver automata. More approaches using maneuver automata for safe motion planning can also be found in [40, 41], and using reachability analysis for verifying the safety of maneuvers in [42, 43].

Another way for obtaining controllers with safety guarantees for nonlinear systems is the combination of control barrier functions with control Lyapunov functions [44–49]. They use quadratic programs to online connect safety constraints and desired performance. Since the computation of the control barrier and control Lyapunov functions is often done by solving sums-of-squares programs, they face the same problem of a fast growing number of parameters with the systems dimension and considered polynomial degree as the LQR-trees methods mentioned before.

A large class of controllers which provide formal guarantees are abstraction-based controllers [50–66]. Most of them obtain a finite state abstraction of the real system and use approaches from automata theory and computer science to compute controllers which can even take complex specifications into account. However, as they often do this by discretizing the state and input spaces, they suffer from the curse of dimensionality, i.e., the exponential growth in computational complexity with the number of states and inputs. Some methods try to avoid computing abstractions of the whole state space but mainly focus on the satisfaction of more complex requirements and only consider undisturbed systems [67–71]. Other approaches for controller synthesis with formal specification (again, only for undisturbed systems) combine genetic programming algorithms with Lyapunov functions [72].

1.4 Organization of the Thesis and Contributions

In this thesis, we present a number of approaches on how to compute safe controllers for disturbed, nonlinear systems by combining control theory, reachability analysis, and optimization.

Chapter 2: Background

We begin by providing some mathematical notations and useful definitions in Chapter 2. This chapter also contains some background on different set representations, on how some basic algorithms for reachability analysis work, on motion primitives and maneuver automata, and on how conformance checking can be used to obtain models which safely contain the behavior of real systems.

Chapter 3: Offline Controller Synthesis

One of the main chapters is Chapter 3, which covers the offline computation of safe controllers for disturbed nonlinear systems. We present four different algorithms, which all solve the problem of controlling all states from some initial set close to a desired final state in finite time, while guaranteeing the satisfaction of state and input constraints for all times. These algorithms are particularly suitable for computing motion primitives offline, which can be used for online control with maneuver automata:

- The first algorithm interpolates open-loop solutions computed for the extreme states of the initial set and thereby obtains a control law for each state in the initial set. By applying this approach iteratively and combining it with reachability analysis, we are able to extend the theory developed for linear systems to nonlinear systems.

- The second approach also interpolates open-loop solutions; however, it uses a different set-representation to obtain the interpolated control law, which leads to a much better scaling of the overall computational complexity.

- In the third approach, instead of interpolating optimal open-loop solutions, we use a continuous feedback controller, whose parameters we obtain by directly optimizing over the reachable sets of the closed-loop dynamics.

- Finally, we present a fourth approach which combines the ideas of the interpolation-based controllers and the continuous feedback controller, thereby keeping the advantages of the respective controllers while overcoming their disadvantages. The resulting controller utilizes a state-dependent feedforward controller, which allows us to obtain an individual reference trajectory for each state of the initial set by solving a single linear program. This individual reference trajectory is then tracked by a continuous feedback controller which is again obtained by optimizing over reachable sets.

Contribution: The main contribution of this chapter are four algorithms which allow us to efficiently compute controllers with formal guarantees for constraint satisfaction for nonlinear systems subject to external disturbances and sensor noise. All optimizations and reachable set computations are performed offline, and the resulting controllers have a simple structure, which makes them suitable for fast systems. The algorithms are easy to apply, and due to the included reachability analysis, there is no need for additional invariant set computations, stability proofs, or finding a Lyapunov function. The results have been published at the International Conference on Hybrid Systems: Computation and Control [216], the IFAC World Congress [217], the American Control Conference [218], and IEEE Transaction on Automatic Control [219]. We applied the control algorithms from this chapter in various works which have been published, for example, at the American Control Conference [220,221], the Intelligent Transportation Systems Conference [215], and the International Symposium on Automated Technology for Verification and Analysis [222].

Based on the algorithms from this chapter, as well as from other chapters, we developed the

MATLAB Toolbox AROC[1], which was introduced at the International Conference on Hybrid Systems: Computation and Control [223]. For the efficient online application of the control law based on interpolating extreme states, we also developed new ways for computing analytical closed-form expressions of convex combinations, which are shown in Appendix A. They were published at the American Control Conference [224] and applied to the robust control of power systems, which we published at the IEEE Power and Energy Society General Meeting [225]. We also developed new methods for the order-reduction of zonotopes and published the results at the International Conference on Decision and Control [226].

Chapter 4: Offline Controller Synthesis for Piecewise Affine Systems

In Chapter 4, we extend the offline controller synthesis from Chapter 3 to hybrid dynamics through the example of piecewise affine systems. Hybrid dynamics combine both continuous and discrete dynamics, which makes computing reachable sets and corresponding controllers challenging. The piecewise affine systems which we consider have different affine dynamics in different parts of the state space. Therefore, every time the system enters a new region, the dynamics change, so that the reachable sets have to be split and computed for each dynamics individually. Since this drastically increases the computational burden, we present a novel way of computing a reference trajectory and then tracking it with a controller based on one of the interpolation-based controllers from Chapter 3. By including the distance to the boundaries of each region into the optimization problems of the reference trajectory and feedback controllers, we are able to reduce the number of splits, thereby significantly reducing the computational effort and improving the control performance.

Contribution: The main contribution of this chapter is the ability to efficiently compute a formally correct controller for disturbed piecewise affine systems, which keeps the states inside the same mode sequence as the reference trajectory, whenever possible. This allows for faster computations, controller synthesis for higher dimensions, as well as simpler control laws than existing solutions. In addition, in contrast to many existing approaches, we do not need to tighten constraints or compute invariant sets. Since all mixed-integer problems are solved offline, the online computation becomes very simple. We also directly consider the case that it might not be possible to avoid splitting the reachable set and, if so, provide countermeasures during the controller synthesis. The results of this chapter are published in the journal Nonlinear Analysis: Hybrid Systems [227].

Chapter 5: Offline Controller Synthesis Using Backward Reachable Sets

All approaches in Chapter 3 optimize over forward reachable sets to obtain the smallest reachable set for a given initial set. In Chapter 5, we are instead interested in the largest initial set for which we can find a controller such that all trajectories starting from this initial set end in a

[1] Available online at `https://aroc.in.tum.de`.

given final set. Thus, we start from the final set and go backwards in time, thereby optimizing over backward reachable sets. Since safely computing backward reachable sets is much harder than computing forward reachable sets when doing controller synthesis, we present a novel way of optimizing backward reachable sets by combining linear approximations of backward reachable sets with optimization over forward reachable sets.

The controllers based on backward reachable sets are particularly suitable as safety-net controllers which safeguard a classical, unverified controller. There exists a wide variety of controllers which work well in practice and are tuned to optimize certain performances such as reduced wear, reduced energy consumption, or improved comfort. While many of these controllers cannot be formally verified, they are safe most of the time. Therefore, we present an approach where we monitor the behavior of the unverified controller and apply it while it is safe, i.e., most of the time. However, when the behavior would become unsafe, the safety controller steps in and ensures safety at all times.

Contribution: The contribution of this chapter is threefold:

(i) We develop a novel backward reachability algorithm, which is not only suitable for efficient optimization of the controller parameters, but is in fact the first algorithm which computes under-approximative backward reachable sets for general nonlinear systems with inputs and disturbances. In addition, this reachable set algorithm allows for an efficient optimization of controller parameters without having to perform the reachable set computation in every iteration.

(ii) We use this new way of computing backward reachable sets to generate a novel set-based controller which maximizes the set of initial states which are guaranteed to be steered into a given final set despite the presence of disturbances. The controller has the same advantages of the controllers in Chapter 3, i.e., it provides formal safety guarantees without the need for computing invariant sets or knowing a Lyapunov function, the optimization is performed offline, and the online controller has a simple structure. In addition to these advantages, our new control approach is especially suitable for the application as a safety net controller.

(iii) We provide a novel way of applying optimal, however, not formally verified, controllers in a safety-critical setting and safeguarding them by our formal, set-based controller. By including efficient reachable set computations for predicting the future behavior of the system controlled by the unverified controller, we ensure that the safety controller takes over if it is necessary to prevent unsafe behavior. We are even able to explicitly consider the computation time for our controller and reachable set computations during online verification.

The results of this chapter are going to be published in the journal IEEE Transaction on Automatic Control [214]. The ideas for efficiently obtaining subsets of reachable sets without having to recompute the whole reachable set, which we use for the reachable set optimization in this chapter, are useful for many other applications as well. We show this in [228], which has been published at the International Conference on Hybrid Systems: Computation and Control.

Chapter 6: Online Controller Synthesis

As an alternative to precomputing everything offline, we discuss safe online control in Chapter 6. We present a robust model predictive control algorithm where we use reachable sets instead of fixed tubes to ensure safety despite disturbances. As typically done in MPC, we only optimize over the reference trajectory and not over the feedback controller in order to reduce the online computation time. By checking the safety using reachability analysis and ensuring that we always have a safe fallback solution if we cannot find a new feasible solution, we are able to guarantee safety while reducing the conservatism caused by a fixed tube size.

Contribution: The main contribution of this chapter is the first robust model predictive control algorithm based on reachable sets that provides formal guarantees for the satisfaction of constraints for nonlinear systems with external disturbances and sensor noise, while both taking computation time into account and not being restricted to a fixed tube-size. The results of this chapter were published at the International Conference on Decision and Control [229].

Chapter 7: Practical Application

We apply the theory from the previous chapter to two applications in Chapter 7: First we obtain a model based on real driving data for an autonomous vehicle and compute a maneuver automaton with over 30,000 different motion primitives. We use this maneuver automaton to compute fail-safe trajectories and ensure their driveability for a number of different traffic scenarios generated from real-world data. In the second half of this chapter, we apply our set-based controller synthesis approaches to a lightweight robotic manipulator, where we compute a safe tracking controller and show its applicability for collision-free planning.

Contribution: The main contribution of this chapter is the demonstration that the algorithms from the previous chapters are applicable to real systems in different domains, despite real-world effects. It also shows the feasibility of generating a large maneuver automaton for fast online application for the autonomous vehicle, as well as the possibility and advantages of combining our controller with classical approaches, like feedback linearization for efficient online control. The results for the autonomous vehicle expand on the methods and results published at the Intelligent Transportation Systems Conference [215]. The results for the robot have been submitted to the journal IEEE Transactions on Robotics [230].

Chapter 8: Conclusion and Future Directions

We finish with Chapter 8, where we give an summary of the findings of this thesis, discuss the obtained solutions, and provide an outlook of possible future research directions.

Chapter 2

Background

In this chapter, we give an overview of the notation used, provide some necessary definitions, describe how to compute reachable sets, how to efficiently represent those sets, and how to check conformance of complex systems.

2.1 Notation

We denote by $\mathbb{R}$ the set of real numbers, with $\mathbb{R}_0^+$ being the set of nonnegative real numbers, and $\mathbb{R}^+$ the set of positive real numbers. The set of positive integers is denoted by $\mathbb{N}$, the set of nonnegative integers by $\mathbb{N}_0$, and the empty set by $\emptyset$. For two real numbers $a, b \in \mathbb{R}$, with $a \leq b$, we denote by $[a, b]$ and (a, b) the closed and open sets between a and b, respectively. We use $[a, b]^n$ to denote an n-dimensional interval (or hypercube) with size $[a, b]$ in each dimension. For a vector $x \in \mathbb{R}^n$, we denote its transpose by x^T, and we refer to its elements with $x_i \in \mathbb{R}$. Likewise, for a matrix $A \in \mathbb{R}^{n \times m}$, we denote its transpose by A^T, and we refer to its elements with $A_{i,j} \in \mathbb{R}$. For a scalar x, we denote the absolute value of x by $|x|$. For a vector $x = [x_1, \ldots, x_n]^T$, we denote by $|x|$ the vector containing the absolute values of the entries of x, i.e., $|x| = [|x_1|, \ldots, |x_n|]^T$. We denote the identity matrix by $I \in \mathbb{R}^{n \times n}$ and the unit vectors by $e^{(i)}$, i.e., $e^{(i)}$ is the i-th column of I. We use $\text{diag}(x)$, where $x \in \mathbb{R}^n$, to denote a diagonal matrix with the entries of x on its diagonal and zeros everywhere else. The vector containing all ones is denoted by $\mathbf{1}$, and the vector containing only zeros is referred to as $\mathbf{0}$. The boundary of a set $\mathcal{S} \subset \mathbb{R}^n$ is denoted by $\mathcal{B}(\mathcal{S})$, and we write $\mathcal{I}(\mathcal{S})$ for the interior of $\mathcal{S}$, such that $\mathcal{S} = \mathcal{B}(\mathcal{S}) \cup \mathcal{I}(\mathcal{S})$.

2.2 Convexity

Let us recall some basics about convex combinations and convex sets. For more details, see e.g. [73].

Definition 1 (Convex Combination)**.** *A vector $x \in \mathbb{R}^n$ is called a convex combination of q*

given vectors $\breve{x}^{(1)}, \ldots, \breve{x}^{(q)} \in \mathbb{R}^n$*, if and only if* x *can be written as*

$$x = \sum_{i=1}^{q} \lambda_i(x) \breve{x}^{(i)},$$

with $\lambda_i(x) \geq 0$*, and* $\sum_{i=1}^{q} \lambda_i(x) = 1$*.*

Convex combinations can be used to define a convex set as follows:

Definition 2 (Convex Set)**.** *A set* $\mathcal{S} \subset \mathbb{R}^n$ *is convex if and only if any convex combination of any two points* $x^{(1)}, x^{(2)} \in \mathcal{S}$ *lies inside the set, i.e.,*

$$\forall \lambda \in [0,1]: \ \lambda x^{(1)} + (1-\lambda) x^{(2)} \in \mathcal{S}.$$

Definition 3 (Extreme Point)**.** *Let* $\mathcal{S} \subset \mathbb{R}^n$ *be a compact*[1] *convex set. A point* $\breve{x}^{(0)} \in \mathcal{S}$ *is called an extreme point of* $\mathcal{S}$ *if it cannot be represented by a convex combination of any two points* $\breve{x}^{(1)}, \breve{x}^{(2)} \in \mathcal{S}, \ \breve{x}^{(1)} \neq \breve{x}^{(0)}, \ \breve{x}^{(2)} \neq \breve{x}^{(0)}$*, i.e.,*

$$\nexists \lambda \in (0,1) \text{ such that } \lambda \breve{x}^{(1)} + (1-\lambda) \breve{x}^{(2)} = \breve{x}^{(0)}.$$

Definition 4 (Convex Hull)**.** *The convex hull of a set* $\mathcal{S} \subset \mathbb{R}^n$*, denoted by* $\operatorname{conv}(\mathcal{S})$*, is the set of all convex combinations of points in* $\mathcal{S}$*, i.e.,*

$$\operatorname{conv}(\mathcal{S}) = \left\{ \sum_{i=1}^{q} \lambda_i \breve{x}^{(i)} \;\middle|\; \breve{x}^{(i)} \in \mathcal{S}, \ \lambda_i \geq 0, \ \sum_{i=1}^{q} \lambda_i = 1, \ q \in \mathbb{N} \right\}.$$

From the two previous definitions, it follows that any compact convex set is the convex hull of its extreme points.

2.3 Set Operations

Let us now define the set-based operations for addition, subtraction, and multiplication. We begin with set-based addition.

Definition 5 (Minkowski Sum)**.** *Given sets* $\mathcal{A}, \mathcal{B} \subseteq \mathbb{R}^n$*, the addition of these two sets, also known as the Minkowski sum, is defined as*

$$\mathcal{A} \oplus \mathcal{B} = \{a + b \mid a \in \mathcal{A}, \ b \in \mathcal{B}\}.$$

We use $\bigoplus$ as the set-based summation symbol, analogous to the standard $\sum$ notation. The counterpart to the set-based addition, the set-based subtraction, also known as Minkowski difference is defined as follows:

Definition 6 (Minkowski Difference)**.** *Given sets* $\mathcal{A}, \mathcal{B} \subset \mathbb{R}^n$*, the subtraction of these two sets, also known as the Minkowski difference, is defined as the complement of the Minkowski sum, i.e.,*

$$\mathcal{A} \ominus \mathcal{B} = \{c \in \mathbb{R}^n \mid c \oplus \mathcal{B} \subseteq \mathcal{A}\}.$$

[1] A set is compact if it is closed and bounded.

Note that $\mathcal{X} \ominus \mathcal{Y} \oplus \mathcal{Y} \subseteq \mathcal{X}$. Finally, let us define the multiplication of a set with a matrix:

Definition 7 (Linear Map)**.** *Given a set $\mathcal{A} \subseteq \mathbb{R}^n$ and a matrix $B \in \mathbb{R}^{m \times n}$, the product of the matrix and the set is defined as*

$$B\mathcal{A} = \{Ba \in \mathbb{R}^m \mid a \in \mathcal{A}\}.$$

This matrix set multiplication is also referred to as a linear map.

2.4 Set Representations

Definition 8 (Polyhedron)**.** *A polyhedron is a set defined by the intersection of p half-spaces $\mathcal{H}_1 \cap \cdots \cap \mathcal{H}_p$, $\mathcal{H}_i = \{x \in \mathbb{R}^n \mid C_i x \leq d_i\}, i \in \{1, \ldots, p\}$, i.e.,*

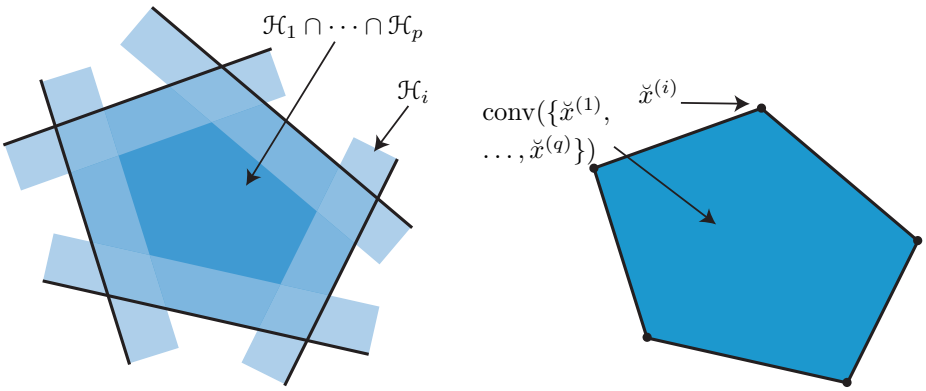
$$\mathcal{P} = \{x \in \mathbb{R}^n \mid Cx \leq d\},$$

with $C \in \mathbb{R}^{p \times n}$ consisting of C_i as its rows and $d \in \mathbb{R}^p$ denoting the vector of entries d_i.

We use the shorthand notation $\langle C, d \rangle_H$ to refer to a polyhedron in half-space representation.

Definition 9 (Polytope)**.** *A polytope is a closed and bounded polyhedron. Instead of representing a polytope as the intersection of half-spaces, it can also be expressed as the convex hull of q extreme points $\breve{x}^{(1)}, \ldots, \breve{x}^{(q)} \in \mathbb{R}^n$, also known as vertices:*

$$\mathcal{P} = \text{conv}(\{\breve{x}^{(1)}, \ldots, \breve{x}^{(q)}\}) \subset \mathbb{R}^n.$$



(a) Half-space representation **(b)** Extreme point representation

Figure 2.1: Two representations of a polytope: (a) As the intersection of p half-spaces $\mathcal{H}_1 \cap \cdots \cap \mathcal{H}_p$ or (b) as the convex hull of q extreme points $\breve{x}^{(1)}, \ldots, \breve{x}^{(q)}$.

A visualization of a polytope represented by half-spaces and extreme points is shown in Fig. 2.1. We call a polytope $\mathcal{P} \subset \mathbb{R}^n$ with $n + 1$ extreme points a simplex, if n of those extreme points are linearly independent. Zonotopes, a special type of polytopes, are illustrated in Fig. 2.2 and defined as follows:

Definition 10 (Zonotope)**.** *A set is called a zonotope if it can be written as*

$$\mathcal{Z} = \left\{ x \in \mathbb{R}^n \;\middle|\; x = c + \sum_{i=1}^{p} \alpha_i g^{(i)},\ \alpha_i \in [-1, 1] \right\}.$$

Therein, $c \in \mathbb{R}^n$ defines the center of the zonotope, and $g^{(i)} \in \mathbb{R}^n, i \in \{1, \dots, p\}$, are p generators. The generators can also be combined in the generator matrix $G \in \mathbb{R}^{n \times p}$, which contains the generators as its columns. We use $\langle c, g^{(1)}, \dots, g^{(p)} \rangle$ and $\langle c, G \rangle$, respectively, as a more concise notation of $\mathcal{Z}$. The order o of a zonotope is defined as the quotient $o = \frac{p}{n}$.

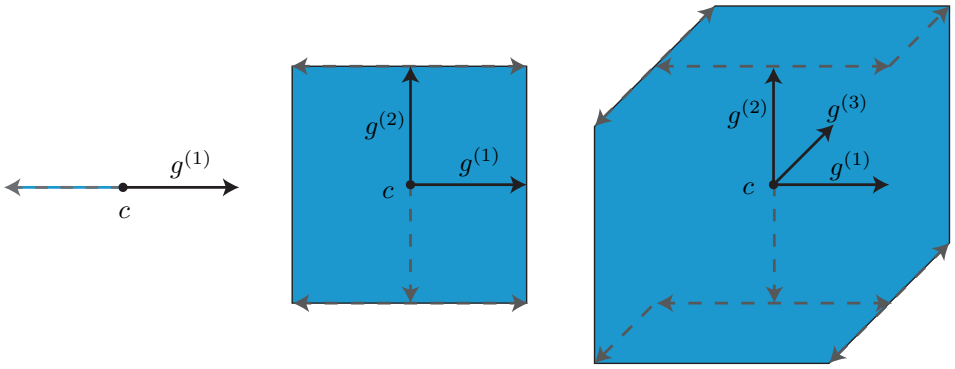


Figure 2.2: Illustration of the construction of a zonotope in $\mathbb{R}^2$. It is defined by its center c and the superposition of the generators $g^{(1)}, g^{(2)}$, and $g^{(3)}$ scaled between -1 and 1. The generators are shown as solid black arrows, and the dashed arrows show different combinations of generators, which correspond to the extreme points of the zonotope.

A zonotope with n linearly independent generators is called a parallelotope. Zonotopes are point-symmetric to their center and have a number of beneficial properties: While a zonotope, as a special type of polytope, can be expressed by its vertices or half-spaces, the generator representation allows a very concise set representation. For example, to represent a parallelotope in $\mathbb{R}^n$, we need 2^n extreme points, but only n generators. In addition, they are closed under Minkowski sum and linear maps, two main operations during the reachability analysis found in [74]. Both operations can be performed very easily: the Minkowski-sum of two zonotopes $\mathcal{Z}_1 = \langle c_{\mathcal{Z}_1}, g^{(1)}_{\mathcal{Z}_1}, \dots, g^{(p)}_{\mathcal{Z}_1} \rangle \subset \mathbb{R}^n, \mathcal{Z}_2 = \langle c_{\mathcal{Z}_2}, g^{(1)}_{\mathcal{Z}_2}, \dots, g^{(q)}_{\mathcal{Z}_2} \rangle \subset \mathbb{R}^n$ is obtained as

$$\mathcal{Z}_1 \oplus \mathcal{Z}_2 = \langle c_{\mathcal{Z}_1} + c_{\mathcal{Z}_2}, g^{(1)}_{\mathcal{Z}_1}, \dots, g^{(p)}_{\mathcal{Z}_1}, g^{(1)}_{\mathcal{Z}_2}, \dots, g^{(q)}_{\mathcal{Z}_2} \rangle,$$

and the linear map of $\mathcal{Z}_1$ under $L \in \mathbb{R}^{n \times n}$ is

$$L\mathcal{Z}_1 = \langle Lc_{\mathcal{Z}_1}, Lg^{(1)}_{\mathcal{Z}_1}, \dots, Lg^{(p)}_{\mathcal{Z}_1} \rangle.$$

A more general class of zonotopes are polynomial zonotopes [76], which can describe non-convex sets. In this thesis, we use a sparse representation of polynomial zonotopes, which are defined as follows [75]:

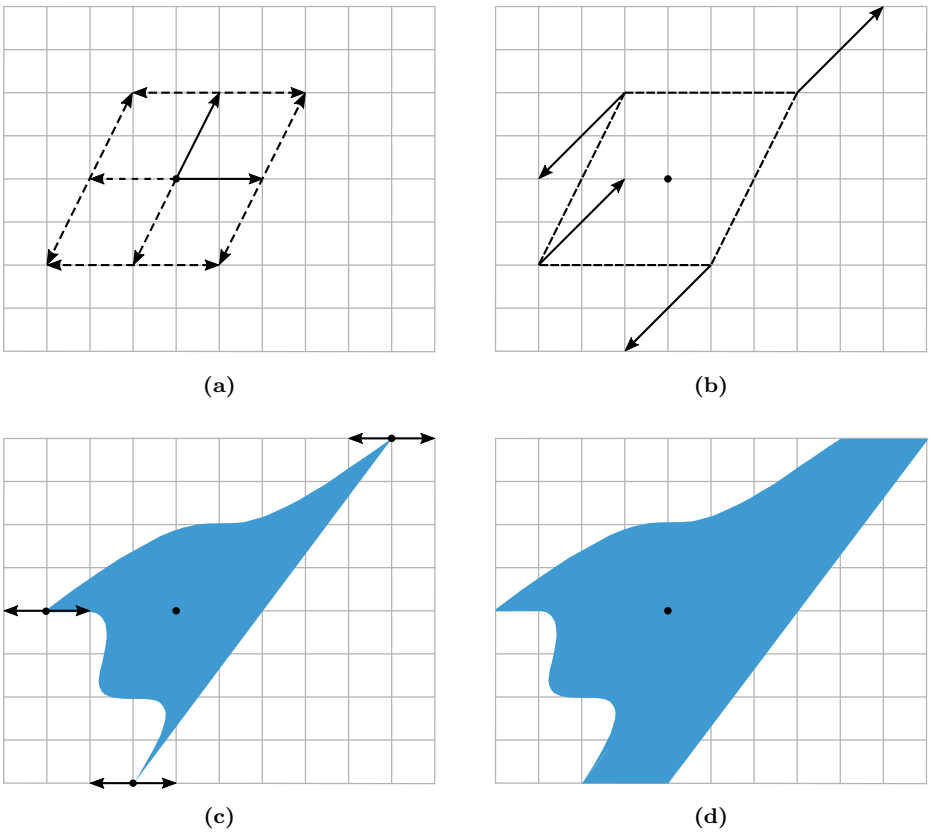


Figure 2.3: Illustration based on that of [75, Ex. 1] showing the construction of the polynomial zonotope (2.2): (a) Set which is spanned by the first two dependent generators $[2,0]^T$ and $[1,2]^T$ around the starting point $[4,4]^T$. (b) Addition of the third dependent generator $[2,2]^T$. (c) Due to its multiplication by the higher-order, mixed term $\delta_1^3\delta_2$, this leads to a non-convex set. This set is enlarged by the independent generator $[1,0]^T$, which leads to the final set shown in (d).

Definition 11 (Polynomial Zonotope)**.** *A set is called a polynomial zonotope if it can be written as*

$$\mathcal{PZ} = \left\{ x \in \mathbb{R}^n \;\middle|\; x = c + \sum_{i=1}^{v} \left(\prod_{k=1}^{y} \delta_k^{E_{k,i}} \right) g^{(i)} + \sum_{j=1}^{v_I} \mu_j g_I^{(j)},\ \delta_k, \mu_j \in [-1,1] \right\}. \tag{2.1}$$

Here, $c \in \mathbb{R}^n$ is the starting point, $g^{(i)} \in \mathbb{R}^n, i \in \{1,\dots,v\}$, are the dependent generators, $g_I^{(j)} \in \mathbb{R}^n, j \in \{1,\dots,v_I\}$, are the independent generators, and $E \in \mathbb{N}_0^{y \times v}$ is the matrix that stores the polynomial exponents.

Polynomial zonotopes are closed under quadratic and higher-order maps, which makes them useful for the reachable set computation of nonlinear systems. Zonotopes are a special class of polynomial zonotopes with only independent generators. Zonotopes offer a simpler set rep-

resentation compared to polynomial zonotopes, and especially for linear systems, polynomial zonotopes do not have any advantages over zonotopes. For a simpler representation (and faster algorithms), we mainly describe the algorithms in this thesis for normal zonotopes. Most algorithms can be generalized to polynomial zonotopes as well. Even without any changes, polynomial zonotopes can be directly used for a more accurate reachable set computation in the presented algorithms. We illustrate the construction of the polynomial zonotope

$$\mathcal{PZ} = \left\{ \begin{bmatrix} 4 \\ 4 \end{bmatrix} + \begin{bmatrix} 2 \\ 0 \end{bmatrix} \delta_1 + \begin{bmatrix} 1 \\ 2 \end{bmatrix} \delta_2 + \begin{bmatrix} 2 \\ 2 \end{bmatrix} \delta_1^3 \delta_2 + \begin{bmatrix} 1 \\ 0 \end{bmatrix} \mu_1 \;\middle|\; \delta_1, \delta_2, \mu_1 \in [-1, 1] \right\} \tag{2.2}$$

in Fig. 2.3.

2.5 Nonlinear Systems and their Analysis

For most of this thesis, we consider disturbed, nonlinear, continuous-time systems of the form

$$\dot{x}(t) = f(x(t), u(t), w(t)), \tag{2.3}$$

with states $x(t) \in \mathbb{R}^n$, inputs $u(t) \in \mathbb{R}^m$, disturbances $w(t) \in \mathcal{W} \subset \mathbb{R}^\omega$, and time $t \in \mathbb{R}_0^+$. We do not require any stochastic properties for $w(\cdot)$; we only assume that any possible disturbance trajectory is bounded at any point in time in the compact set $\mathcal{W}$. We denote this by $w(\cdot) \in \mathcal{W}$, which is shorthand for $w(t) \in \mathcal{W},\ \forall t \in [0, t_f]$, where $t_f \in \mathbb{R}_0^+$ is the final time. The same shorthand is also used for state and input trajectories throughout the thesis.

Definition 12 (Solution)**.** *The solution of* (2.3)*, with initial state* $x^{(0)}$*, input* $u(\cdot)$*, and disturbance* $w(\cdot)$ *at time* t*, is denoted by* $\xi(x^{(0)}, u(\cdot), w(\cdot), t)$ *and satisfies the following two properties:*

$$\begin{aligned} &\xi(x^{(0)}, u(\cdot), w(\cdot), 0) = x^{(0)}, \\ &\dot{\xi}(x^{(0)}, u(\cdot), w(\cdot), t) = f\Big(\xi\big(x^{(0)}, u(\cdot), w(\cdot), t\big), u(t), w(t)\Big),\ \forall t \in \mathbb{R}_0^+. \end{aligned}$$

If we consider an undisturbed system, we use $\xi(x^{(0)}, u(\cdot), 0, t)$ to denote the solution without disturbances, i.e., $\mathcal{W} = \{0\}$.

In most parts of this thesis, we do not explicitly consider measurement errors to avoid further complicating the notation and description. However, measurement errors can be easily included in all approaches presented in this thesis. We model the measurement of system (2.3) by a function h, returning the measured state $\hat{x}(t)$ based on the actual state $x(t)$ and some measurement error $\nu(t)$ from a compact set of possible measurement errors $\mathcal{V} \subset \mathbb{R}^o$:

$$\hat{x}(t) = h(x(t), \nu(t)),\ \nu(t) \in \mathcal{V}. \tag{2.4}$$

The set of possible actual states based on a measurement $\hat{x}(t)$ is given by

$$\hat{\mathcal{X}}(t) = \{x(t) \in \mathbb{R}^n \mid \exists \nu(t) \in \mathcal{V} : h(x(t), \nu(t)) = \hat{x}(t)\}. \tag{2.5}$$

If not all states are measurable, $\hat{\mathcal{X}}(t)$ can also be obtained by a set-based observer [77–79]. In this case, we also include possible observer errors in the uncertain measurement. We assume

that all relevant states can be obtained from such an observer. Therefore, to simplify the notation, we do not distinguish between measured outputs and measured states.

If we use the measured state $\hat{x}(t)$ to compute the control input $u_{ctrl}(\hat{x}(t), t)$, we can express the closed-loop dynamics as

$$\dot{x}(t) = f(x(t), u_{ctrl}(\hat{x}(t), t), w(t)).$$

We see that the measurement errors are a special case of disturbance which only affects the input to the feedback controller. This can be directly included in the reachable set computation, which is introduced next.

2.6 Reachability Analysis

Let us first formally define reachable sets and afterwards explain the basic idea of computing over-approximations of reachable sets.

Definition 13 (Reachable Set)**.** *For system* (2.3)*, the reachable set* $\mathcal{R}_{t,\mathcal{U},\mathcal{W}}(\mathcal{X}_0) \subset \mathbb{R}^n$ *for a time* t*, inputs* $u(\cdot) \in \mathcal{U} \subset \mathbb{R}^m$*, disturbances* $w(\cdot) \in \mathcal{W} \subset \mathbb{R}^\omega$*, and a set of initial states* $\mathcal{X}_0 \subset \mathbb{R}^n$ *at time* t_0 *is the set of end states of trajectories starting in* $\mathcal{X}_0$ *after time* t*, i.e.,*

$$\mathcal{R}_{t,\mathcal{U},\mathcal{W}}(\mathcal{X}_0) = \{x \in \mathbb{R}^n \mid \exists x^{(0)} \in \mathcal{X}_0, \exists u(\cdot) \in \mathcal{U}, \exists w(\cdot) \in \mathcal{W} : \\ \xi(x^{(0)}, u(\cdot), w(\cdot), t_0 + t) = x\}.$$

The reachable set over a time interval $[t_1, t_2]$ *is the union of all reachable sets for these time points, i.e.,*

$$\mathcal{R}_{[t_1,t_2],\mathcal{U},\mathcal{W}}(\mathcal{X}_0) = \bigcup_{t \in [t_1,t_2]} \mathcal{R}_{t,\mathcal{U},\mathcal{W}}(\mathcal{X}_0).$$

If we consider the reachable set for a system with feedback $u_{fb}(x(t), t)$, then we denote by $\mathcal{R}_{t,u_{fb},\mathcal{W}}(\mathcal{X}_0)$ the reachable set obtained for the closed-loop dynamics

$$\dot{x}(t) = f(x(t), u_{fb}(x(t), t), w(t)).$$

If we consider systems without disturbances, we use $\mathcal{R}_{t,\mathcal{U},0}(\mathcal{X}_0)$ to show that $\mathcal{W} = \{0\}$. Note that for the sake of simplicity, this reachable-set notation does not explicitly contain the starting time t_0. If the dynamics are time invariant, the actual starting time does not make a difference, as only the duration t matters. If the dynamics are time varying, it should be clear from the context at what time t_0 the reachable set computation starts.

For most systems, it is not possible to compute exact reachable sets [80]. To still be safe, we therefore compute over-approximations instead, which can be computed with any desired accuracy [81]. In the remainder of the thesis, most of the time, we do not distinguish between exact reachable sets and over-approximated reachable sets. Therefore, if we refer to a reachable set, we usually mean an over-approximated reachable set. All guarantees obtained from over-approximated reachable sets still hold for the real system; we might simply be too conservative at times.

We incorporate reachable sets in all control methods presented in this thesis. However, we do not require a specific algorithm for computing these sets. As there exist many different algorithms already and it is to be expected that more and better algorithms are developed in the future, we make our algorithms as general as possible, such that they benefit from future developments in reachability-analysis tools. We have two basic assumptions for the reachable set algorithms: First, we assume that they compute the reachable sets for single time points as well as for time intervals. As we are fine with over-approximations for both types of reachable sets, this assumption is not very restrictive. Our second requirement is that the reachability algorithms are able to work with zonotopes as set representation. This is due to the beneficial properties that zonotopes have, not only for reachability analysis, but also for set-based controller synthesis. It also results from the fact that, currently, several of the most efficient reachability algorithms for nonlinear systems utilize zonotopes [2]. If this ever changes, our algorithms can be adapted to work with other set representations as well.

Let us now give a brief overview of two example algorithms for computing over-approximative reachable sets. We present the basic idea of the techniques from [74], which are implemented in the MATLAB toolbox CORA [82] and which are used throughout this thesis in all numerical and practical examples. Let us start by considering a disturbed, linear, time-invariant system of the form

$$\dot{x}(t) = Ax(t) + w(t), \tag{2.6}$$

with $x(\cdot) \in \mathbb{R}^n$ denoting the state of the system and $w(\cdot) \in \mathcal{W} \subset \mathbb{R}^n$ denoting an uncontrollable disturbance, which is bounded by the compact set $\mathcal{W}$. For a simpler notation, we assume that the disturbances are centered around the origin (see [74] for the general case). Usually in reachability analysis, one is interested in the set of all possible trajectories resulting from all possible inputs. If we have control over the inputs, they usually depend on the state or time, and we can treat them differently than uncontrollable disturbances. Therefore, we only consider disturbances as inputs in (2.6). If there is a state-dependent feedback controller, we choose A such that it contains the closed-loop dynamic. In later chapters, we describe how to treat other kinds of inputs. The reachability algorithm divides the time into short intervals and uses the superposition principle to separate the reachable set into two parts: one with the autonomous dynamics and one resulting from the disturbance $\mathcal{W}$.

To compute the reachable set of a time interval $[0, t_f]$, starting from a given initial set $\mathcal{X}_0$, the interval is first divided into N time steps of length Δt. For an easier notation the shorthand notation $t_i := i\,\Delta t$ is introduced. The reachable set $\mathcal{R}_{[0,t_f],0,\mathcal{W}}(\mathcal{X}_0)$ is then computed via the following four steps (see Fig. 2.4 for the first three steps).

(i) The homogeneous part, i.e., the part without disturbance effects, of the reachable set after the first time step Δt is computed (see Fig. 2.4(a)). This is done by simply multiplying the initial set with the matrix exponential function based on the system dynamics A:

$$\mathcal{R}^h_{t_1} = e^{A\Delta t}\mathcal{X}_0.$$

(ii) The convex hull $\text{conv}(\mathcal{X}_0, \mathcal{R}^h_{t_1})$ of the initial set and the homogeneous part of the reachable

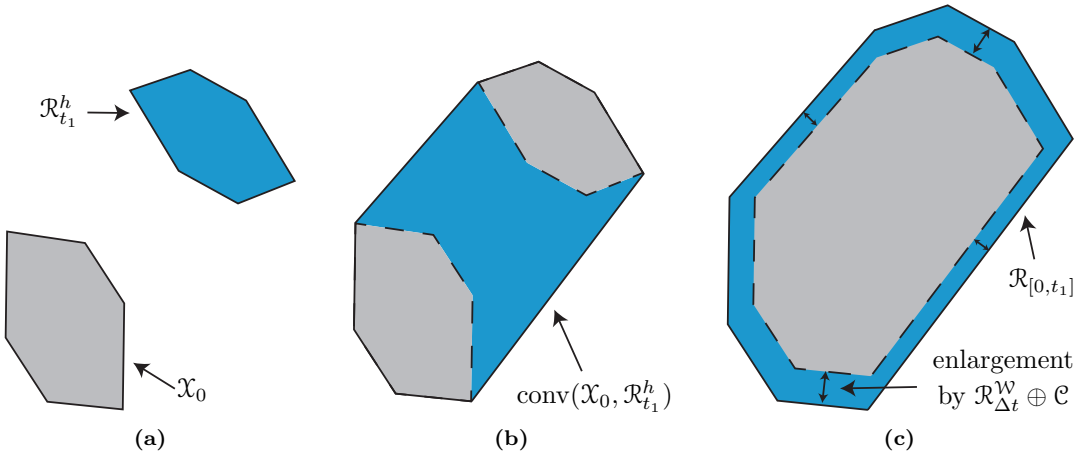


Figure 2.4: Computation of the reachable set for the first time interval $[0, t_1]$: (a) We first compute the homogeneous solution $\mathcal{R}^h_{t_1}$ at time t_1, neglecting the disturbances. (b) Then we compute the convex hull of $\mathcal{X}_0$ and $\mathcal{R}^h_{t_1}$ to approximate the reachable set between these two time points. (c) Finally, we enlarge the set to take disturbances and curvature of trajectories into account.

set after one time step is computed (Fig. 2.4(b)). This approximates the reachable set within the time interval $[0, t_1]$ under the assumption that all trajectories are straight lines.

(iii) The set $\mathrm{conv}(\mathcal{X}_0, \mathcal{R}^h_{t_1})$ is enlarged by the sets $\mathcal{R}^{\mathcal{W}}_{\Delta t} = \mathcal{R}_{\Delta t,0,\mathcal{W}}(\emptyset)$ to account for the disturbances and by $\mathcal{C}$ for the curvature of the trajectories, since they are not straight lines, thereby making the result over-approximative (see Fig. 2.4(c)):

$$\mathcal{R}_{[0,t_1]} = \mathrm{conv}(\mathcal{X}_0, \mathcal{R}^h_{t_1}) \oplus \mathcal{R}^{\mathcal{W}}_{\Delta t} \oplus \mathcal{C}.$$

(iv) Steps (i)–(iii) only have to performed a single time. For all following time intervals $[t_k, t_{k+1}]$, it is sufficient to propagate the reachable set from the last step forward by multiplying it by the exponential matrix and adding the disturbance set

$$\mathcal{R}_{[t_k,t_{k+1}]} = e^{A\Delta t}\mathcal{R}_{[t_{k-1},t_k]} \oplus \mathcal{R}^{\mathcal{W}}_{\Delta t}.$$

Details on how to compute the sets $\mathcal{R}^{\mathcal{W}}_{\Delta t}$ and $\mathcal{C}$ can be found in [74]. Linear, time-invariant systems have the advantage that the superposition principle holds. It can be used to independently compute the homogeneous dynamics and the effects from the disturbances. In addition, it allows us to forward-propagate the solution from the previous steps in Step (iv), thereby drastically simplifying the computations.

This algorithm can be extended to systems with uncertain or time-varying parameters and nonlinear as well as hybrid dynamics, see [74]. Let us now briefly describe the main idea for the nonlinear case, as this is used for many of our applications. It works in the following five steps:

(i) For nonlinear systems, the superposition principle does not hold anymore. To still be able to efficiently compute the reachable sets, at each time step, the system dynamics are locally linearized to obtain $f(x,u,w) \in f_{lin}(x,u,w) \oplus \mathcal{L}$, with $f_{lin}(x,u,w)$ describing the linearized dynamics without any linearization errors and $\mathcal{L}$ denoting the set of possible linearization errors. Note that this abstraction by a linear system has to be newly computed for each time interval.

(ii) Considering only the linearized dynamics $f_{lin}(x,u,w)$, Steps (i)–(iii) from before are performed to obtain $\mathcal{R}^{lin}_{[t_k,t_{k+1}]}$ without yet considering any linearization errors.

(iii) The linearized system can be seen as the first part of an infinite Taylor series. Therefore, the linearization error can be computed by bounding the remaining terms of this series, which can be done in an over-approximative fashion as described in [74]. Its effects result in the reachable set $\mathcal{R}^{err}$.

(iv) Due to the linearized dynamics, the reachable set can be obtained as the superposition of the linearized dynamics without linearization error and the reachable set due to linearization errors as

$$\mathcal{R}_{[t_k,t_{k+1}]} = \mathcal{R}^{lin}_{[t_k,t_{k+1}]} \oplus \mathcal{R}^{err}.$$

(v) Steps (i)–(iv) are repeated for each time interval $[t_k, t_{k+1}]$.

The linearization error $\mathcal{L}$ becomes larger the further away a state is from the linearization point; therefore, it depends on the size of the reachable set. At the same time, the size of the actual reachable set depends on the size of the linearization error. As a result, the linearization error cannot simply be computed in a single step. Instead, in [74], it is computed in an iterative fashion which leads to longer computation times compared to the linear systems. If the linearization error ever becomes too large, one has to either start with a smaller initial set size or split the set during the reachable set computation to reduce the linearization errors. Bounding the linearization error is also a way to limit the conservativeness of this approach.

There exists a number of improvements, extensions, and new algorithms to make reachability analysis less conservative. For example, [76] reduces the effects of abstraction errors by using polynomialization instead of linearization. However, in general, larger initial sets and system dynamics with higher nonlinearities lead to larger abstraction errors in the reachable set computation. To consider disturbance effects and abstraction errors new generators are added to the reachable set in each iteration. Therefore, the order of the zonotope increases over time. In the following chapters, we take this into account in our notation of reachable sets by indicating that the number of generators p_k is dependent on the iteration k. To limit the computational effort, one can use order-reduction methods [74, 83], [226] to over-approximate the reachable set by a zonotope with lower order when the number of generators becomes too high.

2.7 Maneuver Automata

As described in Sec. 1.2, maneuver automata allow fast planning by combining discrete motion primitives to complex maneuvers.

Definition 14 (Maneuver Automaton)**.** *A maneuver automaton* $\mathbf{MA} = \{\mathcal{M}, \mathcal{D}\}$ *is a tuple of a set of motion primitives* $\mathcal{M}$ *and a set of discrete transitions* $\mathcal{D} \subset \mathcal{M} \times \mathcal{M}$ *defining which motion primitives can be followed by each other.*

The motion primitives are precomputed and therefore allow us to perform computationally expensive tasks like reachability analysis offline in advance. In this thesis, we only consider safe motion primitives, which contain a safe controller and its corresponding reachable set. They are defined as follows:

Definition 15 (Motion Primitive)**.** *A motion primitive*

$$\mathbf{MP} = \{x_{ref}(\,\cdot\,), u_{ref}(\,\cdot\,), t_f, \mathcal{X}_0, \mathcal{X}_f, u_{ctrl}(\,\cdot\,), \mathcal{R}_{[0,t_f],u_{ctrl},\mathcal{W}}(\mathcal{X}_0)\}$$

is again a tuple, containing a reference trajectory $x_{ref}(\,\cdot\,)$*, the corresponding input trajectory* $u_{ref}(\,\cdot\,)$*, a duration* t_f*, initial and final sets* $\mathcal{X}_0$ *and* $\mathcal{X}_f$*, respectively, a safe controller* $u_{ctrl}(\,\cdot\,)$*, and the reachable set of the safe controller* $\mathcal{R}_{[0,t_f],u_{ctrl},\mathcal{W}}(\mathcal{X}_0)$*.*

There exists a transition from one motion primitive $\mathbf{MP}_i$ with final set $\mathcal{X}_f^{(i)}$ to another motion primitive $\mathbf{MP}_j$ with initial set $\mathcal{X}_0^{(j)}$, i.e., $(\mathbf{MP}_i, \mathbf{MP}_j) \in \mathcal{D}$ if and only if $\mathcal{X}_f^{(i)} \subseteq T(\mathcal{X}_0^{(j)})$, where $T(\mathcal{X}_0^{(j)})$ is a valid transformation depending on the system dynamics. For example, if the system dynamics are independent of the initial state, it is admissible to translate the initial state of the second motion primitive into the final set of the first motion primitive. In this case, it is only important that the final set is completely contained in the shifted initial set. We give an example for such a transformation in the numerical example in Sec. 3.3.7.

2.8 Conformance Checking

All results from formal verification are based on underlying assumptions. The guarantees only hold as long as the assumptions are also satisfied. In our case, one of the most important assumptions is the model of our system. Choosing the correct model is a fundamental problem, as it is never possible to include all details of the real world in an exact, mathematical model.

Therefore, we do not assume that a single model captures the exact real-world behavior, but rather that it models the main parts and that the remaining effects are over-approximated by uncertainties in the form of disturbances and measurement errors. Still, the question remains of how we can be sure that the model with uncertainties represents the real world well enough. A solution to this problem is provided by conformance checking [84], also known as conformance testing. In conformance checking, one uses the measurements from a real system or simulations from a high-fidelity model to check if the real behavior can be explained with the uncertain model used in synthesis. If this is the case for all tests, the model is said to conform to the real

system. If one or more measurements do not match the model, we have to change the model and/or increase the uncertainty sets. This is done until conformance is shown for all tests.

By systematically testing the system and afterwards increasing the uncertainties by a safety margin, this approach allows us to increase the confidence in the model used. A major advantage over the classical testing of the closed-loop system with the final controller is that we can test the system independently of a specific controller, either open-loop or with an auxiliary controller, before we start with the actual controller synthesis. We are then able to provide guarantees for any closed-loop system.

We do not explicitly consider conformance checking in our controller synthesis algorithms. We simply assume that we are given an uncertain model which captures the real world. This model can be obtained by conformance checking or by any other reliable technique. Our algorithms are therefore independent of the exact modeling process. In Ch. 7, when we apply our algorithms to real systems, we use conformance checking to obtain the model which we use for the controller synthesis. There are two major approaches for conformance checking, and we briefly introduce both of them.

2.8.1 Trace Conformance

The idea of trace conformance [84, 85] is that we try to reproduce measured trajectories with the uncertain model. This is done by finding disturbance and measurement error trajectories which explain the obtained measurements. We thereby ensure that the obtained model is able to reproduce the observed real-world behavior.

To test trace conformance, one records a number of input traces $U_i = [u(t_1), ..., u(t_K)] \in \mathbb{R}^{m\times K}$, $i \in \{1, \dots, I\}$, which are applied to the system and the corresponding measurement traces $\hat{X}_i = [\hat{x}(t_1), ..., \hat{x}(t_K)] \in \mathbb{R}^{n\times K}$ which are measured from the system, both at discrete points in time $T_i = [t_1, ..., t_K] \in \mathbb{R}^K$. These test traces are used to check conformance as defined as follows [215]:

Definition 16 (Trace Conformance)**.** *A model, consisting of the differential equation* (2.3)*, the measurement function* (2.4)*, as well as disturbance and measurement noise sets* $\mathcal{W}$ *and* $\mathcal{V}$ *is said to be trace conformant, if for every test case* $(U_i, \hat{X}_i, T_i)$ *a model trace* (X_i, V_i, W_i) *with* $X_i = [x(t_1), ..., x(t_K)] \in \mathbb{R}^{n\times K}$*,* $V_i = [\nu(t_1), ..., \nu(t_K)] \in \mathbb{R}^{o\times K}$*,* $W_i = [w(t_1), ..., w(t_K)] \in \mathbb{R}^{\omega\times K}$*, exists, for which the following holds:*

$$\forall t_k \in \{t_1, \dots, t_K\}: \quad x(t_{k+1}) = x(t_k) + \int_{t_k}^{t_{k+1}} f\left(x(\tau), u(t_k), w(t_k)\right) d\tau \tag{2.7}$$

$$\wedge\ \hat{x}(t_k) = h(x(t_k), \nu(t_k)) \tag{2.8}$$

$$\wedge\ \nu(t_k) \in \mathcal{V} \wedge w(t_k) \in \mathcal{W}. \tag{2.9}$$

Checking if a valid model trace exists for a given test case is done by solving a constraint satisfaction problem for equations (2.7)–(2.9). This problem is similar to a constrained optimal control problem, where the disturbances and measurement noise are used as the system inputs. Therefore, efficient numerical solvers from optimal control can be used to find feasible model traces.

2.8.2 Reachset Conformance

In trace conformance, we try to reproduce each measured behavior with an disturbance and measurement error trajectory. This is a strong property which is sometimes hard to ensure. An alternative notion of conformance is reachset conformance [86]. Instead of checking if we can find a single trajectory which matches the measurement, in reachset conformance, the measurement only has to lie inside the reachable set of the uncertain model. While this is a weaker notation, it is also easier to verify and sufficient to ensure safety [86]. As this is the main focus of this work, reachset conformance is a viable alternative for obtaining reliable models.

To test reachset conformance, we simply compute the reachable set of our model and check whether the measurements are contained there:

$$\hat{x}(t_k) \in h\left(\mathcal{R}_{t_k,u(\cdot),\mathcal{W}}(\mathcal{X}_0), \mathcal{V}\right),\ \forall t_k \in T_i,\ \forall x(t_k) \in X_i,\ \forall i \in \{1,\dots,I\},$$

where we use the shorthand notation

$$h\left(\mathcal{R}_{t_k,u(\cdot),\mathcal{W}}(\mathcal{X}_0), \mathcal{V}\right) := \left\{h(x(t_k), \nu(t_k)) \mid x(t_k) \in \mathcal{R}_{t_k,u(\cdot),\mathcal{W}}(\mathcal{X}_0),\ \nu(t_k) \in \mathcal{V}\right\}.$$

Chapter 3

Offline Controller Synthesis

3.1 Introduction

In this chapter, we present several algorithms for solving reach-avoid problems by combining optimal control techniques with reachability analysis. Because they involve computations of reachable sets, these algorithms might not be performed in real time; however, they are especially applicable for computing motion primitives for safe maneuver automata. Thus, we want to find controllers which steer all states from an initial set into the smallest possible final set after a fixed time despite disturbances. This lets us concatenate each motion primitive with as many others as possible, making the maneuver automaton more connected and thereby allowing more flexibility during online planning. We begin with a formal problem statement before presenting four different control approaches which can be used to solve this problem.

3.2 Problem Statement

The task is to find a control law $u_{ctrl}(x,t)$ for system (2.3) which guarantees that all states in an initial set $\mathcal{X}_0 \subset \mathbb{R}^n$ are steered into a final set $\mathcal{R}_{t_f,u_{ctrl},\mathcal{W}}(\mathcal{X}_0) \subset \mathbb{R}^n$ around a desired final state $x^{(f)}$ after time t_f despite the disturbance set $\mathcal{W}$. Depending on the application, it might also be beneficial to minimize the applied inputs. We minimize the size of the final set while considering input costs by solving

$$\min_{u_{ctrl}} \|(\mathcal{R}_{t_f,u_{ctrl},\mathcal{W}}(\mathcal{X}_0) - x^{(f)})\|_1 + \gamma \int_0^{t_f} \|u_{ctrl}(\mathcal{R}_{t,u_{ctrl},\mathcal{W}}(\mathcal{X}_0),t)\|_1 \mathrm{d}t, \tag{3.1}$$

where $\gamma \in \mathbb{R}_0^+$ is a factor used to weight the input costs in relation to the state costs. We use the short form $\|\mathcal{R} - x^{(f)}\|_1 := \max_{x\in\mathcal{R}} \|x - x^{(f)}\|_1$ to denote the maximum 1-norm between the desired final state $x^{(f)}$ and any state in the set $\mathcal{R} \subset \mathbb{R}^n$. Similarly, we use $u_{ctrl}(\mathcal{R},t) := \{u \in \mathbb{R}^m \mid \exists x \in \mathcal{R} : u_{ctrl}(x,t) = u\}$ to denote the set of possible inputs of our controller for set $\mathcal{R}$ and $\|u_{ctrl}(\mathcal{R},t)\|_1 := \max_{x\in\mathcal{R}} \|u_{ctrl}(x,t)\|_1$ the input with maximum 1-norm. If desired, we can also weight each state and input individually by multiplying the states and inputs by weight matrices. For a simpler presentation, we omit these weight matrices and only use γ.

In addition to its numerical advantages, the reason for using the 1-norm for our cost function is that many target sets are axis-aligned boxes. Therefore, we prefer a set whose bounding axis-aligned box is as small as possible. When using the 1-norm, we minimize the sum of the side lengths of this bounding box. If we used, for example, the 2-norm instead, we would minimize the sum of the squared sides, therefore not appropriately considering smaller dimensions. The extreme case would be using the infinity norm, as illustrated with the following example: Let us consider a two-dimensional set whose maximum expanse in the first dimension is denoted by s_1 and in the second dimension by s_2, with $s_1 \geq s_2$. In the case that there exist constraints which prevent us from reducing the size of s_1, there is no change in the infinity norm if we reduce the size of s_2, while any cost function using the 1-norm would actually benefit from reducing s_2.

We assume that the initial set $\mathcal{X}_0 = \langle c_{x,0}, g_{x,0}^{(1)}, \dots, g_{x,0}^{(p_0)} \rangle$ and the disturbance set $\mathcal{W} = \langle c_1, g_w^{(1)}, \dots, g_w^{(r)} \rangle$ are zonotopes or can be over-approximated by zonotopes. Note that no extra conservatism is added in the most common case when they are boxes. Furthermore, we consider constraints on the states and inputs of the form

$$\xi(x^{(0)}, u(\,\cdot\,), w(\,\cdot\,), t) \in \mathcal{S}, \quad \forall t \in [0, t_f], \tag{3.2}$$

$$u(t) \in \mathcal{U}, \quad \forall t \in [0, t_f], \tag{3.3}$$

where $\mathcal{S}$ and $\mathcal{U}$ are both polyhedral sets of the form

$$\mathcal{S} = \{x \in \mathbb{R}^n \mid C_\mathcal{S} x \leq d_\mathcal{S}\},$$
$$\mathcal{U} = \{u \in \mathbb{R}^m \mid C_\mathcal{U} x \leq d_\mathcal{U}\},$$

with $C_\mathcal{S} \in \mathbb{R}^{\varrho \times n}, d_\mathcal{S} \in \mathbb{R}^\varrho, C_\mathcal{U} \in \mathbb{R}^{\varsigma \times m}$, and $d_\mathcal{U} \in \mathbb{R}^\varsigma$. Note that for most practical applications, the inputs are bounded, which leads to polytopic input constraints.

During offline computation, the locations of most non-convex constraints, such as other traffic participants in automated driving, are not known. Therefore, we use this approach to compute motion primitives offline in advance, where we take convex input constraints, e.g., maximum acceleration or steering, and convex state constraints, e.g., maximum velocity, into account. The non-convex dynamic constraints are handled during online planning using a maneuver automaton containing these motion primitives (see Fig. 3.1[1] and Sec. 1.2).

For most of this chapter, we want to find a final set $\mathcal{X}_f$ which is as small as possible. If instead the task is to steer all states into a given final set, then we would have to adapt our algorithms by adding this as an additional constraint. In this case, however, it might be possible that no solution exists, depending on the choice of constraints, final time, and final set.

[1] Fig. 3.1 has been previously published in [219] © 2021 IEEE.

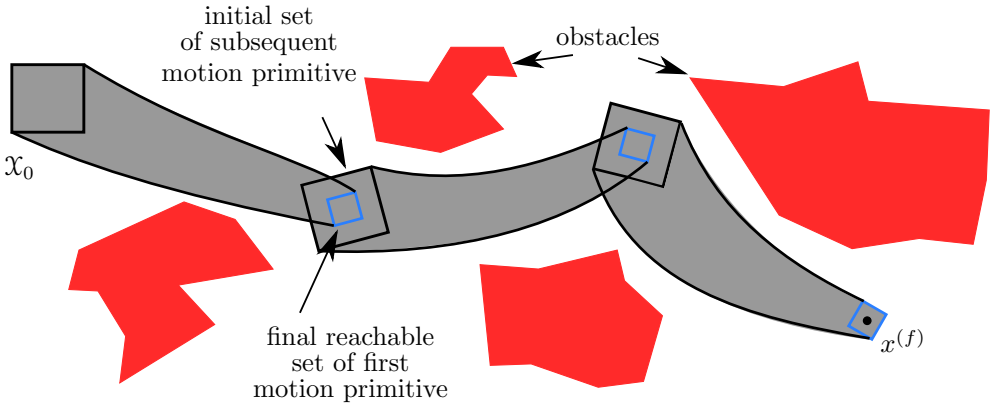


Figure 3.1: Motion primitives (gray) are used to steer all states from $\mathcal{X}_0$ to $x^{(f)}$ while avoiding non-convex obstacles (red). The final set (blue box) of one motion primitive is always contained inside the initial set (black box) of the following motion primitive.

3.3 Convex Control

In this section[1], we present the first algorithm for solving the previously-stated problem. For single states and no disturbances, such a constrained optimal control problem can be solved very fast and efficiently. As discussed in the introduction of this thesis, the difficulty arises when we want to obtain a solution for every state in the initial set. As we cannot solve an optimization problem for each state, we have to find a way to generalize the finitely many solutions which we are able to obtain to all other states of the initial set. The idea of the convex interpolation controller is to solve the optimization problem only for the extreme states of the initial set and to use convex combinations to interpolate the solutions of the extreme states to obtain control inputs for all interior states. This concept works directly for linear systems, and we show that we can extend it to disturbed and nonlinear systems by iteratively applying the controller in combination with reachability analysis.

For linear systems without disturbances, it is well known that for a convex initial set of states and a convex set of constant inputs, the reachable set after a fixed time is again a convex set. This follows directly from the superposition principle of linear systems [87]. Moreover, the extreme points of the reachable set are the end points of trajectories starting from the extreme points of the initial set, see Fig. 3.2 (a). This means that by computing only the trajectories for the extreme points of the initial set, we can obtain the whole reachable set by calculating the convex hull of the end points of these trajectories. This correlation is also used for reachability analysis of linear systems, where the reachable set can be obtained by computing convex combinations of the extreme states of a set of inputs, see, e.g., [88]. It is also used in parametrized tube-based MPC [23], where control inputs are also computed as convex combinations of extreme input values.

[1] This section and most of its figures are based on [216].

In our case, we are not interested in the reachable set for all possible inputs or the scenario in which we use the same input for all extreme states. Rather, we want to steer each extreme state as close as possible to the desired final state $x^{(f)}$. If we consider the initial set $\mathcal{X}_0$ to be given by a polytope defined by its q extreme states $\breve{x}^{(i)},\ i \in \{1, \dots, q\}$, we are looking for each of the extreme states $\breve{x}^{(i)}$ for a piecewise-continuous control input $\breve{u}^{(i)}(\,\cdot\,),\ i \in \{1, \dots, q\}$, which steers this state close the desired final state $x^{(f)}$. It follows that the convex hull of these states is small and close to the desired final state. In order for all trajectories starting inside the initial set to also end in the convex hull of these extreme trajectories, we cannot use the same input trajectory for all states, but have to choose them carefully. For our convex controller, we obtain this control input by using a convex combination of the input trajectories of the extreme states. For an arbitrary state $x^{(0)} \in \mathcal{X}_0$, we first express it as a convex combination of the extreme states $\breve{x}^{(i)}$ by choosing $\lambda_i(x^{(0)})$ such that

$$\begin{aligned} x^{(0)} &= \sum_{i=1}^{q} \lambda_i(x^{(0)}) \breve{x}^{(i)}, \\ \lambda_i(x^{(0)}) &\geq 0,\ \forall i \in \{1, \dots, q\}, \\ \sum_{i=1}^{q} \lambda_i(x^{(0)}) &= 1. \end{aligned} \tag{3.4}$$

We then use the same parameters $\lambda_i(x^{(0)})$ to compute the corresponding control input $u_{conv}(x^{(0)}, \cdot\,)$ for the state $x^{(0)}$ as a convex combination of the control inputs $\breve{u}^{(i)}(\,\cdot\,)$ of the extreme states $\breve{x}^{(i)}$, i.e.,

$$u_{conv}(x^{(0)}, \cdot\,) = \sum_{i=1}^{q} \lambda_i(x^{(0)}) \breve{u}^{(i)}(\,\cdot\,). \tag{3.5}$$

This is illustrated in Fig. 3.2. Note that the controller $u_{conv}(x^{(0)}, \cdot\,)$ provides an open-loop control input. Feedback is achieved by iteratively applying (3.5), as discussed later. As we see in Sec. 3.3.3, it is even possible to obtain closed-form expressions of the convex combinations, so that the system of inequalities in (3.4) does not have to be solved online.

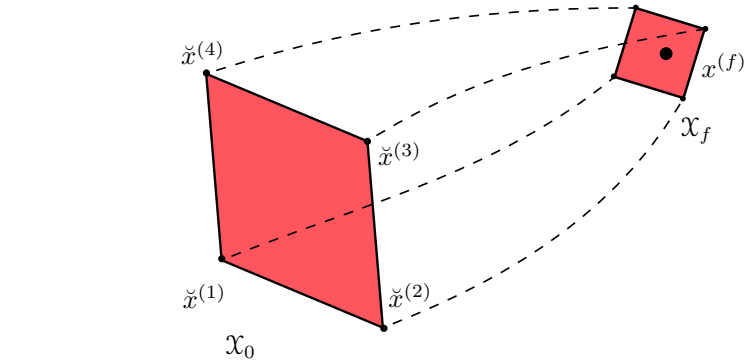
3.3.1 Linear Systems

Before we consider disturbed, nonlinear systems, we illustrate first how our approach works for undisturbed, linear systems of the form

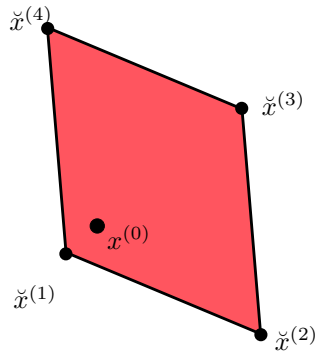
$$\dot{x}(t) = Ax(t) + Bu(t), \tag{3.6}$$

with $x(t) \in \mathbb{R}^n, u(t) \in \mathbb{R}^m, A \in \mathbb{R}^{n \times n}$, and $B \in \mathbb{R}^{n \times m}$. In order to solve the control problem, we perform the following two steps:

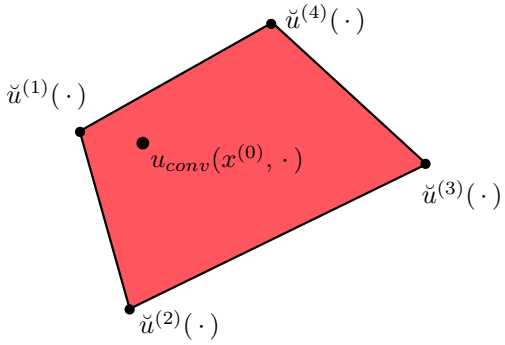
Step 1: For each of the extreme states $\breve{x}^{(i)},\ i \in \{1, \dots, q\}$, of the initial set, we solve a



(a) Extreme Trajectories



(b) State Space



(c) Input Space

Figure 3.2: Basic idea of the convex control approach: (a) Compute input trajectories which control the extreme states $\breve{x}^{(i)}$ of the initial set $\mathcal{X}_0$ close to the desired final state $x^{(f)}$. (b) Express state $x^{(0)} \in \mathcal{X}_0$ as a convex combination of extreme states $\breve{x}^{(i)}$. (c) Use the same convex combination to compute the corresponding control input $u_{conv}(x^{(0)},\,\cdot\,)$ using the control inputs $\breve{u}^{(i)}(\,\cdot\,)$ of the extreme states.

constrained optimization problem of the form

$$\begin{aligned} \breve{u}^{(i)}(\,\cdot\,) = \underset{\breve{u}^{(i)}(\,\cdot\,)}{\arg\min} \, & \|(\xi(\breve{x}^{(i)}, \breve{u}^{(i)}(\,\cdot\,), 0, t_f) - x^{(f)})\|_1 + \gamma \int_0^{t_f} \|\breve{u}^{(i)}(t)\|_1 \mathrm{d}t \\ \text{s.t. } & \forall t \in [0,t] : \xi(\breve{x}^{(i)}, \breve{u}^{(i)}(\,\cdot\,), 0, t) \in \mathcal{S}, \\ & \breve{u}^{(i)}(t) \in \mathcal{U}, \end{aligned} \tag{3.7}$$

to find an input sequence which returns a solution that ends as close as possible to $x^{(f)}$ while satisfying the state and input constraints (3.2)–(3.3). Note that for this control approach, the exact form of the cost function or used norm in (3.7) does not make a difference for the theory and can therefore be freely chosen depending on the specific problem. For other control algorithms which we present in later sections, the use of the 1-norm in the cost function is beneficial.

Step 2: For a given state in the initial set, we express it as a convex combination of the

extreme states by solving (3.4) and use the same convex combination of the corresponding input sequences (3.5) to control it to the final set.

By applying these two steps, we obtain for each extreme state $\breve{x}^{(i)}$ an input sequence $\breve{u}^{(i)}(\,\cdot\,)$, such that the corresponding state trajectory ends close to the desired final state $x^{(f)}$ after a fixed time t_f. If all input sequences $\breve{u}^{(i)}(\,\cdot\,)$ and corresponding state trajectories $\xi(\breve{x}^{(i)}, \breve{u}^{(i)}(\,\cdot\,), 0, \,\cdot\,)$ satisfy the input and state constraints, respectively, then all trajectories starting in the initial set $\mathcal{X}_0$ under the convex control law (3.5) end in the a priori known compact set

$$\mathcal{X}_f = \mathrm{conv}(\xi(\breve{x}^{(1)}, \breve{u}^{(1)}(\,\cdot\,), 0, t_f), \ldots, \xi(\breve{x}^{(q)}, \breve{u}^{(q)}(\,\cdot\,), 0, t_f)).$$

Moreover, all trajectories satisfy the state constraints (3.2) and input constraints (3.3) for all times.

This directly results from the way we compute the control law and from the convexity of linear systems. Using (3.4) and (3.5), it follows from the superposition principle that $\forall t \in [0, t_f]$:

$$\begin{aligned}
&\xi(x^{(0)}, u_{conv}(x^{(0)}, \,\cdot\,), 0, t) \\
&= \xi\left(\sum_{i=1}^{q} \lambda_i(x^{(0)})\breve{x}^{(i)}, \sum_{i=1}^{q} \lambda_i(x^{(0)})\breve{u}^{(i)}(\,\cdot\,), 0, t\right) \\
&= \sum_{i=1}^{q} \lambda_i(x^{(0)})\xi(\breve{x}^{(i)}, \breve{u}^{(i)}(\,\cdot\,), 0, t),
\end{aligned}$$

i.e., any trajectory starting in the initial set lies inside the convex set of the extreme trajectories. Since the extreme trajectories satisfy the polyhedral state constraints, any inner trajectory satisfies the state constraints as well. The control inputs are convex combinations of the extreme inputs, which are contained in the polyhedral input set $\mathcal{U}$. Therefore, it follows from convexity that the control inputs are in the set $\mathcal{U}$ as well.

The important part of this new control approach is the second step. While the first step is performed offline and stores the results, the second step applies the online control algorithm without having to solve any optimal control problems online. This is in contrast to model predictive control, for example.

Although the presented approach has favorable properties, it is an open-loop control. Because of the missing feedback, it is not robust against any disturbances or model-mismatches. Intermediate steps which take disturbances into account are necessary for feedback control. In the case of nonlinear systems, these steps are always necessary, even without disturbances. Therefore, we consider the robust case directly for nonlinear systems in the next subsection.

3.3.2 Nonlinear Systems with Disturbances

Since nonlinear dynamics in general do not preserve convexity [89], convex control cannot be applied in the same way as in the linear case. Instead, we divide the control problem into intermediate steps and iteratively apply the convex control law to steer the system along a reference trajectory. For each time step, we compute the reachable set (see Sec. 2.6), thereby

ensuring that the constraints are always satisfied despite disturbances and nonlinear dynamics. By recomputing the convex control inputs in each time step, we realize feedback and counteract the effects from disturbances. Through the use of reachability analysis, we have a separation of concerns: the optimization provides performance, while the reachability analysis provides guarantees for the satisfaction of all constraints. Therefore, the guarantees still hold if we cannot find optimal solutions for the optimal control problems as long as we find solutions which satisfy the constraints. The new control approach is presented in Alg. 1 and is illustrated in Fig. 3.3. It consists of three major steps:

Algorithm 1 Offline Part of the Convex Control Algorithm for Nonlinear Systems with Disturbances

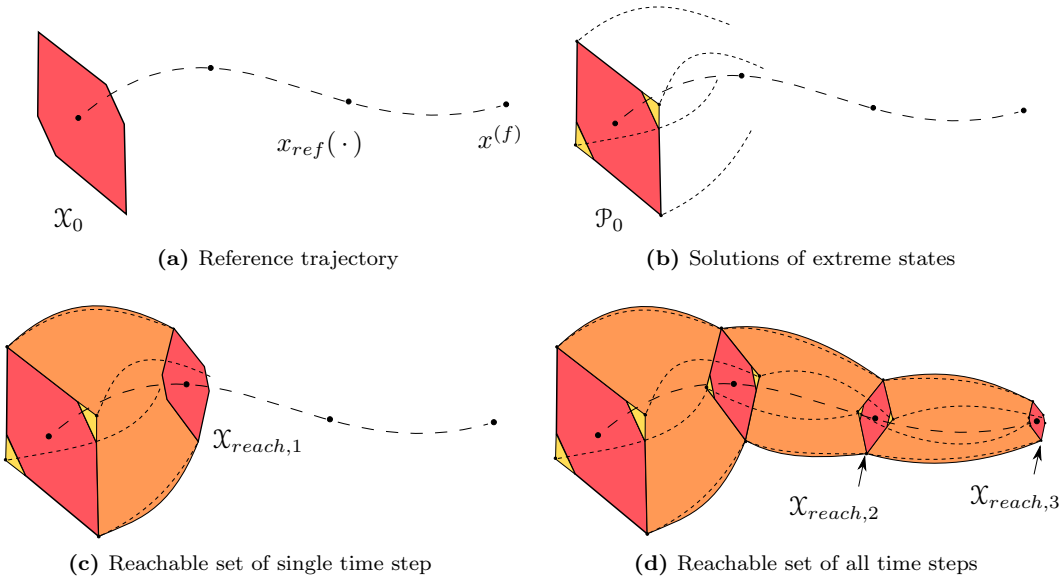
Input: system dynamics $f(x, u, w)$, initial set $\mathcal{X}_0$, desired final state $x^{(f)}$, total time t_f, number of iterations N, constraint sets $\mathcal{S}, \mathcal{U}, \mathcal{S}_{ref}, \mathcal{U}_{ref}, \bar{\mathcal{S}}$, disturbance set $\mathcal{W}$, weighting factors γ, γ_{ref}
Output: motion primitive **MP**

1: $(x_{ref}(\cdot), u_{ref}(\cdot)) \leftarrow$ solution of optimization problem (3.8) for the reference trajectory
2: Initialize: $\mathcal{X}_{reach,0} \leftarrow \mathcal{X}_0$
3: **for** $k = 0, \dots, N-1$ **do**
4: $\quad \mathcal{P}_k \leftarrow$ parallelotope over-approximation of $\mathcal{X}_{reach,k}$
5: $\quad (\breve{x}^{(1,k)}, \dots, \breve{x}^{(2^n,k)}) \leftarrow$ extreme states of $\mathcal{P}_k$ using (3.9)
6: $\quad$ **for** $i = 1, \dots, 2^n$ **do**
7: $\quad\quad \breve{u}^{(i,k)}(\cdot) \leftarrow$ solution of optimization problem (3.10) for each extreme states $\breve{x}^{(i,k)}$
8: $\quad$ **end for**
9: $\quad u_{conv}(x, t_k) \leftarrow$ convex control law (3.5) using $\breve{u}^{(i,k)}$ and $\breve{x}^{(i,k)},\ i \in \{1, \dots, 2^n\}$
10: $\quad \mathcal{R}_{[t_k, t_{k+1}], u_{conv}, \mathcal{W}}(\mathcal{X}_0) \leftarrow \mathcal{R}_{[0, \Delta t], u_{conv}, \mathcal{W}}(\mathcal{X}_{reach,k})$
11: $\quad \mathcal{X}_{reach,k+1} \leftarrow \mathcal{R}_{\Delta t, u_{conv}, \mathcal{W}}(\mathcal{X}_{reach,k})$
12: **end for**
13: **MP** $\leftarrow \{x_{ref}(\cdot), u_{ref}(\cdot), t_f, \mathcal{X}_0, \mathcal{X}_{reach,N}, u_{conv}(\cdot), \mathcal{R}_{[0,t_f], u_{conv}, \mathcal{W}}(\mathcal{X}_0)\}$

Step 1: We first compute a reference trajectory which steers the center $c_{x,0}$ of the initial set $\mathcal{X}_0$ as close as possible to the final state $x^{(f)}$ (see Fig. 3.3(a) and Alg. 1, line 1) by solving the following nonlinear optimization problem for the nominal system:

$$\begin{aligned} u_{ref}(\cdot) = \underset{u_{ref}(\cdot)}{\arg\min} \, & \|(\xi(c_{x,0}, u_{ref}(\cdot), 0, t_f) - x^{(f)})\|_1 + \gamma_{ref} \int_0^{t_f} \|u_{ref}(t)\|_1 \mathrm{d}t \\ \text{s.t. } & \forall t \in [0, t_f] : \xi(c_{x,0}, u_{ref}(\cdot), 0, t) \in \mathcal{S}_{ref}, \\ & u_{ref}(t) \in \mathcal{U}_{ref}, \end{aligned} \tag{3.8}$$

where $\mathcal{S}_{ref} \subseteq \mathcal{S}$ and $\mathcal{U}_{ref} \subseteq \mathcal{U}$ are tightened state and input constraints, respectively. Since we solve the optimization problem for the undisturbed dynamics and only for the center of the initial set, we have to tighten the nominal state and input constraints such that all states of the disturbed system still satisfy the original constraints and such that we have input capacities left to counteract the initial offset from the center. Similarly, the input weight $\gamma_{ref} \in \mathbb{R}_0^+$ can be chosen higher than γ to ensure that enough input capacities are left to find control inputs for the



(a) Reference trajectory **(b)** Solutions of extreme states

(c) Reachable set of single time step **(d)** Reachable set of all time steps

Figure 3.3: Convex control for nonlinear systems: (a) Compute a reference trajectory $x_{ref}(\cdot)$ from the center of the initial set $\mathcal{X}_0$ to the final state $x^{(f)}$. (b) Over-approximate the initial set by a parallelotope $\mathcal{P}_0$ and compute optimal trajectories for the extreme states. (c) Compute the reachable set $\mathcal{X}_{reach,1}$ for one time step under the convex control law while taking all possible disturbance effects into account. (d) Repeat the procedure.

extreme states. By restricting the inputs to be piecewise constant, this control problem can be efficiently solved using numerical optimization algorithms such as direct multiple-shooting [3]. We denote the resulting reference trajectory by

$$x_{ref}(\cdot) = \xi(x_{ref}(0), u_{ref}(\cdot), 0, \cdot).$$

For the subsequent steps, we divide the trajectory into N steps of length $\Delta t = \frac{t_f}{N}$ and we use again the shorthand notation $t_i := i\Delta t$.

Step 2: While Step 1 is performed only once, we perform Steps 2 and 3 for each time step k, $k \in \{0, \ldots, N-1\}$ (see Alg. 1, line 3). At time step k, we over-approximate the reachable set $\mathcal{X}_{reach,k}$ of the previous time step by a parallelotope $\mathcal{P}_k$, i.e., $\mathcal{X}_{reach,k} \subseteq \mathcal{P}_k$ (Alg. 1, line 4). At the first time step $k = 0$, we over-approximate the initial set $\mathcal{X}_0$ (Alg. 1, line 2). There exist efficient algorithms to formally compute parallelotope over-approximations for zonotopes, see e.g., [226]. We use parallelotopes, since they offer a good combination of a small number of extreme states and enclosed volume, and since analytical, closed-form expressions for convex combinations in parallelotopes exist (see Sec. 3.3.3).

For the parallelotope $\mathcal{P}_k$, we compute the 2^n extreme states $\breve{x}^{(1,k)}, \ldots, \breve{x}^{(2^n,k)}$ (Alg. 1, line 5). Computing the extreme states of a parallelotope $\mathcal{P} = \langle c_\mathcal{P}, g_\mathcal{P}^{(1)}, \ldots, g_\mathcal{P}^{(n)} \rangle$ can be done in a

numerically stable way by adding all 2^n combinations of the generators, i.e.,

$$\{\breve{x}^{(1)}, \dots, \breve{x}^{(2^n)}\} = c_{\mathcal{P}} \pm g_{\mathcal{P}}^{(1)} \pm \dots \pm g_{\mathcal{P}}^{(n)}, \tag{3.9}$$

where we use $\pm$ to indicate that each generator can be added either with a plus or minus sign. This is an advantage over general polytopes in half-space representation, for which it is much harder and numerically less reliable to compute the extreme states. For each extreme state $\breve{x}^{(i,k)}$ we then solve a nonlinear optimal control problem with the nominal system dynamics for one time step (Alg. 1, line 7):

$$\begin{aligned} \breve{u}^{(i,k)}(\,\cdot\,) = \underset{\breve{u}^{(i,k)}(\,\cdot\,)}{\arg\min} \, & \|(\xi(\breve{x}^{(i,k)}, \breve{u}^{(i,k)}(\,\cdot\,), 0, \Delta t) - x_{ref}(t_{k+1})\|_1 + \gamma \int_{t_k}^{t_{k+1}} \|\breve{u}^{(i,k)}(t)\|_1 \mathrm{d}t \\ \text{s.t. } & \forall t \in [0, \Delta t] : \xi(\breve{x}^{(i,k)}, \breve{u}^{(i,k)}(\,\cdot\,), 0, t) \in \bar{\mathcal{S}}, \\ & \breve{u}^{(i,k)}(t) \in \mathcal{U}, \end{aligned} \tag{3.10}$$

where $\bar{\mathcal{S}}$, with $\mathcal{S}_{ref} \subseteq \bar{\mathcal{S}} \subseteq \mathcal{S}$, denotes some tightened state constraints. We use these as a heuristic to have a buffer for effects from the superposition of nonlinear dynamics and disturbances. We discuss later how they can be obtained. Note that we do not need to tighten the input constraints, as the inputs are unaffected by these effects. By solving the optimization problem (3.10), we steer all extreme states as close as possible to the optimal reference trajectory while still considering the input costs.

Step 3: In the third step, we use the control inputs of the extreme states of the parallelotope $\mathcal{P}_k$ to obtain the convex control law in (3.5) for each state inside the reachable set of the last step $\mathcal{X}_{reach,k}$. After computing the convex control law $u_{conv}(x, \,\cdot\,)$ (Alg. 1, line 9), we compute the reachable set $\mathcal{R}_{[0,\Delta t],u_{conv},\mathcal{W}}(\mathcal{X}_{reach,k})$ (Alg. 1, line 10).

The over-approximation of the reachable set at the end of the current time step is the initial set for the next step $k+1$ (Alg. 1, line 11), i.e.,

$$\mathcal{X}_{reach,k+1} = \mathcal{R}_{\Delta t,u_{conv},\mathcal{W}}(\mathcal{X}_{reach,k}),$$

and we use this to continue with Step 2. By iterating Steps 2 and 3 for all N time steps, we obtain with $\mathcal{X}_f := \mathcal{X}_{reach,N}$ the over-approximation of the final reachable set of all states starting in the initial set $\mathcal{X}_0$ despite disturbances.

To ensure the satisfaction of the state constraints for all time points with finitely many operations, we simply check for the reachable sets of different time intervals. For the reachability analysis as described in Sec. 2.6, each reachable set $\mathcal{R}_{[0,\Delta t],u_{conv},\mathcal{W}}(\mathcal{X}_{reach,k})$ is represented by one or more zonotopes, depending on the time step size of the reachable set computation algorithm. To check that each of these zonotopes satisfies polyhedral state constraints of the form $\mathcal{S} = \{x \in \mathbb{R}^n | C_{\mathcal{S}} x \leq d_{\mathcal{S}}\}$, we use the following lemma.

Lemma 1. *A zonotope $\mathcal{Z} = \langle c, g^{(1)}, \dots, g^{(p)} \rangle$ satisfies polyhedral constraints of the form $\mathcal{C} = \{x \in \mathbb{R}^n \mid Cx \leq d\}$ if*

$$Cc + \sum_{i=1}^{p} |Cg^{(i)}| \leq d, \tag{3.11}$$

where the absolute value and less or equal operators are both performed element-wise.

Proof. Using the zonotope representation results in

$$Cx \leq d,\ \forall x \in \mathcal{Z}$$
$$\Leftrightarrow Cc + \sum_{i=1}^{p} \alpha_i C g^{(i)} \leq d,\ \forall \alpha_i \in [-1, 1].$$

The left side of the above inequality can be further bounded by

$$Cc + \sum_{i=1}^{p} \alpha_i C g^{(i)} \leq Cc + \sum_{i=1}^{p} |\alpha_i C g^{(i)}| \leq Cc + \sum_{i=1}^{p} \underbrace{|\alpha_i|}_{\leq 1} |C g^{(i)}| \leq Cc + \sum_{i=1}^{p} |C g^{(i)}|.$$

In fact, there exists an α with $\alpha_i = \pm 1$, such that the computed bound is touched. □

After the controller is computed offline using Alg. 1, we save the computed extreme states $\breve{x}^{(i,k)}$ together with the corresponding input trajectories $\breve{u}^{(i,k)}(\,\cdot\,)$ in a look-up table. In each time step during the online application of the convex controller, the current state $x(t_k)$ is expressed as a convex combination of the corresponding extreme states $\breve{x}^{(i,k)}$. The control input $u_{conv}(x(t_k),\,\cdot\,)$ is obtained from the convex combination of the extreme inputs $\breve{u}^{(i,k)}(\,\cdot\,)$ using the same parameters $\lambda_{i,k}(x(t_k))$ as described at the beginning of this section in (3.4) and (3.5). We can use the resulting controller and corresponding reachable set as a motion primitive for a maneuver automaton (Alg. 1, line 13).

The results of the convex control approach for nonlinear systems are summarized in the following theorem:

Theorem 1. *We consider a nonlinear system with disturbances* (2.3) *and with state and input constraints* (3.2)–(3.3)*. We assume that we have found a convex controller for this system using Algorithm 1. If*

$$\mathcal{R}_{[0,t_f],u_{conv},\mathcal{W}}(\mathcal{X}_0) \subseteq \mathcal{S}, \tag{3.12}$$

then any trajectory $\xi(x^{(0)}, u_{conv}(x(t_k),\,\cdot\,), w(\,\cdot\,),\,\cdot\,)$ *which starts in the initial set will end after time* t_f *in* $\mathcal{X}_f := \mathcal{X}_{reach,N}$*, i.e.,*

$$\xi(x^{(0)}, u_{conv}(x(t_k),\,\cdot\,), w(\,\cdot\,), t_f) \in \mathcal{X}_f,\ \forall x^{(0)} \in \mathcal{X}_0,\ \forall w(\,\cdot\,) \in \mathcal{W}.$$

Moreover, every trajectory satisfies the state constraints, and the applied inputs satisfy the input constraints despite the presence of disturbances, i.e., $\forall x^{(0)} \in \mathcal{X}_0,\ \forall w(\,\cdot\,) \in \mathcal{W},\ \forall t \in [0, t_f]$:

$$\xi(x^{(0)}, u_{conv}(x(t_k),\,\cdot\,), w(\,\cdot\,), t) \in \mathcal{S}\ \wedge\ u_{conv}(x(t_k), t) \in \mathcal{U}.$$

Proof. From the definition of the reachable set and the way we compute $\mathcal{X}_f$ as the over-approximation of the final reachable set, it follows that $\forall x^{(0)} \in \mathcal{X}_0,\ \forall w(\,\cdot\,) \in \mathcal{W}$:

$$\xi(x^{(0)}, u_{conv}(x(t_k),\,\cdot\,), w(\,\cdot\,), t_f) \in \mathcal{R}_{t_f,u_{conv},\mathcal{W}}(\mathcal{X}_0) = \mathcal{X}_f.$$

In the same way, it follows from assumption (3.12) that $\forall x^{(0)} \in \mathcal{X}_0,\ \forall w(\,\cdot\,) \in \mathcal{W},\ \forall t \in [0, t_f]$:

$$\xi(x^{(0)}, u_{conv}(x(t_k),\,\cdot\,), w(\,\cdot\,), t) \in \mathcal{R}_{[0,t_f],u_{conv},\mathcal{W}}(\mathcal{X}_0) \subseteq \mathcal{S},$$

and therefore, the state constraints are satisfied for any trajectory starting in the initial set.

When computing the control inputs $\breve{u}^{(i,k)}(\,\cdot\,)$ of the extreme states $\breve{x}^{(i,k)}$ in (3.10), we restrict them to lie in the input set $\mathcal{U}$ at all times. Since the input set is a polyhedron, and since any convex combination of points in a polyhedral set lies again in the polyhedral set [73], it follows that any convex combination of these inputs, and therefore any control input from our convex controller, satisfies the input constraints. □

Tightened State Constraints

In general, it is hard to know in advance how much tighter the new state constraints have to be, and there exists no general solution for this problem in the literature for disturbed, nonlinear systems. Since we combine controller synthesis with reachable set computation, we can check offline if the controller satisfies the real state constraints, and if not, adapt the tightened state constraints for the violated dimensions. By iteratively tightening the nominal state constraints offline in advance until the actual state constraints are satisfied by the real system, we obtain a formally correct controller for the online application. The reachability analysis in [74] relies on linearizing the dynamics and over-approximating the linearization errors and disturbance. Therefore, we are able to use these over-approximations to obtain a good initial estimate of how much we have to tighten the state constraints.

3.3.3 Closed-Form Expression of Convex Combinations

When applying the proposed convex control law (3.5), we have to find the parameters $\lambda_i(x)$ to express a state x as a convex combination of the extreme states $\breve{x}^{(i)}$ at each time step. Utilizing solvers for this problem is computationally expensive, especially for high-dimensional systems, and they would only provide an implicit solution. The computation time would restrict the sampling times of our controller, and the implicit solutions would prohibit the application of reachability analysis, which relies on an explicit, closed-form expression of the closed-loop dynamics.

To overcome these problems, closed-form expressions of convex combinations of simplices, parallelotopes, and general polytopes are presented in Appendix A. As mentioned before, parallelotopes offer a good combination of enclosed volume and a small number of extreme states. While simplices, for example, have the advantage that they have only $n+1$ extreme states, the enclosed volume is smaller and has an "impractical" shape for our application purposes. On the other hand, objects like higher-order zonotopes or general polytopes may better describe certain shapes, but when applied, the number of vertices increases significantly. Therefore we use parallelotopes to over-approximate the reachable set for the convex control computation in Step 2 in Sec. 3.3.2. We use these parallelotope over-approximations only to obtain the input combination and use the actual high-order zonotope for reachability analysis. In doing so, we avoid the error due to this over-approximation so that it has no significant impact on the reachability analysis.

The following theorem shows how to obtain closed-form expressions of convex combinations for parallelotopes:

Theorem 2. *We consider a parallelotope* $\mathcal{P} \subset \mathbb{R}^n$ *given by*

$$\mathcal{P} = \{x \in \mathbb{R}^n \mid x = c_\mathcal{P} + G_\mathcal{P}\alpha,\ \alpha_i \in [-1, 1]\},$$

with $c_\mathcal{P} \in \mathbb{R}^n$ *and* $G_\mathcal{P} \in \mathbb{R}^{n\times n}$*, and which has* 2^n *extreme states* $\breve{x}^{(1)}, \ldots, \breve{x}^{(2^n)}$*, see* (3.9)*. Given a state* $x \in \mathcal{P}$*, this state can be expressed as a convex combination of the extreme states as*

$$x = \sum_{i=1}^{2^n} \lambda_i(x)\breve{x}^{(i)},$$

where the parameters $\lambda_i(x),\ i \in \{1, \ldots, 2^n\}$*, are given by the following closed-form expression*

$$\lambda_i(x) = \prod_{j=1}^{n} \mu_{i,j}, \tag{3.13}$$

where

$$\mu_{i,j} = \begin{cases} x'_j, & \text{if } \alpha_j(\breve{x}^{(i)}) = 1 \\ 1 - x'_j, & \text{if } \alpha_j(\breve{x}^{(i)}) = -1. \end{cases} \tag{3.14}$$

Here, x' *is the transformed state of* x *under the affine transformation*

$$x' = \frac{1}{2}G_\mathcal{P}^{-1}\left(x - c_\mathcal{P}\right) + \frac{1}{2}\mathbf{1} \tag{3.15}$$

and x'_j *denotes its* j*-th entry.*

The proof, as well as results for sets other than parallelotopes, can be found in Appendix A. Note that for a point $\breve{x}^{(i)} \in \mathcal{P}$ to be an extreme point, the entries in the corresponding parameter vector $\alpha(\breve{x}^{(i)})$ must all be ± 1; therefore, one of the cases in (3.14) is always satisfied. Also, $G_\mathcal{P}^{-1}$ always exists since $G_\mathcal{P}$ is quadratic and has full rank by the definition of a parallelotope.

With this theorem, we are able to precompute the convex combinations for the parallelotopes at different points in time. Since we know the parallelotopes in advance, we can compute all matrices and matrix inverses of Thm. 2 offline. During the online computation, we simply plug in the current state in (3.15) and use the result to compute the parameters $\lambda_i(x)$. The computation of all $\lambda_i(x)$ for a ten-dimensional parallelotope can be performed in around $0.1ms$, which is over 200 times faster than using linear programming solvers, see Table A.1 in Appendix A.

3.3.4 Linear Approximation of the Convex Control Approach

The convex controller is a nonlinear controller, as can be seen by looking at the closed-form expressions of $\lambda_i(x)$ in (3.13), where the different entries x'_j are multiplied with each other. Although a convex combination is a linear combination of the extreme states, the parameters $\lambda_i(x)$ have a nonlinear dependency on the initial state for systems with dimensions greater than one (see (3.13) and (3.14)). This increases the nonlinearity of the whole closed-loop system, which leads to larger computation errors during the reachability computation, as well as to a higher computational complexity of the reachability computation itself [74].

In order to overcome these problems, we present an alternative approach where we use a linear approximation of the convex control approach. To do so, we take advantage of the fact that efficient techniques for reachability analysis use zonotopes as set representation [1,2]. We use the zonotope representation of the state set to compute a corresponding zonotope representation for the inputs. We modify only the third step in Sec. 3.3.2 by adding the input approximation, as shown in Alg. 2. For simpler computations, we restrict our considerations to piecewise-constant inputs, which are constant during each time interval $[t_k, t_{k+1})$.

Algorithm 2 Convex Control Algorithm for Nonlinear Systems with Disturbances Using Linear Input Combinations

Input: system dynamics $f(x,u,w)$, initial set $\mathcal{X}_0$, desired final state $x^{(f)}$, total time t_f, number of iterations N, constraint sets $\mathcal{S}, \mathcal{U}, \mathcal{S}_{ref}, \mathcal{U}_{ref}, \bar{\mathcal{S}}$, disturbance set $\mathcal{W}$, weighting factors γ, γ_{ref}
Output: motion primitive **MP**

⋮ (Lines 1 to 8 as in Alg. 1)

9: $\mathcal{Z}_{U,k} \leftarrow$ optimal zonotope approximation by solving (3.19)
10: $u_{zono}(x,t_k) \leftarrow$ linear approximated convex control law (3.18) using $\mathcal{Z}_{U,k}$
11: $\mathcal{R}_{[t_k,t_{k+1}],u_{zono},\mathcal{W}}(\mathcal{X}_0) \leftarrow \mathcal{R}_{[0,\Delta t],u_{zono},\mathcal{W}}(\mathcal{X}_{reach,k})$
12: $\mathcal{X}_{reach,k+1} \leftarrow \mathcal{R}_{\Delta t,u_{zono},\mathcal{W}}(\mathcal{X}_{reach,k})$
13: **end for** (from line 12 of Alg. 1)
14: **MP** $\leftarrow \{x_{ref}(\cdot), u_{ref}(\cdot), t_f, \mathcal{X}_0, \mathcal{X}_{reach,N}, u_{zono}(\cdot), \mathcal{R}_{[0,t_f],u_{zono},\mathcal{W}}(\mathcal{X}_0)\}$

Any state x in a parallelotope $\mathcal{P}$ is uniquely defined by the parameters $\alpha(x)$, i.e., $x = c_\mathcal{P} + G_\mathcal{P}\alpha(x)$, and therefore $\alpha(x)$ can be obtained by

$$\alpha(x) = G_\mathcal{P}^{-1}(x - c_\mathcal{P}), \tag{3.16}$$

where $G_\mathcal{P}^{-1}$ exists since $\mathcal{P}$ is a parallelotope.

In every time step $k \in \{0,\dots,N-1\}$, we want to find an input zonotope $\mathcal{Z}_{U,k}$ with center $c_{\mathcal{Z}_{U,k}}$ and generator matrix $G_{\mathcal{Z}_{U,k}}$ such that we obtain the corresponding control input for any state $x(t_k) \in \mathcal{P}_k$ just by using $\alpha(x(t_k))$ in

$$u_{zono}(x(t_k),t) = c_{\mathcal{Z}_{U,k}} + G_{\mathcal{Z}_{U,k}}\alpha(x(t_k)), \quad \forall t \in [t_k, t_{k+1}). \tag{3.17}$$

By plugging (3.16) into (3.17), we obtain

$$u_{zono}(x(t_k),t) = c_{\mathcal{Z}_{U,k}} + G_{\mathcal{Z}_{U,k}} G_{\mathcal{P}_k}^{-1}(x(t_k) - c_{\mathcal{P}_k}), \quad \forall t \in [t_k, t_{k+1}), \tag{3.18}$$

which is linear in $x(t_k)$ as desired.

We now have to choose the center $c_{\mathcal{Z}_{U,k}}$ and generator matrix $G_{\mathcal{Z}_{U,k}}$ of $\mathcal{Z}_{U,k}$, which best match the desired inputs. Clearly, as this is a linear approximation of the nonlinear convex input combinations, we cannot match the input for every state in $\mathcal{P}_k$ exactly. We choose $c_{\mathcal{Z}_{U,k}}$ and $G_{\mathcal{Z}_{U,k}}$ by solving an optimization problem such that the sum of the differences between the optimal inputs for the extreme states $\breve{u}^{(i,k)}$ and the inputs from the input zonotope $\mathcal{Z}_{U,k}$ for

the corresponding $\alpha(\breve{x}^{(i,k)})$ is minimized (Alg. 2, line 9):

$$\min_{c_{\mathcal{Z}_{U,k}}, G_{\mathcal{Z}_{U,k}}} \sum_{i=1}^{2^n} \left\| \breve{u}^{(i,k)} - (c_{\mathcal{Z}_{U,k}} + G_{\mathcal{Z}_{U,k}} \alpha(\breve{x}^{(i,k)})) \right\| \tag{3.19}$$

$$\text{s.t. } \forall i \in \{1, \dots, 2^n\} : \; c_{\mathcal{Z}_{U,k}} + G_{\mathcal{Z}_{U,k}} \alpha(\breve{x}^{(i,k)}) \in \mathcal{U}, \tag{3.20}$$

where we are free to choose which norm to use in (3.19). Now, we have the desired linear control law $u_{zono}(\,\cdot\,)$ for any state in $\mathcal{P}_k$ with (3.18). Using Alg. 2 and therefore replacing u_{conv} by u_{zono} in Thm. 1 ensures that the results of Thm. 1 also hold in the case of the new control law. We do not change anything about the reachability computation, with the exception of considering the new control law u_{zono}. Therefore, our algorithm is still sound, and we have guarantees for satisfaction of the state constraints. Due to both (3.20) and the fact that $\mathcal{U}$ is convex, it holds that $\forall k$,

$$\mathcal{Z}_{U,k} = \text{conv}\left(u_{zono}(\breve{x}^{(1,k)}, t_k), \dots, u_{zono}(\breve{x}^{(2^n,k)}, t_k)\right) \subseteq \mathcal{U},$$

and therefore the input constraints are satisfied as well.

3.3.5 Possible Extensions

We briefly discuss three possible extensions which can be used to improve the control performance in practice. To keep the presentation of the control approach easy to follow, we previously focused on the simplest case and now show how to improve the presented concepts.

Changing Control Inputs

For the convex control approach, we need to solve many open-loop optimal control problems in order to obtain the inputs for the extreme states. There exists a variety of efficient numerical tools. To bound the computational complexity, many of these tools restrict the inputs to be piecewise constant. However, if we restrict the inputs to be piecewise constant, we cannot expect to find a good solution if we only optimize over too short a horizon. If we consider, for example, a simple double integrator

$$x(t_{k+1}) = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} x(t_k) + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u(t_k),$$

where we start at $x(t_0) = [0, 0]^T$ and want to end after a single time step at $x^{(d)} = [10, 0]^T$, then the best input which minimizes $\|\xi(x(t_0), u(t_0), 0, \Delta t) - x^{(d)}\|_1$ is $u(t_0) = 0$. Even if we repeatedly optimize over this short horizon of 1 step while keeping the desired state $x^{(d)} = [10, 0]^T$, the optimal solution for this limited horizon is always zero. However, if we extend the horizon, this problem can be overcome. For our example, we can optimally solve it with a horizon of 2 steps with $u(t_0) = 10$ and $u(t_1) = -10$. Therefore, if we want to take advantage of the numerical solvers for our optimal control problems, it might be beneficial if we allow multiple input changes during each time interval.

We can consider multiple piecewise-constant control inputs during each time interval $[t_k, t_{k+1}]$ by simply switching the values of the inputs for the extreme states and thereby changing the

inputs for the interior states as well. For the convex control law, which is defined with general input trajectories, this works as described before. For the linear approximation, we simply compute a zonotope for each piecewise-constant input interval. Each zonotope best approximates the inputs during its respective interval and we then switch between these zonotopes.

Longer Optimization Horizon

Analogously to the problem discussed before, it might be beneficial to consider a longer time horizon for the optimization in general. We can do this by simply increasing the duration of each time interval $[t_k, t_{k+1}]$. However, since we only obtain measurements at times t_k, this would lead to fewer measurements and therefore possibly worse control performance due to longer open-loop periods. Instead, we can optimize over a longer time horizon spanning multiple time intervals and only apply the inputs for the first time interval. Then, after the reachability analysis for a single time interval, we again start a new optimization for a long time horizon, similar to the moving horizon of MPC [12]. To do so, we simply increase the length of the optimization horizon in (3.10) and also include states further in the future to the cost function. The reachable set computation remains unchanged, as we still only compute the reachable set for a single time interval. As we get closer to the final time t_f, we shrink the optimization horizon to not optimize for states after t_f.

Weighting of Individual States and Inputs

Depending on the actual system, certain states or inputs might be harder to control, more critical, or simply have a different scale than others and therefore must be weighted differently. To do this, we can simply include a weight matrix in which we multiply the states and inputs in addition to the simple weight γ. If we consider a longer optimization horizon, it is also possible to have these weights be time varying, e.g., to value the distance to the final state more than those to intermediate states.

3.3.6 Implementation

In this subsection, we give an overview of how the system controlled by our convex interpolation controller can be expressed in a form such that standard techniques as described in Sec. 2.6 can be applied to compute the reachable sets. To this end, we need to express the closed-loop dynamics in the form $\dot{x}(t) = f(x(t), u(x,t), w(t))$. While the convex control law and its linear approximation define a control law for each state, they depend on the states from the last measurement at time t_k, i.e., the control law at time $t \in [t_k, t_{k+1})$ has the form $u(x(t_k), t)$. Since the state is continuously evolving due to the continuous-time dynamics, we need to store the information about the initial state of this time interval $x(t_k)$. We do this by introducing an extended state space of the form $x_e(t) := \begin{bmatrix} x(t) \\ x(t_k) \end{bmatrix}$ with dynamics

$$\dot{x}_e(t) = \begin{bmatrix} \dot{x}(t) \\ \dot{x}(t_k) \end{bmatrix} = \begin{bmatrix} f\left(x(t), u_{ctrl}(x(t_k), t), w(t)\right) \\ \mathbf{0} \end{bmatrix}. \tag{3.21}$$

The control law $u_{ctrl}(x(t_k), t)$ refers either to the convex control law $u_{conv}(x(t_k), t)$, which we evaluate by plugging $x(t_k)$ into the closed-form description of $\lambda(x(t_k))$ from Thm. 2, or to the zonotope approximation $u_{zono}(x(t_k), t)$ from (3.18).

Starting with $\mathcal{X}_{reach,k} = \langle c_{x,k}, G_{x,k} \rangle$, we express the extended reachable set simply by stacking the two sets

$$\mathcal{X}_{e,k} = \left\langle \begin{bmatrix} c_{x,k} \\ c_{x,k} \end{bmatrix}, \begin{bmatrix} G_{x,k} \\ G_{x,k} \end{bmatrix} \right\rangle.$$

Starting from $\mathcal{X}_{e,k}$, we can use standard tools for reachability analysis as described in Sec. 2.6 to compute the reachable set based on the dynamics (3.21) for the desired time interval. This representation can also be used for the case that the inputs might change during the time interval $[t_k, t_{k+1}]$. In this case, we simply compute the reachable set for one time interval with fixed inputs and use the reachable set as the initial set for the next time interval, where the closed-loop system dynamics are changed based on the new $\breve{u}^{(i)}(\,\cdot\,)$ and $\mathcal{Z}_{U,k+1}$, respectively. Because we only adjust the input values based on the changes in the input trajectories of the extreme states, but do not have new measurements, we keep the $x(t_k)$ values constant. After we obtain a new measurement, i.e., after each iteration of our overall algorithm, we start a new reachable set computation based on the new measurements.

3.3.7 Numerical Example

In this section, we provide a numerical example to show the applicability of the proposed control approach for a constrained, nonlinear system. We choose a kinematic model of a vehicle, which is broadly used to model the most important dynamics of a car [90]:

$$\begin{aligned} \dot{v} &= a + w_1, \\ \dot{\Psi} &= b + w_2, \\ \dot{p}_x &= v\,\cos(\Psi), \\ \dot{p}_y &= v\,\sin(\Psi), \end{aligned} \tag{3.22}$$

where the states v, Ψ, p_x, and p_y are the velocity, orientation, and positions in the x and in y directions, respectively. The acceleration a and the normalized steering angle b are the inputs, and w_1 and w_2 are additive disturbances. They are constrained to lie in the intervals $a \in [-9.81, 9.81]\,\frac{m}{s^2}$, $b \in [-0.4, 0.4]\,\frac{rad}{s}$, $w_1 \in [-0.5, 0.5]\,\frac{m}{s^2}$, and $w_2 \in [-0.02, 0.02]\,\frac{rad}{s}$.

We use our convex control approach to compute a maneuver automaton for this model. For a compact illustration, we present the maneuver automaton only for the three motion primitives *drive straight*, *turn left*, and *turn right*, at velocities around $20\,\frac{m}{s}$. However, we can add additional motion primitives simply by repeating the procedure for other final states.

Following the techniques introduced in Sec. 1.2 and Sec. 2.7, a motion primitive $\mathbf{MP}_j$ can be concatenated to another motion primitive $\mathbf{MP}_i$ if the reachable set $\mathcal{X}_f^{(i)}$ of $\mathbf{MP}_i$ lies completely in the initial set $\mathcal{X}_0^{(j)}$ of $\mathbf{MP}_j$, see Fig. 1.2(b). Since the car dynamics are independent of the absolute position and orientation, we can transform the initial set $\mathcal{X}_0^{(j)}$ of $\mathbf{MP}_j$ by translating its position and orientation, and by rotating it to be aligned with the final set $\mathcal{X}_f^{(i)}$. We do this

using the following transformation:

$$T(\mathcal{X}_0^{(j)}) = \mathrm{rot}(\Psi_f^{(i)})\mathcal{X}_0^{(j)} \oplus [0, \Psi_f^{(i)}, p_{x,f}^{(i)}, p_{y,f}^{(i)}]^T,$$

where $\Psi_f^{(i)}, p_{x,f}^{(i)}, p_{y,f}$ refers to the end states of the reference trajectory of $\mathbf{MP}_i$ and where

$$\mathrm{rot}(\Psi_f^{(i)}) = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & \cos(\Psi_f^{(i)}) & -\sin(\Psi_f^{(i)}) \\ 0 & 0 & \sin(\Psi_f^{(i)}) & \cos(\Psi_f^{(i)}) \end{bmatrix}$$

is a rotation matrix which rotates the p_x and p_y states by $\Psi_f^{(i)}$. Therefore, for building a maneuver automaton, it suffices if all motion primitives start at the origin for Ψ, x, and y dimensions and only the velocity has to be discretized.

We choose for all motion primitives the initial set as the box $[19.8, 20.2]\,\frac{m}{s} \times [-0.02, 0.02]\,rad \times [-0.2, 0.2]\,m \times [-0.2, 0.2]\,m$. We apply our convex control approach three times for different final states, corresponding to the three motion primitives. The final states are given for the *drive straight* motion primitive by $\left[20\,\frac{m}{s}, 0\,rad, 20\,m, 0\,m\right]^T$ and for the *turn left* and *turn right* motion primitives by $\left[20\,\frac{m}{s}, \pm 0.2\,rad, 19.87\,m, \pm 1.99\,m\right]^T$. Each final state must be reached in one second. We divide the main trajectory into 10 sections and apply four different piecewise-constant control values for each section of the reference trajectory. For the local controllers, we choose the cost function

$$J_{corner} = (\breve{x}^{(i,k+1)} - x_{ref}(t_{k+1}))^T Q (\breve{x}^{(i,k+1)} - x_{ref}(t_{k+1})),$$

with $\breve{x}^{(i,k+1)} = \xi(\breve{x}^{(i,k)}, \breve{u}^{(i,k)}(\,\cdot\,), 0, \Delta t)$ and $Q = \mathrm{diag}([2, 5, 1, 1])$. For this example, we choose a quadratic cost function, as there exist efficient solvers for optimal control problems with this kind of cost function.

Results

We implement our approach in MATLAB and use the ACADO toolbox [91] to solve the optimal control problems with a multiple shooting algorithm. For the reachability computation, we use the CORA toolbox [82], where the disturbances are handled as an uncontrollable input. As the CORA toolbox works with zonotopes, we use the techniques presented in Sec. 3.3.4 to obtain a parallelotope approximating the inputs.

The computation of each motion primitive takes around 10 seconds, which can be performed offline. Of these 10 seconds, 42% is required for computing the reachable sets and around 38% for computing the control laws for the extreme states. The computations are performed on a computer with a 3.1 GHz dual-core Intel i7 processor and 16 GB memory and without using parallel computations. The online computation of the input values can be performed in around $0.01\,ms$, making it applicable to fast systems.

The whole reachable set is shown for the *turn left* motion primitive in Fig. 3.4. The initial sets (black) and final sets for all motion primitives are plotted in Fig. 3.5, where we shift the final sets by the desired final states $x^{(f)}$ to have a better comparison. For the convex

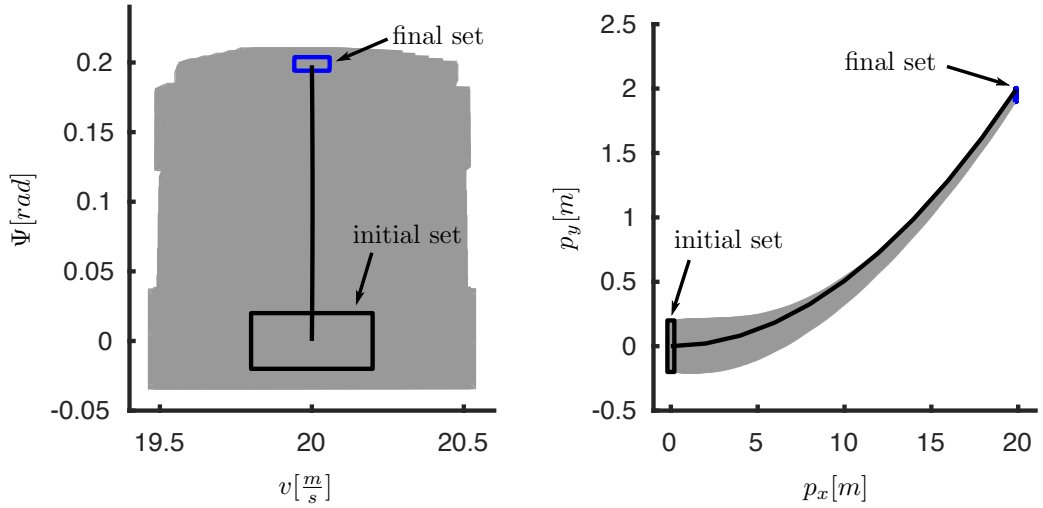


Figure 3.4: Reachable sets for the *turn left* motion primitive with the convex controller, projected onto the (v, Ψ) and the (p_x, p_y) planes. The initial set is plotted in black, the final set in blue, and the reachable set for all times between in gray. The black line shows the reference trajectory $x_{ref}(\cdot)$.

controller, all shifted final sets (blue) are completely contained in the initial set. Therefore, we are able to connect all motion primitives with each other and obtain a fully connected maneuver automaton, as illustrated in Fig. 3.6. The controller satisfies the input constraints at all times due to the way we computed it.

Comparison with LQR Controller

For comparison, we have also implemented an LQR tracking controller [37] (red, solid in Fig. 3.5). It uses the same reference trajectory. In each step, we linearize the system around the corresponding state on the reference trajectory and compute an LQR controller. We use the same Q matrix as for the corner trajectories. To weight the inputs, we choose the R matrix for the Ricatti equation to be the identity matrix. As we see in Fig. 3.5, the shifted reachable sets exceed the initial sets, making it impossible to combine the motion primitive with other motion primitives. Moreover, since the LQR controller does not take input constraints into account, it uses inputs for b in (3.22) up to $0.54\,\frac{rad}{s}$, which is more than the maximally allowed value.

By decreasing the weights of the inputs, we can move the final sets inside the initial sets; however, the input constraint violation increases even more in this case. If we increase the weights on the inputs to $R = 4I$, we obtain a maximal $b = 0.40\,\frac{rad}{s}$, which barely satisfies the constraint. However, the reachable sets (red, dashed in Fig. 3.5) are very large and not suitable for a maneuver automaton at all. Therefore, we cannot achieve both a good final set and input satisfaction with LQR controllers. This shows the advantage of the convex control approach, which optimizes the reachable set while ensuring the satisfaction of the constraints.

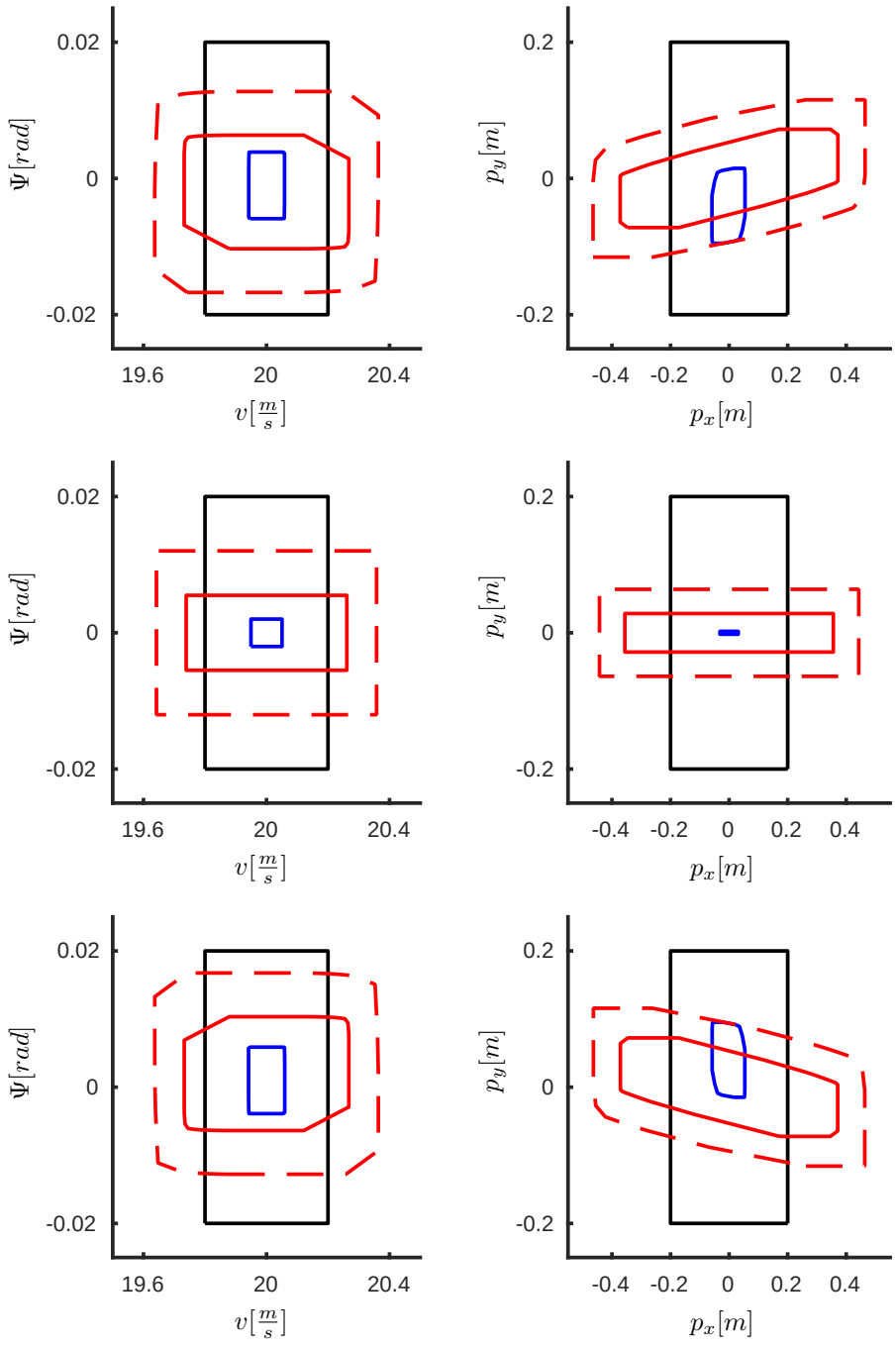


Figure 3.5: Initial (black) and shifted final sets (blue) of the convex controller, projected onto the (v, Ψ) and the (p_x, p_y) planes, for the *turn left* (top), *drive straight* (center), and *turn right* (bottom) motion primitives. For comparison the final sets of two LQR controllers (red).

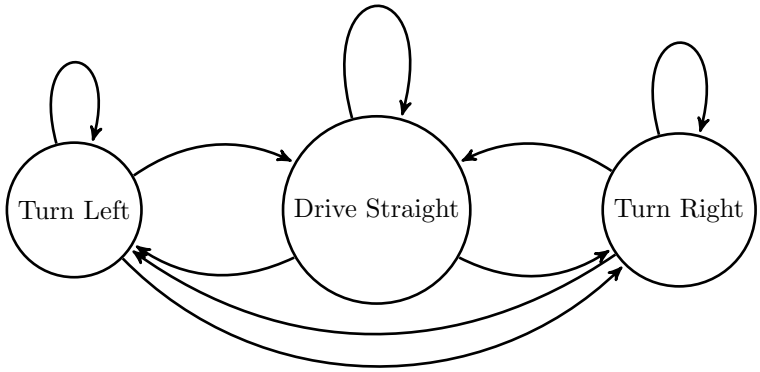
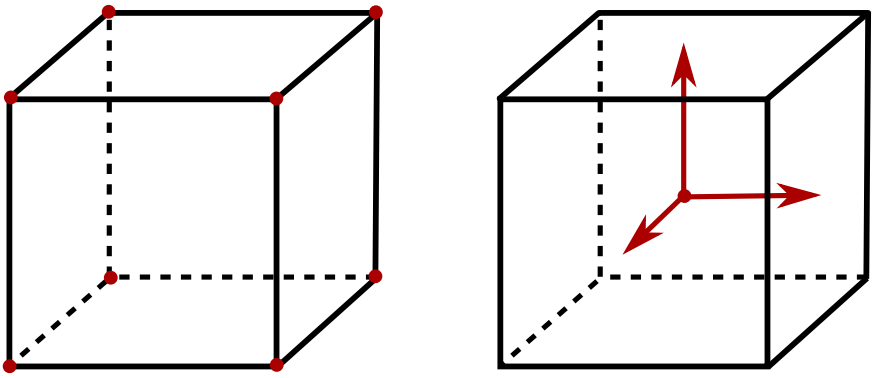


Figure 3.6: Resulting maneuver automaton for the car model. Since the automaton is fully connected, any combination of motion primitives is possible.

3.4 Generator Interpolation Controller

In the last section, we presented the convex control approach, where we use the convex combination of the inputs for the extreme state to obtain an interpolated control input for all states of the initial set. Although we can solve each single optimization problem very quickly and the numerical results demonstrate good performance, the number of extreme states scales poorly with the dimension of the state space, even for parallelotopes with 2^n. It therefore gets too large for higher dimensions.

Instead of interpolating the extreme states, we present in this section[1] another approach which works similarly to the convex control approach; however, we are using the generators of a zonotope to obtain the control inputs. As mentioned in Sec. 2.4, for zonotopes, the number of generators scales much better than the number of extreme states; for example, a parallelotope in $\mathbb{R}^n$ only has n generators. This is visualized for a box as a special case of parallelotope, in Fig. 3.7.



(a) Extreme point representation **(b)** Generator representation

Figure 3.7: Representation of a box in $\mathbb{R}^3$: (a) with $2^n = 8$ extreme points (depicted as red dots) or (b) with a center and $n = 3$ generators (depicted as red arrows which start from the center of the box).

Similarly to the previous section, we first present the idea for linear systems before we extend the approach to disturbed and nonlinear systems.

3.4.1 Linear Systems

As we discussed for the convex control approach, solving (3.1) for disturbed nonlinear systems for every possible state of the initial set is not feasible, as there are uncountably many. We therefore restrict our consideration at first to linearized dynamics: there we can apply the superposition principle to overcome this problem by interpolating finitely many solutions.

[1] This section and most of its figures are based on [218] © 2016 IFAC.

Let us consider the following discrete-time, linear, time-varying system

$$x(t_{k+1}) = A_k x(t_k) + B_k u(t_k), \tag{3.23}$$

with $A_k \in \mathbb{R}^n$ and $B_k \in \mathbb{R}^m$ denoting the time-varying state and input matrices at time t_k. We examine the more general case of time-varying linear systems, since we obtain them when linearizing the nonlinear dynamics in the later part. The restriction to discrete-time dynamics allows us to simplify the optimization problem and to have finitely many optimization variables.

Let us start by expressing the evolution of a single state $x^{(0)} \in \mathcal{X}_0$ as a function of the control input for the linear dynamics (3.23):

$$\xi(x^{(0)}, u(\,\cdot\,), 0, t_N) = \bar{A}x^{(0)} + \sum_{k=0}^{N-1} \bar{B}_k u(x^{(0)}, t_k), \tag{3.24}$$

where we use the shorthand notations:

$$\begin{aligned} \bar{A} &:= A_{N-1} \dots A_0, \\ \bar{B}_k &:= A_{N-1} \dots A_{k+1} B_k, \quad \forall k \in \{0, \cdots, N-2\}, \end{aligned}$$

with $\bar{B}_{N-1} := B_{N-1}$. Since the initial set is given by the zonotope $\mathcal{X}_0 = \langle c_{x,0}, g_{x,0}^{(1)}, \dots, g_{x,0}^{(p_0)} \rangle$ or can be over-approximated by one, we can express (3.24) as

$$\xi(x^{(0)}, u(\,\cdot\,), 0, t_N) = \bar{A}\left(c_{x,0} + \sum_{i=1}^{p_0} \alpha_i(x^{(0)}) g_{x,0}^{(i)}\right) + \sum_{k=0}^{N-1} \bar{B}_k u(x^{(0)}, t_k), \tag{3.25}$$

where $\alpha(x^{(0)})$ is such that

$$x^{(0)} = c_{x,0} + \sum_{i=1}^{p_0} \alpha_i(x^{(0)}) g_{x,0}^{(i)}. \tag{3.26}$$

The structure of zonotopes as a superposition of generators allows us to express the evolution of any state as a weighted combination of the center and generator vectors. Therefore, if we choose a similar structure for the control input, we are able to control every trajectory starting from $\mathcal{X}_0$ by simply finding input trajectories for the center and generators and interpolating between them using the same $\alpha(x^{(0)})$ from (3.26):

$$u_{gen}(x^{(0)}, t) = u(c_{x,0}, t) + \sum_{i=1}^{p_0} \alpha_i(x^{(0)}) u(g_{x,0}^{(i)}, t). \tag{3.27}$$

In the following lemma, we show how an upper bound for the distance of any state in the reachable set to the final state $x^{(f)}$ under this control law can be obtained:

Lemma 2. *Let $\mathcal{R}_{t_f, u_{gen}, 0}(\mathcal{X}_0)$ denote the reachable set for system (3.23) starting from $\mathcal{X}_0 =$*

$\langle c_{x,0}, g_{x,0}^{(1)}, \ldots, g_{x,0}^{(p_0)} \rangle$, *then the following inequality holds*

$$\begin{aligned}
&\|(\mathcal{R}_{t_f,u_{gen},0}(\mathcal{X}_0) - x^{(f)})\|_1 + \gamma \int_0^{t_f} \|u_{gen}(\mathcal{R}_{t,u_{gen},0}(\mathcal{X}_0), t)\|_1 \mathrm{d}t \\
\leq &\left\| \bar{A} c_{x,0} + \sum_{k=0}^{N-1} \bar{B}_k u(c_{x,0}, t_k) - x^{(f)} \right\|_1 + \sum_{i=1}^{p_0} \left\| \bar{A} g_{x,0}^{(i)} + \sum_{k=0}^{N-1} \bar{B}_k u(g_{x,0}^{(i)}, t_k) \right\|_1 \\
&+ \hat{\gamma} \sum_{k=0}^{N-1} \left(\|u(c_{x,0}, t_k)\|_1 + \sum_{i=1}^{p_0} \left\| u(g_x^{(i)}, t_k) \right\|_1 \right),
\end{aligned}$$

with $\hat{\gamma} = \frac{t_f}{N}\gamma$.

Proof. For piecewise-constant inputs as obtained from the discrete dynamics, the input cost term can be written as

$$\gamma \int_0^{t_f} \|u_{gen}(x^{(0)}, t)\|_1 \mathrm{d}t = \hat{\gamma} \sum_{k=0}^{N-1} \|u_{gen}(x^{(0)}, t_k)\|_1. \tag{3.28}$$

Therefore, the cost function can be bounded as follows:

$$\begin{aligned}
&\|(\mathcal{R}_{t_f,u_{gen},0}(\mathcal{X}_0) - x^{(f)})\|_1 + \gamma \int_0^{t_f} \|u_{gen}(\mathcal{R}_{t,u_{gen},0}(\mathcal{X}_0), t)\|_1 \mathrm{d}t \\
&= \max_{x^{(0)} \in \mathcal{X}_0} \left\| \xi(x^{(0)}, u_{gen}(x^{(0)}, \cdot), 0, t_N) - x^{(f)} \right\|_1 + \gamma \int_0^{t_f} \|u_{gen}(x^{(0)}, t)\|_1 \mathrm{d}t \\
&\overset{(3.24),\,(3.28)}{=} \max_{x^{(0)} \in \mathcal{X}_0} \left\| \bar{A} x^{(0)} + \sum_{k=0}^{N-1} \bar{B}_k u_{gen}(x^{(0)}, t_k) - x^{(f)} \right\|_1 + \hat{\gamma} \sum_{k=0}^{N-1} \left\| u_{gen}(x^{(0)}, t_k) \right\|_1 \\
&\overset{(3.26),\,(3.27)}{=} \max_{\alpha \in [-1,1]^{p_0}} \left\| \bar{A} c_{x,0} + \sum_{i=1}^{p_0} \bar{A} g_{x,0}^{(i)} \alpha_i + \sum_{k=0}^{N-1} \bar{B}_k u(c_{x,0}, t_k) + \sum_{i=1}^{p_0} \sum_{k=0}^{N-1} \bar{B}_k u(g_{x,0}^{(i)}, t_k) \alpha_i - x^{(f)} \right\|_1 \\
&\qquad + \hat{\gamma} \sum_{k=0}^{N-1} \left\| u(c_{x,0}, t_k) + \sum_{i=1}^{p_0} u(g_x^{(i)}, t_k) \alpha_i \right\|_1 \\
&\overset{\text{triangle inequal.}}{\leq} \max_{\alpha \in [-1,1]^{p_0}} \left\| \bar{A} c_{x,0} + \sum_{k=0}^{N-1} \bar{B}_k u(c_{x,0}, t_k) - x^{(f)} \right\|_1 + \sum_{i=1}^{p_0} \left\| \left(\bar{A} g_{x,0}^{(i)} + \sum_{k=0}^{N-1} \bar{B}_k u(g_{x,0}^{(i)}, t_k) \right) \alpha_i \right\|_1 \\
&\qquad + \hat{\gamma} \sum_{k=0}^{N-1} \left(\|u(c_{x,0}, t_k)\|_1 + \sum_{i=1}^{p_0} \left\| u(g_x^{(i)}, t_k) \alpha_i \right\|_1 \right) \\
&\leq \max_{\alpha \in [-1,1]^{p_0}} \left\| \bar{A} c_{x,0} + \sum_{k=0}^{N-1} \bar{B}_k u(c_{x,0}, t_k) - x^{(f)} \right\|_1 + \sum_{i=1}^{p_0} \left\| \left(\bar{A} g_{x,0}^{(i)} + \sum_{k=0}^{N-1} \bar{B}_k u(g_{x,0}^{(i)}, t_k) \right) \right\|_1 \underbrace{\|\alpha_i\|_1}_{\leq 1} \\
&\qquad + \hat{\gamma} \sum_{k=0}^{N-1} \left(\|u(c_{x,0}, t_k)\|_1 + \sum_{i=1}^{p_0} \left\| u(g_x^{(i)}, t_k) \right\|_1 \underbrace{\|\alpha_i\|_1}_{\leq 1} \right)
\end{aligned}$$

$$= \left\| \bar{A}c_{x,0} + \sum_{k=0}^{N-1} \bar{B}_k u(c_{x,0}, t_k) - x^{(f)} \right\|_1 + \sum_{i=1}^{p_0} \left\| \bar{A}g_{x,0}^{(i)} + \sum_{k=0}^{N-1} \bar{B}_k u(g_{x,0}^{(i)}, t_k) \right\|_1$$
$$+ \hat{\gamma} \sum_{k=0}^{N-1} \left(\|u(c_{x,0}, t_k)\|_1 + \sum_{i=1}^{p_0} \left\| u(g_x^{(i)}, t_k) \right\|_1 \right).$$

For the first inequality, we use the triangle inequality $\|a+b\|_1 \leq \|a\|_1 + \|b\|_1$, for any vectors $a, b \in \mathbb{R}^n$. The last equality in the proof holds due to the fact that we choose the α_i such that they maximize the 1-norm and since $|\alpha_i| \leq 1, \forall i \in \{1, ..., p_0\}$, they can be neglected. □

Lemma 2 allows us to obtain a new optimization problem, which minimizes an upper bound for the size of the reachable set. While the original optimization problem containing any possible combination of α_i becomes hard to compute, the new formulation decouples the influences of the individual generators. This result can also be interpreted graphically as illustrated in Fig. 3.8: The center of the initial set is steered to the reference state, while the generators are steered to the origin. Since a zonotope is just the superposition of the center and generators, this brings all states close to $x^{(f)}$.

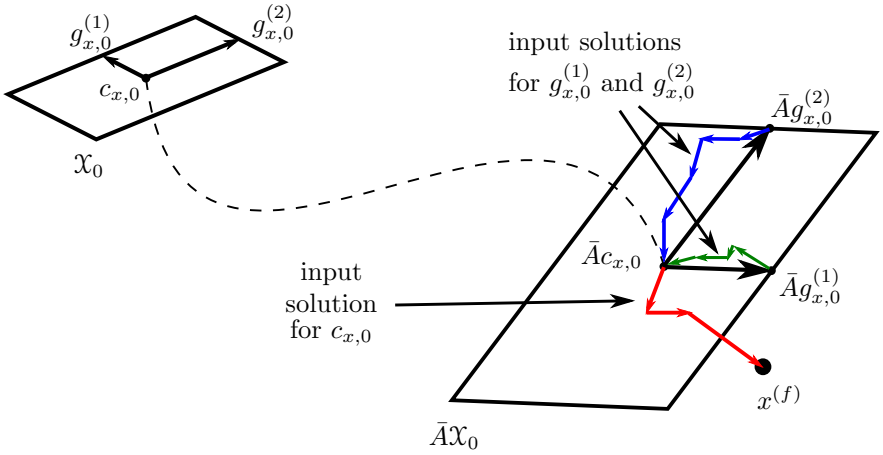


Figure 3.8: Illustration of the control interpolation of the generators: All states of the resulting set should be close to $x^{(f)}$. This is achieved by steering the center $c_{x,0}$ to $x^{(f)}$ and making the generators $g_{x,0}^{(1)}, g_{x,0}^{(2)}$ as small as possible, i.e., steer them to the origin.

The optimization problem is still coupled through the constraints, as the state and overall input must satisfy the state constraints $\mathcal{S}$ and input constraints $\mathcal{U}$, respectively. Since we consider discrete dynamics for now, we only have to check the reachable sets at finitely many time points. We use Lemma 1 from Sec. 3.3 to check the state constraints the same way as before. We must also ensure that the sum of all possible inputs for the center and generators does not exceed the input bounds:

Corollary 1. *The generator interpolation control law* (3.27) *satisfies the input constraints*

$\mathcal{U} = \{u \in \mathbb{R}^m \mid C_\mathcal{U} u \leq d_\mathcal{U}\}\ \forall x^{(0)} \in \mathcal{X}_0$, *if*

$$C_\mathcal{U} u(c_{x,0}, t_k) + \sum_{i=1}^{p_0} |C_\mathcal{U} u(g_{x,0}^{(i)}, t_k)| \leq d_\mathcal{U}, \forall k \in \{0, \dots, N-1\}.$$

Proof. Using the same proof concept as in Lemma 1 results in

$$\begin{aligned} & u(c_{x,0}, t_k) + \sum_{i=1}^{p_0} \alpha_i u(g_{x,0}^{(i)}, t_k) \in \mathcal{U}, \quad \forall \alpha \in [-1,1]^{p_0}, \forall k \in \{0, \dots N-1\} \\ \Leftrightarrow\ & C_\mathcal{U} u(c_{x,0}, t_k) + \sum_{i=1}^{p_0} |C_\mathcal{U} u(g_{x,0}^{(i)}, t_k)| \leq d_\mathcal{U}, \quad \forall k \in \{0, \dots, N-1\}. \end{aligned}$$

□

This simplification is possible by exploiting the symmetry of zonotopes. Let us combine the results from Lemma 2, Lemma 1, and Corollary 1 into a single optimization problem, which is presented in the following theorem.

Theorem 3. *For system* (3.23)*, the following optimization problem minimizes an upper-bound for the cost function in* (3.1) *while taking the state and input constraints into account:*

$$\min_{\substack{u(c_{x,0}), u(g_{x,0}^{(1)}), \\ \dots, u(g_{x,0}^{(p_0)})}} \left\| \begin{bmatrix} \bar{A} c_{x,0} + \bar{B} u(c_{x,0}) \\ \bar{A} g_{x,0}^{(1)} + \bar{B} u(g_{x,0}^{(1)}) \\ \vdots \\ \bar{A} g_{x,0}^{(p_0)} + \bar{B} u(g_{x,0}^{(p_0)}) \end{bmatrix} \right\|_1 + \hat{\gamma} \left\| \begin{bmatrix} u(c_{x,0}) \\ u(g_{x,0}^{(1)}) \\ \vdots \\ u(g_{x,0}^{(p_0)}) \end{bmatrix} \right\|_1 \tag{3.29}$$

s.t. $\forall k \in \{0, \dots, N-1\}$:

$$C_\mathcal{S}\, \xi(c_{x,0}, u(c_{x,0}, \cdot), 0, t_{k+1}) + \sum_{i=1}^{p_0} |C_\mathcal{S}\, \xi(g_{x,0}^{(i)}, u(g_x^{(i)}, \cdot), 0, t_{k+1})| \leq d_\mathcal{S}, \tag{3.30}$$

$$C_\mathcal{U} u(c_{x,0}, t_k) + \sum_{i=1}^{p_0} |C_\mathcal{U} u(g_{x,0}^{(i)}, t_k)| \leq d_\mathcal{U}, \tag{3.31}$$

with the shorthands $\hat{\gamma} := \frac{t_f}{N}\gamma$, $\bar{B} := \left[\bar{B}_0, \cdots, \bar{B}_{N-1}\right]$, $u(g_{x,0}^{(i)}) := [u(g_{x,0}^{(i)}, t_0)^T, \dots, u(g_{x,0}^{(i)}, t_{N-1})^T]^T$, *and equivalently for* $u(c_{x,0})$.

Proof. It follows from Lemma 2, together with the fact that

$$\|a\|_1 + \|b\|_1 = \left\| \begin{bmatrix} a \\ b \end{bmatrix} \right\|_1,$$

that we can obtain the cost function in the form of (3.29). The state constraints (3.30) follow directly from Lemma 1 and the input constraints (3.31) from Corollary 1. □

It is shown in [92, Ch. 6] and [93] that the 1-norm and absolute value can be transformed to linear constraints and cost functions. Therefore, this problem can be solved efficiently in a single linear program.

Depending on the physical meaning of the states, the size of the resulting reachable set in certain dimensions might be more critical than others. Similar to the convex interpolation controller, we can consider this by using a weighting matrix Q, by which we multiply the state costs of the center and each generator, e.g., $\|Q\left(\bar{A}c_{x,0} + \bar{B}u(c_{x,0})\right)\|$. The same can be done for the inputs as well.

3.4.2 Nonlinear Dynamics and Disturbances

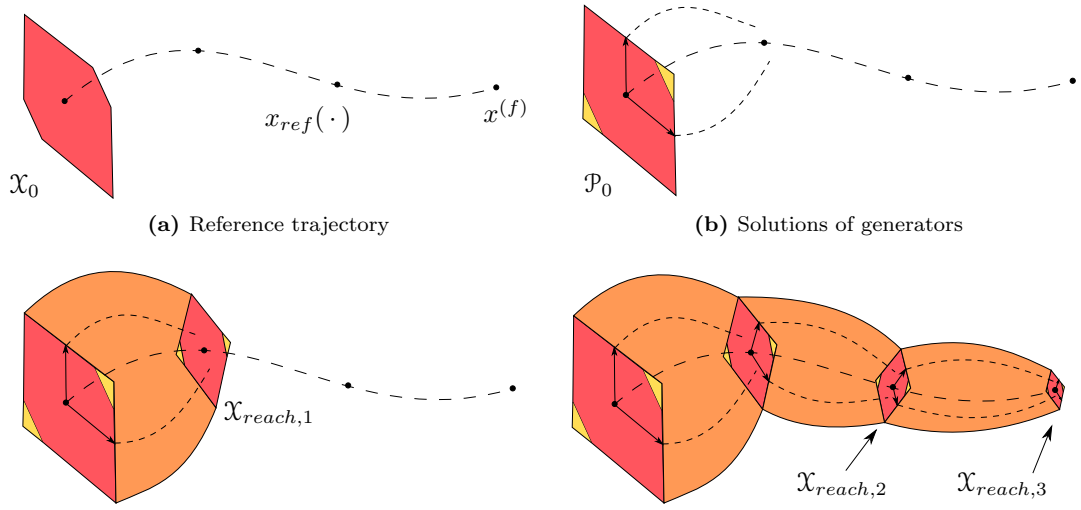
Let us now extend the approach to nonlinear systems with disturbances (2.3). We use a similar approach as we do for the convex interpolation controller: We first compute a reference trajectory and linearize the system along this trajectory in order to be able to apply the theory developed for linear systems. To fully consider the disturbances and the linearization errors, we apply the control law iteratively for short time horizons and compute the reachable set for the times in between. This way, we obtain feedback and guarantee that it satisfies the constraints at all points in time. The approach for nonlinear systems is summarized in Alg. 3 and illustrated in Fig. 3.9. It consists of the following three steps:

Algorithm 3 Offline Part of the Generator Interpolation Control Algorithm for Nonlinear Systems with Disturbances

Input: system dynamics $f(x,u,w)$, initial set $\mathcal{X}_0$, desired final state $x^{(f)}$, total time t_f, number of iterations N, M, constraint sets $\mathcal{S}, \mathcal{U}, \mathcal{S}_{ref}, \mathcal{U}_{ref}, \bar{\mathcal{S}}$, disturbance set $\mathcal{W}$, weighting factors γ, γ_{ref}
Output: motion primitive **MP**

1: $(x_{ref}(\cdot), u_{ref}(\cdot)) \leftarrow$ solution of optimization problem (3.8) for reference trajectory
2: $(A_k, B_k) \leftarrow$ linearization and time-discretization of $f(x,u,0)$ along $x_{ref}(\cdot), u_{ref}(\cdot)$ using (3.32) and (3.33)
3: Initialize: $\mathcal{X}_{reach,0} \leftarrow \mathcal{X}_0$
4: **for** $l = 0, \dots, M-1$ **do**
5: $\quad \mathcal{P}_l \leftarrow$ parallelotope over-approximation of $\mathcal{X}_{reach,l}$
6: $\quad (u(g_{x,l}^{(1)}), \dots, u(g_{x,l}^{(n)})) \leftarrow$ solution of optimization problem (3.34) to obtain feedforward control input for generators of $\mathcal{P}_l$
7: $\quad u_{gen}(x, t_{lh}) \leftarrow$ control law (3.37) using $u_{ref}(\cdot)$ and $u(g_{x,l}^{(1)}), \dots, u(g_{x,l}^{(n)})$
8: $\quad \mathcal{R}_{[t_{lh}, t_{(l+1)h}], u_{gen}, \mathcal{W}}(\mathcal{X}_0) \leftarrow \mathcal{R}_{[0,t_h], u_{gen}, \mathcal{W}}(\mathcal{X}_{reach,k})$
9: $\quad \mathcal{X}_{reach,l+1} \leftarrow \mathcal{R}_{t_h, u_{gen}, \mathcal{W}}(\mathcal{X}_{reach,l})$
10: **end for**
11: **MP** $\leftarrow \{x_{ref}(\cdot), u_{ref}(\cdot), t_f, \mathcal{X}_0, \mathcal{X}_{reach,N}, u_{gen}(\cdot), \mathcal{R}_{[0,t_f], u_{gen}, \mathcal{W}}(\mathcal{X}_0)\}$

Step 1: As done for the convex interpolation controller in Sec. 3.3, we begin by computing a reference trajectory $x_{ref}(\cdot)$, which steers the center $c_{x,0}$ of the initial set $\mathcal{X}_0$ as close as possible to the desired final state $x^{(f)}$ (see Fig. 3.9(a) and Alg. 3, line 1) by solving (3.8). In contrast to the convex interpolation controller, we need to explicitly apply the superposition principle for the generator interpolation controller. Therefore, as an intermediate step, we linearize the system along the reference trajectory (Alg. 3, line 2). To obtain the linearization



(a) Reference trajectory **(b)** Solutions of generators

(c) Reachable set of single time step **(d)** Reachable set of all time steps

Figure 3.9: Applying the generator interpolation control approach to nonlinear systems is similar as done for the convex interpolation controller: (a) Compute a reference trajectory $x_{ref}(\cdot)$ from the center of the initial set $\mathcal{X}_0$ to the final state $x^{(f)}$ and linearize the system along that trajectory. (b) Over-approximate the initial set by a parallelotope $\mathcal{P}_0$ and compute optimal control inputs for its generators. (c) Compute the reachable set $\mathcal{X}_{reach,1}$ for one iteration steps under the optimal control law while taking all possible disturbance effects into account. (d) Repeat the procedure.

points t'_k, we first divide the reference trajectory into N parts of duration $\Delta t = \frac{t_f}{N},\ N \in \mathbb{N}$, and then choose the middle of the time intervals $[t_k, t_{k+1}]$ with $t_k = k\Delta t,\ k \in \{0, \dots, N\}$, i.e., $t'_k = \frac{1}{2}(t_{k+1} - t_k),\ k \in \{0, \dots, N-1\}$. The system matrices of the linear dynamics are then given by

$$A_{c,k} = \left.\frac{\partial f(x,u,0)}{\partial x}\right|_{\substack{x=x_{ref}(t'_k)\\u=u_{ref}(t'_k)}}, \quad B_{c,k} = \left.\frac{\partial f(x,u,0)}{\partial u}\right|_{\substack{x=x_{ref}(t'_k)\\u=u_{ref}(t'_k)}}. \tag{3.32}$$

By restricting the inputs, including those for the reference trajectory, to be constant in each time interval, we can treat the system as a discrete-time, linear system of the form (3.23) with

$$A_k = e^{A_{c,k}\Delta t}, \quad B_k = \int_0^{\Delta t} e^{A_{c,k}\tau} \mathrm{d}\tau \; B_{c,k}. \tag{3.33}$$

We do not consider any affine term, as we are only interested in the system dynamics relative to the reference trajectory. In the next step, we use them to control the generators relative to the center which follows the reference trajectory.

Step 2: As discussed in Sec. 3.3.5, when dealing with piecewise-constant inputs, it is often beneficial to consider several steps. Therefore, we present the approach directly for a time horizon of h time steps, with $N = Mh$ and $M, h \in \mathbb{N}$. At each iteration step $l \in \{0, \dots, M-1\}$, we apply the set-based optimal control approach using the linearized dynamics. We use the

reachable set $\mathcal{X}_{reach,l}$ from the last time step as the initial set and over-approximate it by a parallelotope $\mathcal{P}_l$, i.e., $\mathcal{X}_{reach,l} \subseteq \mathcal{P}_l$ (Alg. 3, line 5). We initialize our approach with $\mathcal{X}_{reach,0} := \mathcal{X}_0$ (Alg. 3, line 3). As done for the convex interpolation controller in Sec. 3.3, we over-approximate the initial set by a parallelotope to reduce the number of generators which are considered in the optimization problem and to have unique control inputs when applying the controller online, as discussed later. Since we compute over-approximations, our approach is still sound, and we ensure the satisfaction of the input constraints for all states.

At each step, we use the previously described generator interpolation control approach for linear dynamics to control the trajectories starting from $\mathcal{P}_l$ as close as possible to $x_{ref}(t_{(l+1)h})$ (see Fig. 3.9(b)) by solving (3.29) for the limited time horizon and with the fixed input trajectory for the center $u(c_{x,l}, t_k) := u_{ref}(t_k),\ \forall k \in \{lh, \dots, (l+1)h-1\}$ (Alg. 3, line 6):

$$\min_{\substack{u(g_{x,l}^{(1)}),\\ \dots, u(g_{x,l}^{(n)})}} \left\| \begin{bmatrix} \bar{A} g_{x,l}^{(1)} + \bar{B} u(g_{x,l}^{(1)}) \\ \vdots \\ \bar{A} g_{x,l}^{(n)} + \bar{B} u(g_{x,l}^{(n)}) \end{bmatrix} \right\|_1 + \hat{\gamma} \left\| \begin{bmatrix} u(g_{x,l}^{(1)}) \\ \vdots \\ u(g_{x,l}^{(n)}) \end{bmatrix} \right\|_1 \tag{3.34}$$

$$\text{s.t. } \forall k \in \{lh, \dots, (l+1)h-1\}:$$

$$C_{\bar{\mathcal{S}}} x_{ref}(t_{k+1}) + \sum_{i=1}^{n} |C_{\bar{\mathcal{S}}}\, \xi(g_{x,l}^{(i)}, u(g_{x,l}^{(i)}, \cdot), 0, t_{k+1})| \leq d_{\bar{\mathcal{S}}}, \tag{3.35}$$

$$C_{\mathcal{U}} u_{ref}(t_k) + \sum_{i=1}^{n} |C_{\mathcal{U}} u(g_{x,l}^{(i)}, t_k)| \leq d_{\mathcal{U}}, \tag{3.36}$$

with the shorthands $\hat{\gamma} := \frac{t_f}{N}\gamma$, $u(g_{x,l}^{(i)}) := [u(g_{x,l}^{(i)}, t_{lh})^T, \dots, u(g_{x,l}^{(i)}, t_{(l+1)h-1})^T]^T$, and where $g_{x,l}^{(1)}, \dots, g_{x,l}^{(n)}$ denote the generators of $\mathcal{P}_l$. This can be solved equivalently to (3.29) efficiently in a single linear program. We use the fixed inputs for the center trajectory to simplify the optimization, ensure that we actually end close to the desired final state, and to stay close to the trajectory for which we linearize the system. However, the approach would also work if we optimize the input trajectory for the center as done in (3.29). As for the linear case, we could also weight different dimensions or inputs individually by using weight matrices.

We use tightened state constraints $\bar{\mathcal{S}} = \langle C_{\bar{\mathcal{S}}}, d_{\bar{\mathcal{S}}} \rangle_H \subseteq \mathcal{S}$ in (3.35), since we only check it at discrete time instances during the optimization and since the optimization is done for linearized and discretized dynamics. As described for the convex controller, we check in the next step if the real state constraints are actually satisfied at all points in time for the real disturbed, nonlinear dynamics. If not, we tighten the state constraints more and compute the reachable set again. As for the convex controller, we do not have to tighten the input constraints, since the inputs which we compute in (3.34) are directly applied to the actual nonlinear system and are unaffected by disturbances.

Step 3: We then compute the reachable set $\mathcal{R}_{[0,t_h],u_{gen},\mathcal{W}}(\mathcal{X}_{reach,l})$ (Alg. 3, line 8) for the original nonlinear system with disturbances if controlled with the previously computed control law $u_{gen}(\,\cdot\,)$ (see Fig. 3.9(c) and Alg. 3, line 7). This reachable set is used to check the satisfaction of the real state constraints according to Lemma 1 for all times. We use the reachable set at the end of the time interval as the initial set of the next iteration step (Alg. 3,

line 9)

$$\mathcal{X}_{reach,l+1} := \mathcal{R}_{t_h,u_{gen},\mathcal{W}}(\mathcal{X}_{reach,l}),$$

and continue with Step 2.

By iterating Steps 2 and 3 for all M time steps, we obtain with $\mathcal{X}_f = \mathcal{X}_{reach,M}$ the final reachable set of all states starting in the initial set $\mathcal{X}_0$ despite disturbances (see Fig. 3.9(d)). For the online use of our control law, we store the generators of the parallelotope over-approximations $\mathcal{P}_l$ and the input zonotopes at sampling times in a look-up table. We can then use the controller with the corresponding reachable set as a motion primitive in a maneuver automaton (Alg. 3, line 11).

Online Use of the Controller In order to apply the controller online, we have to compute the corresponding control input for a given state. If the initial set is a parallelotope, then the control input is unique and we can use the same technique as in Sec. 3.3.4: Given a state $x(t_{(lh)} \in \mathcal{P}_l = \langle c_{x,l}, G_{x,l} \rangle$, any state can be uniquely expressed by the parameter vector $\alpha(x(t_{lh}))$ as

$$\begin{aligned} x(t_{lh}) &= c_{x,l} + G_{x,l}\alpha(x(t_{lh})) \\ \Rightarrow \alpha(x(t_{lh})) &= G_{x,l}^{-1}(x(t_{lh}) - c_{x,l}). \end{aligned}$$

Therefore, the corresponding input for the time interval $[t_{lh}, t_{(l+1)h})$ can be computed as

$$u_{gen}(x, t_k) = u(c_{x,l}, t_k) + \sum_{i=1}^{n} \alpha_i(x(t_{lh}))u(g_{x,l}^{(i)}, t_k) \tag{3.37}$$

for all $k \in \{lh, \ldots, (l+1)h - 1\}$. Since our controller is open-loop during each iteration step $[t_{lh}, t_{(l+1)h})$, we compute h piecewise-constant control inputs with one measurement $x(t_{lh})$, as can be seen in (3.37). Note that all the matrices and their inverses can be computed offline, as they do not depend on the current state $x(t_{lh})$. The inverse $G_{x,l}^{-1}$ always exists since $G_{x,l}$ has full rank, as $\mathcal{P}_l$ is a parallelotope.

The implementation of the reachable set computation can be done the same way as for the convex interpolation controller as described in Sec. 3.3.6 by introducing extra states which store the values of the states at the beginning of the time interval. It is also possible to directly compute the $\alpha_i(x(t_{lh}))$ and store them as the additional states, which simplifies computation.

We omit the theorem for the nonlinear case, as it follows from the same arguments as for the convex controller. The use of the reachable set computation during the controller synthesis ensures that if the over-approximation of the reachable set satisfies the actual state constraints, then every possible trajectory satisfies the state constraints as well. In addition, since the applied inputs only depend on the parallelotopes $\mathcal{P}_l$ at the beginning of each iteration, by including the input constraints (3.36) in the optimization problem, a feasible solution ensures input constraint satisfaction even for the disturbed, nonlinear dynamics. Note that the same extensions as in Sec. 3.3.5 are also possible and useful for the generator interpolation controller.

3.4.3 Numerical Example

Let us now demonstrate the generator interpolation controller for an autonomous vehicle example. We revisit the same example as in Sec. 3.3.7 for the convex interpolation controller. We use the same model, with the same constraints, and the same motion primitives *turn left*, *drive straight*, and *turn right*. We divide the one second time horizon of each motion primitive into $N = 40$ time steps and consider $M = 10$ time intervals with a horizon of $h = 4$ time steps each. We weight the states with a matrix $Q = \mathrm{diag}([13.5, 15, 17.25, 27])$.

Results

We implement the generator interpolation controller again in MATLAB, where we use CVX [94] with the solver SDPT3 to specify and solve the linear program. We use the ACADO toolbox again to compute the reference trajectory and the CORA toolbox for the reachability analysis. The computations are performed on the same computer as in Sec. 3.3.7. The offline computation of the controller for each motion primitive again takes around ten seconds without using parallel computations. The same computation time as that of the convex interpolation controller is reflected in the similar ratio of reachable set computation and optimization: with 43% for the reachable set computation and around 37% for the computation of the generator inputs. Since the online controller has the same form as the linear-approximated convex interpolation controller from Sec. 3.3.4, the online computation consists only of multiplying precomputed matrices with vectors, which can be performed very fast in around $0.01\,ms$.

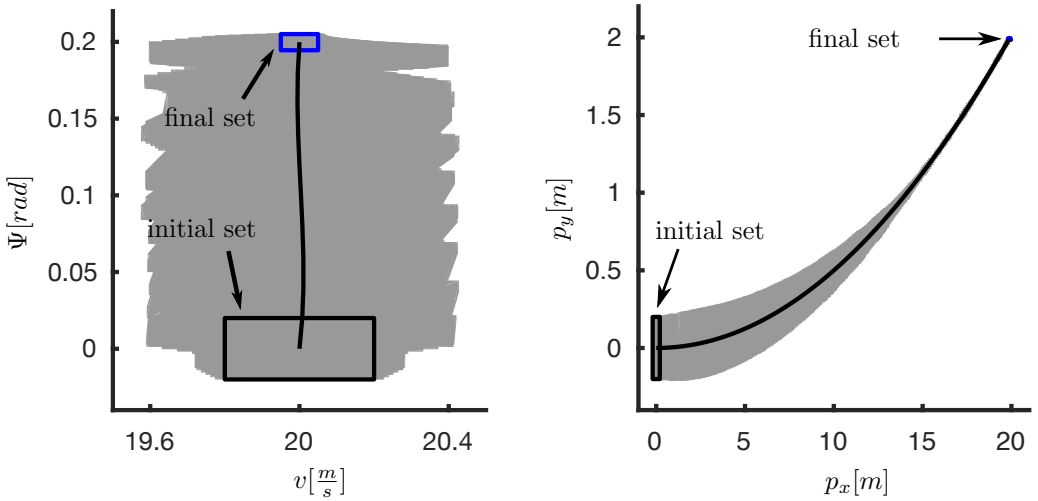


Figure 3.10: Resulting reachable set for the *turn left* motion primitive with the generator interpolation controller, projected onto the (v, Ψ) and the (p_x, p_y) planes. The initial set is plotted in black, the final set in blue, and the reachable set for all times between in gray. The black line shows the reference trajectory $x_{ref}(\cdot)$.

We show the resulting reachable sets in Fig. 3.10. In addition, we show the initial set (black)

and the final reachable set (blue) in Fig. 3.11, where we shift the final sets by the desired final states $x^{(f)}$ to have a better comparison. We see that the (shifted) reachable sets are completely contained inside the initial set for all motion primitives. Therefore, we are again able to connect all motion primitives with each other and obtain a fully connected maneuver automaton, see Fig. 3.6.

Comparison with LQR Tracking Controller

For comparison, we use again an LQR tracking controller as described in Sec. 3.3.7. First, we simply use the same state weights as for our generator interpolation controller and weight the inputs with the identity matrix. The resulting reachable set is plotted in Fig. 3.11 (red, solid), and it is not completely inside the initial set. Moreover, it uses inputs of $b = 1.53\,\frac{rad}{s}$, which is more than 3.8 times the allowed inputs. If we increase the input weights until the input constraints are satisfied, we obtain the reachable set which is plotted with a red, dashed line in Fig. 3.11, and which is much larger than the initial set. As seen before, we cannot achieve both a small final set and input satisfaction with LQR controllers. By minimizing the reachable set while ensuring the constraint satisfaction, our generator interpolation controller is able to provide better solutions which satisfy the constraints.

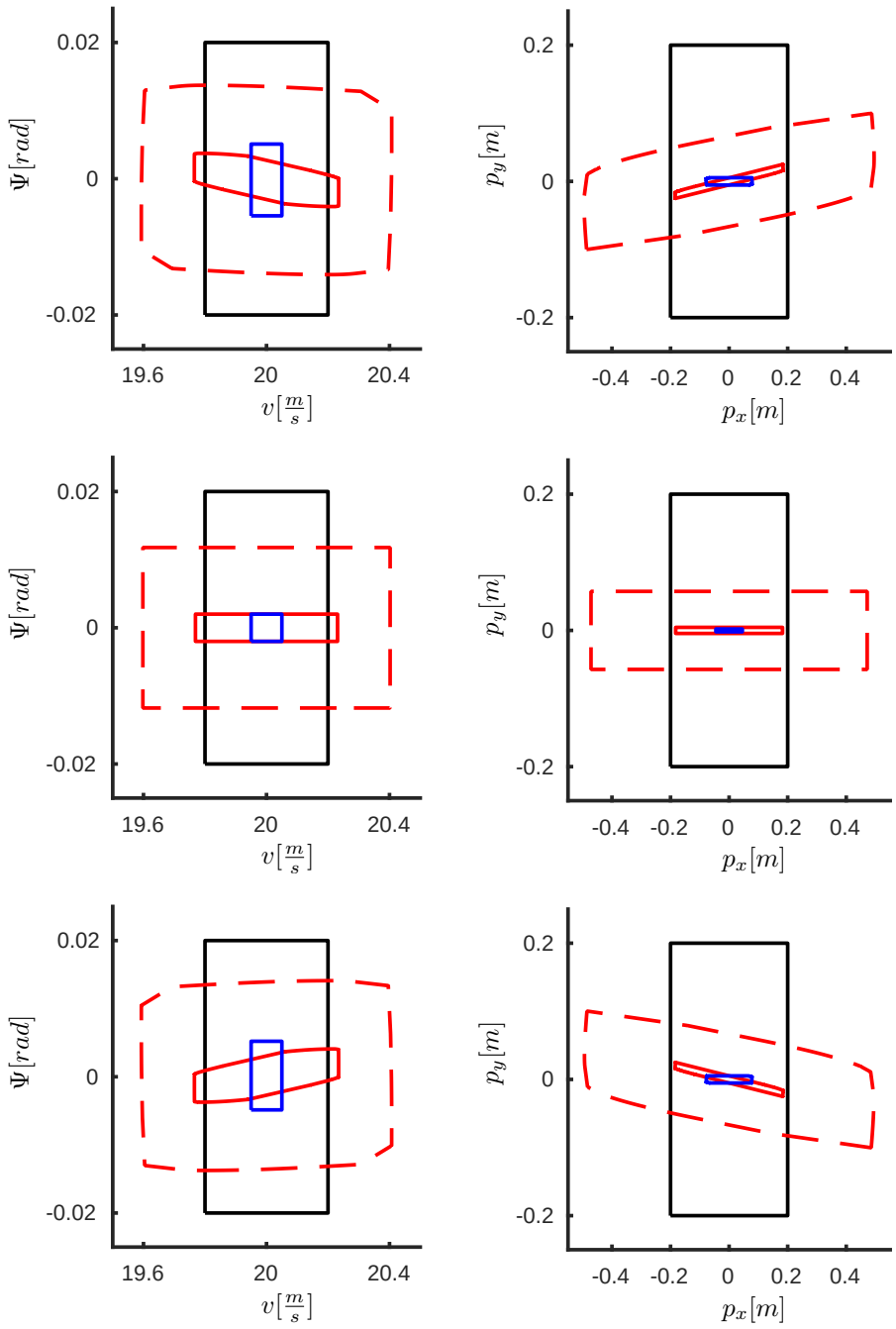


Figure 3.11: Initial (black) and shifted final sets (blue) of the generator interpolation controller, projected onto the (v, Ψ) and the (p_x, p_y) planes, for the *turn left* (top), *drive straight* (center), and *turn right* (bottom) motion primitives. For comparison, the final sets of two LQR controllers (red).

3.5 Continuous Feedback Controller

In the previous two sections, we presented two control approaches in which we interpolate optimal open-loop input trajectories to obtain control laws for a set of possible initial states. For both algorithms, we take advantage of the fact that there exist efficient tools to compute open-loop input trajectories for single initial states which can take state and input constraints into account. By iteratively applying the interpolating control law and adapting the interpolation based on the most recent measurement, we obtain a feedback loop. However, since the feedback happens only at discrete times, the controller is open-loop in between these times and therefore cannot counteract disturbances as quickly as a continuous feedback can.

An alternative way to solve reach-avoid problems that is often used in practice is by combining an optimized open-loop reference trajectory with tracking control, which should drive the error between the actual state and the desired state along the reference trajectory to zero [95]. A common controller type is LQR, which we also use for comparison in the numerical examples of the interpolation controllers. The LQR algorithm computes a feedback matrix K, which minimizes a quadratic cost function on states and inputs for an undisturbed linear system. While the feedback controllers are able to control all states starting in an initial set and can also react to disturbances, as seen in the numerical examples in Sec. 3.3.7 and Sec. 3.4.3, they are not able to take state or input constraints into account.

In this section[1], we present a way to overcome the previously described problems by computing a continuous feedback controller through direct optimization over reachable sets. Our method optimizes the reference trajectory and feedback matrices such that they take state and input constraints into account, despite disturbances, and directly minimize the reachable set at the final time. Due to the continuous feedback, the controller can react to disturbances at any time.

3.5.1 Closed-Loop Controller Optimization

We now illustrate the main idea of our closed-loop controller optimization in comparison to classical open-loop optimal control in Fig. 3.12: In Fig. 3.12(a), we show a classic optimal control approach, in which an optimal input trajectory $u(\cdot)$ for a single initial state $x^{(0)}$ is computed. This input trajectory minimizes the difference between the final state of the trajectory $\xi(x^{(0)}, u(\cdot), 0, t_f)$ and the desired final state $x^{(f)}$. Our new approach is shown in Fig. 3.12(b), where we consider a whole set of initial states for which we compute the tracking controller

$$u_{ctrl}(x(t), t) = u_{ref}(t) + K(t)(x(t) - x_{ref}(t)), \tag{3.38}$$

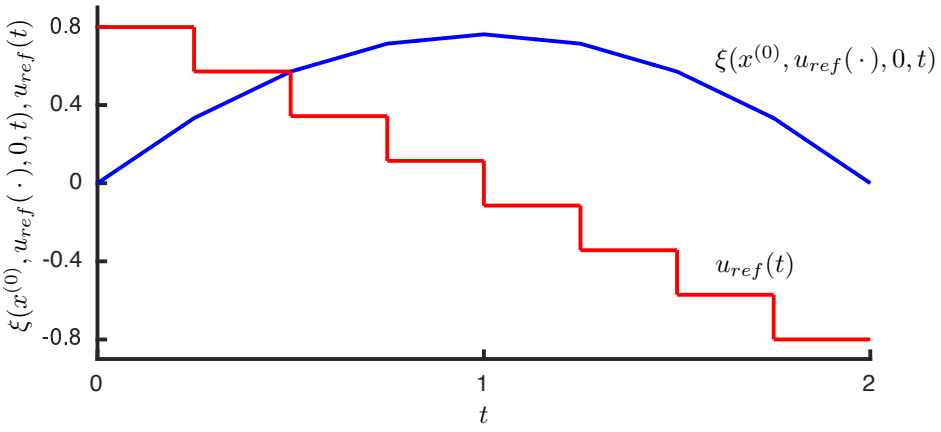
with

$$x_{ref}(t) = \xi(x_{ref}(0), u_{ref}(\cdot), 0, t). \tag{3.39}$$

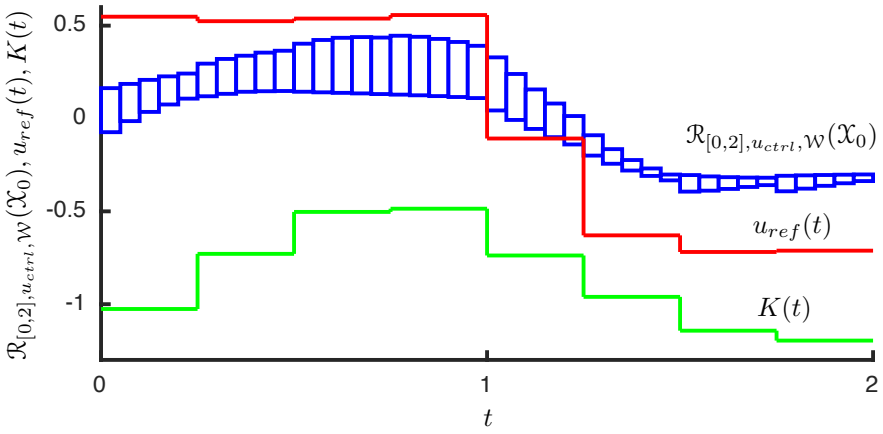
We choose a linear control structure consisting of an open-loop reference input $u_{ref}(t)$ and a time-varying feedback matrix $K(t)$ to simplify the reachability analysis as well as to obtain a

[1] This section and most of its figures are based on [217] © 2017 IEEE.

fast-to-evaluate control law. By solving the optimal control problem for the entire reachable set, we optimize the inputs for all initial states. As discussed later, the idea is also applicable for other controller types.



(a) Classical, open-loop optimization



(b) Novel, set-based closed-loop optimization

Figure 3.12: Comparison trajectory vs. set-based optimization: (a) While classical optimal open-loop control optimizes a single input trajectory for a single initial state, (b) our continuous feedback approach optimizes a reference input and a feedback matrix to minimize the set of solutions for a whole set of initial states. For a better visualization of the approach, the plots do not show the same optimization problem.

For a simpler demonstration and notation, we restrict our consideration to piecewise-constant control inputs and feedback matrices. This also makes the computations easier, similar to the numerical solution of direct multiple shooting. We choose them such that we minimize the reachable set while satisfying state and input constraints. At the beginning, we have a rather large initial set, so we cannot choose the controller to be too aggressive, as this would lead to inputs that are too large. After the reachable set becomes smaller, however, we can make the

controller more aggressive such that we obtain a faster convergence to the reference trajectory.

Therefore, we divide the time horizon in N parts, where we compute a new controller in each part. Since the computational effort depends on the number of optimization variables, we do not optimize all entries of $K(t)$. Instead, we optimize the weighting matrices of LQR controllers, where we restrict the matrices Q_k and R_k to be diagonal matrices for computational efficiency. Since we can further normalize the matrices by choosing the first entry of each Q_k to be equal to 1, we only have to consider $N(n+m-1)$ optimization variables for the feedback matrices, where N denotes the number of different weighting matrices, n the number of states, and m the number of inputs. In addition, we also have Nm optimization variables for the reference trajectory. If desired, we can split the computation of the reference trajectory and the computation of the feedback controller in two problems, similar to the previous approaches. We thereby reduce the overall computational complexity. Clearly, we could also choose a different number of reference inputs or feedback matrices by simply keeping, for example, $K(t)$ constant over a fixed number of reference input values. This can be useful if more reference inputs allow to better reach the desired final state without increasing the optimization variables for the feedback matrices.

To compute the LQR controllers, we linearize the system along the reference trajectory and compute the LQR controller based on the Q_k and R_k matrices. If desired, we can also compute multiple $K(t)$ matrices during each step with the same Q_k and R_k. This can be done by dividing each of the N steps into smaller steps and computing a new LQR controller after each smaller step using the new linearized dynamics at this point. This does not increase the number of optimization variables, and by extension, the complexity of the optimization problem. However, by recomputing the feedback matrices based on the changing linearized dynamics, we adapt the controller to the actual nonlinear dynamics directly while still preserving the simple control structure of a piecewise-linear control law.

The nonlinear optimization problem for the controller $u_{ctrl}(\,\cdot\,)$ (see (3.38)), which depends on the reference trajectory $u_{ref}(\,\cdot\,)$, and the Q_k, R_k matrices, is given by

$$\begin{aligned} \min_{u_{ref}(\,\cdot\,),Q_k,R_k} \; & \|\mathcal{R}_{t_f,u_{ctrl},\mathcal{W}}(\mathcal{X}_0) - x^{(f)}\|_1 + \gamma \int_0^{t_f} \|u_{ctrl}(\mathcal{R}_{t,u_{ctrl},\mathcal{W}}(\mathcal{X}_0),t)\|_1 \mathrm{d}t \\ \text{s.t. } \forall t \in [0,t_f]: \; & \mathcal{R}_{t,u_{ctrl},\mathcal{W}}(\mathcal{X}_0) \subseteq \mathcal{S}, \\ & u_{ctrl}(\mathcal{R}_{t,u_{ctrl},\mathcal{W}}(\mathcal{X}_0),t) \subseteq \mathcal{U}. \end{aligned} \tag{3.40}$$

By incorporating reachability analysis inside the optimization problem, we are able to directly optimize over all possible trajectories in a single nonlinear optimization problem. Therefore, the cost function and the constraint function directly depend on the reachable sets, meaning that we compute the reachable set in every iteration of the optimization algorithm. This is illustrated in Fig. 3.13.

To use efficient nonlinear programming algorithms, we have to express our optimization in the standard form of a cost function which returns a scalar value subject to minimization and a constraint function which returns a vector of values which are all negative if the constraints are satisfied. Let us describe in the following, how we can implement the various parts of the optimization problem and further improve them for repeated evaluation.

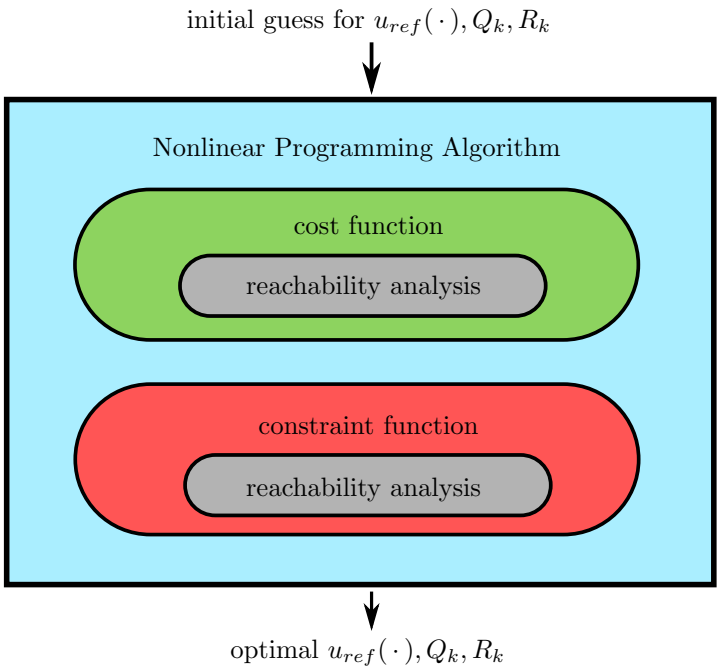


Figure 3.13: Illustration of the approach: reachability analysis is included in the cost and constraint functions of the nonlinear program. By optimizing over the reachable sets in a nonlinear program, we obtain the optimal reference trajectory $u_{ref}(\cdot)$ and controller parameters Q_k, R_k.

3.5.2 Implementation

Similar to the explanations in Sec. 3.3.6, we need to express the closed-loop dynamics in closed-form to allow the use of standard reachable set computation algorithms (see Sec. 2.6). In contrast to the interpolation-based controllers, we do not have to store the last measurement, as we have a continuous feedback based on the current state. To evaluate $K(t)(x_{ref}(t) - x(t))$, we need to know the value of $x_{ref}(t)$ at each point in time. Therefore, we include it in the extended state $x_e(t) := \begin{bmatrix} x(t) \\ x_{ref}(t) \end{bmatrix}$ which has the dynamics

$$\dot{x}_e(t) = \begin{bmatrix} \dot{x}(t) \\ \dot{x}_{ref}(t) \end{bmatrix} = \begin{bmatrix} f\left(x(t), u_{ref}(t) + K(t)\left(x(t) - x_{ref}(t)\right), w(t)\right) \\ f(x_{ref}(t), u_{ref}(t), 0) \end{bmatrix}, \tag{3.41}$$

where the reference trajectory has the same dynamics as the actual system, however, without disturbances.

We compute the reachable set starting from $\mathcal{X}_0 = \langle c_{x,0}, g_{x,0}^{(1)}, \ldots, g_{x,0}^{(p_0)} \rangle$ by initializing the extended dynamics with $x(0) \in \mathcal{X}_0$ and $x_{ref}(0) = c_{x,0}$, as there is no uncertainty for the initial state of the reference trajectory. This leads to an initial zonotope for the extended dynamics which looks like

$$\mathcal{X}_{e,0} = \left\langle \begin{bmatrix} c_{x,0} \\ c_{x,0} \end{bmatrix}, \begin{bmatrix} g_{x,0}^{(1)} \\ \mathbf{0} \end{bmatrix}, \ldots, \begin{bmatrix} g_{x,0}^{(p_0)} \\ \mathbf{0} \end{bmatrix} \right\rangle.$$

From the reachable set computation based on the dynamics (3.41), we then obtain an extended reachable set $\mathcal{R}^e_{[0,t_f],u_{ctrl},\mathcal{W}}(\mathcal{X}_{e,0})$. We compute the reachable set over small time intervals $[\tau_l, \tau_{l+1}] \subset [t_k, t_{k+1}]$, i.e., the reachability time step size $\Delta\tau$ is a fraction of the time step size Δt for which we have constant reference inputs and feedback matrices (see Fig. 3.14). The extended reachable set for each time interval is then described by a zonotope of the form

$$\mathcal{R}^e_{[\tau_l,\tau_{l+1}],u_{ctrl},\mathcal{W}}(\mathcal{X}_{e,0}) = \left\langle \begin{bmatrix} c_{x,l} \\ c_{ref,l} \end{bmatrix}, \begin{bmatrix} g^{(1)}_{x,l} \\ g^{(1)}_{ref,l} \end{bmatrix}, \ldots, \begin{bmatrix} g^{(p_l)}_{x,l} \\ g^{(p_l)}_{ref,l} \end{bmatrix} \right\rangle,$$

where the number of generators are time dependent due to the addition of new generators to consider disturbance effects and linearization errors, as discussed at the end of Sec. 2.6.

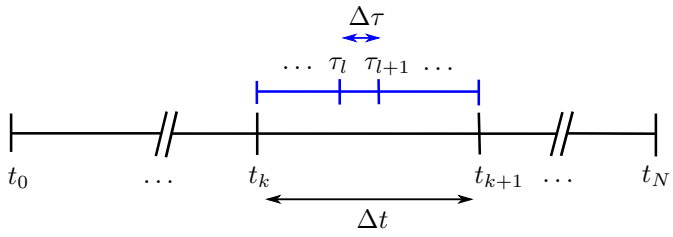


Figure 3.14: Illustration of the two time scales: Δt for the control inputs and $\Delta\tau$ for the reachability analysis.

From the coupled dynamics of the extended system, we know that if we choose an $\alpha \in [-1,1]^{p_l}$, we obtain a combination of

$$\begin{bmatrix} x(t) \\ x_{ref}(t) \end{bmatrix} = \begin{bmatrix} c_{x,l} \\ c_{ref,l} \end{bmatrix} + \sum_{i=1}^{p_l} \alpha_i \begin{bmatrix} g^{(i)}_{x,l} \\ g^{(i)}_{ref,l} \end{bmatrix},$$

with $t \in [\tau_l, \tau_{l+1}]$. While we do not have the information for which exact t this combination holds, we have the coupling of the actual state and the state of the reference trajectory preserved. This allows us to compute the input of this state as

$$\begin{aligned} u_{ctrl}(x(t),t) &= u_{ref}(t) + K(t)(x(t) - x_{ref}(t)) \\ &= u_{ref}(t) + K(t)\left((c_{x,l} - c_{ref,l}) + \sum_{i=1}^{p_l} \alpha_i (g^{(i)}_{x,l} - g^{(i)}_{ref,l})\right). \end{aligned}$$

We choose the interval $[\tau_l, \tau_{l+1}]$ such that $u_{ref}(t)$ and $K(t)$ are constant $\forall t \in [\tau_l, \tau_{l+1}]$, so we do not have to know the exact t for $u_{ref}(t)$ and $K(t)$ either. The zonotopic set representation in combination with the linear control laws allows us to express the set of possible inputs in $[\tau_l, \tau_{l+1}]$ as a zonotope of the following form

$$\begin{aligned} u_{ctrl}(\mathcal{R}_{[\tau_l,\tau_{l+1}],u_{ctrl},\mathcal{W}}(\mathcal{X}_0),t) = \Big\langle & u_{ref}(t) + K(t)(c_{x,l} - c_{ref,l}), K(t)(g^{(1)}_{x,l} - g^{(1)}_{ref,l}), \\ & \ldots, K(t)(g^{(p_l)}_{x,l} - g^{(p_l)}_{ref,l}) \Big\rangle. \end{aligned} \tag{3.42}$$

This zonotope is then used in the cost and constraint function as explained in the following.

3.5.3 Cost Function

To evaluate the first part of the cost function (3.40), $\|\mathcal{R}_{t_f,u_{ctrl},\mathcal{W}}(\mathcal{X}_0) - x^{(f)}\|_1$, we need to find the $x \in \mathcal{R}_{t_f,u_{ctrl},\mathcal{W}}(\mathcal{X}_0)$ which maximizes the 1-norm. We can do this by simply solving a single linear program. If this is not desired, as the repeated evaluation of linear programs in the cost function takes time and might make the problem numerically harder, we can instead compute an upper bound analogous to Lemma 2 as

$$\|\mathcal{R}_{t_f,u_{ctrl},\mathcal{W}}(\mathcal{X}_0) - x^{(f)}\|_1 \leq \|c_{x,f} - x^{(f)}\|_1 + \sum_{i=1}^{p_f} \|g_{x,f}^{(i)}\|_1,$$

with $c_{x,f}$ and $g_{x,f}^{(1)}, \ldots, g_{x,f}^{(p_f)}$ denoting the center and the generators of $\mathcal{R}_{t_f,u_{ctrl},\mathcal{W}}(\mathcal{X}_0)$.

As described in the previous subsection, obtaining the maximum applied input at each point in time is not feasible. Instead, we use (3.42) to obtain the set of applied inputs for each time interval. Since $u_{ctrl}(\mathcal{R}_{[\tau_l,\tau_{l+1}],u_{ctrl},\mathcal{W}}(\mathcal{X}_0), t)$ is a zonotope, we obtain the maximum 1-norm of its states through the same methods used for the state costs, i.e., by either solving a linear program for each interval or by obtaining an upper bound by using Lemma 2. Using the latter, we obtain

$$\begin{aligned}
&\gamma \int_0^{t_f} \|u_{ctrl}(\mathcal{R}_{t,u_{ctrl},\mathcal{W}}(\mathcal{X}_0), t)\|_1 \mathrm{d}t \leq \gamma \Delta\tau \sum_{l=0}^{M-1} \|u_{ctrl}(\mathcal{R}_{[\tau_l,\tau_{l+1}],u_{ctrl},\mathcal{W}}(\mathcal{X}_0), \tau_l)\|_1 dt \\
&= \gamma \Delta\tau \sum_{l=0}^{M-1} \Big(\|u_{ref}(t_l) + K(t_l)(c_{x,l} - c_{ref,l})\|_1 + \sum_{i=1}^{p_l} \|K(t_l)(g_{x,l}^{(i)} - g_{ref,l}^{(i)})\|_1 \Big),
\end{aligned}$$

with $M = \frac{t_f}{\Delta\tau}$ denoting the number of intervals for the reachability analysis. The inequality results from the fact that we consider the maximum input value for the whole interval.

3.5.4 Constraint Function

Checking the state constraints (3.2) for all times can be done easily, as shown in Lemma 1. The reachability analysis algorithm returns over-approximations for the reachable sets of time intervals $\mathcal{R}_{[\tau_l,\tau_{l+1}],u_{ctrl},\mathcal{W}}(\mathcal{X}_0)$. By checking if $\mathcal{R}_{[\tau_l,\tau_{l+1}],u_{ctrl},\mathcal{W}}(\mathcal{X}_0) \in \mathcal{S}$, $\forall l \in \{0, M-1\}$, using the formula from Lemma 1, we ensure the satisfaction of the state constraints at all times. Because we obtain zonotope over-approximations for the inputs of time intervals with (3.42), we can use the same technique for inputs as well.

Since we already include reachability analysis and therefore the formal check of constraints during the optimization, we know that if a feasible solution is found, then all constraints are satisfied. As a result, we do not need an additional verification step afterwards. If the reachable set has to end in a given terminal set, we can also directly include this as a terminal set constraint. In contrast to the iterative optimization of shorter time horizons for the convex and generator interpolation controllers, we optimize over the whole time horizon for the continuous feedback controller and can therefore directly consider such constraints during the optimization.

3.5.5 Extension to Other Control Laws

So far, we have considered linear feedback control laws in this section, as they are easy to implement, have proven their applicability to a wide variety of nonlinear systems in practice, and have favorable properties for the reachable set computation, as they lead to smaller linearization errors compared to nonlinear control laws. The optimization of LQR tracking controllers is just an example of how controller synthesis can be done by optimizing over reachable sets. If different types of control laws are desired, including nonlinear ones, the presented methods can be used very similarly, and we can still benefit from many of the presented simplifications. Since the applied input set (3.42) is not exactly a zonotope for nonlinear control laws, it would have to be computed in an over-approximative fashion, though it would still work.

The basic idea is the same for different control laws: We compute the reachable set in the optimization problem while using the parameters of the control law as optimization variables. By checking the (over-approximated) state and input constraints, we ensure safety at all times. This approach works as long as (i) the closed-loop system can be expressed in a (piecewise) closed-form for which we can compute reachable sets, and (ii) the control laws depend on some parameters which we can optimize.

3.5.6 Numerical Example

Since we have to perform a reachability computation in every iteration of the optimization algorithm, the reachability algorithm accounts for the majority of the overall computation time. Therefore, we present a linear example in this section, as reachability analysis for linear systems can be performed much faster than for nonlinear systems, see Sec 2.6. We choose a platooning example, where vehicles on a highway are supposed to drive in a platoon behind each other. By driving close to the vehicle in front of them, they can save fuel and reduce driving time for human drivers. However, in this scenario there are important safety constraints which must be satisfied at all times, despite the effects of external disturbances. We consider vehicle-to-vehicle communication which allows a central controller design. This example is inspired by the benchmark example proposed in [96].

We consider a platoon with four vehicles, where the dynamics of each vehicle $i \in \{1, 2, 3, 4\}$ is given by

$$\begin{aligned} \dot{p}^{(i)} &= v^{(i)}, \\ \dot{v}^{(i)} &= a^{(i)} + w^{(i)}, \end{aligned}$$

where $p^{(i)}$ denotes the position of the i-th vehicle, $v^{(i)}$ its velocity, and $a^{(i)}$ its acceleration as its input. The disturbances are denoted by $w^{(i)}$. To model the whole platoon, we use the absolute states of the first vehicle and the relative states of the second, third, and fourth vehicles, i.e.,

we consider the eight-dimensional state vector

$$x = \begin{bmatrix} p^{(1)} \\ v^{(1)} \\ p^{(1)} - p^{(2)} - c_s \\ v^{(1)} - v^{(2)} \\ p^{(2)} - p^{(3)} - c_s \\ v^{(2)} - v^{(3)} \\ p^{(3)} - p^{(4)} - c_s \\ v^{(3)} - v^{(4)} \end{bmatrix},$$

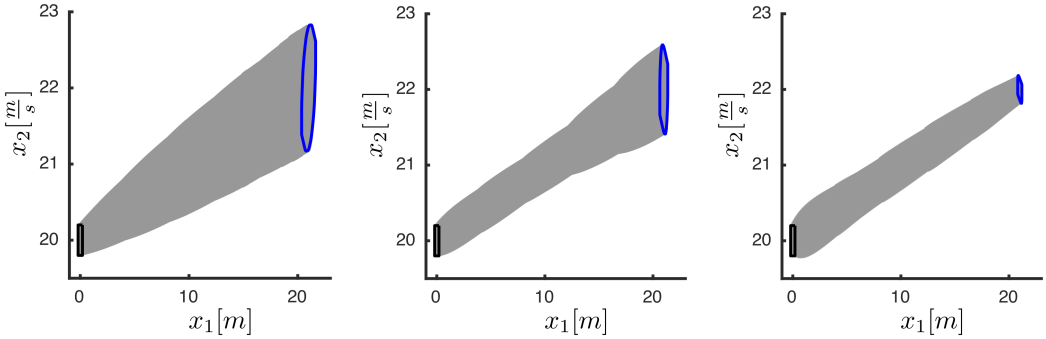
the input vector $u = [a^{(1)}, a^{(2)}, a^{(3)}, a^{(4)}]^T$, and disturbance vector $w = [w^{(1)}, w^{(2)}, w^{(3)}, w^{(4)}]^T$. Therein $c_s \in \mathbb{R}_0^+$ denotes a safety constant, defining a minimal safe distance. The resulting dynamics are given by

$$\begin{aligned} \dot{x}_1 &= x_2, & \dot{x}_2 &= u_1 + w_1, \\ \dot{x}_3 &= x_4, & \dot{x}_4 &= u_1 - u_2 + w_1 - w_2, \\ \dot{x}_5 &= x_6, & \dot{x}_6 &= u_2 - u_3 + w_2 - w_3, \\ \dot{x}_7 &= x_8, & \dot{x}_8 &= u_3 - u_4 + w_3 - w_4. \end{aligned}$$

We assume that all inputs are constrained between $u_i \in [-10, 10]\,\frac{m}{s},\ \forall i \in \{1, 2, 3, 4\}$, and all disturbances vary freely in the interval $w_i \in [-1, 1]\,\frac{m}{s},\ \forall i \in \{1, 2, 3, 4\}$. Moreover, we have the state constraints that the vehicles must keep the minimal safety distances, i.e., $x_3, x_5, x_7 \geq 0$. We consider the following scenario, which can be used as a motion primitive in a platooning maneuver automaton: The vehicles start with initial states ranging freely in the box $\mathcal{X}_0 = [-0.2, 0.2]\,m \times [19.8, 20.2]\,\frac{m}{s} \times [0.8, 1.2]\,m \times [-0.2, 0.2]\,\frac{m}{s} \times [0.8, 1.2]\,m \times [-0.2, 0.2]\,\frac{m}{s} \times [0.8, 1.2]\,m \times [-0.2, 0.2]\,\frac{m}{s}$, i.e., the vehicles drive with different velocities around $20\,\frac{m}{s}$ behind each other. We consider a final state $x^{(f)} = [21\,m, 22\,\frac{m}{s}, 1\,m, 0\,\frac{m}{s}, 1\,m, 0\,\frac{m}{s}, 1\,m, 0\,\frac{m}{s}]^T$ which should be reached after $1\,s$, i.e., the whole platoon should speed up to $22\,\frac{m}{s}$ and align in a safe distance of $c_s + 1\,m$ between each vehicle.

The reference trajectory is split in five time steps, and since the system dynamics are linear, five constant control matrices are computed. We implement the controller in MATLAB and again use the CORA toolbox for the reachability computation. For the optimization, we use MATLAB's `fmincon` function with the `active-set` algorithm. The computations are performed on the same computer as in Sec. 3.3.7. The computation of the optimized controller takes around five minutes, where the reachable set computations taking the majority of the time at 78%. The optimization algorithm terminates after it reaches the maximum number of iterations and finds a feasible solution.

We show the reachable sets at different times during the algorithm in Fig. 3.15. The reachable set changes between different iterations, as it is directly optimized on, and the size of the final set is minimized. In Fig. 3.16, we show the optimized final set in comparison to the initial set for the different dimensions. For the x_1 and x_2 coordinates, we shifted the final reachable set by the final states $x^{(f)}$ for a better comparison. We see that the shifted reachable set of our controller (blue) lies completely in the initial set (black).



(a) Initial guess **(b)** During optimization **(c)** After optimization

Figure 3.15: Illustration of the reachable sets projected onto the (x_1, x_2) plane at different times during the optimization algorithm: (a) the reachable set for the initial guess, (b) during the optimization, and (c) the final reachable set after the optimization. The initial sets are shown in black, the final sets in blue, and the reachable set for all times between in gray.

For comparison, we also compute three LQR tracking controllers using the traditional approach of a reference trajectory in combination with an LQR feedback controller as done in Sec. 3.3.7 and Sec. 3.4.3. For a fair comparison, we use the same reference trajectory as for our controller. Since we value all dimensions the same, we choose the state weighting matrix as the identity $Q = I$. We obtain the first LQR controller with an input weighting matrix $R = I$. We use this controller as an initial guess for our optimization problem; therefore the reachable set can be seen in Fig. 3.15 on the left. The final set is very large and while it satisfies the input constraints, it violates the state constraints. To get an idea of how well a LQR controller can solve the problem, we increase the state weighting matrices manually until any further increase would start to violate the input constraints. We see the resulting final set for $Q = 90I$ and $R = I$ in red, dashed in Fig. 3.16. It is much larger than with our controller and lies outside the initial set.

We can keep increasing the state weighting matrices until $Q = 700I$, for which the final reachable sets (red, solid in Fig. 3.16) lie just inside the initial set; however, this controller uses more than two times of the allowed inputs. Even if we weight the positions and velocities in the state weighting matrix differently, any resulting controller which controls all final states inside the initial set uses inputs which are much larger than allowed.

In conclusion, we are not able to find an LQR controller with constant Q and R matrices which both satisfies the constraints and lies in the final set. This example shows how using the reachability analysis inside the optimization problem leads to controllers with better performance and guaranteed constraint satisfaction, which is not possible with classical LQR controllers, even for this linear example.

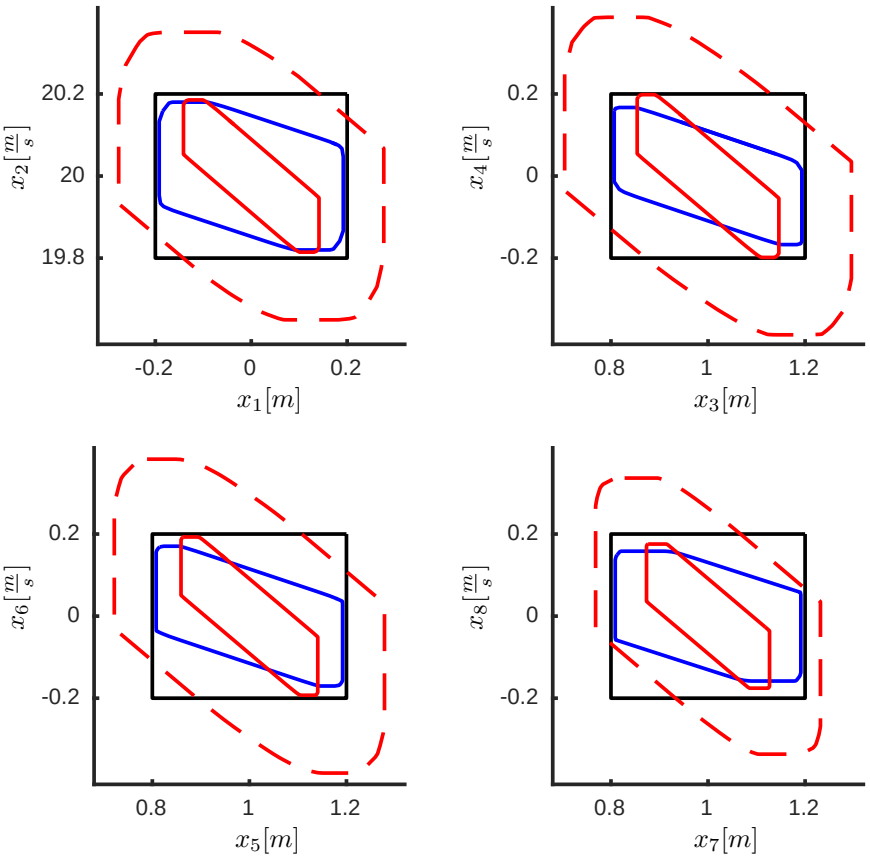


Figure 3.16: Initial (black) and shifted final sets (blue) of our optimized controller u_{ctrl}, projected onto the (x_1, x_2), the (x_3, x_4), the (x_5, x_6), and the (x_7, x_8) planes. For comparison the final sets of two LQR controllers are shown (red).

3.6 Combined Control

3.6.1 Overview

As discussed in the previous section, many classical control schemes tackle the reach-avoid problem in Sec. 3.2 with a feedforward reference trajectory and a feedback controller which tracks this reference trajectory:

$$u_{classic}(x(t), t) = u_{ff}(t) + u_{fb}(x(t), t).$$

While this works well if the actual trajectory starts close to the reference trajectory or if there are no input constraints, this approach struggles with satisfying the input constraints and providing a good performance for states further away from the reference trajectory. If one uses a controller with a high gain to counteract disturbances as fast as possible, it generates large inputs for states far away from the reference trajectory. On the other hand, if the control gain is chosen lower, then it satisfies the input constraints but degrades the ability to quickly counteract disturbances.

To deal with this problem, we used multiple weight matrices Q_k, R_k in the last section. This offers the possibility of increasing the aggressiveness of the controllers over time when the set of possible states becomes smaller. The disadvantage, on the other hand, is that we have to solve a large optimization problem with many optimization variables, as each new set of weighting matrices adds new optimization variables. This leads to many iterations of the optimization algorithm, which becomes computationally expensive as the whole reachable set has to be computed in each iteration.

In the approaches from Sec. 3.3 and Sec. 3.4, we only have to compute the reachable set a single time for the whole time horizon, and therefore the computations are much faster. However, these methods do not provide continuous feedback and therefore cannot counteract disturbances as well as the approach with continuous feedback from Sec. 3.5.

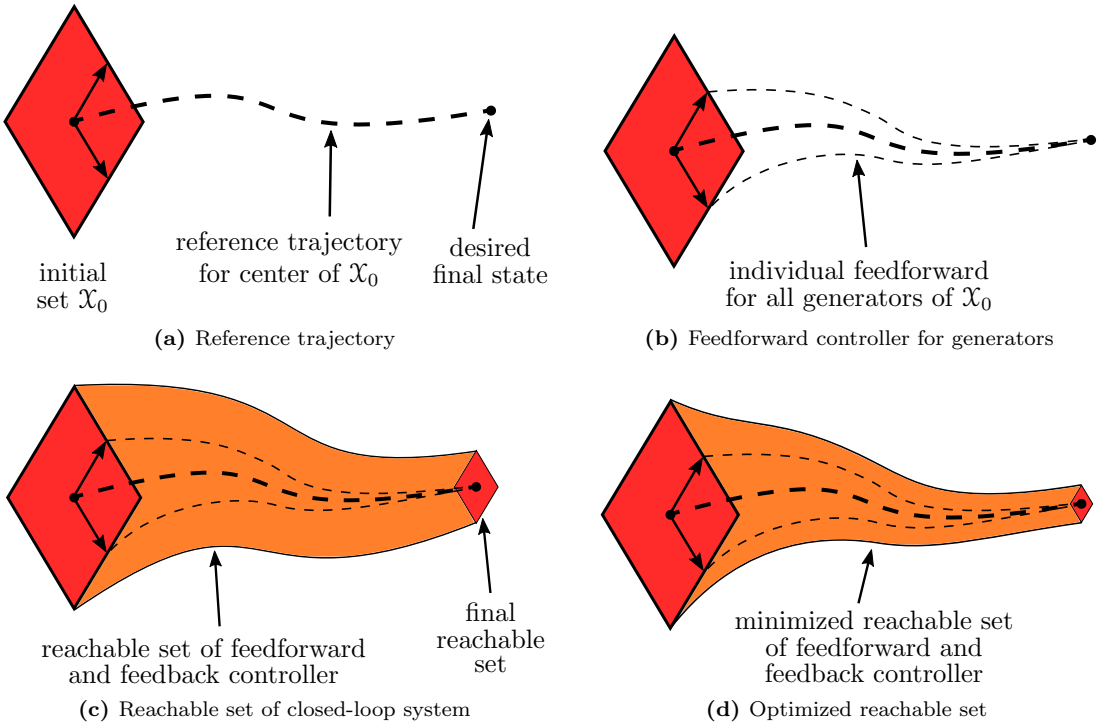
In this section[1], we combine the ideas of the generator interpolation approach with the optimized continuous feedback controller to benefit from the advantages of both approaches while reducing their disadvantages. We thus propose the following idea, which is illustrated in Fig. 3.17: Instead of the single feedforward trajectory, we introduce a state-dependent feedforward controller $u_{ff}(x^{(0)}, t)$, based on the generator interpolation controller, together with a feedback controller $u_{fb}(x(t), t)$, based on the continuous feedback controller from last section:

$$u_{ctrl}(x(t), t) = u_{ff}(x^{(0)}, t) + u_{fb}(x(t), t). \tag{3.43}$$

This allows the feedforward controller for the undisturbed system to steer all states from the initial set into a small final set. We thereby obtain a unique reference trajectory for each of the infinitely many initial states. As a result, the feedback controller only has to counteract the disturbances, allowing it to be much more aggressive without violating input constraints.

We compute the different parts — reference trajectory, feedforward controller for the generators of the initial set, and feedback controller — after each other. We still have to ensure

[1] This section, including the figures, is based on [219] © 2021 IEEE.



(a) Reference trajectory **(b)** Feedforward controller for generators

(c) Reachable set of closed-loop system **(d)** Optimized reachable set

Figure 3.17: In contrast to the interpolation-based approaches, the combined control approach optimizes over the whole time horizon and the closed-loop dynamics: (a) Computing a reference trajectory from the center of the initial set to the desired final state. (b) Computing the feedforward controller for each generator of the initial set, which steers them to the origin. (c) Counteracting disturbances with the feedback controller; minimizing the reachable set by optimizing the controller parameters. (d) Obtaining state-dependent feedforward and feedback controllers with reachable sets of minimal size.

that there are enough remaining input capacities and that the states are far enough from their boundaries to appropriately consider disturbance effects, similar to previous sections. Therefore, we have to tighten the constraints for the reference trajectory and for the feedforward controller, respectively. Let us introduce the new constraint sets $\mathcal{S}_{ref}, \mathcal{S}_{ff}$ and $\mathcal{U}_{ref}, \mathcal{U}_{ff}$ such that

$$\mathcal{S}_{ref} \subseteq \mathcal{S}_{ff} \subseteq \mathcal{S}, \tag{3.44}$$

$$\mathcal{U}_{ref} \subseteq \mathcal{U}_{ff} \subseteq \mathcal{U}. \tag{3.45}$$

In many approaches from literature, such as [22], the above sets are exclusively reserved, resulting in the distribution of input utilization illustrated in Fig. 3.18. In contrast to this static allocation of state and input sets, we introduce a flexible and therefore less conservative distribution, which is shown in Fig. 3.19. We minimize the applied inputs in an optimization problem and are able to use the unused capacities of previous steps in contrast to the previous example.

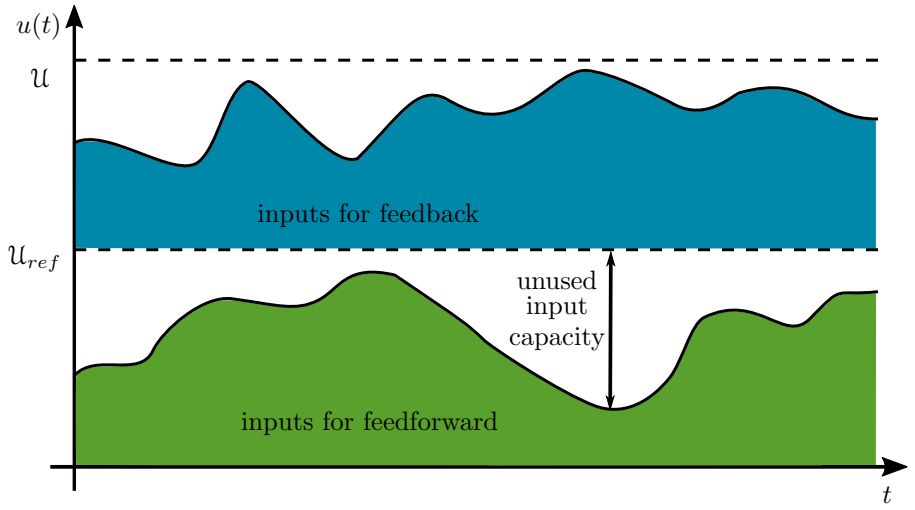


Figure 3.18: Illustration of a classical trajectory tracking approach where the reference trajectory and the feedback controller are designed independently. The controller cannot benefit from the fact that the reference trajectory does not use the full input capacities.

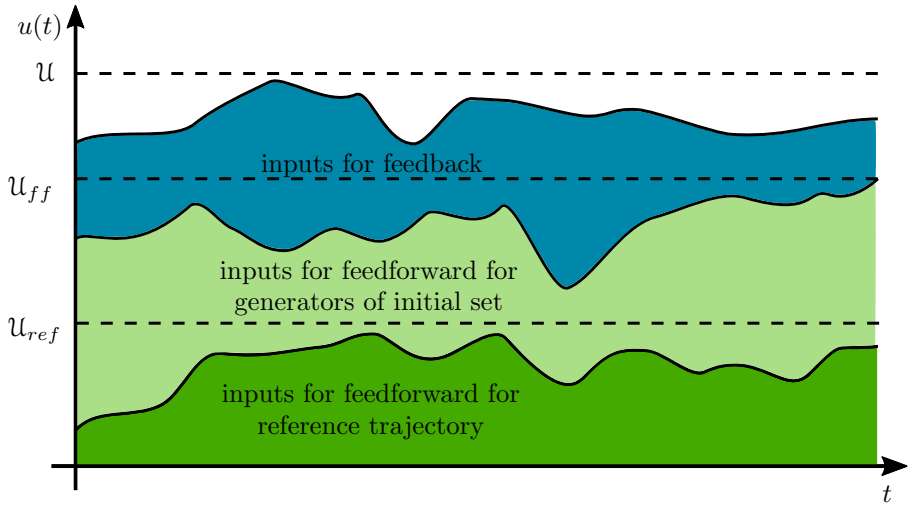


Figure 3.19: Illustration of the three different input constraints which are used in our optimization problem. While we restrict the inputs for initial optimizations, we allow the later optimizations to apply the unused parts of the input constraints of previous optimizations.

The choice of $\mathcal{S}_{ref}, \mathcal{S}_{ff}, \mathcal{U}_{ref}$, and $\mathcal{U}_{ff}$ are design variables. As the computations for the reference trajectory and the feedforward controller can be done very fast, especially compared to the optimization of the feedback controller, we could also minimize over the allowed states and inputs to find the smallest sets such that all states are controlled close enough to the desired

final state for the undisturbed system.

Let us now explain the controller parts in more detail. The resulting synthesis procedure is summarized in Alg. 4 and is explained subsequently.

Algorithm 4 Combined Controller Synthesis Algorithm

Input: system dynamics $f(x,u,w)$, initial set $\mathcal{X}_0$, desired final state $x^{(f)}$, total time t_f, number of iterations N, constraint sets $\mathcal{S},\mathcal{U},\mathcal{S}_{ref},\mathcal{U}_{ref},\mathcal{S}_{ff},\mathcal{U}_{ff}$, disturbance set $\mathcal{W}$, weighting factors $\gamma,\gamma_{ref},\gamma_{ff}$

Output: motion primitive **MP**

1: $(x_{ref},u_{ref}) \leftarrow$ solution of optimization problem (3.8) for the reference trajectory
2: $(A_k,B_k) \leftarrow$ linearization and time-discretization of $f(x,u,0)$ along $x_{ref}(\,\cdot\,),u_{ref}(\,\cdot\,)$ using (3.32) and (3.33)
3: $(u(g_{x,0}^{(1)}),\dots,u(g_{x,0}^{(p_0)})) \leftarrow$ solution of optimization problem (3.47) for feedforward control of the generators of the initial set
4: $(Q,R) \leftarrow$ solution of optimization problem (3.50) for the optimal controller parameters
5: $u_{ctrl}(x,t) \leftarrow$ combined control law (3.43) using $u(g_{x,0}^{(1)}),\dots,u(g_{x,0}^{(p_0)})$ and Q,R
6: $\mathcal{R}_{[0,t_f],u_{ctrl},\mathcal{W}}(\mathcal{X}_0) \leftarrow$ reachable set computed in optimization problem (3.50)
7: **MP** $\leftarrow \{x_{ref}(\,\cdot\,),u_{ref}(\,\cdot\,),t_f,\mathcal{X}_0,\mathcal{X}_{reach,N},u_{ctrl}(\,\cdot\,),\mathcal{R}_{[0,t_f],u_{ctrl},\mathcal{W}}(\mathcal{X}_0)\}$

3.6.2 State-Dependent Feedforward Control

We want to find an individual feedforward control law for each state of the initial set, such that it is steered as close as possible to $x^{(f)}$, i.e.,

$$\begin{aligned} u_{ff}(\,\cdot\,) = \underset{u_{ff}(\,\cdot\,)}{\arg\min}\, &\|(\mathcal{R}_{t_f,u_{ff},0}(\mathcal{X}_0) - x^{(f)})\|_1 + \gamma_{ff}\int_0^{t_f} \|u_{ff}(\mathcal{R}_{t,u_{ff},0}(\mathcal{X}_0),t)\|_1 \mathrm{d}t \\ \text{s.t. } &\forall t\in[0,t_f]: \mathcal{R}_{t,u_{ff},0}(\mathcal{X}_0)\subseteq\mathcal{S}_{ff}, \\ &u_{ff}(\mathcal{R}_{t,u_{ff},0}(\mathcal{X}_0),t)\subseteq\mathcal{U}_{ff}, \end{aligned} \tag{3.46}$$

where we minimize the difference between each state to the desired final state as well as the applied inputs weighted by $\gamma_{ff}\in\mathbb{R}_0^+$.

As discussed before, for a general, nonlinear system with a nonlinear control law, it is not feasible to solve (3.46) for every possible state of the initial set as there are uncountably many. We overcome this problem analogously to the nonlinear approach of the generator interpolation controller in Sec. 3.4.2 by restricting the feedforward control to linearized dynamics with linear control laws, which allows us to interpolate finitely many solutions using the superposition principle of linear systems. We therefore choose the state-dependent feedforward controller similar to the controller from Sec. 3.4 as

$$u_{ff}(x^{(0)},t) = u_{ref}(t) + \sum_{i=1}^{p_0}\alpha_i(x^{(0)})u(g_{x,0}^{(i)},t),$$

where $\alpha_i(x^{(0)})$ describes $x^{(0)}$ in $\mathcal{X}_0$, see (3.26).

To obtain the required input trajectories, we begin by computing the reference trajectory $x_{ref}(\,\cdot\,)$ for the center $c_{x,0}$ of the initial set $\mathcal{X}_0 = \langle c_{x,0}, g_{x,0}^{(1)}, \dots, g_{x,0}^{(p_0)} \rangle$ by solving (3.8) (Alg. 4, line 1). We then linearize and time-discretize the system along the reference trajectory using (3.32) and (3.33) (Alg. 4, line 2) and finally compute the input trajectories for the generator-interpolating feedforward controller (Alg. 4, line 3) by solving

$$\min_{\substack{u(g_{x,0}^{(1)}),\\ \dots, u(g_{x,0}^{(p_0)})}} \left\| \begin{bmatrix} \bar{A} g_{x,0}^{(1)} + \bar{B} u(g_{x,0}^{(1)}) \\ \vdots \\ \bar{A} g_{x,0}^{(p_0)} + \bar{B} u(g_{x,0}^{(p_0)}) \end{bmatrix} \right\|_1 + \hat{\gamma}_{ff} \left\| \begin{bmatrix} u(g_{x,0}^{(1)}) \\ \vdots \\ u(g_{x,0}^{(p_0)}) \end{bmatrix} \right\|_1 \tag{3.47}$$

$$\text{s.t. } \forall k \in \{0, \dots, N-1\} :$$

$$C_{\mathcal{S}_{ff}} x_{ref}(t_{k+1}) + \sum_{i=1}^{p_0} |C_{\mathcal{S}_{ff}} g^{(i)}(t_{k+1})| \leq d_{\mathcal{S}_{ff}}, \tag{3.48}$$

$$C_{\mathcal{U}_{ff}} u_{ref}(t_k) + \sum_{i=1}^{p_0} |C_{\mathcal{U}_{ff}} u(g_{x,0}^{(i)}, t_k)| \leq d_{\mathcal{U}_{ff}}, \tag{3.49}$$

with the shorthands $\hat{\gamma}_{ff} = \frac{t_f}{N}\gamma_{ff}$, $\bar{B} = \left[\bar{B}_0, \cdots, \bar{B}_{N-1}\right]$, and $u(g_{x,0}^{(i)}) = \left[u(g_{x,0}^{(i)}, t_0)^T, \dots, u(g_{x,0}^{(i)}, t_{N-1})^T\right]^T$. Here, (3.48) and (3.49) ensure the satisfaction of the state and input constraints $\mathcal{S}_{ff} = \{x \in \mathbb{R}^n \mid C_{\mathcal{S}_{ff}} x \leq d_{\mathcal{S}_{ff}}\}$ and $\mathcal{U}_{ff} = \{u \in \mathbb{R}^m \mid C_{\mathcal{U}_{ff}} u \leq d_{\mathcal{U}_{ff}}\}$, respectively.

This is all equivalent to the nonlinear case in Sec. 3.4, the sole exception that it is performed only once for the entire time horizon. In Sec. 3.4, we have to iteratively solve optimization problem (3.34) for short time horizons and compute the reachable set to obtain a feedback which counteracts disturbances and linearization errors. With the combined control approach, we use the feedback controller presented next to compensate disturbances and linearization errors. This allows us to only consider undisturbed dynamics for computing the feedforward controller here and therefore to directly optimize the inputs for the entire time horizon.

3.6.3 Feedback Control

We use the same controller for feedback control that we did in Sec. 3.5:

$$u_{fb}(x(t), t) = K(t)(x(t) - x_{ff}(t)),$$

with $x_{ff}(t) = \xi\big(x^{(0)}, u_{ff}(x^{(0)}, \cdot), 0, t\big)$. In contrast to Sec. 3.5, however, here the controller does not have to shrink the size of the reachable set; it merely counteracts the effects from linearization errors and disturbances (see Fig. 3.17(c)–(d)). As a result, we do not need to compute several different weighting matrices Q_k and R_k, only a single pair Q and R. This drastically reduces the number of optimization variables from $N(n+m-1)$ to $n+m-1$ for the same assumptions as in Sec. 3.5, and therefore the computation load for solving the corresponding optimization problem. Note that while we only have a single pair of weight matrices, we again obtain time-dependent feedback matrices $K(t)$, as we still linearize the system along the reference trajectory $x_{ref}(\,\cdot\,)$ to capture the changing linearized dynamics of the nonlinear system. The reachability analysis is also still performed on the actual nonlinear dynamics.

The nonlinear optimization problem (Alg. 4, line 4) for the Q and R matrices is therefore given by

$$\begin{aligned} &\min_{Q,R} \|\mathcal{R}_{t_f,u_{ctrl},\mathcal{W}}(\mathcal{X}_0) - x^{(f)}\|_1 + \gamma \int_0^{t_f} \|u_{ctrl}(\mathcal{R}_{t,u_{ctrl},\mathcal{W}}(\mathcal{X}_0),t)\|_1 \mathrm{d}t \\ &\text{s.t. } \forall t \in [0,t_f] : \mathcal{R}_{t,u_{ctrl},\mathcal{W}}(\mathcal{X}_0) \subseteq \mathcal{S}, \\ &\qquad\qquad u_{ff}(\mathcal{R}_{t,u_{ctrl},\mathcal{W}}(\mathcal{X}_0),t) \subseteq \mathcal{U}. \end{aligned} \tag{3.50}$$

Analogous to problem (3.40), we can use Lemma 2 to compute an upper bound for the cost function and use Lemma 1 and Corollary 1 to check the constraints. Therefore, we can solve this optimization problem with the same tools as in Sec. 3.5, though with fewer optimization variables. Note that the computations of the combined control law (Alg. 4, line 5) and reachable set (Alg. 4, line 6) are actually already performed during the reachable set computation as part of the optimization problem (3.50). Therefore, this is no extra effort. Analogous to the previous sections, we can store the control law with its corresponding reachable set as a motion primitive to be used in a maneuver automaton (Alg. 4, line 7).

Next, we present the implementation of reachability analysis with an extended state space to obtain the set of applied inputs. Analogous to the discussion in Sec. 3.5.5, the presented approach can also be extended for general control laws. Due to the previously mentioned advantages of linear control laws, we again present the approach for this case.

3.6.4 Implementation

As done in Sec. 3.5.2, we begin by expressing our closed-loop dynamics in closed-form. Like in (3.41), we again use additional states to keep track of the reference trajectory of the center, the state-dependent feedforward trajectories, and the inputs from the state-dependent feedforward controllers. Since we linearize dynamics for the feedforward controller, we are able to use the superposition principle and the zonotope notation to store this information efficiently. We therefore define the extended state as

$$x_e(t) := \begin{bmatrix} x(t) \\ x_{ref}(t) \\ \tilde{x}_{ff}(t) \\ \alpha(x^{(0)}) \end{bmatrix},$$

where the state $\tilde{x}_{ff}(t)$ denotes the deviation of the feedforward trajectory with respect to the reference trajectory, i.e.,

$$\tilde{x}_{ff}(t) := x_{ff}(t) - x_{ref}(t).$$

Equivalently, we define

$$\tilde{u}_{ff}(\alpha(x^{(0)}),t) := u_{ff}(\alpha(x^{(0)}),t) - u_{ref}(t),$$

where $\alpha_i(x^{(0)})$ is such that (3.26) holds. Using the combined control law

$$\begin{aligned} u_{ctrl}(x(t),t) &= u_{ff}(\alpha(x^{(0)}),t) + u_{fb}(x(t),t) \\ &= u_{ref}(t) + \sum_{i=1}^{p_0} \alpha_i(x^{(0)}) u(g_{x,0}^{(i)},t) + K(t)(x(t) - \underbrace{x_{ff}(t)}_{=x_{ref}(t)+\tilde{x}_{ff}(t)}), \end{aligned}$$

the extended state has the dynamics

$$\dot{x}_e(t) = \begin{bmatrix} \dot{x}(t) \\ \dot{x}_{ref}(t) \\ \dot{\tilde{x}}_{ff}(t) \\ \dot{\alpha}(x^{(0)}) \end{bmatrix} = \begin{bmatrix} f\left(x(t), u_{ref}(t) + \tilde{u}_{ff}(\alpha(x^{(0)}),t) + K(t)\left(x(t) - x_{ref}(t) - \tilde{x}_{ff}(t)\right), w(t)\right) \\ f(x_{ref}(t), u_{ref}(t), 0) \\ A_k \tilde{x}_{ff}(t) + B_k \tilde{u}_{ff}(\alpha(x^{(0)}),t) \\ \mathbf{0} \end{bmatrix},$$

where

$$\tilde{u}_{ff}(\alpha(x^{(0)}),t) = \left[u(g_{x,0}^{(1)},t), \ldots, u(g_{x,0}^{(p_0)},t)\right] \alpha(x^{(0)}).$$

The extra state for the reference trajectory $x_{ref}(t)$ evolves based on the actual nonlinear dynamics, though without disturbances. The state $\tilde{x}_{ff}(t)$ considers the deviation of the state-dependent feedforward trajectory $x_{ff}(t)$ from the center reference trajectory $x_{ref}(t)$ and evolves based on the linearized dynamics. This ensures that both the center following the reference trajectory, which has been computed based on the nonlinear dynamics, as well as the generators driven by the feedforward control law based on the linearized dynamics, end in the desired states. Since the feedback controller tracks the state-dependent feedforward controller while considering the actual disturbed, nonlinear dynamics, its feedback compensates any errors due to the linearized dynamics and the real system is controlled to the desired states as well. We use the parameter $\alpha(x^{(0)})$ to compute the feedforward inputs based on the initial deviation of $x^{(0)}$ from $c_{x,0}$, similar to the approach we used for the convex interpolation controller and generator interpolation controller, see Sec. 3.3.6.

For an initial set $\mathcal{X}_0 = \langle c_{x,0}, g_{x,0}^{(1)}, \ldots, g_{x,0}^{(p_0)} \rangle$, we define the initial zonotope for the extended dynamics as

$$\mathcal{X}_{e,0} = \left\langle \begin{bmatrix} c_{x,0} \\ c_{x,0} \\ \mathbf{0} \\ \mathbf{0} \end{bmatrix}, \begin{bmatrix} g_{x,0}^{(1)} \\ \mathbf{0} \\ g_{x,0}^{(1)} \\ e^{(1)} \end{bmatrix}, \ldots, \begin{bmatrix} g_{x,0}^{(p_0)} \\ \mathbf{0} \\ g_{x,0}^{(p_0)} \\ e^{(p_0)} \end{bmatrix} \right\rangle,$$

with $e^{(i)}$ denoting the i-th p_0-dimensional unit vector.

This means the actual state starts from $x^{(0)} \in \mathcal{X}_0$ and the reference trajectory from the center of the initial set, i.e., $x_{ref}(0) = c_{x,0}$. The generators $g_{ff}^{(i)}$ of the state-dependent feedforward trajectories are initialized by the set of possible deviations of the initial state $x^{(0)}$ from $c_{x,0}$, i.e., the generators of the initial set: $g_{ff,0}^{(i)} = g_{x,0}^{(i)}, \forall i \in \{1, \ldots, p_0\}$. The parameters $\alpha(x^{(0)})$ store the information of this initial deviation of the initial state $x^{(0)}$ from $c_{x,0}$ and are used to express the feedforward state $\tilde{x}_{ff}(t)$ as a weighted sum of the generators $g_{ff,t}^{(i)}$, i.e.,

$$\tilde{x}_{ff}(t) = \sum_{i=1}^{p_t} \alpha_i(x^{(0)}) g_{ff,t}^{(i)}.$$

Therefore, we initialize them with unit vectors, i.e., $g_{\alpha,0}^{(i)} = e^{(i)}$, where the i-th unit vector corresponds to the i-th generator of the initial set.

Computing the reachable set starting from $\mathcal{X}_{e,0}$, we obtain an extended reachable set $\mathcal{R}^e_{[0,t_f],u_{ctrl},\mathcal{W}}(\mathcal{X}_{e,0})$. For a single time interval $[\tau_l, \tau_{l+1}]$, the extended reachable set is given by a zonotope of the form

$$\mathcal{R}^e_{[\tau_l,\tau_{l+1}],u_{ctrl},\mathcal{W}}(\mathcal{X}_{e,0}) = \left\langle \begin{bmatrix} c_x \\ c_{ref} \\ \mathbf{0} \\ \mathbf{0} \end{bmatrix}, \begin{bmatrix} g_x^{(1)} \\ g_{ref}^{(1)} \\ g_{ff}^{(1)} \\ g_\alpha^{(1)} \end{bmatrix}, \ldots, \begin{bmatrix} g_x^{(p)} \\ g_{ref}^{(p)} \\ g_{ff}^{(p)} \\ g_\alpha^{(p)} \end{bmatrix} \right\rangle,$$

where we omit the subscript denoting the time interval for the center, the generators, and the time-dependent number of generators p, to not further complicate the notation. We can directly use this reachable set to check if states with the actual nonlinear dynamics satisfies the state constraints, i.e., if $\langle c_x, g_x^{(1)}, \ldots, g_x^{(p)} \rangle \subseteq \mathcal{S}$, and to compute the state dependent part of the cost function.

To check the input constraints and to compute the input-dependent part of the cost function, we have to compute the set of possible inputs based on the extended reachable set during each time interval $[\tau_l, \tau_{l+1}]$. For the sake of simplicity, we choose the time interval $[\tau_l, \tau_{l+1}]$ such that the feedforward inputs are constant. Note that since additional generators are added during the reachable set computation, the generators $g_\alpha^{(i)}$ are not necessarily only unit vectors anymore. These new generators, resulting, e.g., from disturbance effects, are not considered during the feedforward control, and consequently only contain zeros in the $g_\alpha^{(i)}$ entries. For a given state

$$x(t) = c_x + \sum_{i=1}^{p} \beta_i(x(t)) g_x^{(i)},$$

with $\beta(x(t)) \in [-1, 1]^p$, the corresponding input is given by

$$\begin{aligned} u_{ctrl}(x(t),t) = & \underbrace{u_{ff}(\alpha(x^{(0)}),t)}_{=u_{ref}(t)+\tilde{u}_{ff}(\alpha(x^{(0)}),t)} + K(t)(x(t) - \underbrace{x_{ff}(t)}_{=x_{ref}(t)+\tilde{x}_{ff}(t)}) \\ = & \, u_{ref}(t) + \sum_{i=1}^{p} \beta_i(x(t)) \left[u\left(g_{x,0}^{(1)}, t\right), \ldots, u\left(g_{x,0}^{(p_0)}, t\right) \right] g_\alpha^{(i)} \\ & + K(t) \left((c_x - c_{ref}) + \sum_{i=1}^{p} \beta_i(x(t)) K(t) \left(g_x^{(i)} - g_{ref}^{(i)} - g_{ff}^{(i)} \right) \right). \end{aligned}$$

Therefore, using the same arguments as in Sec. 3.5.2, the set of inputs during the time interval $[\tau_l, \tau_{l+1}]$ is given by the following zonotope:

$$\begin{aligned} u_{ctrl}\left(\mathcal{R}_{[\tau_l,\tau_{l+1}],u_{ctrl},\mathcal{W}}(\mathcal{X}_0), t\right) = \Big\langle & u_{ref}(t) + K(t)(c_x - c_{ref}), \\ & \left[u(g_{x,0}^{(1)}, t), \ldots, u(g_{x,0}^{(p_0)}, t) \right] g_\alpha^{(1)} + K(t) \left(g_x^{(1)} - g_{ref}^{(1)} - g_{ff}^{(1)} \right), \\ & \vdots \\ & \left[u(g_{x,0}^{(1)}, t), \ldots, u(g_{x,0}^{(p_0)}, t) \right] g_\alpha^{(p)} + K(t) \left(g_x^{(p)} - g_{ref}^{(p)} - g_{ff}^{(p)} \right) \Big\rangle. \end{aligned}$$

This zonotope is then used in the cost and constraint function as described in Sec. 3.5.

3.6.5 Numerical Example

To illustrate the applicability of the combined control approach and its improved performance compared to the approaches we presented in previous sections, we revisit the previous examples as well as a nonlinear tube-based MPC example from [97]. We implement our approach again in MATLAB, using the same toolboxes and optimization functions as before. For a fair comparison, all computations are performed on the same computer as used for the earlier examples.

Linear Example: Vehicle Platoon

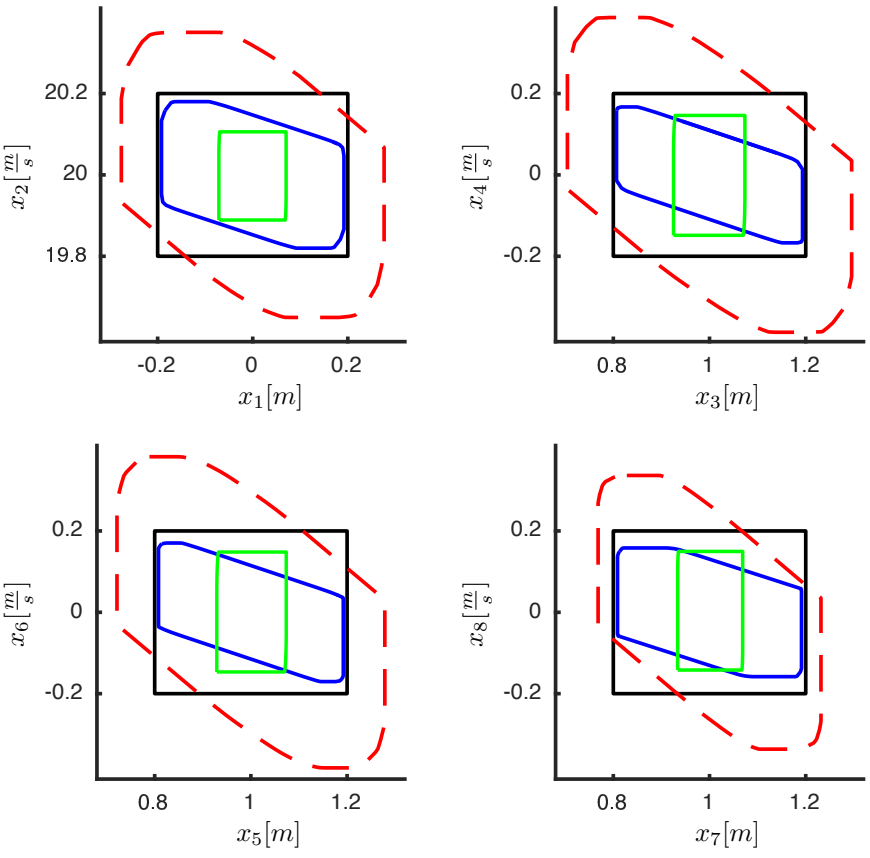


Figure 3.20: Linear platooning example: Initial (black) and shifted final sets (green) of our combined controller, projected onto the (x_1, x_2), the (x_3, x_4), the (x_5, x_6), and the (x_7, x_8) planes. For comparison, the final sets of the continuous feedback controller from Sec. 3.5 (blue) and an LQR controller (red) are shown.

Let us first revisit the linear example from Sec. 3.5.6, where we compute a controller for a platoon with four vehicles. In Fig. 3.20, we show the resulting final reachable sets for the continuous feedback controller from Sec. 3.5.6 in blue and with the new combined approach in green. We see that the new approach is able to obtain much smaller reachable sets and its computation time of around one minute is around five times faster than the old approach. This is quite impressive, as even the old approach performs much better than a constant LQR tracking controller (red).

Nonlinear Example: Kinematic Vehicle

Let us now consider the autonomous vehicle example from Sec. 3.4.3, which we solved with the generator interpolation controller. We use the same input bounds as before; however, to demonstrate the improved performance, we increase the disturbance sets to make the control task harder. To do so, we enlarge each disturbance interval by a factor of four compared to the example in Sec. 3.4.3, resulting in $w_1 \in [-2, 2]\ \frac{m}{s^2}$ and $w_2 \in [-0.08, 0.08]\ \frac{rad}{s}$.

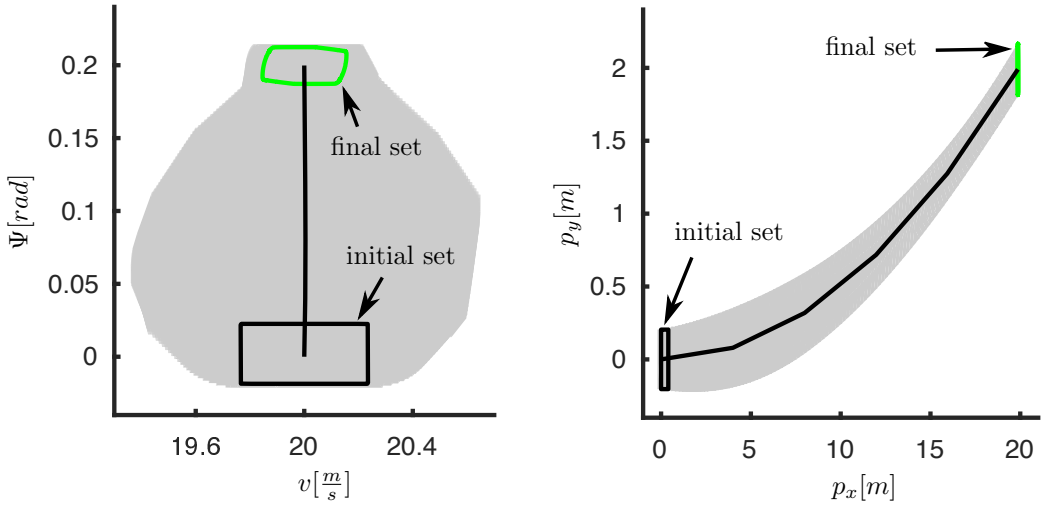


Figure 3.21: Nonlinear car example: Reachable sets for the *turn left* motion primitive with the combined controller, projected onto the (v, Ψ) and the (p_x, p_y) planes. The initial set is plotted in black, the final set in green, and the reachable set for all times between in gray. The black line shows the reference trajectory.

We show the reachable set for the *turn left* motion primitive in Fig. 3.21. For a better illustration, we show again the initial set with the shifted final set in Fig. 3.22. The reachable set of the generator interpolation controller with the new disturbance set is shown in blue and the reachable set of the combined controller in green. We see that the additional feedback controller allows us to end completely inside the shifted initial set, while the old controller, which applies only the feedforward controller in an iterative fashion, results in a large reachable set. The reachable sets of the LQR tracking controllers which we use as a comparison in

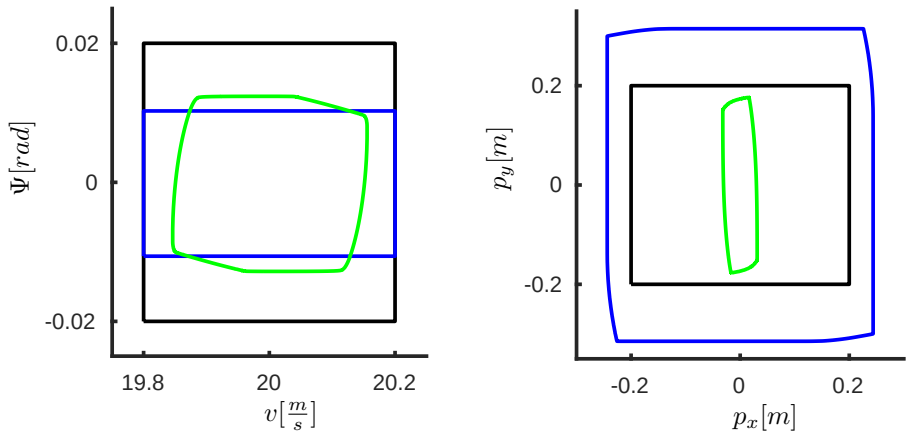


Figure 3.22: Nonlinear car example: Initial (black) and shifted final sets (green) of the combined controller, projected onto the (v, Ψ) and the (p_x, p_y) planes, for the *turn left* motion primitive. For comparison, the final set of the generator interpolation controller from Sec. 3.4 (blue).

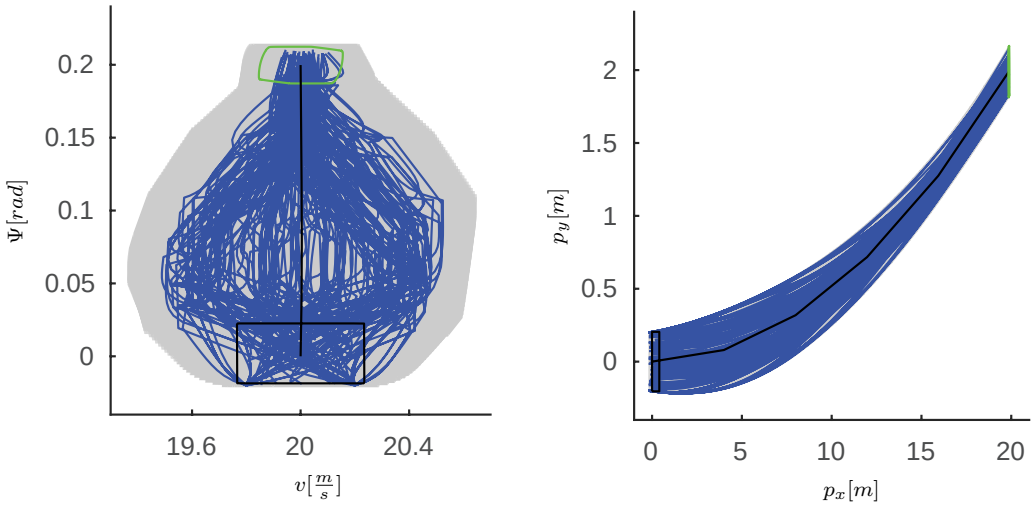


Figure 3.23: Nonlinear car example: Fig. 3.21 with 200 simulations of the combined controller for random initial states and random disturbances, shown in blue.

Sec. 3.4.3 would become much larger with the increased disturbances than the sets in Fig. 3.22, therefore we omit them for clarity.

We compute 200 simulations with random disturbances for this scenario with our new controller and show them in Fig. 3.23. We see that the simulations cover almost the whole reachable set for the (p_x, p_y) plane and a large part of the (v, Ψ) plane. This demonstrates that our approach is not overly conservative, but that the reachable sets are justified by actual disturbance

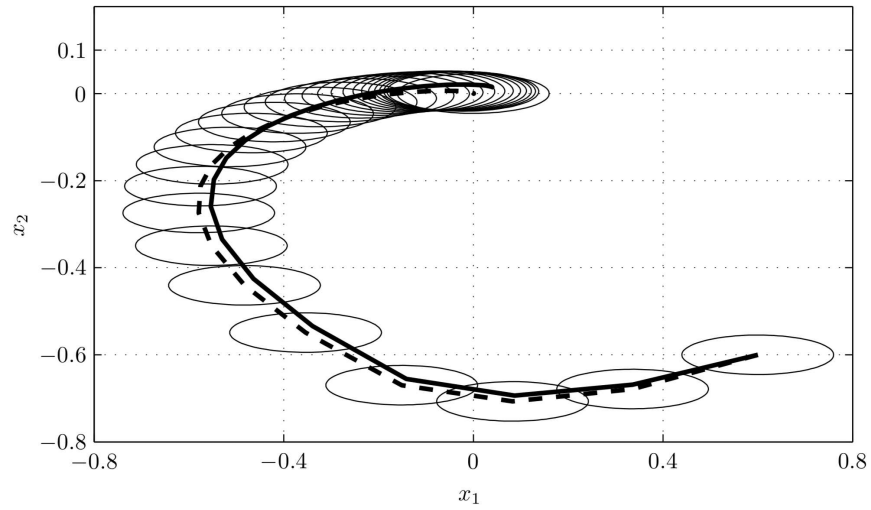
effects. In the same fashion as for the *turn left* motion primitive, we can compute other motion primitives such as *drive straight* or *turn right*, as done in the previous sections. Since the shifted final sets for the combined controller always fit in the initial set, we can concatenate all of them with each other and obtain the fully connected maneuver automaton from Fig. 3.6 even for the enlarged disturbances. The computation of the new controller takes around two minutes, where 70% of the time is spend for computing the reachable sets during the feedback optimization and less than 2% for the optimization of the reference trajectory and state-dependent feedforward controller.

Nonlinear Example: Comparison with Tube-Based MPC

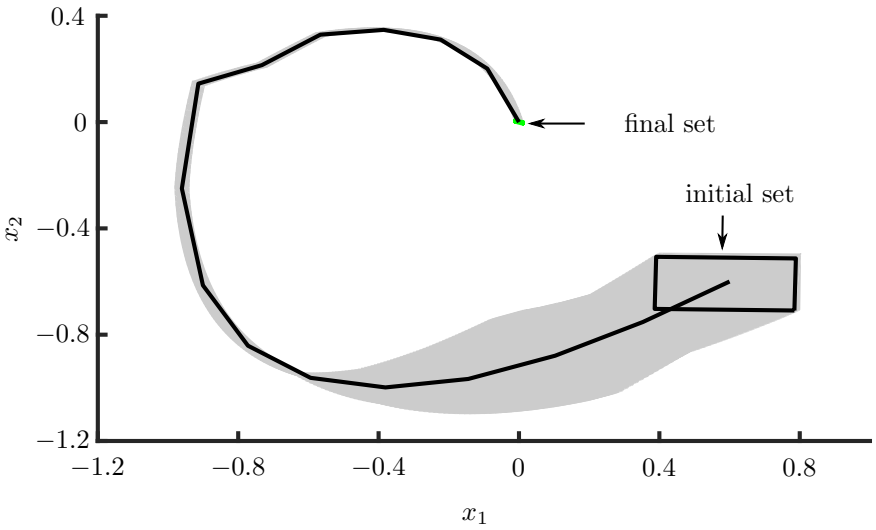
In our last example, we apply our approach to the example from [97]. The system dynamics are given by

$$\begin{aligned}\dot{x}_1 &= -x_1 + 2x_2 + 0.5u,\\ \dot{x}_2 &= -3x_1 + 4x_2 - 0.25x_2^3 - 2u + w,\end{aligned}$$

with the constrained control input $u \in [-2, 2]$ and the bounded disturbance $w \in [-0.1, 0.1]$. The authors from [97] solve the control problem of stabilizing the origin for this unstable system using a nonlinear tube-based MPC controller, which is applicable for continuous-time nonlinear systems. The resulting tube for a single example is shown in Fig. 3.24(a). We solve the same problem with our approach, where we start from an initial set which contains their control invariant set. The results are shown in Fig. 3.24(b) and take less than 30 s to compute. We see that the system with our controller converges to a small set around the origin. Our reachable sets are thus much smaller than the control invariant set from [97], which determines the size of the tube. This shows the advantage of our controller based on the actual reachable sets, rather than a fixed-size tube based on control invariant regions.



(a) Tube-based MPC solution; taken from [97]



(b) Solution from our combined control algorithm

Figure 3.24: Comparison of tube-based MPC with our combined control approach: (a) Reference trajectory and control invariant sets of the tube-based MPC solution. (b) Reachable set (gray) of our controller for the same example with initial set in black and final set in green. The initial set of (a) is contained in the initial set of (b).

3.7 Discussion

In this section, we discuss different aspects of the presented control algorithms and compare them.

3.7.1 Formal Guarantees

One of the advantages of all of our approaches is that they do not require proofs for stability or for constraint satisfaction. Verifying the stability or safety of complex, nonlinear dynamics that are affected by disturbances, restricted by state and input constraints, and controlled by switching control laws is a very hard task when using classical approaches. Finding a Lyapunov function for such a system easily becomes infeasible in practice.

Instead, safety and constraint satisfaction is guaranteed by the included reachability analysis for all initial states and all possible disturbances. By computing all optimization problems and reachable sets offline in advance, we are able to only consider motion primitives in our maneuver automaton for which a feasible solution has been found. Thus, even if we cannot find a feasible controller for a certain motion primitive, we know this offline in advance and can recompute this motion primitive.

3.7.2 Complexity

When we discuss the complexity of our algorithm, we have to distinguish between online and offline complexity. Most critical is the online complexity, as this restricts the sampling times of the controlled system and therefore its performance.

Online Complexity

We perform all optimizations and reachable set computations offline in advance and store the different controller matrices in a look-up table. Online, we only need to apply the controller corresponding to the current motion primitive. Since we use piecewise-linear controllers for all approaches, the online complexity is very low in all cases:

Convex Interpolation Controller The online complexity of the convex interpolation controller consists only of matrix vector multiplications, and in the case of closed-form expression of the convex combinations, plugging the current state into a closed-form formula. Clearly, since we have 2^n extreme points, we have to compute 2^n weights, but each computation can be done very fast (around $0.1\,ms$ for all weights of a ten-dimensional system, see Table A.1 in Appendix A). If we use the linear approximation of the inputs as presented in Sec. 3.3.4, the computation simplifies to a matrix vector multiplication with a complexity of only $\mathcal{O}(n\,m)$. Therefore, we can apply the controller with high sampling times. The efficiency of this approach can also be seen in the fast computation times of around $10\,s$ for the offline part and around $0.01\,ms$ for the online part, see Sec. 3.3.7.

Generator Interpolation Controller The resulting generator interpolation controller has the same structure and therefore the same online complexity as the linear approximation of the convex interpolation controller.

Continuous Feedback Controller The online complexity of the continuous feedback controller is equivalent to other linear feedback controllers. Compared to the previous two control methods, the only difference is the continuous nature of the controller, which requires knowing the reference trajectory at every time step. One way is to store the reference trajectory exactly enough and interpolate between the sampling points. By considering this interpolation during both the controller synthesis and the reachable set computations, the approach is still formal. In this case, the computational burden at each time instance is equivalent to the convex and generator interpolation controllers. Alternatively, the reference trajectory can be integrated online, which requires some numerical integration techniques, e.g., the Runge-Kutta method.

Combined Controller The combined controller is similar to the continuous feedback controller with the difference that the reference trajectory is based on the initial state. Since we have to compute the input as a combination of the center and the generators, this results in a small additional computational effort; however, it does not change the complexity class nor does it significantly change the computation times. It also requires either the integration of a (extended) system or interpolating precomputed reference trajectories for the center and the generators. From the way we computed the generator feedforward, we are able to superpose the precomputed reference trajectories.

Offline Complexity

While the online computation times of all four approaches are very similar, the offline computation times differ much more. All approaches rely on solving some nonlinear programs, at least for the computation of the reference trajectory. Since this is a nonlinear and non-convex optimization problem, one cannot give bounds on its computational complexity, and therefore not on the overall complexity of any approach which uses it. Let us still discuss the various parts which are used in the different algorithms and what that means for actual computation times.

Reference Trajectory: For each approach we have to compute a reference trajectory. While this is done using nonlinear programming, there exist very efficient implementations using direct optimization which solve the problem fast and converge for many initial states [98]. Therefore, for practical examples, the computation of the reference trajectory can be done very fast, and as many applications in MPC show, even in real time. In particular, since we have to compute the reference trajectory only once and we are not dependent on an optimal solution, we can neglect the complexity of the reference trajectory for the overall approach. We can argue that if the optimal control problem is hardly solvable for undisturbed open-loop trajectories, then we cannot expect it to be solved more easily for a set of initial states and disturbances.

Reachability Analysis: The second part which is used in all approaches is the computation of reachable sets. The algorithm from [74] which we use in our numerical examples has a

complexity of $\mathcal{O}(n^3)$, where n denotes the number of states. Since we combine reachability analysis with controller synthesis, we eliminate the need for an additional verification step afterward. For the convex interpolation controller and the generator interpolation controller, we only compute the reachable set for the whole time horizon once. This means that we do not add any computation time compared to synthesizing and verifying a controller separately. Optimizing over the reachable sets leads to longer computation times, though as seen before, to better results as well.

Convex Interpolation Controller Besides the reference trajectory, we have to solve a nonlinear optimal control problem for each extreme state at each iteration step. By using the same efficient algorithms as for the reference trajectory and by considering shorter time horizons, we can solve each optimization problem very fast. Since the number of extreme points of a parallelotope grows with 2^n, the number of optimization problems increases exponentially as well. Though exponential, it is still much better than other comparable algorithms, which rely on discretizing the state space (like explicit MPC or abstraction-based control) and have exponential complexity with a larger base: d^n, where d denotes the number of discretized states in each dimension. The value of d can easily be 30 or 100. The cubic complexity of the reachability analysis is therefore negligible for the overall complexity.

Generator Interpolation Controller The main optimization for the generator interpolation controller is done by solving linear programs. The complexity of solving linear programs depends on the exact implementation and the number and type of constraints. There exist algorithms with polynomial time complexity in the number of optimization variables and constraints [99]. If we consider zonotopes of a fixed order, the number of optimization variables and number of constraints grows polynomially as well (with $\mathcal{O}(n^2)$, if $n > m$, where m denotes the number of inputs). In practice, the computation time for solving linear programs even with many constraints is also very low and does not add much to the overall computation time. If we neglect the reference trajectory, then the remaining algorithm of the generator interpolation controller scales polynomially — as we only solve linear programs and compute reachable sets — and thus much better than the convex interpolation controller.

Continuous Feedback Controller The complexity of the continuous feedback controller is governed by the number of iterations of the nonlinear programming algorithm. At each iteration, we have to compute the reachable set for the whole time horizon with complexity of $\mathcal{O}(n^3)$. The number of required iteration steps depends on the number of optimization variables which are $N(n + m - 1)$ with N denoting the number of different feedback weights Q_k and R_k. Due to the many reachable set computations, this algorithm needs much longer than the previous two.

Combined Controller As seen in the numerical examples, the combined controller needs much less time than the continuous feedback controller, as the number of optimization variables for the overall optimization problem is reduced to $n + m - 1$ and the single additional linear

Table 3.1: Comparison of the four algorithms

Property	Convex Interpolation Controller	Generator Interpolation Controller	Continuous Feedback Controller	Combined Controller
Offline computation time (low/medium dimensions)	low	low	high	medium
Increase of offline computation time for high dimensions	very high	low	high	medium
Online computation time	very low	very low	very low	very low
Capability to consider nonlinearities	good	medium	good	good
Robustness against disturbances	medium	medium	high	high

program for computing the state-dependent feedforward controller can be neglected. Since it still computes many reachable sets, it takes longer than the two interpolation-based controllers.

3.7.3 Optimality

As we rely on superposing different solutions for nonlinear systems, nonlinear programming algorithms that do not guarantee convergence to a global optimal solution, or a combination of both, we can only expect to obtain a local optimum. In fact, there is no efficient method able to obtain globally optimal controllers for disturbed, nonlinear systems [8, 100]. Most optimal control approaches only consider open-loop dynamics or only undisturbed feedback controllers [101, 102]. Our synthesis approaches optimize over the whole reachable set, i.e., all possible trajectories resulting from any possible disturbance realization, and chooses the controller which minimizes those trajectories. This is not done in any comparable method, as most other formal methods consider fixed controllers (e.g., tube-based MPC) or fixed control inputs for the whole set (e.g., abstraction-based methods).

3.7.4 Comparison of the Four Algorithms

All four algorithms have their own advantages and disadvantages. We provide a high-level comparison of the algorithms in Table 3.1. Which of them might be the most suitable depends on the application area. If short offline computation times are needed, then the convex interpolation controller and the generator interpolation controller are the first choice, as they only require a single computation of the whole reachable set.

If the dynamics are quite nonlinear and the state dimension is not too large, the convex interpolation controller has the advantage that it computes the input trajectories for the actual

nonlinear dynamics and for each extreme point. The generator interpolation controller, on the other hand, requires computing linearized dynamics and uses the superposition of the control inputs for the center and the generators, which might not be as accurate for very nonlinear dynamics as the results of the convex interpolation controllers. For larger systems, i.e., with a higher number of state variables, the generator interpolation controller has the big advantage that the number of generators scales linearly compared to the exponential scaling of the number of extreme states for the convex interpolation controller.

In contrast, the continuous feedback controller and the combined controller repeatedly optimize over the reachable sets of the closed-loop system. The optimization over reachable sets ensures the minimization of the actual final set while taking all effects into account and the continuous feedback allows us to counteract disturbances better and faster. This is not the case for the two interpolation algorithms, as the control inputs are computed open-loop while only considering the reachable set of the previous time step. The repeated computation of reachable sets in the optimization algorithm for the continuous feedback controller leads to longer offline computation times, which is usually worth it when considering the better online control performance.

The combined control approach uses a combination of the generator interpolation controller and the continuous feedback controller. Thus, it is not surprising that it has better control results than both and that it is faster than the continuous feedback controller. Therefore, in most cases, the combined control approach is the best choice, especially compared to the continuous feedback controller. All four approaches provide formal guarantees, optimize over solutions, and formally ensure the satisfaction of state and input constraints of disturbed systems. Not many control approaches are able to do this, especially not efficiently. As we see in the examples, all of these approaches are much better than a simple LQR tracking control approach.

3.7.5 Extension to Include Measurement Noise

So far, we did not discuss measurement noise in this chapter in order to have a simpler notation and presentation. To apply the presented approaches to systems with uncertain measurements, only few changes have to be done, which we present now.

For the convex interpolation controller and the generator interpolation controller, we define the parallelotopes $\mathcal{P}_k$ during Step 2 of the nonlinear controller synthesis algorithm as $\mathcal{P}_k \supseteq \{h(x,\nu) \mid x \in \mathcal{X}_{reach,k}, \nu \in \mathcal{V}\}$ and compute the open-loop parts based on this set. This ensures that the resulting control law is feasible for any possibly measured state. For the reachable set computation, we start again from the actual reachable set from the previous time step, i.e., $\mathcal{X}_{reach,k}$, and include the measurement noise as a disturbance which only affects the controller functions by using the measured state $\hat{x}(t_k) = h(x(t_k), \nu(t_k))$ instead of $x(t_k)$ in $u_{conv}(\hat{x}(t_k), t)$ and $u_{gen}(\hat{x}(t_k), t)$, respectively. We thereby ensure that the reachable set includes all possible trajectories which result from our control law with disturbed measurements.

For the continuous feedback controller, we directly optimize over the closed-loop dynamics and therefore simply include the uncertain measurements in the control law $u_{ctrl}(\hat{x}(t), t)$ for the reachable set computation as well.

Finally, for the combined controller, we compute the state-dependent feedforward controller

analogous to the generator interpolation controller by starting from an initial zonotope $\hat{\mathcal{X}}_0$ which satisfies $\hat{\mathcal{X}}_0 \supseteq \{h(x,\nu) \mid x \in \mathcal{X}_0, \nu \in \mathcal{V}\}$. For the feedback controller we include the measurement noise in the feedback control law $u_{ctrl}(\hat{x}(t),t) = u_{ff}(\hat{x}^{(0)},t) + u_{fb}(\hat{x}(t),t)$ for the reachable set computation. An explicit description of the combined controller with measurement noise can also be found in [219].

These ideas also work analogously for the approaches from the next two chapters, which we again present without disturbances to simplify notation.

3.8 Summary

In this chapter, we present four novel control approaches which solve reach-avoid problems offline and can be used to generate robust maneuver automata. All approaches allow us to compute controllers which steer all states from an initial set to a set around a desired final state while satisfying state and input constraints. The approaches work for linear and nonlinear systems even in the presence of external disturbances and sensor noise. The use of reachable set computation allows us to achieve provable safety and formal guarantees for the satisfaction of constraints. All controllers have a linear, time-varying controller structure, which allows very fast online computation times and easy implementation, as most computation tasks can be performed offline in advance.

The first approach allows us to steer all states from an initial set to a final set by computing optimal input trajectories for the extreme states only. By interpolating between these extreme state inputs using convex combinations, we obtain control laws for every state inside the reachable set. The presented control approach is a novel way of viewing closed-loop control by combining the optimized solutions from open-loop control with the stability and robustness of feedback control. By using closed-form expressions for the convex combinations and linear approximations for the control inputs, we provide two ways to make the convex controllers even more efficient and faster.

In the second approach, we exploit the zonotope representation of sets to obtain even more efficient control laws. This is done by computing optimal open-loop input trajectories for the center and generators of the set and interpolate them by superposing the solutions. By linearizing the system and again iteratively applying the optimized controller to the system in combination with reachable set computation, we are once more able to apply the approach to disturbed nonlinear systems. In each step, only a single linear program has to be solved to optimize the inputs for each state in the set. This significantly reduces the computational complexity compared to the convex interpolation controller, so that the generator interpolation control approach scales polynomially with the number of dimensions.

In a third approach, we optimize a continuous feedback controller by directly optimizing over reachable sets. To the best of our knowledge, we use, for the first time, reachability analysis inside the optimization problem to obtain optimal control inputs, not only for a single state, but for a whole continuous set of initial states. This new technique allows us to take constraints and the effects of disturbances into account. The resulting controller is robust against disturbances, and we obtain formal guarantees for the satisfaction of constraints and the resulting reachable

set. By computing the reachable set inside the optimization problem, we are able to directly optimize the size of the reachable set at a final time point.

Finally, we combine the ideas of the second and the third approaches to join their advantages while overcoming their disadvantages. We use the generator interpolation controller to obtain a state-dependent feedforward controller based on zonotopes, which steers all states of the initial set to the desired final set. We combine it with a continuous feedback controller, which we compute by optimizing again over reachable sets. Since it only has to counteract disturbances and linearization errors, not initial deviation, it can be much more aggressive than for classical approaches. In addition, the optimization problem becomes significantly simpler, as the number of optimization variables is drastically reduced.

We show the applicability of each of the approaches in numerical examples, where we compare them to classical approaches such as LQR tracking controllers. All of our controllers demonstrate that they can provide much smaller reachable sets while guaranteeing the satisfaction of state and input constraints despite disturbances.

Compared to existing approaches, our algorithms have a number of advantages: They are able to provide formal guarantees without knowing a Lyapunov function, which can be hard to get for practical application but are required for many classical control approaches (see, e.g., [103]). Compared to approaches using sum-of-squares programming [34–36, 44–49] or abstraction based control [50–71], our new approaches scale much better and thus are applicable for larger systems. Due to the offline computation, the online application is much faster than implicit MPC [12, 22–30] and in many cases requires far less data storage than explicit MPC [14–21]. In contrast to many classical approaches, the consideration of sensor noise is straightforward in our algorithms. To the best of our knowledge, our control approaches are therefore the first which are able to efficiently provide formal guarantees for solving reach-avoid problems for constrained systems even in the presence of external disturbances and sensor noise. This is possible due to the novel close combination of reachability analysis and control theory.

Chapter 4

Offline Controller Synthesis for Piecewise Affine Systems

4.1 Introduction

So far, we considered the control problem for constrained and disturbed nonlinear systems with purely continuous dynamics. There also exists a large interest in the control of hybrid systems, i.e., systems with continuous and discrete dynamics. They are becoming a commonly adopted modeling framework, as there are more and more applications where discrete computation tasks closely interact with the continuous, physical world. Since a number of safety-critical applications are also developed using hybrid models, their control design methods require robust safety guarantees that account for uncertainties. Therefore, we extend the idea for computing safe controllers to hybrid systems in this chapter[1].

While the field of hybrid systems is very wide, we focus on the class of discrete-time piecewise affine (PWA) systems that evolve according to a different affine dynamics based on the value taken by the state in a polyhedral partition of the state space defining the so-called modes. PWA systems therefore offer the required hybrid aspects with changing dynamics in different modes; at the same time, their simplicity allows us to focus on the main differences which arise from the non-continuity in the dynamics without making the discussion too complex. It is possible to further extend the ideas to more general classes of hybrid systems, but many more special cases would have to be considered. This lies outside the scope of the current chapter and would complicate the discussion unnecessarily.

PWA systems are quite common since they naturally arise as models in various application contexts. In addition to their capability in modeling non-smooth dynamics, they can also be used as a model for nonlinear (smooth) systems by using hybridization methods such as those in [85, 104–106]. Various analysis and control problems, such as model reduction [107, 108], controllability and observability [109], identification [110–112], fault detection and estimation,

[1]This chapter, including the figures, is based on [227] © 2019 Elsevier Ltd. Sec. 4.4 as well as the implementation of the optimization for the reference trajectory are mainly the work of Riccardo Vignali.

as well as tracking, stabilization, and control [113–118], have been studied for the class of PWA systems. The vast literature on PWA systems is primarily motivated by their significant modeling capabilities, but also by the fact that their description via affine equations and constraints can simplify analysis and design. They are also equivalent to the class of mixed logical dynamical systems [119].

Challenges for Reachability Analysis

Many reachability algorithms also work for hybrid systems [1, 2] including the one from [74]. One problem in the case of hybrid systems, however, is that zonotopes are not closed under intersection. This makes it difficult to adopt them for the formal verification of PWA systems, since reachable sets can intersect with multiple modes. Different solutions have been proposed for this possibility in the literature, though all with some drawback.

One way is to split the reachable sets, which requires over-approximating the intersection of the reachable set within each mode and starting a new reachable set computation for each new set. In addition to the conservatism resulting from a possible over-approximation of the intersections by simpler geometric sets, the computational complexity increases with each new split [120, 121].

Alternatively, instead of splitting the set, one can apply the combined dynamics of the intersected modes in an over-approximative way. This strategy is used in [122], which deals with continuous-time systems, and leads to computations with interval matrices, which can be rather conservative.

Another solution is proposed in [123], where instead of computing geometric intersections explicitly, these intersections are over-approximated by nonlinear maps which consider the reachable set during a possible intersection time. While this approach has several advantages, its over-approximations are rather conservative if the potential intersection duration is too long.

Related Contributions on PWA System Control Design

Combinatorial problems often appear in control design for PWA systems [124], even if there is no uncertainty affecting its evolution, see for example, [125–132]. To limit the complexity of solving these problems, some approaches exist which use branch-and-bound methods to reduce the number of combinations [133].

Robust control for PWA systems is also addressed in [134] by imposing some robust mode control restriction (see [135]) to ensure that the mode sequence of the controlled system is independent of the disturbance realization. This restriction may limit the possibility of finding a solution. Other optimal control approaches for hybrid systems, such as [10], require solving a Hamilton-Jacobi partial differential equation, which is often prohibitively hard in practice. More recent techniques, on the other hand, use hybrid control Lyapunov functions to stabilize hybrid systems [136]. Reachability and controller synthesis for PWA systems using simplices can be found in [137], and an extension to output feedback for general polytopes is given in [138].

There is a large interest in model predictive control of PWA systems [17, 139, 140], including those with bounded disturbances [141–143]; however, many approaches rely on min-max

optimizations which can be very costly. Tightening constraints is one method for reducing computational costs [144]. A generalization of tube-based MPC [22] to PWA systems is proposed in [145] under the rather restrictive assumption that a common Lyapunov function for the affine dynamics exists. The approach requires the tightening of constraints and the online solution of a mixed integer program, where binary variables account for mode switching. A variant of the standard extension of [22] to PWA systems is proposed in [146], where the actual mode of the real system is used when defining the nominal trajectory. Alternatively explicit MPC solutions are possible, such as in [147], which computes feedback control policy by resorting to set invariance and multiparametric programming for constrained PWA systems affected by a bounded disturbance. The resulting PWA state feedback control law is optimal, but quite complex to represent and apply online; for example, a polyhedral partition of 417 regions is required for the PWA controller for the two-dimensional example with two modes in [147].

Proposed Set-Based Control Approach

By extending the generator interpolation controller to PWA systems, we are able to provide formal guarantees while scaling better than existing approaches. Our control approach for PWA systems consists of two parts: a smart choice of the reference trajectory, which directly takes the distances to mode boundaries into account and thereby aims at reducing the number of splits, as demonstrated in Fig. 4.1 and the generator interpolation controller from Sec. 3.4 for tracking this reference trajectory. The extension is not trivial due to the hybrid dynamics of the system and the possible occurrence of mode splitting as discussed before.

In contrast to verification, however, the control input can be chosen to avoid splitting completely, or at least reduce the number of splits. Without splits, the PWA system becomes a time-varying affine system which drastically simplifies the set-based feedback control design.

4.2 Problem Formulation

We consider a discrete-time PWA system whose state $x \in \mathbb{R}^n$ evolves according to the following equation:

$$x(k+1) = \begin{cases} A^{(1)}x(k) + B_u^{(1)}u(k) + B_w^{(1)}w(k) + f^{(1)}, & x(k) \in \mathcal{M}^{(1)} \\ \quad \vdots & \\ A^{(s)}x(k) + B_u^{(s)}u(k) + B_w^{(s)}w(k) + f^{(s)}, & x(k) \in \mathcal{M}^{(s)} \end{cases} \tag{4.1}$$

with inputs $u(k) \in \mathbb{R}^m$ and disturbances $w(k) \in \mathcal{W} \subset \mathbb{R}^\omega$. As in the previous chapter, we consider a set of initial states $\mathcal{X}_0 = \langle c_{x,0}, g_{x,0}^{(1)}, \dots, g_{x,0}^{(p_0)} \rangle$ and a set of possible disturbances $\mathcal{W} = \langle c_w, G_w \rangle$, and assume that they are given as zonotopes (or can be over-approximated by them). Without loss of generality, we assume for a simpler notation that this disturbance zonotope is centered around the origin, i.e., $c_w = \mathbf{0}$. If this was not the case, it would mean that there is a constant disturbance effect acting on the system and we can simply add the $B_w^{(i)}c_w$ effect to the affine part $f^{(i)}$. Since we only consider discrete-time systems in this chapter, we use a discrete $k \in \mathbb{N}_0$ instead of t as a time variable to simplify notation.

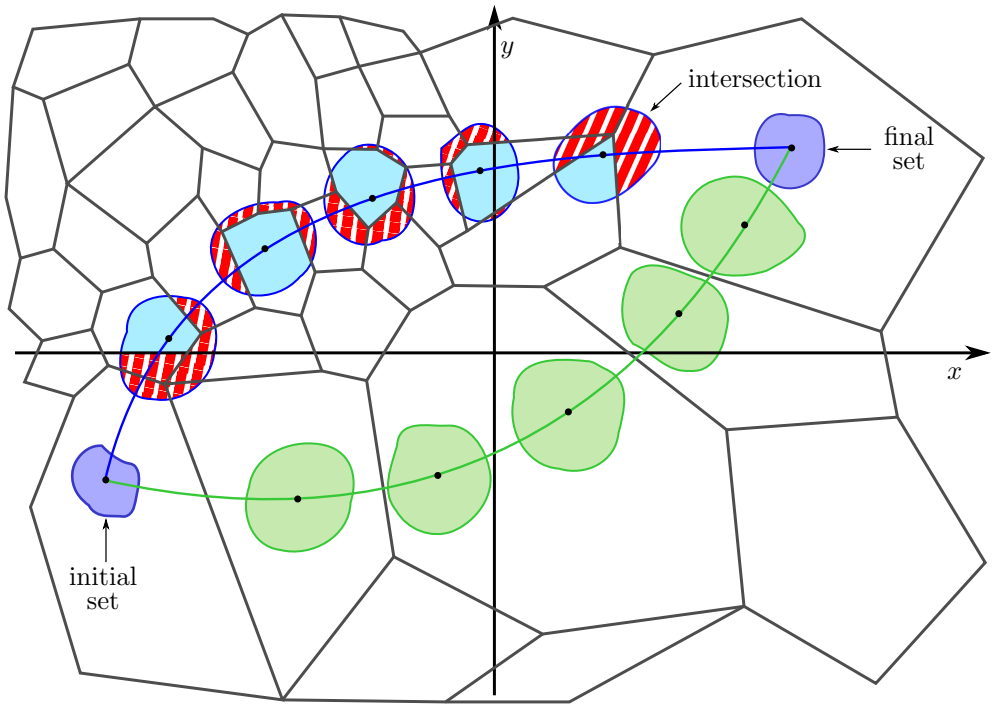


Figure 4.1: Example of two reference trajectories (blue and green lines) and reachable set propagation according to the dynamics associated with the reference mode sequence: tracking the green trajectory is much easier because reachable sets continue to evolve within the reference mode sequence. While the reachable sets of the blue trajectory have a similar size, they often intersect with the mode boundaries shown in gray.

The system dynamics in each set $\mathcal{M}^{(i)}$ representing mode i is defined by the matrices $A^{(i)}, B_u^{(i)}, B_w^{(i)}$ and vector $f^{(i)}$. We therefore distinguish between the continuous values of state $x(k)$, which evolves in discrete time, and the discrete mode i. We assume that each set $\mathcal{M}^{(i)} \subset \mathbb{R}^n$ is a polyhedron with half-space representation $\langle C_{\mathcal{M}}^{(i)}, d_{\mathcal{M}}^{(i)} \rangle_H$, where $C_{\mathcal{M}}^{(i)} \in \mathbb{R}^{v_i \times n}$ and $d_{\mathcal{M}}^{(i)} \in \mathbb{R}^{v_i}$. We refer to the hyperplanes defining the boundary of $\mathcal{M}^{(i)}$ as the mode boundary.

System (4.1) is said to be well posed if the collection of modes $\{\mathcal{M}^{(1)}, \ldots, \mathcal{M}^{(s)}\}$ forms a polyhedral subdivision of the state space, that is if $\cup_{i=1}^{s} \mathcal{M}^{(i)} = \mathbb{R}^n$, each $\mathcal{M}^{(i)}$ is of dimension n, and the intersection $\mathcal{M}^{(i)} \cap \mathcal{M}^{(j)}$, $i \neq j$, is either empty or a common proper face of both polyhedra. We assume that the PWA system we are dealing with is well posed. Without loss of generality, we also assume that the initial set $\mathcal{X}_0$ is a subset of a single mode, i.e., $\mathcal{X}_0 \subseteq \mathcal{M}^{(i)}$, for some $i \in \{1, \ldots, s\}$. If this was not the case, we can simply divide the initial set into several subsets, such that each subset satisfies this requirement, and compute a controller for each subset.

We address finite-horizon control of system (4.1) over $[0, N]$ and aim again for the minimum reachable set around a desired state $x^{(f)}$ for all trajectories starting in $\mathcal{X}_0$ while minimizing the

weighted input costs. Therefore, we want to minimize the following cost function:

$$\min_{u_{ctrl}(\,\cdot\,)} \|\mathcal{R}_{N,u_{ctrl},\mathcal{W}}(\mathcal{X}_0) - x^{(f)}\|_1 + \gamma \sum_{k=0}^{N-1} \|u_{ctrl}(\mathcal{R}_{k,u_{ctrl},\mathcal{W}}(\mathcal{X}_0), k)\|_1, \tag{4.2}$$

which is the discrete-time version of (3.1). Like in the previous chapter, we assume that the system is subject to input and time-varying state constraints:

$$\xi(x^{(0)}, u(\,\cdot\,), w(\,\cdot\,), k) \in \mathcal{S}_k, \ \forall k \in \{1, \ldots, N\}, \tag{4.3}$$

$$u(k) \in \mathcal{U}, \ \forall k \in \{0, \ldots, N-1\}, \tag{4.4}$$

and we assume again that they are all polyhedral sets with half-space representations $\mathcal{S}_k = \langle C_{\mathcal{S}_k}, d_{\mathcal{S}_k} \rangle_H$ and $\mathcal{U} = \langle C_U, d_U \rangle_H$, where $C_{\mathcal{S}_k} \in \mathbb{R}^{\varrho \times n}, d_{\mathcal{S}_k} \in \mathbb{R}^{\varrho}, C_{\mathcal{U}} \in \mathbb{R}^{\varsigma \times m}$, and $d_{\mathcal{U}} \in \mathbb{R}^{\varsigma}$. In this chapter, we explicitly discuss time-varying state constraints. We can also consider such constraints for all other approaches in this thesis; however, as they would significantly complicate the notation, we omit time-varying state constraints in the other chapters. The discrete-time system in this chapter allows us to show the approach directly for the time-varying case without much notational overhead. In the next section, we present an overview of our approach to solve (4.2), which is an extension of the ideas from Sec. 3.4 with differences due to the hybrid dynamics.

4.3 Overview of the Approach

Let us introduce our approach shown in Fig. 4.2, which illustrates the following steps:

Step 1. Compute reference trajectory (Fig. 4.2(a)). We start by computing a reference trajectory for the nominal PWA system (i.e., without uncertainties, see Sec. 4.4), which at time N should be close to the desired final state $x^{(f)}$ while both satisfying the state and input constraints and considering the distance to the mode boundaries to avoid splitting reachable sets. The sequence of modes $\mathcal{M}_0, \ldots, \mathcal{M}_N$ followed by the reference trajectory is called the main mode sequence. Each $\mathcal{M}_k$ takes values in the set of the PWA system modes $\{\mathcal{M}^{(1)}, \ldots, \mathcal{M}^{(s)}\}$.

Step 2. Compute reachable sets along mode sequence (Fig. 4.2(b)). At each iteration k, we design a control law $u_{ctrl,k}(\,\cdot\,)$ for a prediction horizon of M steps ahead, which steers all the states of the reachable sets starting from the initial set of this iteration $\mathcal{X}^*(k)$ close to the corresponding reference trajectory. We minimize the applied inputs and the size of reachable sets, and weight their distance to the boundaries of the main mode sequence so that, where possible, splits are avoided. At the same time, the controller ensures that the state and input constraints are satisfied despite the presence of disturbances. The result of this optimization is a sequence of predicted reachable sets $\mathcal{X}(k+1|k), \ldots, \mathcal{X}(k+M|k)$, with $\mathcal{X}(k+l|k) := \mathcal{R}_{l,u_{ctrl,k},\mathcal{W}}(\mathcal{X}^*(k))$, together with the corresponding time-varying control law $u_{ctrl,k}(\,\cdot\,)$.

Step 3. Apply the first input (Fig. 4.2(c)). At each iteration, we only retain the control law at time k and the reachable set $\mathcal{X}(k+1) := \mathcal{R}_{1,u_{ctrl,k},\mathcal{W}}(\mathcal{X}^*(k))$. We only use the

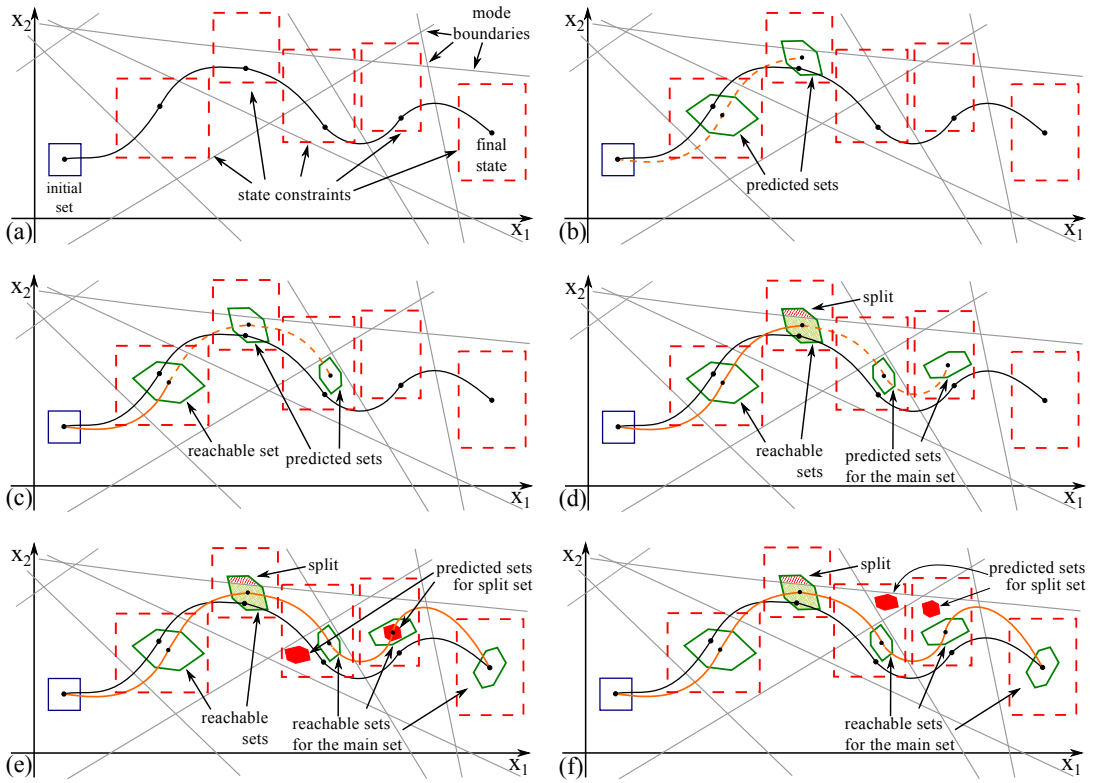


Figure 4.2: Overview of the overall approach with prediction horizon $M = 2$ and horizon after splits $M_{split} = 2$. (a) Computation of the reference trajectory and main switching sequence $\mathcal{M}_0, \ldots, \mathcal{M}_N$. (b) Computation of the first M predicted reachable sets $\mathcal{X}(1|0), \ldots, \mathcal{X}(M|0)$ starting from the initial set. (c) Application of the first control law and fixation of the set $\mathcal{X}^*(1)$ (orange solid line). Computation of a new collection of M sets $\mathcal{X}(2|1), \ldots, \mathcal{X}(M+1|1)$, starting from $\mathcal{X}^*(1)$. (d) Application of the first control law. There is now a split with a mode boundary; therefore, computation of M new sets $\mathcal{X}(3|2), \ldots, \mathcal{X}(M+2|2)$ starting from the main set $\mathcal{X}^*(2) = \mathcal{X}(2) \cap \mathcal{M}_2$. This is continued until the main set reaches the final set. For each split there are two possibilities: (e) Case I: The split set is steered back inside the main reachable set in M_{split} steps. No branching is needed. (f) Case II: The split set is not steered back inside the main reachable set in M_{split} steps: it will be used as a new initial set.

portion of the reachable set that lies inside the main mode as an initial set for each iteration, i.e., $\mathcal{X}^*(k) = \mathcal{R}_k \cap \mathcal{M}_k$. We repeat the procedure of Steps 2 and 3 for all k's until we reach the final time step. A sequence of sets $\mathcal{X}^*(1), \ldots, \mathcal{X}^*(N)$, called the main reachable set sequence, is determined.

Step 4. Check for splitting (Fig. 4.2(d)). If the main reachable set is completely inside the modes of the reference trajectory (no split occurred) and the state and input constraints have been satisfied, the problem is solved. If not, we start from the first time step k_0 where a split has been detected ($k_0 = 2$ in Fig. 4.2(d)).

Step 5a. Easy recovery possible (Fig. 4.2(e)). If we can steer the sets $\mathcal{X}(k_0) \cap \mathcal{M}^{(i)} \neq \emptyset$ with $\mathcal{M}^{(i)} \neq \mathcal{M}_{k_0}$ (i.e., those parts of the main reachable set outside of the main mode sequence) back to the main reachable set in M_{split} time steps, we only have to consider the main reachable set from time $k_0 + M_{split}$ onward, as it contains the split sets.

Step 5b. Easy recovery not possible (Fig. 4.2(f)). If Step 5a is not possible for some of the split sets, we branch each of them and treat them as new initial sets for which we compute a new reference trajectory each (computed as in Step 1) for the residual time horizon $[k_0, N]$. It is also necessary to recompute the reachable sets along the mode sequence at Step 2 starting from $\mathcal{X}(k_0) \cap \mathcal{M}^{(i)} \neq \emptyset$, with $\mathcal{M}^{(i)} \neq \mathcal{M}_{k_0}$, at time k_0.

Next, we explain the algorithms for the reference trajectory computation and the set-based control design.

4.4 Reference Trajectory Computation

We now explain how to generate the main reference trajectory (Fig. 4.2(a)) over the time horizon $[0, N]$; the computation for $[k_0, N]$ when splitting occurs (Fig. 4.2(f)) is done analogously. We consider the PWA system operating in nominal conditions and choose the initial state as $x_{ref}(0) = c_{x,0}$, with $c_{x,0}$ denoting the centers of the initial set $\mathcal{X}_0$.

We design a trajectory that satisfies the state and input constraints (4.3)–(4.4), while getting as close as possible to the desired final state $x^{(f)}$, minimizing the applied inputs, and keeping the state evolution as distant as possible from the mode and state constraints boundaries for each time step. As done in the previous chapter, we can also use a tightened version of the state and input constraints for the computation of the reference trajectory, if required. Here, it is not as crucial, as we include the distance from the boundaries of the state constraints as well as the applied inputs in the optimization problem.

The computation of the reference trajectory is formulated as a mixed integer linear program (MILP) and makes use of the following equivalent mixed logical dynamical (MLD) form of the (nominal) PWA system (4.1):

$$\begin{aligned} x_{ref}(k+1) &= Ax(k) + B_u u_{ref}(k) + B_\delta \delta(k) + B_z z(k), \\ E_x x_{ref}(k) &+ E_u u_{ref}(k) + E_\delta \delta(k) + E_z z(k) \leq E_{aff}, \end{aligned} \tag{4.5}$$

where $\delta(k) \in \{0,1\}^\kappa$ is a binary vector that determines which of the s modes is active at time k and $z(k) \in \mathbb{R}^\iota$ is an auxiliary real-valued variable (see [119] for details). Each mode $\mathcal{M}^{(i)}$ in (4.1) is encoded via a binary vector $\delta^{(i)}$, so that $\delta(k) = \delta^{(i)}$ if and only if $x(k) \in \mathcal{M}^{(i)}$. Linear input constraints are encoded in the MLD representation via E_u and E_{aff}. The MLD reformulation (4.5) simplifies the computations as it allows us to express all different dynamics of the original PWA system (4.1) corresponding to the different modes in a single compact form. Via the *big-M* technique [109, 119], it can be structured so that when $\delta(k) = \delta^{(i)}$, they collapse to the dynamics of the current mode $x_{ref}(k+1) = A^{(i)} x_{ref}(k) + B_u^{(i)} u_{ref}(k) + f^{(i)}$ and $C_{\mathcal{M}}^{(i)} x_{ref}(k) \leq d_{\mathcal{M}}^{(i)}$.

We start by considering a single time step k and a single mode i. The intersection between the state constraint set $\mathcal{S}_k$ and mode $\mathcal{M}^{(i)}$ is a polydron which we denote by $\mathcal{P}^{(i)}(k)$ and which

has the half-space representation $\langle C_{\mathcal{P}}^{(i)}(k), d_{\mathcal{P}}^{(i)}(k)\rangle_H$, with $C_{\mathcal{P}}^{(i)}(k) \in \mathbb{R}^{J_k^{(i)} \times n}$ and $d_{\mathcal{P}}^{(i)}(k) \in \mathbb{R}^{J_k^{(i)}}$.

Let $\text{dist}(x_{ref}(k), \mathcal{P}_j^{(i)}(k))$ denote the signed Euclidean distance of point $x_{ref}(k)$ from the j-th hyperplane $C_{\mathcal{P},j}^{(i)}(k)x_{ref}(k) = d_{\mathcal{P},j}^{(i)},\ j \in \{1, \dots, J_k^{(i)}\}$, given by

$$\text{dist}\big(x_{ref}(k), \mathcal{P}_j^{(i)}(k)\big) = \frac{-C_{\mathcal{P},j}^{(i)}(k)x_{ref}(k) + d_{\mathcal{P},j}^{(i)}(k)}{\|C_{\mathcal{P},j}^{(i)}(k)\|_2}, \tag{4.6}$$

which is positive if and only if $x_{ref}(k)$ lies inside $\mathcal{P}^{(i)}(k)$. Then we can introduce the auxiliary variable $d^{(i)}(k)$:

$$d^{(i)}(k) = \begin{cases} \min\limits_{j \in \{1,\dots,J_k^{(i)}\}} \text{dist}\big(x_{ref}(k), \mathcal{P}_j^{(i)}(k)\big), & x(k) \in \mathcal{M}^{(i)} \\ 0, & \text{otherwise,} \end{cases} \tag{4.7}$$

and rephrase the condition $x_{ref}(k) \in \mathcal{P}^{(i)}(k)$ as $d^{(i)}(k) > 0$.

Using (4.6), definition (4.7) can be translated via the *big-M* technique (see [109], [119]) to the following mixed integer inequalities (in the following we drop the dependence on time k to simplify the notation):

$$\begin{aligned} &\|C_{\mathcal{P},1}^{(i)}\|_2 d^{(i)} \le -\Gamma(2\delta^{(i)} - \mathbf{1})^T\delta - C_{\mathcal{P},1}^{(i)}x_{ref} + d_{\mathcal{P},1}^{(i)} + \Gamma\|\delta^{(i)}\|_1 \\ &\quad\vdots \qquad\qquad\qquad\qquad \vdots \\ &\|C_{\mathcal{P},J^{(i)}}^{(i)}\|_2 d^{(i)} \le -\Gamma(2\delta^{(i)} - \mathbf{1})^T\delta - C_{\mathcal{P},J^{(i)}}^{(i)}x_{ref} + d_{\mathcal{P},J^{(i)}}^{(i)} + \Gamma\|\delta^{(i)}\|_1 \end{aligned} \tag{4.8}$$

$$\begin{aligned} &d^{(i)} \le \Gamma(2\delta_1^{(i)} - 1)\delta_1 - \Gamma\delta_1^{(i)} + \Gamma \\ &d^{(i)} \ge -\Gamma(2\delta_1^{(i)} - 1)\delta_1 + \Gamma\delta_1^{(i)} - \Gamma \\ &\quad\vdots \qquad\qquad\qquad\qquad \vdots \\ &d^{(i)} \le \Gamma(2\delta_\kappa^{(i)} - 1)\delta_\kappa - \Gamma\delta_\kappa^{(i)} + \Gamma \\ &d^{(i)} \ge -\Gamma(2\delta_\kappa^{(i)} - 1)\delta_\kappa + \Gamma\delta_\kappa^{(i)} - \Gamma, \end{aligned} \tag{4.9}$$

where Γ is a large enough constant that can be computed exactly (see [109]). It is easy to verify that the first (second) set of inequalities becomes tight if and only if the state x_{ref} belongs (does not belong) to mode $\mathcal{M}^{(i)}$, i.e., when the binary variable δ is (not) equal to $\delta^{(i)}$. For example, let us take the right hand side of equation (4.8): the term $-\Gamma(2\delta^{(i)} - \mathbf{1})^T\delta = -\Gamma\sum_{j=1}^{\kappa}(2\delta_j^{(i)} - 1)\delta_j$ is always equal to or greater than $-\Gamma\|\delta^{(i)}\|_1$, and, in particular, is equal to $-\Gamma\|\delta^{(i)}\|_1$ if and only if $\delta = \delta^{(i)}$. This makes the sum $-\Gamma(2\delta^{(i)} - \mathbf{1})^T\delta + \Gamma\|\delta^{(i)}\|_1$ equal to 0 if and only if $\delta = \delta^{(i)}$ (tight constraint), and in all other cases greater than Γ (loose constraint).

A similar reasoning can be applied to (4.9): for them to become tight, and therefore enforce $d^{(i)} = 0$, it suffices that only one element of the vector δ differs from the corresponding element in the vector $\delta^{(i)}$. Therefore, if we consider the set of constraints (4.8) and (4.9) for all modes $i \in \{1, \dots, s\}$, we obtain the condition that all the scalars $d^{(i)}$ are equal to 0, except the one corresponding to the mode containing the state x_{ref}, which assumes the value of the minimum distance of x_{ref} from the closest boundary.

We determine the reference input $u_{ref}(0), \dots, u_{ref}(N-1)$ by maximizing the sum of all $d^{(i)}(k),\ i \in \{1, \dots, s\}$, at all time steps $k \in \{1, \dots, N-1\}$ to stay far from mode and constraint

set boundaries, while imposing that $x_{ref}(N)$ gets close to the desired final state $x^{(f)}$ according to the 1-norm and minimizing the inputs:

$$\min_{u_{ref}(0),\dots,u_{ref}(N-1)} \|x_{ref}(N) - x^{(f)}\|_1 + \gamma \sum_{k=0}^{N-1} \|u_{ref}(i)\|_1 - \gamma_d \sum_{k=0}^{N-1} \sum_{i=1}^{s} d^{(i)}(k) \tag{4.10}$$

$$\text{s.t. } V(k) \begin{bmatrix} x_{ref}(k) \\ d^{(1)}(k) \\ \vdots \\ d^{(s)}(k) \\ \delta(k) \end{bmatrix} \leq R(k), \ \forall k \in \{1, \dots, N\},$$

$$\text{MLD dynamics (4.5)}, \ \forall k \in \{0, \dots, N-1\}, \text{ with } x_{ref}(0) = c_{x,0},$$

$$x_{ref}(k) \in \mathcal{S}_k,$$

$$u_{ref}(k) \in \mathcal{U},$$

with weight $\gamma_d \in \mathbb{R}_0^+$ and with $V(k)$ and $R(k)$ as

$$V = \begin{bmatrix} \begin{bmatrix} P_A^{(1)} \\ \mathbf{0}_{2\kappa,n} \end{bmatrix} & H^{(1)} & & \mathbf{0} & D^{(1)} \\ \vdots & & \ddots & & \vdots \\ \begin{bmatrix} P_A^{(s)} \\ \mathbf{0}_{2\kappa,n} \end{bmatrix} & \mathbf{0} & & H^{(s)} & D^{(s)} \end{bmatrix}, \ R = \begin{bmatrix} R^{(1)} \\ R^{(2)} \\ \vdots \\ R^{(s)} \end{bmatrix},$$

where $\mathbf{0}_{2\kappa,n}$ denotes a $2\kappa \times n$ matrix containing only zeros and with

$$H^{(i)} = \begin{bmatrix} \|C_{\mathcal{P},1}^{(i)}\|_2 \\ \|C_{\mathcal{P},2}^{(i)}\|_2 \\ \vdots \\ \|C_{\mathcal{P},J^{(i)}}^{(i)}\|_2 \\ 1 \\ -1 \\ 1 \\ -1 \\ \vdots \\ 1 \\ -1 \end{bmatrix}, \quad R^{(i)} = \begin{bmatrix} d_{\mathcal{P},1}^{(i)} + \Gamma\|\delta^{(i)}\| \\ d_{\mathcal{P},2}^{(i)} + \Gamma\|\delta^{(i)}\| \\ \vdots \\ d_{\mathcal{P},J^{(i)}}^{(i)} + \Gamma\|\delta^{(i)}\| \\ \Gamma - \Gamma\delta_1^{(i)} \\ \Gamma - \Gamma\delta_1^{(i)} \\ \Gamma - \Gamma\delta_2^{(i)} \\ \Gamma - \Gamma\delta_2^{(i)} \\ \vdots \\ \Gamma - \Gamma\delta_\kappa^{(i)} \\ \Gamma - \Gamma\delta_\kappa^{(i)} \end{bmatrix},$$

and

$$D^{(i)} = \begin{bmatrix} & \Gamma(2\delta^{(i)} - \mathbf{1})^T & & \\ & \Gamma(2\delta^{(i)} - \mathbf{1})^T & & \\ & \vdots & & \\ & \Gamma(2\delta^{(i)} - \mathbf{1})^T & & \\ \Gamma(1 - 2\delta_1^{(i)}1) & 0 & \dots & 0 \\ \Gamma(1 - 2\delta_1^{(i)}) & 0 & \dots & 0 \\ 0 & \Gamma(1 - 2\delta_2^{(i)}) & \dots & 0 \\ 0 & \Gamma(1 - 2\delta_2^{(i)}) & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \Gamma(1 - 2\delta_\kappa^{(i)}) \\ 0 & 0 & \dots & \Gamma(1 - 2\delta_\kappa^{(i)}) \end{bmatrix}.$$

Since the 1-norm can be rewritten as a linear term by introducing auxiliary variables and linear constraints [92, 93], problem (4.10) is an MILP. Note that to compute the reference trajectory, we are not adding extra binary variables other than the δ variables that are already embedded in the MLD formulation to represent the modes structure.

4.5 Set-Based Control Design

In this section, we address the synthesis of the state feedback control law $u_{ctrl}(x, k)$ (Fig. 4.2(b)–(d)). We discussed in Sec. 3.3.5, how optimizing over a longer horizon and only using the control law for the first (few) time interval(s) can be beneficial. Here, we explicitly consider this case by operating according to a receding horizon strategy, i.e., we design the control inputs $u_{ctrl,h}(x, w(\cdot), k)$ that make the system robustly track the reference trajectory along the look-ahead time horizon $k \in \{h + 1, \cdots, h + M\}$, starting from $x \in \mathcal{X}^*(h)$ at time h, and then use the first part for the time-dependent overall control strategy $u_{ctrl}(\cdot)$ at time h, i.e.,

$$u_{ctrl}(x, h) := u_{ctrl,h}(x, w(\cdot), h), \ \forall x \in \mathcal{X}^*(h). \tag{4.11}$$

As we see later, by assigning only the first input of $u_{ctrl,h}(x, w(\cdot), h)$, the control law $u_{ctrl}(x, h)$ remains independent of the actual disturbance realization, i.e., our assumption that the disturbance cannot be measured still holds.

In this chapter, we explicitly consider possible future disturbances during the optimization of the control inputs, as mode switches can strongly depend on the disturbance effects. For an easier presentation in the previous chapter, we considered the nominal dynamics during the optimization of the control inputs and only considered the disturbances during the reachable set computation. The same ideas presented here can also be used for the controllers for continuous dynamics from the previous chapter. In the case of nonlinear dynamics, it might be more challenging to get a good estimate for the disturbance effects due to linearization. However, due to the following verification through reachability analysis, it suffices to get approximations of the linearization errors, similar to the idea that the linearized dynamics only approximate the actual nonlinear dynamics.

Let us now consider the computation of the control law for one time instance. For a simpler notation and without loss of generality, we consider the computation at time $h = 0$. Therefore, we start from $\mathcal{X}_0 = \langle c_{x,0}, g_{x,0}^{(1)}, \dots, g_{x,0}^{(p_0)} \rangle$. At later times, we start from $\mathcal{X}^*(h)$. By construction, $\mathcal{X}_0$ as well as $\mathcal{X}^*(h)$ always belongs to a single mode.

Let $\rho : \mathbb{R}^n \to \{1, \dots, s\}$ be the map that associates the number of the corresponding active mode to $x \in \mathbb{R}^n$, i.e., $\rho(x) = i \Leftrightarrow x \in \mathcal{M}^{(i)}$. With this map, we express the evolution of a single state $x^{(0)} \in \mathcal{X}_0$ as a function of the control input, based on our assumption that $x(k) \in \mathcal{M}^{\rho(x_{ref}(k))}$, $\forall k \in \{0, \dots, M\}$, i.e., it stays in the main mode sequence. For ease of notation, we denote by $A_j, B_{u,j}, B_{w,j}, f_j$ matrices and vectors active in the mode $\rho(x_{ref}(j))$. This allows us to obtain a similar notation to the one from Sec. 3.4, and we can write the trajectory at time k starting from $x^{(0)} \in \mathcal{X}_0$ similar to (3.24) as:

$$\xi(x^{(0)}, u_{ctrl,0}(\,\cdot\,), w(\,\cdot\,), k) = \bar{A}x^{(0)} + \sum_{j=0}^{k-1} \left(\bar{B}_{u,j} u_{ctrl,0}(x^{(0)}, w(\,\cdot\,), j) + \bar{B}_{w,j} w(j) + \bar{A}_{f,j} f_j \right) \tag{4.12}$$

for $k \in \{1, \dots, M\}$, where we use the shorthand notations:

$$\begin{aligned} \bar{A} &:= A_{k-1} \dots A_0, \\ \bar{A}_{f,j} &:= A_{k-1} \dots A_{j+1}, \quad j \in \{0, \cdots, k-2\}, \end{aligned}$$

with $\bar{A}_{f,k-1} := I$, i.e., the identity matrix,

$$\bar{B}_{u,j} := A_{k-1} \dots A_{j+1} B_{u,j}, \quad j \in \{0, \cdots, k-2\},$$

with $\bar{B}_{u,k-1} := B_{u,k-1}$, and $\bar{B}_{w,j}$ being defined analogously to $\bar{B}_{u,j}$. Using the zonotope representation of the initial set $\mathcal{X}_0 = \langle c_{x,0}, g_{x,0}^{(1)}, \dots, g_{x,0}^{(p_0)} \rangle$ and of the disturbance set $\mathcal{W} = \langle \mathbf{0}, g_w^{(1)}, \dots, g_w^{(r)} \rangle$, we can express (4.12) analogous to (3.25) as

$$\begin{aligned} \xi(x^{(0)}, u_{ctrl,0}(\,\cdot\,), w(\,\cdot\,), k) = \bar{A} &\left(c_{x,0} + \sum_{i=1}^{p_0} \alpha_i(x^{(0)}) g_{x,0}^{(i)} \right) \\ + \sum_{j=0}^{k-1} &\left(\bar{B}_{u,j} u_{ctrl,0}(x^{(0)}, w(\,\cdot\,), j) + \bar{B}_{w,j} \sum_{i=1}^{r} \beta_i(w(j)) g_w^{(i)} + \bar{A}_{f,j} f_j \right), \end{aligned} \tag{4.13}$$

where $\alpha_i(x^{(0)})$ is such that

$$x^{(0)} = c_{x,0} + \sum_{i=1}^{p_0} \alpha_i(x^{(0)}) g_{x,0}^{(i)}$$

and $\beta_i(w(j))$ is such that

$$w(j) = \sum_{i=1}^{r} \beta_i(w(j)) g_w^{(i)}.$$

We obtain the same structure as in Sec. 3.4, with the difference that we explicitly consider the disturbance effects here. Therefore, we can use a similar structure for the controller as in (3.27), which enables us to control every trajectory starting from $\mathcal{X}_0$ for any disturbance trajectory:

$$u_{ctrl,0}(x^{(0)}, w(\cdot), j) = u(c_{x,0}, j) + \sum_{i=1}^{p_0} \alpha_i(x^{(0)}) u\big(g_{x,0}^{(i)}, j\big) + \sum_{l=0}^{j-1} \sum_{i=1}^{r} \beta_i(w(l)) u\big(g_w^{(i)}, l, j\big). \quad (4.14)$$

where

$$\sum_{l=0}^{-1} \sum_{i=1}^{r} \beta_i(w(l)) u\big(g_w^{(i)}, l, 0\big) := \mathbf{0}, \quad (4.15)$$

i.e., we can only react to disturbances which happened in the past. New disturbances are added at each time point. Since they are always from the same set of disturbances, we use $u\big(g_w^{(i)}, l, j\big)$ to denote the control input at time j for the i-th generator of the disturbances which affected the system at time l. As we do not know the exact disturbance trajectories in advance, (4.14) only considers inputs for disturbances of previous time steps $l < j$, which are known at time j. When we use the first time step of this control law (4.14) as an auxiliary controller for our receding horizon implementation (4.11), it therefore depends solely on the initial state and involves only $u(c_{x,0}, 0)$ and $u\big(g_{x,0}^{(1)}, 0\big), \dots, u\big(g_{x,0}^{(p_0)}, 0\big)$. The actual disturbances of previous time steps are contained in the measured initial state, and we consider the set of possible future disturbances to obtain trajectories which satisfy the constraints and avoid splits despite any future disturbances.

We find the input trajectories $u(c_{x,0}, \cdot)$, $u\big(g_{x,0}^{(1)}, \cdot\big), \dots, u\big(g_{x,0}^{(p_0)}, \cdot\big)$, and $u\big(g_w^{(1)}, \cdot, \cdot\big), \dots, u\big(g_w^{(r)}, \cdot, \cdot\big)$ along the finite horizon $[0, M-1]$ by solving a linear program analogous to (3.29)–(3.31) in Sec. 3.4. In the linear program, we minimize the reachable sets around the reference trajectory, minimize the applied inputs, and maximize the distance of the reachable sets to the mode boundaries. The upper bound of the costs for the distance from the reference trajectory and the applied inputs in each time step can be obtained similar to Lemma 2:

Corollary 2. *Let $\mathcal{R}_{k,u_{ctrl,0},\mathcal{W}}(\mathcal{X}_0)$ denote the reachable set resulting from control law (4.14) at time k, then the following inequality holds:*

$$\begin{aligned}
&\left\|\mathcal{R}_{k,u_{ctrl,0},\mathcal{W}}(\mathcal{X}_0) - x_{ref}(k)\right\|_1 + \gamma \left\|u_{ctrl,0}\left(\mathcal{R}_{k-1,u_{ctrl,0},\mathcal{W}}(\mathcal{X}_0), \mathcal{W}, k-1\right)\right\|_1 \\
\leq &\left\|\bar{A}c_{x,0} - x_{ref}(k) + \sum_{j=0}^{k-1}\left(\bar{A}_{f,j} f_j + \bar{B}_{u,j} u(c_{x,0}, j)\right)\right\|_1 + \sum_{i=1}^{p_0}\left\|\bar{A} g_{x,0}^{(i)} + \sum_{j=0}^{k-1} \bar{B}_{u,j} u\big(g_{x,0}^{(i)}, j\big)\right\|_1 \\
&+ \sum_{j=0}^{k-2}\sum_{i=1}^{r}\left\|\bar{B}_{w,j} g_w^{(i)} + \sum_{l=j+1}^{k-1} \bar{B}_{u,l} u\big(g_w^{(i)}, j, l\big)\right\|_1 + \sum_{i=1}^{r}\left\|\bar{B}_{w,k-1} g_w^{(i)}\right\|_1 \\
&+ \gamma\left(\|u(c_{x,0}, k-1)\|_1 + \sum_{i=1}^{p_0}\left\|u\big(g_{x,0}^{(i)}, k-1\big)\right\|_1 + \sum_{j=0}^{k-2}\sum_{i=1}^{r}\left\|u\big(g_w^{(i)}, j, k-1\big)\right\|_1\right). \quad (4.16)
\end{aligned}$$

Proof. Let us introduce the shorthand notations α_i for $\alpha_i(x_0)$ and $\beta_{i,j}$ for $\beta_i(w(j))$. The fol-

lowing holds:

$$
\begin{aligned}
&\left\|\mathcal{R}_{k,u_{ctrl,0},\mathcal{W}}(\mathcal{X}_0) - x_{ref}(k)\right\|_1 + \gamma \left\|u_{ctrl,0}\left(\mathcal{R}_{k-1,u_{ctrl,0},\mathcal{W}}(\mathcal{X}_0), \mathcal{W}, k-1\right)\right\|_1 \\
&\overset{(4.12)}{=} \max_{x^{(0)}\in\mathcal{X}_0, w(\,\cdot\,)\in\mathcal{W}} \left\| \bar{A}x^{(0)} + \sum_{j=0}^{k-1}\left(\bar{B}_{u,j}u_{ctrl,0}(x^{(0)}, w(\,\cdot\,), j) + \bar{B}_{w,j}w(j) + \bar{A}_{f,j}f_j\right) - x_{ref}(k)\right\|_1 \\
&\qquad + \gamma \left\|u_{ctrl,0}(x^{(0)}, w(\,\cdot\,), k-1)\right\|_1 \\
&\overset{\substack{(4.13),\\(4.14)}}{=} \max_{\alpha_i,\beta_{i,j}\in[-1,1]} \left\| \bar{A}\left(c_{x,0} + \sum_{i=1}^{p_0}\alpha_i g_{x,0}^{(i)}\right)\right. \\
&\qquad\quad + \sum_{j=0}^{k-1}\bar{B}_{u,j}\left(u(c_{x,0}, j) + \sum_{i=1}^{p_0}\alpha_i(x^{(0)})u\big(g_{x,0}^{(i)}, j\big) + \sum_{l=0}^{j-1}\sum_{i=1}^{r}\beta_{i,l}u\big(g_w^{(i)}, l, j\big)\right) \\
&\qquad\quad \left. + \sum_{j=0}^{k-1}\left(\bar{B}_{w,j}\sum_{i=1}^{r}\beta_{i,j}g_w^{(i)} + \bar{A}_{f,j}f_j\right) - x_{ref}(k)\right\|_1 \\
&\qquad\quad + \gamma \left\|u(c_{x,0}, k-1) + \sum_{i=1}^{p_0}\alpha_i u\big(g_{x,0}^{(i)}, k-1\big) + \sum_{j=0}^{k-2}\sum_{i=1}^{r}\beta_{i,j}u\big(g_w^{(i)}, j, k-1\big)\right\|_1 \\
&\overset{\substack{\text{see}\\\text{below}}}{=} \max_{\alpha_i,\beta_{i,j}\in[-1,1]} \left\| \bar{A}c_{x,0} + \sum_{j=0}^{k-1}\bar{B}_{u,j}u(c_{x,0}, j) + \sum_{i=1}^{p_0}\left(\bar{A}\alpha_i g_{x,0}^{(i)} + \sum_{j=0}^{k-1}\bar{B}_{u,j}\alpha_i u\big(g_{x,0}^{(i)}, j\big)\right)\right. \\
&\qquad\quad + \sum_{j=0}^{k-1}\sum_{i=1}^{r}\bar{B}_{w,j}\beta_{i,j}g_w^{(i)} + \sum_{j=0}^{k-2}\sum_{i=1}^{r}\sum_{l=j+1}^{k-1}\bar{B}_{u,l}\beta_{i,j}u\big(g_w^{(i)}, j, l\big) + \sum_{j=0}^{k-1}\bar{A}_{f,j}f_j - x_{ref}(k)\Big\|_1 \\
&\qquad\quad + \gamma \left\|u(c_{x,0}, k-1) + \sum_{i=1}^{p_0}\alpha_i u\big(g_{x,0}^{(i)}, k-1\big) + \sum_{j=0}^{k-2}\sum_{i=1}^{r}\beta_{i,j}u\big(g_w^{(i)}, j, k-1\big)\right\|_1 \\
&\overset{\substack{\text{triangle}\\\text{inequal.}}}{\leq} \max_{\alpha_i,\beta_{i,j}\in[-1,1]} \left\| \bar{A}c_{x,0} - x_{ref}(k) + \sum_{j=0}^{k-1}\left(\bar{A}_{f,j}f_j + \bar{B}_{u,j}u(c_{x,0}, j)\right)\right\|_1 \\
&\qquad\quad + \sum_{i=1}^{p_0}\left\|\alpha_i\left(\bar{A}g_{x,0}^{(i)} + \sum_{j=0}^{k-1}\bar{B}_{u,j}u\big(g_{x,0}^{(i)}, j\big)\right)\right\|_1 \\
&\qquad\quad + \sum_{j=0}^{k-2}\sum_{i=1}^{r}\left\|\beta_{i,j}\left(\bar{B}_{w,j}g_w^{(i)} + \sum_{l=j+1}^{k-1}\bar{B}_{u,l}u\big(g_w^{(i)}, j, l\big)\right)\right\|_1 + \sum_{i=1}^{r}\left\|\beta_{i,k-1}\bar{B}_{w,k-1}g_w^{(i)}\right\|_1 \\
&\qquad\quad + \gamma\left(\|u(c_{x,0}, k-1)\|_1 + \sum_{i=1}^{p_0}\left\|\alpha_i u\big(g_{x,0}^{(i)}, k-1\big)\right\|_1 + \sum_{j=0}^{k-2}\sum_{i=1}^{r}\left\|\beta_{i,j}u\big(g_w^{(i)}, j, k-1\big)\right\|_1\right)
\end{aligned}
$$

$$
\begin{aligned}
&= \max_{\alpha_i,\beta_{i,j}\in[-1,1]} \left\| \bar{A}c_{x,0} - x_{ref}(k) + \sum_{j=0}^{k-1} \left(\bar{A}_{f,j} f_j + \bar{B}_{u,j} u(c_{x,0}, j)\right) \right\|_1 \\
&\quad + \sum_{i=1}^{p_0} \underbrace{\|\alpha_i\|_1}_{\leq 1} \left\| \bar{A} g_{x,0}^{(i)} + \sum_{j=0}^{k-1} \bar{B}_{u,j} u\big(g_{x,0}^{(i)}, j\big) \right\|_1 \\
&\quad + \sum_{j=0}^{k-2} \sum_{i=1}^{r} \underbrace{\|\beta_{i,j}\|_1}_{\leq 1} \left\| \bar{B}_{w,j} g_w^{(i)} + \sum_{l=j+1}^{k-1} \bar{B}_{u,l} u\big(g_w^{(i)}, j, l\big) \right\|_1 + \sum_{i=1}^{r} \underbrace{\|\beta_{i,k-1}\|_1}_{\leq 1} \left\| \bar{B}_{w,k-1} g_w^{(i)} \right\|_1 \\
&\quad + \gamma \left(\|u(c_{x,0}, k-1)\|_1 + \sum_{i=1}^{p_0} \underbrace{\|\alpha_i\|_1}_{\leq 1} \left\| u\big(g_{x,0}^{(i)}, k-1\big) \right\|_1 + \sum_{j=0}^{k-2} \sum_{i=1}^{r} \underbrace{\|\beta_{i,j}\|_1}_{\leq 1} \left\| u\big(g_w^{(i)}, j, k-1\big) \right\|_1 \right) \\
&= \left\| \bar{A}c_{x,0} - x_{ref}(k) + \sum_{j=0}^{k-1} \left(\bar{A}_{f,j} f_j + \bar{B}_{u,j} u(c_{x,0}, j)\right) \right\|_1 + \sum_{i=1}^{p_0} \left\| \bar{A} g_{x,0}^{(i)} + \sum_{j=0}^{k-1} \bar{B}_{u,j} u\big(g_{x,0}^{(i)}, j\big) \right\|_1 \\
&\quad + \sum_{j=0}^{k-2} \sum_{i=1}^{r} \left\| \bar{B}_{w,j} g_w^{(i)} + \sum_{l=j+1}^{k-1} \bar{B}_{u,l} u\big(g_w^{(i)}, j, l\big) \right\|_1 + \sum_{i=1}^{r} \left\| \bar{B}_{w,k-1} g_w^{(i)} \right\|_1 \\
&\quad + \gamma \left(\|u(c_{x,0}, k-1)\|_1 + \sum_{i=1}^{p_0} \left\| u\big(g_{x,0}^{(i)}, k-1\big) \right\|_1 + \sum_{j=0}^{k-2} \sum_{i=1}^{r} \left\| u\big(g_w^{(i)}, j, k-1\big) \right\|_1 \right).
\end{aligned}
$$

Note that at one point, we switched the indices of $u\big(g_w^{(i)}, l, j\big)$ to $u\big(g_w^{(i)}, j, l\big)$ for a better presentation. In the control law (4.14), we express which inputs are used at each time step j to counteract the disturbance effects of previous time steps $j < l$. In order to show which inputs are used to counteract each disturbance generator, we rewrite the sums after the third equality sign. There, we used the fact that the following changes hold for any function $f(j,l)$ which depends on the parameters j and l :

$$
\begin{aligned}
&\sum_{j=1}^{k-1} \sum_{l=0}^{j-1} f(j,l) \\
&= \sum_{l=0}^{0} f(1,l) + \sum_{l=0}^{1} f(2,l) \cdots + \sum_{l=0}^{k-2} f(k-1,l) \\
&= \big(f(1,0)\big) + \big(f(2,0) + f(2,1)\big) + \cdots + \big(f(k-1,0) + f(k-1,1) + \cdots + f(k-1,k-2)\big) \\
&= \big(f(1,0) + f(2,0) + \cdots + f(k-1,0)\big) + \big(f(2,1) + \cdots + f(k-1,1)\big) + \cdots + \big(f(k-1,k-2)\big) \\
&= \sum_{j=1}^{k-1} f(j,0) + \sum_{j=2}^{k-1} f(j,1) + \cdots + \sum_{j=k-1}^{k-1} f(j,k-2) \\
&= \sum_{l=0}^{k-2} \sum_{j=l+1}^{k-1} f(j,l)
\end{aligned}
$$

Note that we start the sum with $j = 1$, as $u\big(g_w^{(i)}, l, 0\big) = \mathbf{0}$, $\forall l \in \mathbb{N}_0$, see (4.15).

□

We use $cost(\mathcal{R}_{k,u_{ctrl,0},\mathcal{W}}(\mathcal{X}_0))$ to refer to the upper bound of the cost function (4.16). In contrast to the minimization in Sec. 3.4, where we optimized the final set after the whole time horizon in the linear case and the reachable set after a single iteration step in the nonlinear case, we use a receding horizon in this chapter. As common for receding horizon optimization such as MPC, we therefore consider the sizes of the reachable sets at each time step. Thus, we state the cost function for a single time step and later minimize the sum of the cost terms over the whole time horizon.

The result of Corollary 2 can be interpreted graphically: as shown in Fig. 4.3, we exploit the available control capabilities and steer the center of the initial set towards the reference state and the generators towards the origin, thereby aiming at bringing all states close to $x_{ref}(k)$. In contrast to the approach from Sec. 3.4 and its illustration in Fig. 3.8, we consider additional generators for the disturbances. Since we can only counteract disturbances which happened in previous time steps, the disturbances at the last time step $k-1$ appear unaffected in (4.16).

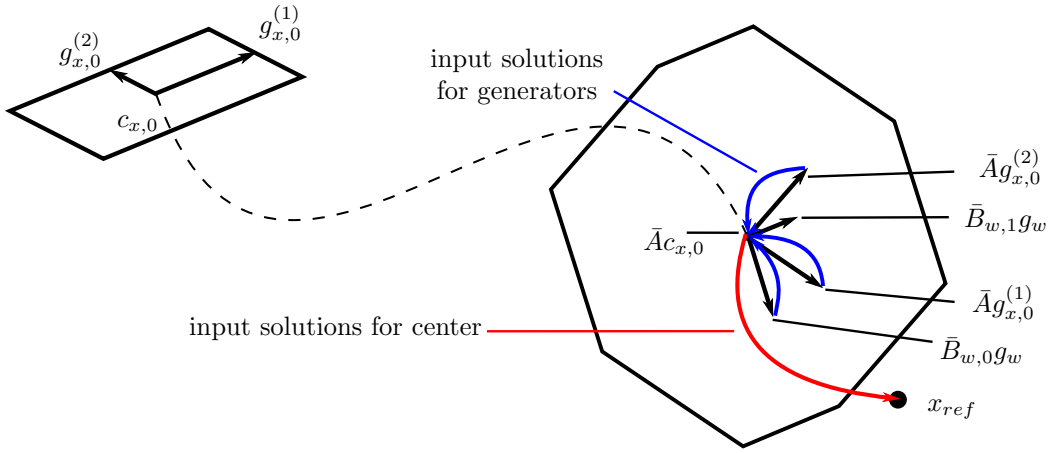


Figure 4.3: Illustration of the set-based control approach similar to Fig. 3.8, but with explicit consideration of disturbances: We obtain a small reachable set close to $x_{ref}(k)$ by steering the center of the initial set to $x_{ref}(k)$, and the generators of the initial set as well as those resulting from disturbances to the origin (here for the disturbance set $\langle \mathbf{0}, g_w \rangle$ and a horizon of two). The last disturbance generator $\bar{B}_{w,1}g_w$ is not affected, as we cannot directly measure the disturbances.

In addition to the costs from the size of the final set and the applied inputs, we also wish to stay away from the mode boundaries to minimize the number of splits. To achieve this, we weight the minimal distance of the reachable sets to the mode boundaries, similarly to what we do in the optimization of the reference trajectory in (4.8):

$$cost(d) := -\sum_{k=1}^{M} d(k) \tag{4.17}$$

$$\text{s.t. } \forall k \in \{1,\dots,M\},\ \forall j \in \{1,\dots,v(k)\}:$$

$$\|C_{\mathcal{M},j}(k)\|_2 d(k) \leq C_{\mathcal{M},j}(k)c_{x,k} + \sum_{i=1}^{p_k} |C_{\mathcal{M},j}(k)g^{(i)}_{x,k}| - d_{\mathcal{M},j}(k), \tag{4.18}$$

where $C_{\mathcal{M},j}(k)$ and $d_{\mathcal{M},j}(k)$ represent the j-th row (of $v(k)$ rows total) of the matrix and vector defining the half-space representation of $\mathcal{M}_k$, i.e., the mode in the main switching sequence at time k, and $c_{x,k}, g_{x,k}^{(1)}, \dots, g_{x,k}^{(p_k)}$ denote the center and generators of the reachable sets $\mathcal{R}_{k,u_{ctrl,0},\mathcal{W}}(\mathcal{X}_0)$. Since additional generators from the disturbance effects are added in every time step, the number of generators p_k of the reachable sets increases over time.

We use Lemma 1 to check the state constraints $\mathcal{S}_k = \langle C_{\mathcal{S}_k}, d_{\mathcal{S}_k} \rangle_H$ with

$$C_{\mathcal{S}_k} c_{x,k} + \sum_{i=1}^{p_k} |C_{\mathcal{S}_k} g_{x,k}^{(i)}| \leq d_{\mathcal{S}_k}. \tag{4.19}$$

To satisfy the input constraints, we have to ensure that the sum of all possible inputs for the center and generators does not exceed the input bounds. This can be checked analogously to Corollary 1:

Corollary 3. *The control law* $u_{ctrl,0}(x^{(0)}, w(\,\cdot\,), j)$ *from* (4.14) *satisfies the input constraints* $\mathcal{U} = \{u \in \mathbb{R}^m \mid C_{\mathcal{U}} u \leq d_{\mathcal{U}}\}$ *if* $\forall j \in \{0, \dots, M-1\}$:

$$C_{\mathcal{U}} u(c_{x,0}, j) + \sum_{i=1}^{p_0} |C_{\mathcal{U}} u\big(g_{x,0}^{(i)}, j\big)| + \sum_{l=0}^{j-1} \sum_{i=1}^{r} |C_{\mathcal{U}} u\big(g_w^{(i)}, l, j\big)| \leq d_{\mathcal{U}}. \tag{4.20}$$

Proof. Using the same proof concept as in Corollary 1, it follows that

$$\begin{aligned} & u(c_{x,0}, j) + \sum_{i=1}^{p_0} \alpha_i u\big(g_{x,0}^{(i)}, j\big) + \sum_{l=0}^{j-1} \sum_{i=1}^{r} \beta_{i,l} u\big(g_w^{(i)}, l, j\big) \in \mathcal{U}, \\ & \qquad \forall \alpha \in [-1,1]^{p_0},\ \beta \in [-1,1]^{r \times j},\ \forall j \in \{0, \dots M-1\} \\ \Leftrightarrow\ & C_{\mathcal{U}} u(c_{x,0}, j) + \sum_{i=1}^{p_0} |C_{\mathcal{U}} u\big(g_{x,0}^{(i)}, j\big)| + \sum_{l=0}^{j-1} \sum_{i=1}^{r} |C_{\mathcal{U}} u\big(g_w^{(i)}, l, j\big)| \leq d_{\mathcal{U}},\ \forall j \in \{0, \dots, M-1\}. \end{aligned}$$

□

Using (4.17) together with the results from Corollary 2, Lemma 1, and Corollary 3, allows us to express everything in a single optimization problem similar to optimization problem (3.29)–(3.30):

$$\begin{aligned} \min_{u(c_{x,0},\,\cdot\,), u(g_{x,0}^{(i)},\,\cdot\,), u(g_w^{(i)},\,\cdot\,)} & \sum_{k=1}^{M} cost(\mathcal{R}_{k,u_{ctrl,0},\mathcal{W}}(\mathcal{X}_0)) + \gamma_d\ cost(d) \\ \text{s.t. } \forall k \in \{1, \dots, M\} &: (4.18),\ (4.19),\ (4.20), \end{aligned} \tag{4.21}$$

where $\gamma_d \in \mathbb{R}_0^+$ is a weight used to adjust the importance of being far away from the mode boundaries. Our new optimization problem has a similar structure as the optimization problem (3.29)–(3.30). Since the only additional cost term and constraint are both linear, we obtain again a single linear program.

Problem (4.21) provides the inputs $u(c_{x,0},\,\cdot\,)$ and $u(g_{x,0}^{(i)},\,\cdot\,)$, which can be used to compute

i) the feedback law $u_{ctrl}(x,0)$ at time 0 via (4.11):

$$u_{ctrl}(x,0) = u(c_{x,0},0) + \sum_{i=1}^{p_0} \alpha_i(x) u(g_{x,0}^{(i)},0), \tag{4.22}$$

where $\alpha_i(x)$, $i \in \{1,\dots,p_0\}$, are such that

$$x = c_{x,0} + \sum_{i=1}^{p_0} \alpha_i(x) g_{x,0}^{(i)}; \tag{4.23}$$

ii) the reachable set $\mathcal{X}^*(1) = \mathcal{R}_{1,u_{ctrl},\mathcal{W}}(\mathcal{X}^*(0)) \cap \mathcal{M}_1$, where $\mathcal{M}_1$ is the mode to which $x_{ref}(1)$ belongs (see Step 2 in Section 4.3).

Splits and Recovery

The procedure described above is iteratively repeated for N time steps (with a shrinking horizon of $N-k$ for the last M iterations). Afterwards, we check for each split with any mode that has occurred (Fig. 4.2(d)) whether the split set can be steered back into the main sequence or not (Figs. 4.2(e) and (f), respectively). We do this by solving optimization problem (4.21) for a time horizon of M_{split}, starting at the time step k_0 when the split occurred and with the additional constraints

$$d(k) \geq 0, \forall k \in \{k_0+1,\dots,k_0+M_{split}\}$$

and

$$\mathcal{X}^{split}(k_0+M_{split}) \subseteq \mathcal{X}^*(k_0+M_{split}),$$

where we denote by $\mathcal{X}^{split}(k_0+M_{split})$ the reachable set starting from the split with a single mode.

These additional constraints ensure that the reachable set of the split set is steered back into the main switching sequence after one time step and completely stays inside this mode sequence until it is a subset of the main reachable set $\mathcal{X}^*(k_0+M_{split})$. Therefore, we only need to solve a single linear program inside the main switching sequence. Without these constraints, we would have to consider the combinatorial problem of all possible modes the set could enter and therefore the different dynamics which could be applied. This would lead to an MILP similar to the reference trajectory. The second constraint $\mathcal{X}^{split}(k_0+M_{split}) \subseteq \mathcal{X}^*(k_0+M_{split})$ can be easily checked using the half-space representation of $\mathcal{X}^*(k_0+M_{split})$ together with Lemma 1. Note that if $k_0+M_{split} < N$, we use the horizon $N-k_0$ instead.

If the optimization problem is not feasible, we cannot steer the reachable set back in M_{split} steps along the main switching sequence. In this case, we start a new optimization problem with a new reference trajectory and with a reduced time horizon of $N-k_0$ steps.

Obtaining the Online Control Law

The outcome of the overall control design procedure described in this section is a collection of zonotopes per time step representing the reachable set (possibly split over different modes

and over-approximated) and the associated set-based controllers. Both of them are finitely parameterized and have a low space complexity (which scales linearly with the number of states, time steps, and splits). Analogous to the previous chapter, the controller and corresponding reachable set can be stored as a motion primitive for a maneuver automaton.

Once the value x of the state at time k is available during online application, it is possible to determine the input $u_{ctrl}(x, k)$ using (4.22). Several possibilities exist for obtaining the parameters $\alpha_i(x)$ in (4.22): the easiest way is by solving the system of inequalities (4.23). Since this is a linear feasibility program, it can be solved fast online.

If faster computation times are required, then we can use the closed-form expressions of convex combinations from Appendix A: by interpolating the α values of the extreme points of our zonotope, we can use this technique to obtain them through a closed-form expression. It is also possible to over-approximate the reachable set in each time step by a parallelotope, analogously to the nonlinear case of the generator interpolation controller in Sec. 3.4. In the case of a parallelotope, we can obtain the α values by simply inverting the generator matrix. This allows us to obtain linear controllers, which work analogously to the disturbance feedback controllers discussed in [148].

4.6 Numerical Example

We demonstrate the effectiveness of our approach on the quadruple tank benchmark described in [149]. The system is nonlinear with 4 states $h_1, \dots, h_4$ and 2 control inputs v_1, v_2, that denote the tank levels and the voltage inputs to the two pumps, respectively. The equations of the system are

$$\begin{aligned}
\dot{h}_1 &= -\frac{a_1}{A_1}\sqrt{2gh_1} + \frac{a_3}{A_1}\sqrt{2gh_3} + \frac{\gamma_1 k_1}{A_1}v_1, \\
\dot{h}_2 &= -\frac{a_2}{A_2}\sqrt{2gh_2} + \frac{a_4}{A_2}\sqrt{2gh_4} + \frac{\gamma_2 k_2}{A_2}v_2, \\
\dot{h}_3 &= -\frac{a_3}{A_3}\sqrt{2gh_3} + \frac{(1-\gamma_2) k_2}{A_3}v_2, \\
\dot{h}_4 &= -\frac{a_4}{A_4}\sqrt{2gh_4} + \frac{(1-\gamma_1) k_1}{A_4}v_1,
\end{aligned}$$

where the values of the various parameters can be found in [149]. The states are constrained between $0\,cm$ and $20\,cm$, and the inputs are bounded between $0\,V$ and $12\,V$. To obtain a PWA system, we linearize the dynamics around different points and consider the approximation error as an additive disturbance that we want to counteract (one disturbance for each dynamics equation). The state space is therefore partitioned into 16 modes obtained by splitting each axis into the two intervals $[0, 3]\,cm$ and $(3, 20]\,cm$. For each mode, a linear approximation is obtained by Taylor expansion around the centers of the intervals. We bound the resulting linearization errors and include them with the external disturbances in a disturbance set $\mathcal{W} = [-1, 2]^4\,V$. We model the disturbances acting on the system through the same input matrix as the inputs, i.e., $B_w^{(s)} = B_u^{(s)},\ \forall s \in \{1, \dots, 16\}$.

We choose a sampling time of 1 second, $N = 20$ for the time horizon, $M = 3$ for the prediction horizon, and $M_{split} = 7$ for the horizon after a split. The time-varying state constraints are

$$\left.\begin{array}{ll} 0.55k\,cm \le h_i(k) \le 5 + 0.55k\,cm, & i = 1 \\ 0.15k\,cm \le h_i(k) \le 2 + \dfrac{k^2}{16}\,cm, & i = 2 \\ 4\,cm \le h_i(k) \le 13\,cm, & i = 3 \\ 4\,cm \le h_i(k) \le 11\,cm, & i = 4 \end{array}\right\} \ \forall k \in \{1,\dots,10\},$$
$$3\,cm \le h_i(k) \le 10\,cm, \quad \forall i \in \{1,\dots,4\} \ \} \ \forall k \in \{11,\dots,20\}.$$

The initial set $\mathcal{X}_0$ is given as a box with edges of lengths $[0.6\,cm, 0.6\,cm, 0.2\,cm, 0.2\,cm]$ around its center $c_{x,0} = [1\,cm, 1\,cm, 12\,cm, 7\,cm]^T$ and the terminal state $x^{(f)}$ is set to $[9\,cm, 6.5\,cm, 6.5\,cm, 6.5\,cm]^T$.

Results

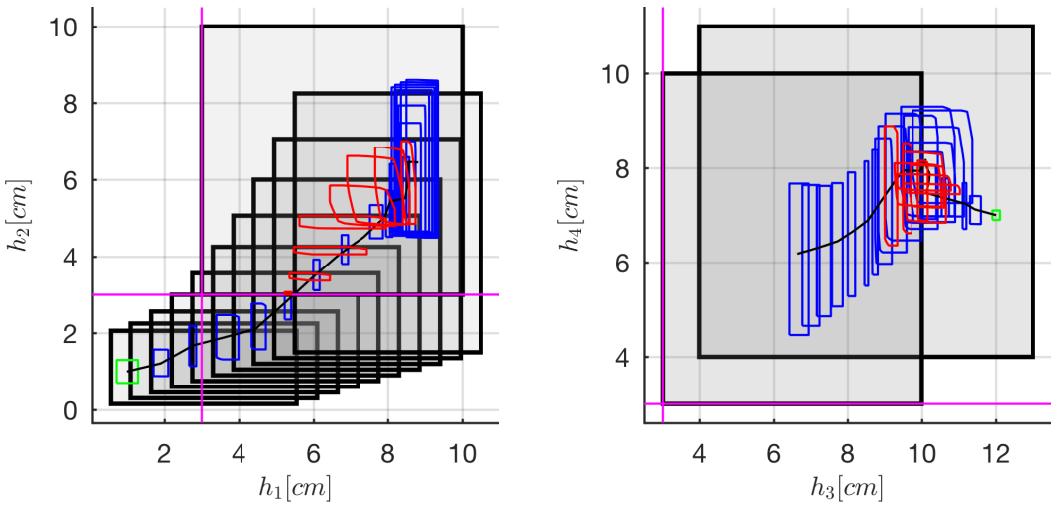


Figure 4.4: Results from numerical example: Reference trajectory (black), initial set (green), main reachable sets (blue), and state constraints (gray) at all time steps, projected onto the (h_1, h_2) plane (left) and (h_3, h_4) plane (right). The mode boundaries are plotted in magenta. At time $k = 5$, a split set originates from an intersection with the boundary $h_2 = 3\,cm$ (in red). The corresponding reachable sets (in red) merge back with the main reachable set after $M_{split} = 7$ steps.

We run our algorithm on the same computer as in the previous chapter, see Sec. 3.3.7; the MILP for the reference trajectory computation is solved via CPLEX, whereas the linear programming problems for the set-based controller computation are solved as in Sec. 3.4.3 with CVX using the solver SDPT3. As in the previous chapter, we again use the CORA toolbox for the zonotope computations. The overall computations take 2.5 minutes.

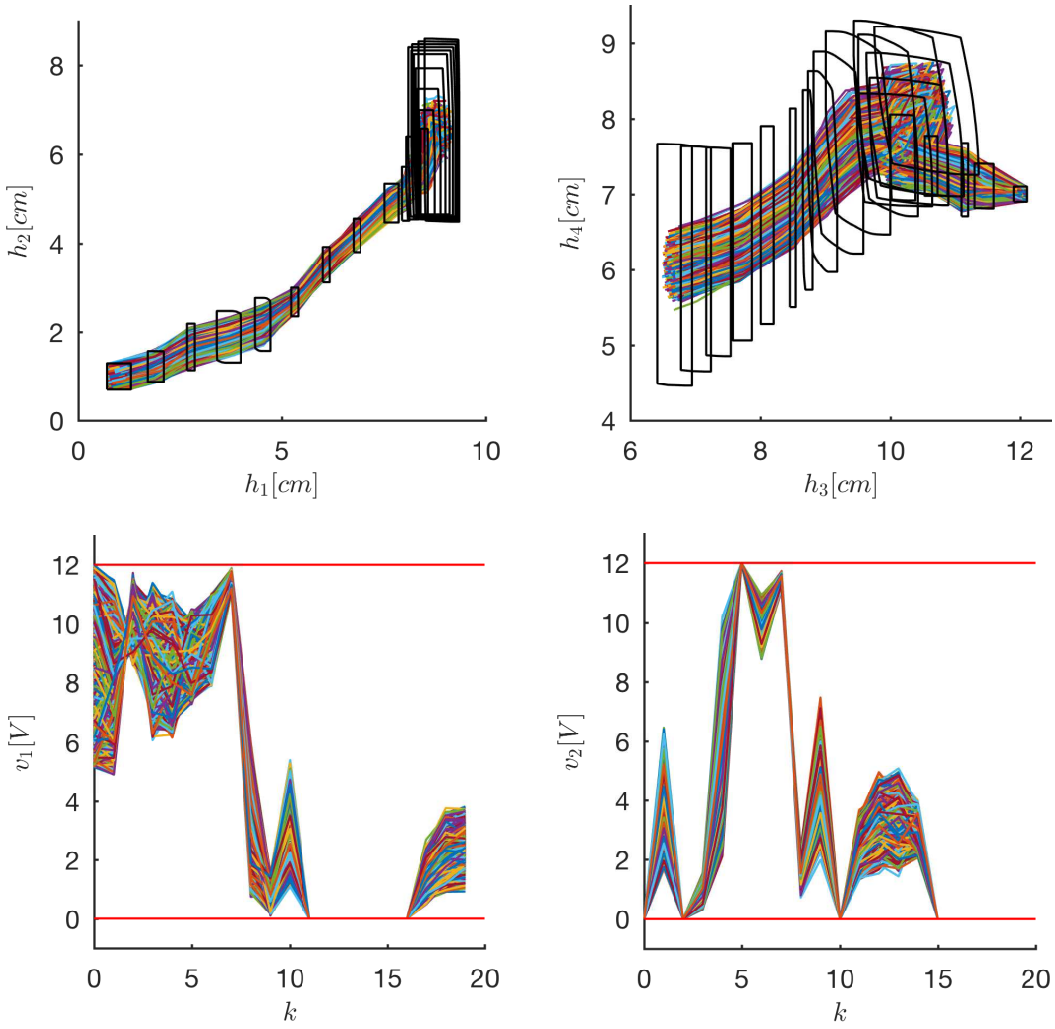


Figure 4.5: 5,000 simulation runs of the resulting controller with random initial states and random disturbances. Top: Resulting state trajectories and reachable sets from Fig. 4.4. Bottom: Corresponding input trajectories with inputs bounds plotted in red.

The results are presented in Fig. 4.4, where we project the resulting sets on the (h_1, h_2) and (h_3, h_4) planes. We see that the initial set is steered along the reference trajectory and always stays inside the state constraints. It can also be seen that both the reference trajectory and the sets stay away from the mode edges, if possible. In particular, this can be seen in Fig. 4.4 (left) where the reachable set at the second time step is entirely maintained below the mode boundary $h_1 = 3\,cm$ (magenta) and is entirely steered above the boundary at the third time step to prevent splitting. This is not possible at $k = 5$, as can be seen in Fig. 4.4 (left) when h_2 crosses the line $h_2 = 3\,cm$, so that a split occurs. However, the split set is rather small (shown in red). Due to the different dynamics and disturbances, the evolution of the split set

over time is different from the evolution of the main set (red sets propagating); nonetheless, we are able to steer the split set inside the main reachable set in $M_{split} = 7$ time steps, thus avoiding branching. As a result, we can compute the controller via one MILP for the reference trajectory, 20 linear programs for the main reachable set, and a single linear program to steer the split set back to the main reachable set.

In addition, we also present the results for 5,000 simulations of the controller with random initial states and random disturbances from their respective sets in Fig. 4.5. We see that all state trajectories are indeed inside the reachable sets and therefore also satisfy the state constraints. The same is true for the input trajectories, which all satisfy the input constraints despite disturbances.

Comparison with Modified Approaches

For comparison, we also solve the problem with a modified version of our approach, where we do not weight the distance to the boundaries of modes or state constraints in the cost function when computing the reference trajectory. As a consequence, the new reference trajectory stays close to the lower boundaries of the time-varying state constraints, while just barely satisfying them. The set-based controller tries to stay close to this trajectory while still satisfying the state constraints. This works for a few steps; however, after that, the receding horizon optimization problem becomes infeasible. It is impossible to find a set-based controller which steers the whole reachable set inside the state constraints of the next time step because the lower bound of the state constraints increases faster than the controller can steer all states from the current reachable set.

We also consider another variant of our algorithm, where we weight the distances to the state constraint boundaries for the reference trajectory, but we do not weight the distances to the mode boundaries for the reference trajectory or the set-based controller. In this case, we obtain a feasible solution for the main reachable set; however, we have to split two times. Moreover, it is not possible to steer any of these split sets into the main reachable set sequence after $M_{split} = 7$ time steps. This therefore leads to two additional MILP and 27 additional linear programs and more than twice the computation time. Even if we consider the distance to the mode boundaries during the computation of the reference trajectory, we end up with two splits, one of which cannot be steered back into the main sequence if we do not consider the distance to the mode boundaries during the computation of the set-based controller. While decreasing the computation time and the number of control laws which need to be stored, our approach increases the costs for the size of the reachable set $cost(\mathcal{R}_{k,u_{ctrl,0},\mathcal{W}}(\mathcal{X}_0))$ by only 10%, compared to the approach where we ignore the costs for the distance to the mode boundaries. This shows how directly considering the distance to the boundaries of the mode sets and the state constraints is beneficial for obtaining good controller results and how we are able to minimize the overall computational effort and time by carefully choosing the cost functions.

Comparison with Existing Tools

For an additional comparison, we try to solve the problem at hand with Pessoa [150], a tool to compute abstraction-based controllers. The advantage of this correct-by-construction method is that one obtains a finite state machine and can use tools which are able to handle linear temporal logic specifications. As discussed before, the general bottleneck of these approaches is the requirement to compute an abstraction, which scales exponentially in the state and input dimensions. To reduce the size of the abstraction, we confine the considered state space to the set $\mathcal{S} = [0, 10]\,cm \times [0, 10]\,cm \times [3, 15]\,cm \times [3, 11]\,cm$. We discretize the states with a resolution of $0.5\,cm$ and the inputs with a resolution of $3\,V$, thereby obtaining a discrete abstraction with 187,425 states and 25 inputs. For simplicity's sake, we do not include any time dependence in the problem, i.e., we do not consider the time-varying state constraints nor do we specify at what time the final set has to be reached. We could include this time dependency by adding the time as an additional state, which would increase the number of states and therefore the computation time by a factor of 20. Since Pessoa synthesizes controllers of the form "stay in set $\mathcal{A}$" or "reach set $\mathcal{A}$ while staying in set $\mathcal{B}$", we change the task to "reach set $\mathcal{G}$ while staying in set $\mathcal{S}$".

Even if we choose the goal set to be the whole final state constraint, i.e., $\mathcal{G} = [3, 10]\,cm \times [3, 10]\,cm \times [3, 10]\,cm \times [3, 10]\,cm$, we are not able to find a controller which works for all states of the initial set, not even if we ignore disturbances. We have also tried it with a state resolution of $1\,cm$ and an input resolution of $1\,V$, which also leads to the same result that no controller exists which satisfies the specifications for all initial states with the abstracted model. With a finer resolution of the abstraction, e.g., a state resolution of $0.5\,cm$ and an input resolution of $1\,V$, it might be possible to find a controller. However, with a resolution of $1\,cm$ for states and $1\,V$ for inputs, for which we could not find a controller, the computation of the abstraction already takes around 8 hours. If we use a state resolution of $0.5\,cm$, we need 13.2 times as many discrete states compared to the state resolution of $1\,cm$, which would lead to a computation time of over 105 hours. If we also include the time dependencies, we would even have a computation time of almost 3 months. This shows that for this example our approach scales much better. Please note that if one obtains a controller with Pessoa, one can reuse it for different initial states in the abstracted state space, while for our approach, we need to compute a new controller for an initial state outside of our original initial set.

We also compare our approach with the one proposed in [151]. There, the authors reformulate the robust control as an MILP problem that simultaneously takes into account all the possible (unknown) switching sequences that the system may follow while satisfying the specification. We tested the approach from [151] on our problem and were not able to find a solution after several hours of computation. This is not unexpected since the number of binary variables appearing in the MILP used in [151] scales exponentially with the number of time instants. The computational burden of a time horizon of 20 steps is therefore already too high. The approach proposed in this chapter also resorts to MILP, but the number of binary variables required in our case only scales linearly with the time horizon — thus, our approach is able to cope with larger instances of the problem.

4.7 Discussion

Online Computational Complexity

As discussed at the end of Sec. 4.5, there are multiple ways to obtain the online control law. None of them is computationally hard. If we simply want to find a vector α which describes the measured state inside the reachable set of the current time step, this can be done by solving a linear feasibility problem. As this can be done using linear programming, where we can even optimize the used input, and linear programs scale polynomially [99], this can be done fast online. By restricting the maximum order of the zonotope representation, we can further limit the computational effort to our needs. If we restrict it to a parallelotope at the beginning of each step, we can simply invert the generator matrix as done for the generator interpolation controller in Sec. 3.4. In this case, we do not have to solve any linear program, only perform a simple matrix vector multiplication.

Offline Computational Complexity

The offline algorithm described in Section 4.3 solves an MILP every time we compute a reference trajectory for which efficient solvers exist (see, e.g., CPLEX); nonetheless, the complexity of an MILP is, in the worst case, exponential in the number of binary variables. In our case, the binary variables are represented by the vector $\delta(k)$ for each time step, and therefore their number increases linearly with the problem horizon and also linearly with the number of modes. We benefit from the fact that in our approach, the extra requirement of staying far from mode boundaries does not require the addition of extra binary variables. To reduce the computational effort, since the state constraints have to be satisfied at each time step, it is possible to prune the exploration tree by removing any switching sequence that contains modes that do not intersect the state constraint sets at some time step.

For the set-based controller design, if all reachable sets stay within the main mode sequence, we have to solve N linear programs which scale polynomially in the size of the optimization variables and constraints [99]. In our case, these variables scale linearly in the prediction horizon M and linearly in the state and input dimensions, if we fix the zonotope order. This can be done by applying zonotope reduction methods [226] which reduce the number of generators in the considered zonotopes to lower the computational effort. Therefore, the controller synthesis problem can be solved very efficiently.

Note that we cannot give a fixed bound on the number of splits; however, we expect that splits do not occur too frequently in practical examples, especially because we try to avoid them by maximizing the distance to the mode boundaries. If the split set can be steered back in the main mode after M_{split} time steps, this adds only a single linear program to the computational effort. If we cannot steer it back in M_{split} time steps, a new reference trajectory has to be computed, which requires solving a new MILP.

Due to the exponential scaling of the computational effort for solving an MILP compared to the polynomial scaling for linear problems, the complexity of the MILP for computing the reference trajectory governs the overall complexity of the proposed algorithm. Therefore, if we

denote the number of splits which cannot be steered back by S, the overall complexity of our algorithm is $(S+1)\,\mathcal{O}(\text{MILP})$.

Optimality

For linear problems and mixed-integer linear problems, it is possible to find the global optimal solution [152, 153]. Therefore, we are able to find the optimal solution to our optimal control problems according to our cost and constraint functions. Since we consider the additional requirements of staying away from mode boundaries to avoid splits of the reachable sets, the resulting solutions are in general not the optimal solutions of the unmodified problem. Computing those solutions, however, would require either performing the complex MILP optimization online for the current state, or solving and storing the solution for all states under all disturbances with all possible mode sequences, which easily becomes far too complex to compute or store. Therefore, by considering the costs for avoiding the splits, we obtain a solution, which might be non-optimal, but which avoids the split over too many mode sequences and which allows us to efficiently compute and store the controllers for a whole set of solutions despite disturbance effects. Note that limiting the optimization horizon M of our set-based controller to $M < N$ can also lead to non-optimal solutions. However, this can be overcome by always optimizing over the whole prediction horizon N, and therefore results in a trade-off between computation time and quality of the solutions.

4.8 Summary

In this chapter, we extend the ideas from the previous chapter to the hybrid dynamics of piecewise affine systems. In our new approach, we account for the possible increase of the (offline) computational effort of the state feedback controller by explicitly aiming to prevent the disturbance from activating too many modes.

This is achieved by a two-step approach where, in the first step, we compute a nominal reference trajectory which stays far from the mode boundaries, and, in the second step, we track this reference trajectory using a set-based feedback controller which minimizes the reachable sets along the reference trajectory while avoiding splitting of the reachable sets over modes. If we are able to avoid splitting, our synthesis method reduces to a single MILP for the reference trajectory computation and N linear problems, where N is the horizon length. Otherwise, a mode recovery procedure (linear problem) and, possibly, a branching of the reference trajectory (MILP on the residual time horizon) are activated.

Like the approaches in Ch. 3, most computations and in particular all MILPs are performed offline. The resulting control law is very simple to store and apply online. This provides significant advantages over approaches from the literature, which face the difficulty that many modes lead to many possible mode sequences, which leads to challenging combinatorial problems [124, 151]. This effect becomes even worse when considering a set of initial states or disturbances, as the number of possible mode sequences rapidly increases in this case, which limits the applicability of approaches like [125–132]. By computing the reference trajectory for a

single state without uncertainty, we benefit from the efficient optimization tools which exist for this simpler case. By optimizing the reachable sets to avoid splitting over multiple modes, we actively avoid the branching of mode sequences, which allows us to use the efficient approaches from the previous chapter and benefit from their advantages and guarantees. Therefore, our new algorithm is not only more efficient than many existing ones, but we are also able to provide formal guarantees.

In contrast to many MPC approaches [17, 139–143], we do not need to solve a complex optimization problem online, nor do we require finding a Lyapunov function or have the restriction that a common Lyapunov function has to exist for all modes [145]. We also do not need to tighten constraints or compute invariant sets. In contrast to existing approaches, we take the possibility that the reachable set may be split explicitly into account and provide countermeasures when designing the reference trajectory and the feedback controller. We also do not face the same problems as approaches using abstraction-based control [150] or multiparametric programming [147], whose computational effort becomes too high for most practical applications or whose explicit forms are complex to store and apply online. We demonstrated this for a quadruple tank benchmark example. While our new approach was able to solve this problem, it was not possible with existing approaches [150] and [151].

Chapter 5

Offline Controller Synthesis Using Backward Reachable Sets

5.1 Introduction

So far, we presented a number of control approaches which minimize reachable sets and ensure safety at all times, despite disturbances and sensor noise. However, even though safety is undeniably the most important feature for a successful application of autonomous systems, there are also other important aspects which determine success in practice, e.g., comfort, energy consumption, and long-term wear. There exists a wide variety of classical controllers which focus on these performance goals and are often used in practice, e.g., standard MPC [12]. However, due to the way they are designed, they cannot be formally verified to prove safety in all situations. Formal control algorithms, including those from previous chapters, on the other hand, focus on ensuring safety under all possible situations, thereby possibly requiring large input values and fast input changes opposing comfort, energy consumption, and long-term wear goals.

To benefit from the advantages of optimal and safe control, we pursue the following idea in this chapter[1]: We combine a formal controller serving as a safety net with an optimal, unverified controller which minimizes some cost function. While worst-case behavior has to be considered for safety, in most situations, such behavior does not occur and an optimal controller can achieve better performance. To ensure safety, the formal safety controller monitors the optimal controller by computing its reachable set for the near future. As long as this set is safe, the optimal controller is applied. Otherwise, the safety controller takes over to keep the system safe. Thus, we combine the optimality of classical control with the safety of formal controller synthesis.

In principle, we can use any of the synthesis approaches from the previous chapters to compute a safety controller. However, there, we focus on minimizing the reachable set as quickly as possible. Therefore, the sets in which the controller can guarantee finding a safe

[1]This chapter and most of its figures are based on [214] © 2022 IEEE.

solution would be small so that the safety controller would take over most of the time. In the second part of this chapter, we address this problem by optimizing over backward reachable sets instead. By fixing the final reachable set, we compute backwards in time the maximum set for which we can find a safety controller, thereby maximizing the set in which the optimal controller can safely operate and thus minimizing interventions from the safety controller.

Related Work on Under-Approximative Reachable Sets

When using approximations of reachable sets to ensure safety, one has to compute over-approximations for forward reachable sets, as used in the previous chapters, and under-approximations for backward reachable sets [154]. While the idea is simple, the computation of under-approximative backward reachable sets is not straightforward. There exist fewer methods than for forward reachable set computation like [1, 2] and they are often more conservative than the forward algorithms. They exist for linear systems [155–157], piecewise affine systems [158], and polynomial systems [159–161]. Only a few consider general nonlinear systems, e.g., [162–167], of which most consider systems without inputs [162–166]. The combination of backward reachable sets and controller design is done in [168], however only for discrete-time, linear systems without disturbances.

Since we consider nonlinear systems, let us review the approaches for these systems in more detail. Several approaches, mostly for polynomial systems, use the Hamilton-Jacobi framework to find under-approximations of reachable sets [159, 161, 162]. While they obtain quite good results for low-dimensional polynomial systems, they do not scale well with higher dimensions or polynomial degrees. Recent approaches use decomposition to deal with the unfavorable computational complexity of their methods [169], at least for suitable classes of nonlinear systems, where the dynamics are loosely coupled. Other approaches use Taylor models in combination with polytopic level sets to under-approximate the reachable set [165]. They have the disadvantage that they might result in unconnected sets, where only one is the actual under-approximation, which then requires computationally expensive checks to find the proper set. Computational challenges also arise when trying to solve practical problems with the approach in [163], which uses so-called polynomial level-set functions to represent the under-approximative reachable sets and is too hard to compute for larger state spaces. Another method is proposed in [166], which has been extended for systems with inputs in [167]. Their results are promising, however, they only compute under-approximations of projections on the coordinate axes. To the best of our knowledge, there exists no approach which computes actual under-approximative reachable sets for nonlinear systems with inputs. Therefore, and since we need a special preservation of input effects for our controller synthesis, which is not provided by any of the existing techniques, we develop a new approach. This new approach uses optimization to maximize the size of backward reachable sets and uses a forward reachability algorithm based on [76] to verify the results.

Related Work on Safety Nets

The idea of a controller safeguarding other controllers has been previously proposed, e.g., by [170, 171]. There, stability is proven for certain regions with a simple controller, which takes over if a more complex controller fails to satisfy certain safety constraints. This framework is extended in [172, 173] by including the online computation of reachable sets to identify the regions from which the verified controller can recover the system to a stable region. Computing a safe invariant region for a safety controller and ensuring that a second controller cannot leave this region is done in invariance control [174, 175]. Combining the satisfaction of safety constraints with performance controllers can also be achieved using control barrier functions and has been applied to different types of systems, such as autonomous cars and robotic applications [44–48]. A related method is that of reference governors [176–178], which monitor the system and modify the inputs when constraints would be violated. The idea of using reachability analysis together with a safety controller is also used for motion planning in [43] and for safe reinforcement learning in [179]. Another type of safety net controller named shielding is presented in [180], which can also be used for safe reinforcement learning [181].

While there exist already different approaches for safety nets, they often only consider systems without disturbances [171–175, 177] and some only discrete automata [180, 181]. The approaches [43, 179] can deal with disturbances; however, they rely on Hamilton-Jacobi reachability analysis and even though there have been advancements for special types of systems, in general their computational effort scales exponentially with the number of states and is therefore restricted to low-dimensional systems [182]. Similar restrictions hold for approaches using control barrier functions such as [44–49]: While it has been shown that one can consider disturbances in the controller design based on control barrier functions, see [49], as they use sums-of-squares programming to obtain the control barrier and control Lyapunov functions, they face a fast growth of computational effort with the system dimension and considered polynomial degree as well.

5.2 Problem Formulation

As in Ch. 3, we consider a disturbed, nonlinear, continuous-time system (2.3) with polyhedral constraints on the states and inputs, (3.2) and (3.3), respectively. Rather than looking for the smallest final set for a given initial set, the task in this chapter is to find for a given final set $\mathcal{X}_f \subset \mathbb{R}^n$ the largest initial set $\mathcal{X}(t_0) \subset \mathbb{R}^n$. More specifically, we are looking for the initial set with maximum volume around a desired state $x^{(0)}$ and the corresponding verified control law $u_{ver}(x,t)$ such that for system (2.3), all solutions starting at $t_0 = 0$ in $\mathcal{X}(t_0)$ end in the final set $\mathcal{X}_f$ at time t_f despite disturbances and while satisfying the constraints.

5.3 Online Control with Safety Net

For the online control, we assume that we have found a safe sequence of motion primitives, which connect the initial state with the goal state. We obtain this sequence by using discrete

planning techniques as described in Sec. 1.2. We discuss how to compute the motion primitives offline using backward reachable sets in Sec. 5.4 and Sec. 5.5.

Let us start by describing how our online controller works, which is summarized in Alg. 5 and illustrated in Fig. 5.1. We explain this for a single motion primitive to simplify the notation. At the end of each motion primitive, we simply switch to the following one. During the online application, we repeatedly need to decide when to use the unverified, optimal controller or the verified, safety controller. To this end, we use a safety net controller which acts in discrete-time and allows switching to and from the optimal controller at discrete switching times $t_k := k\Delta t,\ k \in \mathbb{N}_0$, where $\Delta t = \frac{t_f}{M},\ M \in \mathbb{N}$. Similarly, we only start a new optimization for the optimal controller once in each time step, so that the control behavior remains predictable. Besides this restriction, we are flexible with the form of the unverified controller and also allow continuous-time controllers.

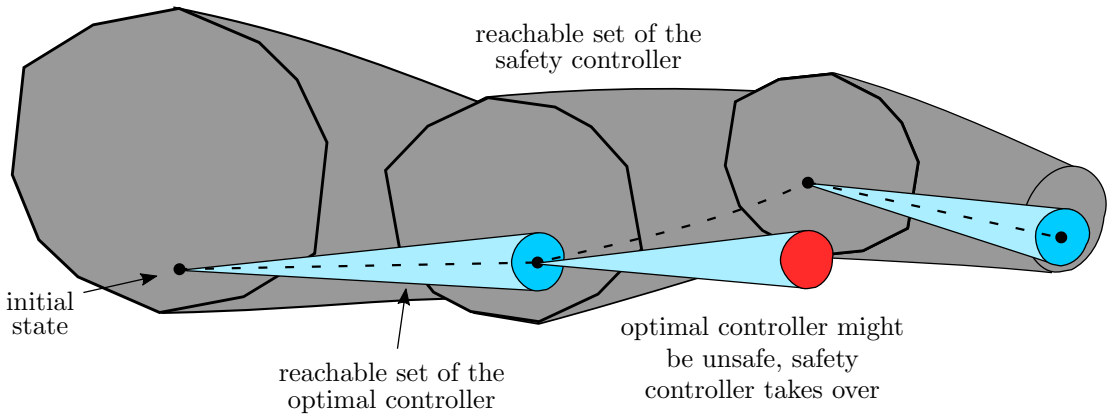


Figure 5.1: Overview of the online control: Based on the measurement of the current state, we compute the reachable set for the optimal controller. If it ends inside the reachable set of the safety controller and if it satisfies all constraints, we apply the optimal controller (see first time step). If this is not the case, we switch to the safety controller (see second time step). After the intervention of the safety controller, we check if the optimal controller is safe once again, and switch back to it if safety can be ensured (see third step).

For each switching time t_k, we have to decide which controller we want to apply in the next time interval. Let us assume that we are applying a feasible controller $u_{appl}(\,\cdot\,)$ for the previous time interval $[t_{k-1}, t_k]$: either the unverified, optimal controller $u_{unv}(\,\cdot\,)$ or the verified, safety controller $u_{ver}(\,\cdot\,)$. Since the computation of the reachable set takes some time, we need to measure the state $x(t_k - t_c)$ and start the computations at time $t_k - t_c$, to know at t_k which controller to apply during time period $[t_k, t_{k+1}]$. Here, $t_c \leq \Delta t$ denotes an allocated computation time consisting of (i) the computation time of the optimal controller, (ii) the computation time of the verifier, and (iii) a buffer time. It can be chosen by testing how long these computations take on the used machine. If the actual computation time takes longer, we simply apply the safety controller for the next time step to ensure safety.

Algorithm 5 Online Safety Net Control Algorithm

Input: system dynamics $f(x,u,w)$, disturbance set $\mathcal{W}$, time step size Δt, maximum computation time t_c, motion primitive **MP**, unverified controller $u_{unv}(\cdot)$, state measurements $x(\cdot)$
Output: control inputs $u_{appl}(\cdot)$

1: **while** $t < t_f$ **do**
2: $x(t_k - t_c) \leftarrow$ measured state at $t = t_k - t_c$
3: $\hat{\mathcal{X}}(t_k|t_k - t_c) \leftarrow \mathcal{R}_{t_c,u_{appl},\mathcal{W}}(x(t_k - t_c))$
4: $\hat{\mathcal{X}}(t_{k+1}|t_k - t_c) \leftarrow \mathcal{R}_{\Delta t,u_{unv},\mathcal{W}}(\hat{\mathcal{X}}(t_k|t_k - t_c))$
5: **if** computation finished before t_k and satisfaction of (5.1)–(5.3) **then**
6: $u_{appl}(\cdot) \leftarrow u_{unv}(\cdot),\ \forall t \in [t_k, t_{k+1})$
7: **else**
8: $u_{appl}(\cdot) \leftarrow u_{ver}(\cdot),\ \forall t \in [t_k, t_{k+1})$
9: **end if**
10: **end while**

We begin by measuring the state $x(t_k - t_c)$ (Alg. 5, line 2), where we do not explicitly consider measurement errors for a simpler notation. Note that the presented ideas would also work in the presence of measurement errors and we would simply obtain sets of possible initial states, as we show in detail in Ch. 6. Based on the measurement of $x(t_k - t_c)$, we first compute the reachable set at time t_k for the currently applied controller $u_{appl}(\cdot)$ (Alg. 5, line 3):

$$\hat{\mathcal{X}}(t_k|t_k - t_c) := \mathcal{R}_{t_c,u_{appl},\mathcal{W}}(x(t_k - t_c)),$$

where we use the notation $\hat{\mathcal{X}}(t_2|t_1)$ to refer to the reachable set at time t_2 which is computed based on the information at time t_1. Starting from $\hat{\mathcal{X}}(t_k|t_k - t_c)$, we then compute the reachable set at t_{k+1} (Alg. 5, line 4) using the unverified, optimal controller $u_{unv}(\cdot)$:

$$\hat{\mathcal{X}}(t_{k+1}|t_k - t_c) := \mathcal{R}_{\Delta t,u_{unv},\mathcal{W}}(\hat{\mathcal{X}}(t_k|t_k - t_c)).$$

Next, we check if the controller would satisfy the state constraints

$$\mathcal{R}_{[0,\Delta t],u_{unv},\mathcal{W}}(\hat{\mathcal{X}}(t_k|t_k - t_c)) \subseteq \mathcal{S} \tag{5.1}$$

and the input constraints

$$u_{unv}(x, t_k + \tau) \in \mathcal{U},\ \forall x \in \mathcal{R}_{\tau,u_{unv},\mathcal{W}}(\hat{\mathcal{X}}(t_k|t_k - t_c)),\ \forall \tau \in [0, \Delta t), \tag{5.2}$$

during the whole time interval $[t_k, t_{k+1}]$ and if it would end completely inside the reachable set of the safety controller at time t_{k+1} (Alg. 5, line 5):

$$\hat{\mathcal{X}}(t_{k+1}|t_k - t_c) \subseteq \mathcal{R}_{t_{k+1},u_{ver},\mathcal{W}}(\mathcal{X}(t_0)). \tag{5.3}$$

This means that the solution of the optimal controller can leave the reachable set of the safety controller as long as it satisfies the state and input constraints and returns to the reachable set of the safety controller at t_{k+1}. This ensures that we can switch at t_{k+1} to the safety controller, if no safe solution of the optimal controller exists for $[t_{k+1}, t_{k+2}]$.

If all three constraints are satisfied and the computations are finished before time t_k, we can safely apply the optimal controller during the time interval $[t_k, t_{k+1})$ (Alg. 5, line 6); otherwise, we switch to the safety controller (Alg. 5, line 8).

Theorem 4. *Given a motion primitive* **MP** *from Alg. 6 connecting the initial state $x^{(0)}$ with the desired final set $\mathcal{X}_f \subseteq \mathcal{S}$ and satisfying the state constraints* (3.2)*, Alg. 5 returns the input trajectory $u_{appl}(\cdot)$. This input trajectory satisfies the input constraints* (3.3) *and results in a state trajectory $\xi(x^{(0)}, u_{appl}(\cdot), w(\cdot), \cdot)$, which satisfies the state constraints* (3.2) *and reaches the final set, i.e., $\xi(x^{(0)}, u_{appl}(\cdot), w(\cdot), t_f) \in \mathcal{X}_f$.*

Proof. The proof follows by construction of the algorithm and is shown by induction:
Base Case: At time t_0, we know a safe solution from the safety controller for the time interval $[t_0, t_1]$ which satisfies the state constraints by assumption and the input constraints by construction (as described later in Sec. 5.5).
Induction Hypothesis: If we know a safe controller for the time interval $[t_{k-1}, t_k]$, we can always get a safe solution for the following time interval $[t_k, t_{k+1}]$.
Induction Step: When we measure the state $x(t_k - t_c)$, we know that we are inside the reachable set of the currently applied controller. By the definition of reachable sets, applying the same controller ensures that the reachable set $\mathcal{R}_{t_c,u_{appl},\mathcal{W}}(x(t_k - t_c))$ ends inside the previously computed reachable set, i.e.,

$$\mathcal{R}_{t_c,u_{appl},\mathcal{W}}(x(t_k - t_c)) \subseteq \mathcal{R}_{\Delta t,u_{appl},\mathcal{W}}(\hat{\mathcal{X}}(t_{k-1}|t_{k-1} - t_c)) \overset{(5.3)}{\subseteq} \mathcal{R}_{t_k,u_{ver},\mathcal{W}}(\mathcal{X}(t_0)).$$

Since $\mathcal{R}_{t_k,u_{ver},\mathcal{W}}(\mathcal{X}(t_0))$ is the reachable set of the safety controller at a switching time t_k, we can always safely switch at t_k to the safety controller. Therefore, we always check the constraints with the optimal controller for the time interval $[t_k, t_{k+1}]$ at time $t_k - t_c$: If this is safe, we apply the optimal controller for the next time interval; otherwise, we switch to the safety controller at time t_k to obtain a safe solution for the next time interval $[t_k, t_{k+1}]$. □

In the next two sections, we discuss how to construct the safety controller and motion primitives by using backward reachability analysis.

5.4 Backward Controller Synthesis for Linear, Discrete-Time Systems

Before we consider nonlinear systems for synthesizing safe controllers, we discuss the case of discrete-time, linear, time-varying systems without disturbances of the form (3.23). We assume that the matrices A_k have full rank, i.e., they are invertible. If we obtain them as time-discretized versions of continuous-time systems, this assumption is satisfied, as the matrix exponential $e^{A\Delta t}$ is always invertible [183, Ch. 7.2, Thm. 2]. In this case, the backward reachable set can be computed exactly. Afterwards, we extend the idea presented in this section to continuous-time, nonlinear systems with disturbances in Sec. 5.5.

5.4.1 Backward Reachable Set

To obtain the backward reachable set of (3.23), we express the evolution of a single state $x(t_k)$ after h steps analogously to (3.24) as

$$\xi(x(t_k), u(\cdot), 0, t_h) = \bar{A}x(t_k) + \sum_{i=k}^{k+h-1} \bar{B}_i u(t_i), \tag{5.4}$$

where we use the shorthand notations

$$\begin{aligned}\bar{A} &:= A_{k+h-1} \dots A_k, \\ \bar{B}_i &:= A_{k+h-1} \dots A_{i+1} B_i, \quad \forall i \in \{k, \cdots, k+h-2\},\end{aligned}$$

and $\bar{B}_{k+h-1} := B_{k+h-1}$. Solving (5.4) for $x(t_k)$ leads to

$$x(t_k) = \bar{A}^{-1} \left(x(t_{k+h}) + \sum_{i=k}^{k+h-1} (-1)\bar{B}_i u(t_i) \right). \tag{5.5}$$

After replacing concrete values with sets, we obtain the backward reachable set

$$\mathcal{X}(t_k) = \bar{A}^{-1} \left(\mathcal{X}(t_{k+h}) \oplus \bigoplus_{i=k}^{k+h-1} (-1)\bar{B}_i \mathcal{U} \right). \tag{5.6}$$

For zonotopic sets $\mathcal{X}(t_{k+h}) = \langle c_{k+h}, g_{k+h}^{(1)}, \dots, g_{k+h}^{(p_{k+h})} \rangle$ and $\mathcal{U} = \langle c_u, g_u^{(1)}, \dots, g_u^{(q)} \rangle$, the backward reachable set is

$$\begin{aligned}\mathcal{X}(t_k) &= \bar{A}^{-1} \left(\mathcal{X}(t_{k+h}) \oplus \bigoplus_{i=k}^{k+h-1} (-1)\bar{B}_i \mathcal{U} \right) \\ &= \Big\{ x \in \mathbb{R}^n \,\Big|\, x = \bar{A}^{-1} c_{k+h} - \sum_{i=k}^{k+h-1} \bar{A}^{-1}\bar{B}_i c_u + \sum_{i=1}^{p_{k+h}} \alpha_i \bar{A}^{-1} g_{k+h}^{(i)} - \sum_{i=k}^{k+h-1} \sum_{j=1}^{q} \beta_{i,j} \bar{A}^{-1}\bar{B}_i g_u^{(j)}, \\ &\qquad \alpha_i, \beta_{i,j} \in [-1, 1] \Big\} \\ &=: \Big\{ x \in \mathbb{R}^n \,\Big|\, x = \hat{c} + \sum_{i=1}^{\hat{p}} \hat{\alpha}_i \hat{g}^{(i)}, \hat{\alpha}_i \in [-1, 1] \Big\},\end{aligned} \tag{5.7}$$

where

$$\begin{aligned}\hat{c} &:= \bar{A}^{-1} c_{k+h} - \sum_{i=k}^{k+h-1} \bar{A}^{-1}\bar{B}_i c_u, \\ \hat{g}^{(i)} &:= \bar{A}^{-1} g_{k+h}^{(i)}, \ \forall i \in \{1, \dots, p_{k+h}\}, \\ \hat{g}^{(p_{k+h} + i\,q + j)} &:= -\bar{A}^{-1}\bar{B}_{k+i} g_u^{(j)}, \ \forall i \in \{0, \dots, h-1\}, \ \forall j \in \{1, \dots, q\}.\end{aligned}$$

Here, p_{k+h} denotes the number of generators of $\mathcal{X}(t_{k+h})$ and $\hat{p} := p_{k+h} + hq$. By introducing $\hat{c}, \hat{g}^{(1)}, \dots, \hat{g}^{(\hat{p})}$, we obtain a simpler notation and are able to handle the generators resulting from the states and inputs in the same way. Note that we can always obtain zonotopic sets for $\mathcal{X}$ and $\mathcal{U}$ by under-approximating the original sets with zonotopes.

5.4.2 Resulting Online Control Law

Let us now discuss how to obtain the control law steering the system from a state $x \in \mathcal{X}(t_k)$ to the desired set $\mathcal{X}(t_{k+h})$. Each state $x \in \mathcal{X}(t_k)$ can be expressed as a superposition of the generators $\hat{g}^{(i)}$ by finding the corresponding parameters $\hat{\alpha}_i(x)$ (see (5.7)). Since, in general, the choice of $\hat{\alpha}_i(x)$ is not unique, we solve the following optimization problem:

$$\min_{\hat{\alpha}(x)} \sum_{i=1}^{\hat{p}} \rho_i |\hat{\alpha}_i(x)| \tag{5.8}$$

$$\text{s.t. } x = \hat{c} + \sum_{i=1}^{\hat{p}} \hat{\alpha}_i(x)\hat{g}^{(i)}, \tag{5.9}$$

$$-1 \leq \hat{\alpha}_i(x) \leq 1, \ \forall i \in \{1, \dots, \hat{p}\}, \tag{5.10}$$

with weights $\rho_i \in \mathbb{R}_0^+$. We then multiply $G_u = [g_u^{(1)}, \dots, g_u^{(q)}]$ with the factors $\hat{\alpha}_i(x)$ corresponding to the input generators at time step $k+j$ to obtain the following piecewise-constant control law:

$$u(x, t_{k+j}) = c_u + G_u[\hat{\alpha}_{p_{k+h}+j\,q+1}(x), \dots, \hat{\alpha}_{p_{k+h}+(j+1)\,q}(x)]^T, \ \forall j \in \{0, \dots, h-1\}.$$

It follows from the superposition principle of linear systems that if we use the weighted combination of the corresponding inputs, we ensure that any state $x \in \mathcal{X}(t_k)$ is steered in the desired set, too. This is analogous to the ideas from Sec. 3.3.1 and Sec. 3.4.1. Since some generators $\hat{g}^{(i)}$ in (5.7) result from states converging to the desired set without inputs and others from generators which are controlled to the origin by the inputs, the choice of ρ_i balances the applied inputs to the size of the final set: Higher weights for $\rho_1, \dots, \rho_{p_{k+h}}$ in (5.8) lead to solutions which use higher inputs and end closer to the center of $\mathcal{X}(t_{k+h})$. Higher weights for $\rho_{p_{k+h}+1}, \dots, \rho_{\hat{p}}$, on the other hand, punish large inputs and will lead to solutions which save input capacities on the price of ending closer to the boundaries of $\mathcal{X}(t_{k+h})$.

Similar to the previous approaches, the optimization problem in (5.8) is a linear program and can therefore be solved online. For instance, solving the linear problem for the numerical example at the end of this chapter only takes around $3\,ms$ for a MATLAB implementation, and even less when using C++. If computation time or effort is a bigger concern, one can also simply apply the first solution which satisfies constraints (5.9)–(5.10). If even faster solutions are required, we can again express the zonotope as a convex combination of its extreme states and use the techniques from Appendix A to obtain weights $\hat{\alpha}_i(x)$ by simply plugging x into this closed-form expression.

5.5 Backward Controller Synthesis for Nonlinear, Continuous-Time Systems

Let us now extend the presented ideas to nonlinear, continuous-time systems, for which the superposition principle no longer holds and where the backward reachable sets cannot be computed exactly or efficiently in an under-approximative way.

Algorithm 6 Backward Reachable Set Control Algorithm

Input: system dynamics $f(x,u,w)$, final set $\mathcal{X}_f$, desired initial state $x^{(0)}$, total time t_f, number of iterations N, M, constraint sets $\mathcal{S}, \mathcal{U}, \mathcal{S}_{ref}, \mathcal{U}_{ref}$, weighting factor γ_{ref}, maximum number of generators $\bar{p}$

Output: motion primitive **MP**

1: $(x_{ref}(\cdot), u_{ref}(\cdot)) \leftarrow$ solution of optimization problem (3.8) for the reference trajectory
2: $(A_k, B_k) \leftarrow$ linearization and time-discretization of $f(x,u,0)$ along $x_{ref}(\cdot), u_{ref}(\cdot)$ using (3.32) and (3.33)
3: Initialize: $\mathcal{X}(\tau_{Mh}) \leftarrow \mathcal{X}_f$
4: **for** $l = M, \dots, 1$ **do**
5: $\quad \bar{\mathcal{X}}(\tau_{lh}) \leftarrow$ zonotope with at most $\bar{p}$ generators under-approximating the set $\mathcal{X}(\tau_{lh})$ using (5.12)–(5.15)
6: $\quad \mathcal{X}(\tau_{(l-1)h}) \leftarrow$ backward reachable set of linearized dynamics (A_k, B_k) starting from $\bar{\mathcal{X}}(\tau_{lh})$ using (5.6)
7: $\quad u_{conv}(x, \tau_{(l-1)h}) \leftarrow$ control law (5.16) based on $\mathcal{U}$, $u_{ref}(\cdot)$ and $\mathcal{X}(\tau_{(l-1)h})$
8: $\quad \mathcal{R}_{[0,\tau_h],u_{ver},\mathcal{W}}(\mathcal{X}(\tau_{(l-1)h}, s)) \leftarrow$ scalable forward reachable set from (5.18)
9: $\quad s^* \leftarrow$ solution of optimization problem (5.26) to obtain optimal scaling factors
10: $\quad \mathcal{X}^*(\tau_{(l-1)h}) \leftarrow$ rescaled $\mathcal{X}(\tau_{(l-1)h}, s^*)$ using s^*
11: $\quad \mathcal{R}_{[\tau_{(l-1)h},\tau_{lh}],u_{ver},\mathcal{W}}(\mathcal{X}^*(t_0)) \leftarrow$ forward reachable set $\mathcal{R}_{[0,h\,\Delta\tau],u_{ver},\mathcal{W}}(\mathcal{X}^*(\tau_{(l-1)h}))$
12: **end for**
13: **MP** $\leftarrow \{x_{ref}(\cdot), u_{ref}(\cdot), t_f, \mathcal{X}^*(t_0), \mathcal{X}_f, u_{ver}(\cdot), \mathcal{R}_{[0,t_f],u_{ver},\mathcal{W}}(\mathcal{X}^*(t_0))\}$

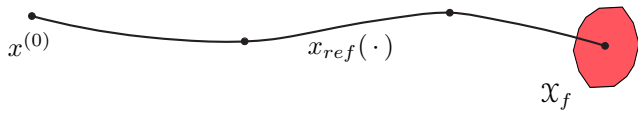
5.5.1 Overview

To obtain an estimate of the backward reachable set, we linearize the dynamics without disturbances for a short time horizon, so that we can use the techniques from Sec. 5.4. In the next step, we compute the forward reachable set considering the actual disturbed, nonlinear dynamics of the controlled system and scale it until it ends in the original final set $\mathcal{X}_f$. We iterate these steps for the whole time horizon $[0, t_f]$, as presented in Alg. 6 and Fig. 5.2.

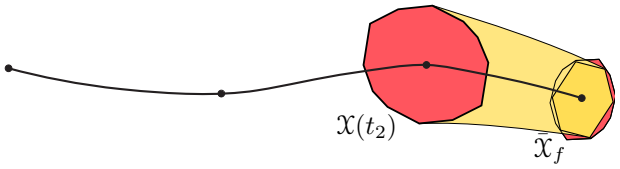
5.5.2 Reference Trajectory

We begin by computing an optimal reference trajectory $x_{ref}(\cdot)$ from the center of the final set $\mathcal{X}_f$ to the desired initial state $x^{(0)}$ (see Fig. 5.2(a) and Alg. 6, line 1) by simply solving (3.8) for the backward case. As in Sec. 3.3.2, we divide the reference trajectory in N steps of length $\Delta\tau = \frac{t_f}{N}$ and restrict the input $u_{ref}(\cdot)$ to be piecewise constant in each time step. Afterwards, we linearize the system along $x_{ref}(\cdot)$ using (3.32) and (3.33) (Alg. 6, line 2).

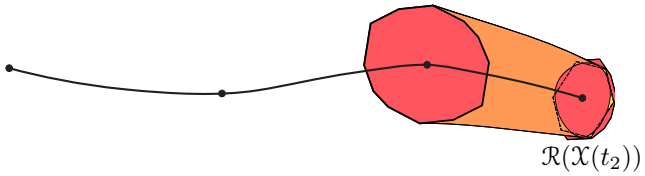
To improve computational efficiency, we group the N pieces of the reference trajectory into M parts consisting of h time steps $\Delta\tau$ each. Since N, M, and h are design parameters, we can choose them such that $N = Mh$ holds with $M, h \in \mathbb{N}$. We iteratively compute the backward reachable set for all time intervals $[\tau_{(l-1)h}, \tau_{lh}]$, $l \in \{1, \dots, M\}$, with $\tau_k := k\Delta\tau$, $k \in \mathbb{N}_0$, starting at time $\tau_{Mh} = t_f$ (Alg. 6, line 3). These intervals $[\tau_{(l-1)h}, \tau_{lh}]$ are exactly the time



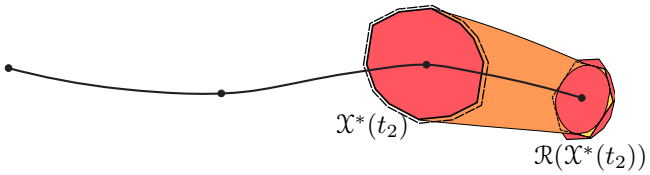
(a) Reference trajectory



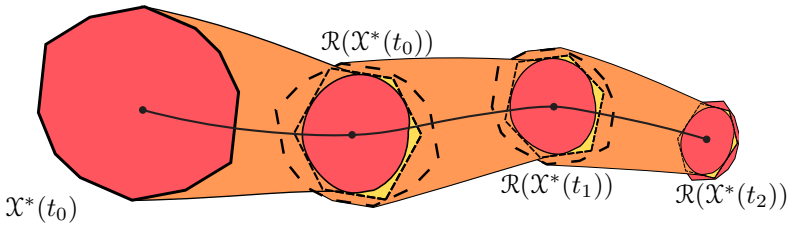
(b) Backward reachable set



(c) Forward reachable set



(d) Scaled forward reachable set



(e) Reachable set whole time horizon

Figure 5.2: Illustration of the backward controller synthesis for nonlinear systems: (a) We compute a reference trajectory $x_{ref}(\cdot)$ from the center of the final set $\mathcal{X}_f$ to the desired initial state $x^{(0)}$ and divide it into M parts (here, $M = 3$). (b) For computational reasons, we under-approximate the final set by a zonotope $\bar{\mathcal{X}}_f$ with fewer generators. We then compute the backward reachable set $\mathcal{X}(t_2)$ for the linearized dynamics for a short time horizon. (c) Starting from $\mathcal{X}(t_2)$, we compute the forward reachable set $\mathcal{R}(\mathcal{X}(t_2)) := \mathcal{R}_{\Delta t, u_{ver}, \mathcal{W}}(\mathcal{X}(t_2))$ based on the disturbed, nonlinear dynamics. (d) If $\mathcal{R}(\mathcal{X}(t_2))$ does not end inside the under-approximated final set $\bar{\mathcal{X}}_f$ (see (c)), we scale the backward reachable set $\mathcal{X}^*(t_2)$ until $\mathcal{R}(\mathcal{X}^*(t_2))$ ends inside the desired set. (e) We iteratively repeat these steps $M = 3$ times until we obtain the desired initial set $\mathcal{X}^*(t_0)$.

intervals $[t_{l-1}, t_l]$ after which we can switch between the safety and the optimal controller during online control (see Sec. 5.3).

Note that we again use a reference trajectory for obtaining motion primitives; however, in principle, our approach also works without it. Not using a reference trajectory might be interesting for computing a region of attraction or a control invariant set.

5.5.3 Zonotope Order Reduction

Due to the addition of input generators in the backward reachable set computation, the number of generators and therefore the order of the reachable sets increases during each iteration. To limit the computational effort, we restrict the generator number in each iteration by under-approximating the reachable set $\mathcal{X}(\tau_{lh})$ with a zonotope $\bar{\mathcal{X}}(\tau_{lh}) \subseteq \mathcal{X}(\tau_{lh})$ of a maximum number of $\bar{p}$ generators, as illustrated in Fig. 5.2(b) (Alg. 6, line 5). There exists a number of methods to compute an enclosing zonotope with fewer generators, see for example [74, 83], [226]. However, for our case, we need an under-approximative approach, which is harder to compute.

Given a zonotope $\mathcal{X} = \langle c, g^{(1)}, \dots, g^{(p)} \rangle$ we first sort the generators according to their length:

$$\|g^{(a_1)}\|_2 \geq \cdots \geq \|g^{(a_p)}\|_2, \tag{5.11}$$

where $a \in \mathbb{N}^p$ is a vector of indices that defines the order. To obtain a reduced-order zonotope with $\bar{p} \geq n$ generators, we keep the $\eta = \bar{p} - n$ best generators and under-approximate the zonotope defined by the remaining generators with a box:

$$\bar{\mathcal{X}} = \langle c, g^{(a_1)}, \dots, g^{(a_\eta)}, \delta_1 e^{(1)}, \dots, \delta_n e^{(n)} \rangle, \tag{5.12}$$

where $e^{(i)} \in \mathbb{R}^n$ is the i-th unit vector. The scaling factors $\delta_1, \dots, \delta_n$ are determined by the following linear program:

$$\max_{\delta_1,\dots,\delta_n} \sum_{i=1}^{n} \delta_i \tag{5.13}$$

$$\text{s.t. } \langle \mathbf{0}, \delta_1 e^{(1)}, \dots, \delta_n e^{(n)} \rangle \subseteq \langle \mathbf{0}, g^{(a_{\eta+1})}, \dots, g^{(a_p)} \rangle, \tag{5.14}$$

$$\delta_i \geq 0, \ \forall i \in \{1, \dots, n\}. \tag{5.15}$$

To check in (5.14) if the reduced-order zonotope $\bar{\mathcal{X}}$ is contained in the original zonotope $\mathcal{X}$, we can use Lemma 1 after converting the zonotope $\mathcal{X}$ to a polytope in half-space representation according to [74, Theorem 2.1]. For high-dimensional zonotopes $\mathcal{X}$ or for those with many generators, computing a half-space representation can become computationally challenging. In this case, the zonotope containment problem (5.14) can also be solved using the approaches from [184] and [185].

5.5.4 Backward Reachable Set

Let us now describe how to compute an estimation of the backward reachable set based on the linearized dynamics for each of the larger time intervals $[\tau_{(l-1)h}, \tau_{lh}],\ l \in \{1, \dots, M\}$ (see Sec. 5.5.2). For each time interval, we apply (5.6) to compute the backward reachable set as

illustrated in Fig. 5.2(b) (Alg. 6, line 6). Since we use a reference trajectory, we cannot use the whole inputs for the feedback controller, but have to reduce the available inputs by the inputs already used for the reference trajectory. We do this by introducing for each input generator $g_u^{(j)}$ a scaling factor $\phi_j(\tau_i) := 1-|\theta_j(\tau_i)| \in [0,1]$, where $\theta_j(\tau_i) \in [-1,1]$ are chosen such that $c_u + \sum_{j=1}^{q} \theta_j(\tau_i) g_u^{(j)} = u_{ref}(\tau_i)$. By substituting the input generators by their scaled counterparts $\phi_j(\tau_i) g_u^{(j)}$, the backward reachable set $\mathcal{X}(\tau_{(l-1)h})$ starting from the under-approximated previous set $\bar{\mathcal{X}}(\tau_{lh}) = \langle \bar{c}_{lh}, \bar{g}_{lh}^{(1)}, \dots, \bar{g}_{lh}^{(\bar{p})} \rangle$ can be expressed as a zonotope analogous to (5.7):

$$\begin{aligned}\mathcal{X}(\tau_{(l-1)h}) = \Big\{ x \in \mathbb{R}^n \;\Big|\; x = x_{ref}(\tau_{(l-1)h}) + \sum_{i=1}^{\bar{p}} \alpha_i \bar{A}^{-1} \bar{g}_{lh}^{(i)} - \sum_{i=(l-1)h}^{lh-1} \sum_{j=1}^{q} \beta_{i,j} \bar{A}^{-1} \bar{B}_i \phi_j(\tau_i) g_u^{(j)}, \\ \alpha_i, \beta_{i,j} \in [-1,1] \Big\} \\ =: \Big\{ x \in \mathbb{R}^n \;\Big|\; x = \hat{c} + \sum_{i=1}^{\hat{p}} \hat{\alpha}_i \hat{g}^{(i)},\ \hat{\alpha}_i \in [-1,1] \Big\},\end{aligned}$$

where

$$\begin{aligned}\hat{c} &:= x_{ref}(\tau_{(l-1)h}), \\ \hat{g}^{(i)} &:= \bar{A}^{-1} \bar{g}_{lh}^{(i)},\ \forall i \in \{1, \dots, \bar{p}\}, \\ \hat{g}^{(\bar{p}+i\,q+j)} &:= -\bar{A}^{-1} \bar{B}_{(l-1)h+i} \phi_j(\tau_{(l-1)h+i}) g_u^{(j)},\ \forall i \in \{0, \dots, h-1\},\ \forall j \in \{1, \dots, q\},\end{aligned}$$

with $\hat{p} = \bar{p} + hq$.

The consequences of the used reference input on the reachable set are illustrated in Fig. 5.3: Since the reference input reduces the available input generators for the controller (see Fig. 5.3(a)), it leads to a smaller backward reachable set (see Fig. 5.3(b)). However, this set is centered around the reference trajectory, so it is exactly the part of the original backward reachable set which we are interested in. Since the over-approximation errors in reachable set computations depend on the size of the reachable set, we obtain less conservative results this way.

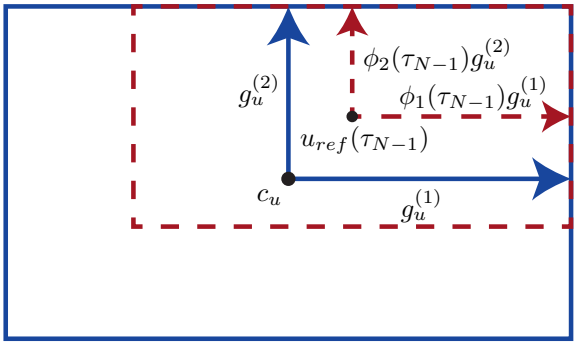
We calculate the control law for the nonlinear continuous-time system (Alg. 6, line 7) analogously to the linear case in Sec. 5.4.2 by solving a linear program equivalent to (5.8) so that $\forall i \in \{0, \dots, h-1\}$:

$$u_{ver}(x, \tau_{k+i}) = u_{ref}(\tau_{k+i}) + G_u(\tau_{k+i}) [\hat{\alpha}_{\bar{p}_{lh}+i\,q+1}(x), \dots, \hat{\alpha}_{\bar{p}_{lh}+(i+1)\,q}(x)]^T, \tag{5.16}$$

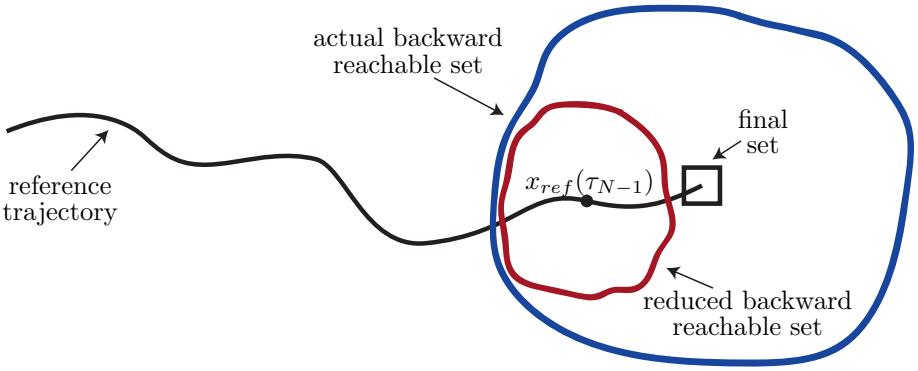
where we again denote the matrix containing all (scaled) input generators by $G_u(\tau_{k+i}) := [\phi_1(\tau_{k+i}) g_u^{(1)}, \dots, \phi_q(\tau_{k+i}) g_u^{(q)}]$. Due to the time-dependent weighting with $\phi_j(\tau_{k+i})$ which accounts for the reference inputs, the matrix $G_u(\tau_{k+i})$ becomes time dependent.

5.5.5 Reachability Analysis

After computing the backward reachable set based on the linearized, time-discretized dynamics without disturbances, we have to check if its forward reachable set with the real, nonlinear dynamics actually ends in the desired final set, see Fig. 5.2(c). If this is not the case, we have to shrink the initial set obtained from the backward reachability analysis by reducing the length of



(a) Comparison input zonotopes



(b) Comparison backward reachable sets

Figure 5.3: Illustration of the input scaling due to the reference trajectory: (a) As we use some input for the reference input $u_{ref}(\cdot)$, we do not have the full input set (blue) available for the feedback controller, but a reduced input zonotope (red). (b) Therefore, the backward reachable set of the reduced input set is not as big as the original one, but it is centered around the reference trajectory as desired.

the generators until the forward reachable set ends up inside the desired final set as illustrated in Fig. 5.2(d).

With most existing reachability approaches, including the ones presented in Sec. 2.6, it is not possible to see how the size of the generators of the reachable set depends on the size of the generators of the initial set and of the applied inputs. This requires us to recompute the whole reachable set for each optimization loop, as is the case for our control approaches using forward reachable sets in Sec. 3.5 and Sec. 3.6. To overcome this problem, we subsequently present a new approach, which maintains the correlation between the generators of the initial set and the generators of the final set so that we can simply minimize an algebraic function to optimize the backward reachable set. The details of this new approach can be found in [228].

Let us start with $\mathcal{X}_0 := \mathcal{X}(\tau_{(l-1)h})$, which we obtain from the backward reachable set computation, and introduce a scaling factor $s_i \in [0, 1]$ for every zonotope factor $\hat{\alpha}_i$. The initial set

with scaling factors is defined as

$$\mathcal{X}_0 := \left\{ x \in \mathbb{R}^n \;\middle|\; x = \hat{c} + \sum_{i=1}^{\hat{p}} s_i \hat{\alpha}_i \hat{g}^{(i)}, \hat{\alpha}_i \in [-1,1] \right\}. \tag{5.17}$$

We use the notation $\mathcal{X}_0 = \mathcal{X}_0(s)$ to denote that the initial set depends on the vector of scaling factors $s = [s_1, \dots, s_{\hat{p}}]^T \in \mathbb{R}^{+\hat{p}}$. The resulting forward reachable set $\mathcal{R}_f(s) := \mathcal{R}_{h\Delta\tau, u_{ver}, \mathcal{W}}(\mathcal{X}_0(s))$, using [76] is represented as a polynomial zonotope:

$$\mathcal{R}_f(s) = \left\{ x \in \mathbb{R}^n \;\middle|\; x = c_f + \sum_{i=1}^{v} \left(\prod_{k=1}^{y} s_k^{E_{k,i}} \right) \left(\prod_{k=1}^{y} \delta_k^{E_{k,i}} \right) g_f^{(i)} + \sum_{j=1}^{v_I} \mu_j g_{I,f}^{(j)},\ \delta_k, \mu_j \in [-1,1] \right\}. \tag{5.18}$$

The expression in (5.18) contains the desired analytical relation describing how changes in the initial set $\mathcal{X}_0(s)$ influence the final reachable set $\mathcal{R}_f(s)$ (Alg. 6, line 8). The independent generators $g_{I,f}$, resulting from disturbances and over-approximation errors, are the only part which we cannot influence and which therefore always increase the final reachable set $\mathcal{R}_f(s)$. Since they are usually rather small compared to the dependent generators, they do not affect the results much. If they get too large, we can always refine the reachable set computation, see [76]. Alternatively, if the reachable set is much smaller after scaling, one can do an extra iteration and compute the new reachable set starting from the reduced initial set, which will lead to smaller over-approximations due to linearization errors. For our numerical examples in Sec. 5.6, this was never necessary. Let us demonstrate in the following numerical example how the analytical relations between initial set and reachable set can be used to quickly find subsets which satisfy a given constraint:

Example 1. *We consider the example shown in Fig. 5.4, where the initial set is*

$$\mathcal{X}_0(s) = \left\{ x \in \mathbb{R}^n \;\middle|\; x = \begin{bmatrix} -8 \\ -2 \end{bmatrix} + \begin{bmatrix} 1 \\ -0.5 \end{bmatrix} s_1\delta_1 + \begin{bmatrix} 0 \\ 1 \end{bmatrix} s_2\delta_2,\ \delta_1, \delta_2 \in [-1,1] \right\},$$

resulting in the final forward reachable set (see Fig. 5.4)

$$\mathcal{R}_f(s) = \left\{ x \in \mathbb{R}^n \;\middle|\; x = \begin{bmatrix} 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 2 \\ 0 \end{bmatrix} s_1\delta_1 + \begin{bmatrix} 0 \\ 2 \end{bmatrix} s_2\delta_2 + \begin{bmatrix} 1 \\ 1 \end{bmatrix} s_1^3 s_2 \delta_1^3 \delta_2 + \begin{bmatrix} 0 \\ 0.5 \end{bmatrix} \mu_1, \right.$$
$$\left. \delta_1, \delta_2, \mu_1 \in [-1,1] \right\}. \tag{5.19}$$

Our goal is to find suitable scaling factors $s^* = [s_1^*\ s_2^*]^T,\ s_1^*, s_2^* \in [0,1]$, such that the final reachable set $\mathcal{R}_f(s^*)$ starting from the scaled initial set $\mathcal{X}_0(s^*)$ satisfies the constraint $[1\ 1]^T x \leq 5$ that is depicted in red in Fig. 5.4. Since $\mathcal{X}_0(s)$ and $\mathcal{R}_f(s)$ both depend on the same scaling factors s, we determine s^* such that $\forall x \in \mathcal{R}_f(s^*) :\ [1\ 1]^T x \leq 5$, which is identical to

$$\max_{x \in \mathcal{R}_f(s^*)} [1\ 1]^T x \leq 5. \tag{5.20}$$

Since it holds that

$$\{[1\ 1]^T x \mid x \in \mathcal{R}_f(s^*)\} \overset{(5.19)}{=} \{x \in \mathbb{R} \mid 2s_1^*\delta_1 + 2s_2^*\delta_2 + 2{s_1^*}^3 s_2^* \delta_1^3 \delta_2 + 0.5\mu_1,\ \delta_1, \delta_2, \mu_1 \in [-1,1]\},$$

the constraint in (5.20) *can be equivalently formulated as*

$$\max_{\delta_1,\delta_2,\mu_1\in[-1,1]} \; 2s_1^*\delta_1 + 2s_2^*\delta_2 + 2{s_1^*}^3 s_2^*\delta_1^3\delta_2 + 0.5\mu_1 \leq 5 \tag{5.21}$$

An example for scaling factors satisfying (5.21) *are* $s_1^* = 1$ *and* $s_2^* = 0.625$. *We show the reachable set* $\mathcal{R}_f(s^*)$ *for the scaled initial set* $\mathcal{X}_0(s^*)$ *in Fig. 5.4 and see that it in fact satisfies the constraint.*

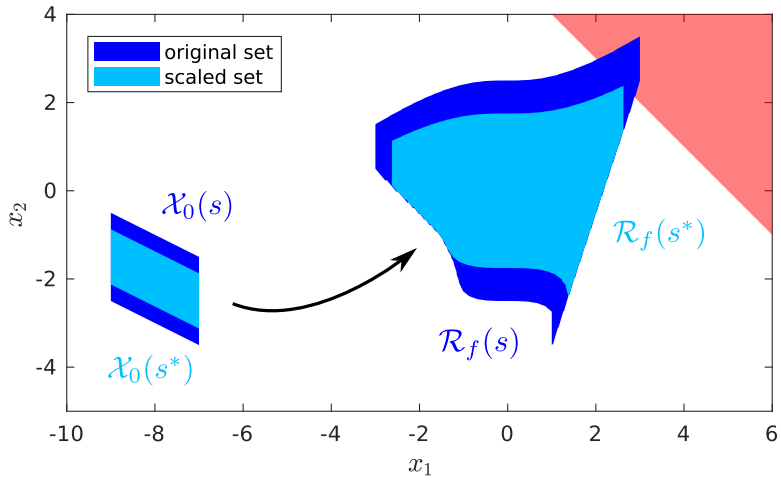


Figure 5.4: Visualization of Example 1: The original initial set $\mathcal{X}_0(s)$ and its corresponding final reachable set $\mathcal{R}_f(s)$ with $s = [1, 1]^T$ are depicted in dark blue, where $\mathcal{R}_f(s)$ violates the constraint shown in red. We reduce the scaling factors s^* such that the final set with the new scaling factors satisfies the constraint. The scaled final set $\mathcal{R}_f(s^*)$ and the corresponding scaled initial set $\mathcal{X}_0(s^*)$ are shown in light blue.

5.5.6 Optimization Problem

Using the ideas demonstrated in Example 1, we are now interested in finding scaling factors $s_i \in [0, 1]$ which maximize the volume of the initial set $\mathcal{X}_0(s)$ for which the forward reachable set $\mathcal{R}_f(s)$ still ends up inside the desired under-approximated final set $\mathcal{X}_g := \bar{\mathcal{X}}(\tau_{lh})$. Experiments in Sec. 5.6 have shown that it is advantageous to only scale the generators resulting from the states, i.e., $s_1, \ldots, s_{\bar{p}}$ and keep the scaling factors corresponding to the input generators equal to one, i.e., $s_i = 1$, $\forall i \in \{\bar{p}+1, \ldots, \hat{p}\}$. This restriction both ensures that we do not change the resulting control law and reduces the number of optimization variables, which leads to a faster convergence to good solutions.

Objective Function Our goal is to maximize the volume of the initial set $\mathcal{X}_0(s)$, i.e., $\max_s \; \mathrm{Vol}(\mathcal{X}_0(s))$. The volume of a zonotope $\mathcal{Z} = \langle c, g^{(1)}, \ldots, g^{(\hat{p})} \rangle$ can be calculated as fol-

lows [186]:

$$\mathrm{Vol}(\mathcal{Z}) = 2^n \sum_{i=1}^{n_{comb}} \left| \det\left(\left[g^{\left(\mathfrak{G}_1^{(i)}\right)}, \dots, g^{\left(\mathfrak{G}_n^{(i)}\right)} \right] \right) \right|,$$

where $\det(\cdot)$ refers to the determinant of a matrix. We denote by $\left\{ g^{\left(\mathfrak{G}_1^{(i)}\right)}, \dots, g^{\left(\mathfrak{G}_n^{(i)}\right)} \right\}$ the n_{comb} possible n-membered subsets of the generator set $\left\{ g^{(1)}, \dots, g^{(\hat{p})} \right\}$. With the scaled initial set $\mathcal{X}_0(s)$ as defined in (5.17), this results in

$$\begin{aligned}
\max_s \ \mathrm{Vol}(\mathcal{X}_0(s)) &= \max_s \ 2^n \sum_{i=1}^{n_{comb}} \left| \det\left(\left[s_{\mathfrak{G}_1^{(i)}} g^{\left(\mathfrak{G}_1^{(i)}\right)}, \dots, s_{\mathfrak{G}_n^{(i)}} g^{\left(\mathfrak{G}_n^{(i)}\right)} \right] \right) \right| \\
&= \max_s \ 2^n \sum_{i=1}^{n_{comb}} \left| \det\left(G_i\right) \det\left(\mathrm{diag}\left(\left[s_{\mathfrak{G}_1^{(i)}}, \dots, s_{\mathfrak{G}_n^{(i)}} \right] \right) \right) \right| \\
&= \max_s \ 2^n \sum_{i=1}^{n_{comb}} \left| \det\left(G_i\right) \right| \left(\prod_{j=1}^{n} s_{\mathfrak{G}_j^{(i)}} \right),
\end{aligned} \tag{5.22}$$

where we combine each generator subset to a matrix $G_i := \left[g^{\left(\mathfrak{G}_1^{(i)}\right)}, \dots, g^{\left(\mathfrak{G}_n^{(i)}\right)} \right]$. Since for high dimensions computing the volume using (5.22) can become hard, we can use p-radius minimization or segment length minimization [187] as alternatives to approximate the volume in a scalable way.

Constraints We have to consider three constraints: (i) the final forward reachable set $\mathcal{R}_f(s) := \mathcal{R}_{h\,\Delta\tau, u_{ver}, \mathcal{W}}(\mathcal{X}_0(s))$ must be located inside the desired under-approximated final set $\mathcal{X}_g := \bar{\mathcal{X}}(\tau_{lh})$, (ii) the reachable set for the whole time interval must be inside the state constraint set $\mathcal{S}$, and (iii) the scaling factors s_i for the state generators must be between 0 and 1, i.e.,

$$s_i \in [0, 1], \ \forall i \in \{1, \dots, \bar{p}\}. \tag{5.23}$$

In theory, one could also consider scaling factors which are larger than one. In this case, one would have to recompute the reachable set to ensure that the actual reachable set still satisfies the constraints, as the analytic relation is only over-approximative for subsets of the original set. However, linearization errors and external disturbances all increase the size of the reachable set for the nonlinear system compared to the backward reachable set based on the linearized and undisturbed dynamics. Therefore, in all practical examples, we have to reduce the size of the initial set rather than increase it. Worst case, the constraint (5.23) leads to some conservatism, but it ensures that the formal guarantees always hold.

As we see in (5.21) of Example 1, checking if the polynomial zonotope $\mathcal{R}_f(s)$ is located inside the target set results in an optimization problem which is computationally expensive to solve. Therefore, we compute a zonotope over-approximation $\bar{\mathcal{R}}_f(s)$ of $\mathcal{R}_f(s)$ from (5.18) according to [75, Prop. 4]:

$$\bar{\mathcal{R}}_f(s) = \left\{ x \in \mathbb{R}^n \;\middle|\; x = \bar{c}_f(s) + \sum_{i=1}^{v} \bar{\delta}_i \bar{g}_f^{(i)}(s) + \sum_{j=1}^{v_I} \mu_j g_{I,f}^{(j)}, \ \bar{\delta}_i, \mu_j \in [-1, 1] \right\},$$

where $\bar{c}_f(s)$ and $\bar{g}_f^{(i)}(s)$ are polynomial functions in s. Using Lemma 1, the check if $\bar{\mathcal{R}}_f(s)$ is located inside the half-space representation of $\mathcal{X}_g = \langle C_{\mathcal{X}_g}, d_{\mathcal{X}_g} \rangle_H$ simplifies to checking the following inequality:

$$C_{\mathcal{X}_g}\bar{c}_f(s) + \sum_{i=1}^{v} |C_{\mathcal{X}_g}\bar{g}_f^{(i)}(s)| + \sum_{j=1}^{v_I} |C_{\mathcal{X}_g} g_{I,f}^{(j)}| \leq d_{\mathcal{X}_g}. \tag{5.24}$$

If the computation of the half-space representation of $\mathcal{X}_g$ becomes too complex, we can again use the approximations from [184] or [185].

As shown in [228], the reachable sets of intermediate time intervals

$$\mathcal{R}_{[\tau_k,\tau_{k+1}]}(s) := \mathcal{R}_{[\tau_k,\tau_{k+1}],u_{ver},\mathcal{W}}(\mathcal{X}_0(s)),\ \forall k \in \{0,\ldots,h-1\},$$

can be expressed in the same way and depend on the same scaling factors s. Therefore, we can apply the same method to also ensure that none of the reachable sets $\mathcal{R}_{[\tau_k,\tau_{k+1}]}(s)$ with their over-approximations

$$\bar{\mathcal{R}}_{[\tau_k,\tau_{k+1}]}(s) = \left\{ x \in \mathbb{R}^n \;\middle|\; x = \bar{c}_k(s) + \sum_{i=1}^{v} \bar{\delta}_i \bar{g}_k^{(i)}(s) + \sum_{j=1}^{v_I} \mu_j g_{I,k}^{(j)},\ \bar{\delta}_i, \mu_j \in [-1,1] \right\},$$

violate the state constraints $\mathcal{S}$ by checking

$$C_{\mathcal{S}}\bar{c}_k(s) + \sum_{i=1}^{v} |C_{\mathcal{S}}\bar{g}_k^{(i)}(s)| + \sum_{j=1}^{v_I} |C_{\mathcal{S}} g_{I,k}^{(j)}| \leq d_{\mathcal{S}},\ \forall k \in \{0,\ldots,h-1\}. \tag{5.25}$$

Here, we assume for a simpler notation that the reachable set algorithm has the same time step size as our control algorithm. If the reachable set algorithm has a smaller time step size, we simply have to check the constraints for all of the smaller time intervals as done in Sec. 3.5 and illustrated in Fig. 3.14.

Optimization Problem With the objective function (5.22) and the constraints (5.23)–(5.25) as derived above, the optimization problem becomes a nonlinear program:

$$\begin{aligned}
&\max_{s_1,\ldots,s_{\bar{p}}} \; 2^n \sum_{i=1}^{n_{comb}} |\det(G_i)| \left(\prod_{j=1}^{n} s_{\mathcal{G}_j^{(i)}} \right) \\
&\text{s.t. } C_{\mathcal{X}_g}\bar{c}_f(s) + \sum_{i=1}^{v} |C_{\mathcal{X}_g}\bar{g}_f^{(i)}(s)| + \sum_{j=1}^{v_I} |C_{\mathcal{X}_g} g_{I,f}^{(j)}| \leq d_{\mathcal{X}_g}, \\
&\quad C_{\mathcal{S}}\bar{c}_k(s) + \sum_{i=1}^{v} |C_{\mathcal{S}}\bar{g}_k^{(i)}(s)| + \sum_{j=1}^{v_I} |C_{\mathcal{S}} g_{I,k}^{(j)}| \leq d_{\mathcal{S}}, \forall k \in \{0,\ldots,h-1\}, \\
&\quad s_i \in [0,1],\ \forall i \in \{1,\ldots,\bar{p}\}.
\end{aligned} \tag{5.26}$$

Since the explicit computation of the forward reachable set is not performed during optimization, (5.26) can be solved efficiently. Based on the optimized scaling factors s^* which are the solution of optimization problem (5.26) (Alg. 6, line 9), we first compute the scaled initial set $\mathcal{X}_0(s^*)$ (Alg. 6, line 10) and then the forward reachable set starting from it, $\mathcal{R}_{[0,h\,\Delta\tau],u_{ver},\mathcal{W}}(\mathcal{X}_0(s^*))$, which results in the reachable set for the considered time interval $\mathcal{R}_{[\tau_{(l-1)h},\tau_{lh}],u_{ver},\mathcal{W}}(\mathcal{X}^*(t_0))$ (Alg. 6, line 11).

5.6 Numerical Example

Safety Controller

We demonstrate our approach for the car example from Sec. 3.3.7. For the safety controller, we construct a maneuver automaton consisting of several motion primitives, each computed with our backward-reachable-set-based control approach from Alg. 6. For the sake of simplicity, we consider the same three motion primitives as in Sec. 3.3.7: *turn left*, *drive straight*, and *turn right*. For all motion primitives, we consider the final set $\mathcal{X}_f = x^{(f)} \oplus [-0.2, 0.2]\,\frac{m}{s} \times [-0.02, 0.02]\,rad \times [-0.2, 0.2]\,m \times [-0.2, 0.2]\,m$, where $x^{(f)} = \left[20\,\frac{m}{s}, \pm 0.2\,rad, 19.87\,m, \pm 1.99\,m\right]^T$, for the *turn left* and *turn right* motion primitives and $x^{(f)} = \left[20\,\frac{m}{s}, 0\,rad, 20\,m, 0\,m\right]^T$ for the *drive straight* motion primitive. We are looking for a initial set $\mathcal{X}(t_0)$ around the desired initial state $x^{(0)} = \left[20\,\frac{m}{s}, 0\,rad, 0\,m, 0\,m\right]^T$, which is as large as possible and for which all trajectories can be steered to the final set after one second. We divide the one second time horizon into $N = 40$ time steps, and consider $M = 10$ time intervals with a horizon of $h = 4$ time steps each, which results in $\Delta t = 0.1\,s$ and $\Delta\tau = 0.025\,s$. During the under-approximation of the reachable sets, we reduce the number of generators to $\bar{p} = 12$.

We implement the control approach again in MATLAB with CORA for the reachable set computations and ACADO for optimizing the reference trajectory. The computations are performed on the same computer as in the previous chapters, see Sec. 3.3.7. The offline computation of the controller for each motion primitive takes around 70 seconds without using parallel computations. Around 77% of this time is used for computing reachable sets (both forward and backward). When we solve this example without the analytical relations between initial set and final set, the numerical optimization algorithm requires us to compute over fifty reachable sets for each time step, compared to only one when using the analytical relations.

We show the resulting reachable set for the *turn left* motion primitive in Fig. 5.5. In Fig. 5.6, we show the obtained initial sets (green) together with the final sets (blue) for all three motion primitives, where we again shift the final sets to lie in the initial sets as done in the numerical examples in Ch. 3. We see that the final sets are much smaller than the initial sets for all motion primitives. Therefore, we are able to connect all motion primitives with each other and obtain the fully connected maneuver automaton shown in Fig. 3.6.

Optimal Controllers

For comparison and a better illustration, we present two different online controllers: first, an LQR-based tracking controller and second, a model predictive controller.

LQR Tracking Controller Instead of using just a single LQR controller, we compute L of them with different weights $Q_{LQR}^{(i)}, R_{LQR}^{(i)},\ i \in \{1, \dots, L\}$, and select the best one online. To compute the LQR controllers, we use the reference trajectory $x_{ref}(\cdot)$ and the corresponding input $u_{ref}(\cdot)$ from the current motion primitive and linearize the system along it. Based on the linearized dynamics, we compute the LQR matrices $K^{(i)}(t),\ i \in \{1, \dots, L\}$, by solving the algebraic Riccati equation [37] for the weights $Q_{LQR}^{(i)}, R_{LQR}^{(i)}$. The LQR tracking controller is

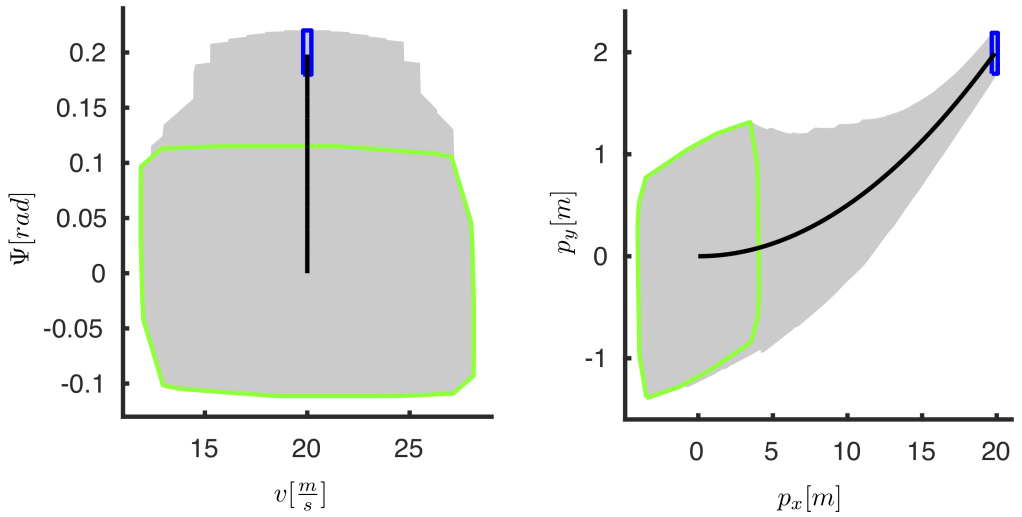


Figure 5.5: Reachable sets of the safety controller for a *turn left* motion primitive projected onto the (v, Ψ) and the (p_x, p_y) planes. The initial set is shown in green and the final set in blue.

therefore given by

$$u_{LQR}(x(t)) = u_{ref}(t) + K^{(i)}(t)(x(t) - x_{ref}(t)).$$

For our example, we compute $L = 2$ controllers, where we keep the state weights constant as $Q_{LQR} = \text{diag}([0.2, 10, 31.2, 1])$ and choose the input weights as $R^{(1)}_{LQR} = \text{diag}([50, 170])$ and $R^{(2)}_{LQR} = \text{diag}([60, 200])$. Lower input weights lead to more aggressive controllers which apply higher inputs to reduce any tracking errors faster and vice versa.

At any point during the online application, we compute the reachable set and check the constraints (5.1)–(5.3) for each controller $K^{(i)}(t)$. Afterwards, we apply the one which satisfies the constraints and has the smallest normalized error to the reference trajectory. This allows us to always aim for the best performance, and the different controllers increase the chances to find one which satisfies the state and input constraints. The usage of multi-core processors allows us to parallelize the computations. If none of the controllers satisfies the constraints, we switch to the safety controller, as described in Sec. 5.3.

Model Predictive Controller Our second controller is a standard MPC that tracks the reference trajectory by optimizing over a horizon of $H = 6$. It has a time step size of $\Delta t_{MPC} = P\Delta\tau$ which, for computational efficiency, is larger than the internal time step size $\Delta\tau$ of the

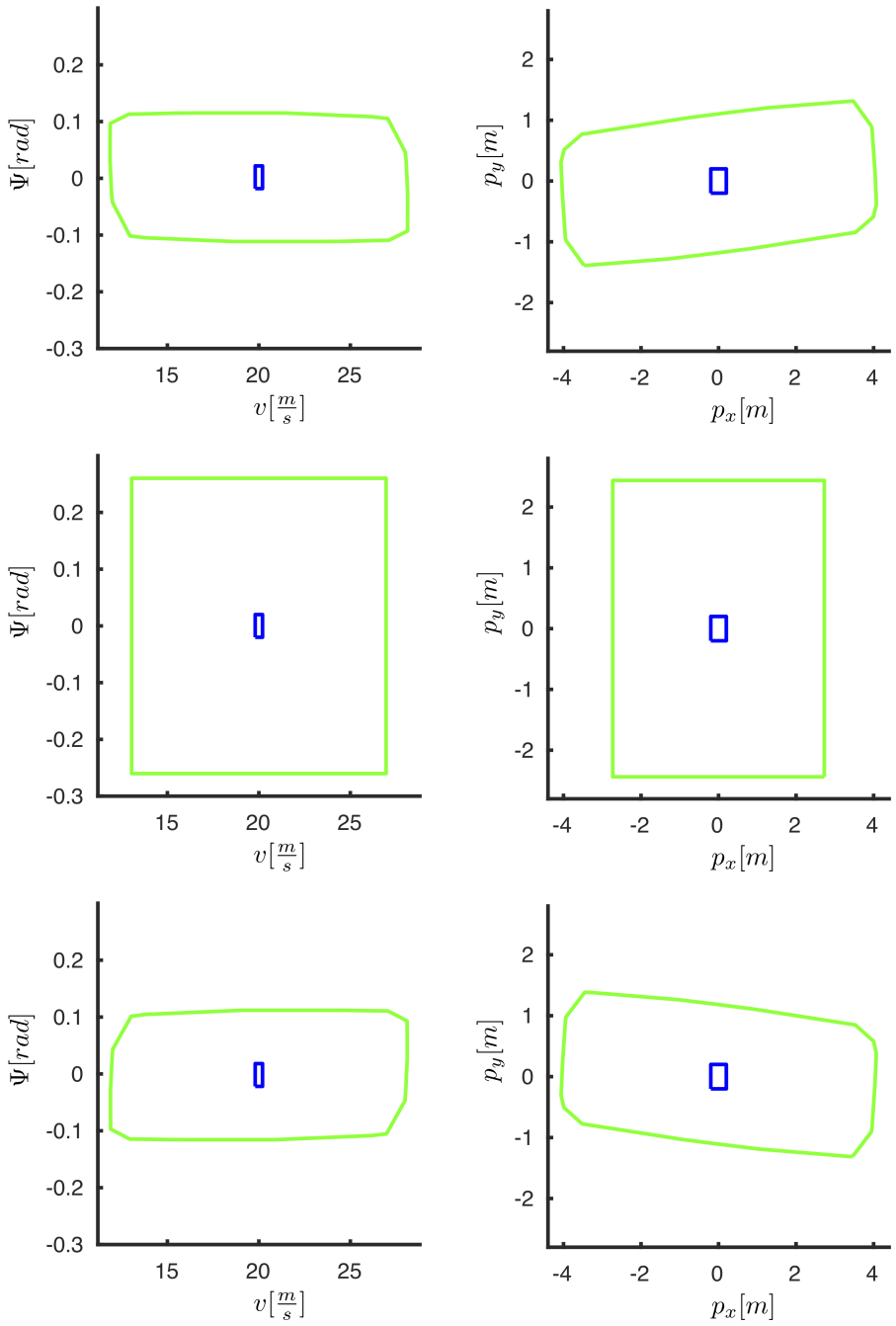


Figure 5.6: Initial (green) and shifted final sets (blue) of the safety controller, projected onto the (v, Ψ) and the (p_x, p_y) planes, for the *turn left* (top), *drive straight* (center), and *turn right* (bottom) motion primitives.

safety controller by a factor of $P = 2$. Its optimization problem is given by

$$\begin{aligned}
\min_{u_{MPC}(\,\cdot\,|\tau_k)} \; & x_d(\tau_{k+PH}|\tau_k)^T Q_{MPC} x_d(\tau_{k+PH}|\tau_k) \\
& + \sum_{i=0}^{H-1} u_{MPC}(\tau_{k+Pi}|\tau_k)^T R_{MPC} u_{MPC}(\tau_{k+Pi}|\tau_k), \qquad (5.27) \\
\text{s.t. } \forall i \in \{0, \ldots, H-1\} : & \\
& u_{MPC}(\tau_{k+Pi}|\tau_k) \in \mathcal{U}, \\
& \xi(x(\tau_k), u_{MPC}(\,\cdot\,|\tau_k), 0, \tau_{P(i+1)}) \in \mathcal{X}^*(\tau_{k+P(i+1)}),
\end{aligned}$$

with $x_d(\tau_{k+PH}|\tau_k) := \xi(x(\tau_k), u_{MPC}(\,\cdot\,|\tau_k), 0, \tau_{PH}) - x_{ref}(\tau_{k+PH})$. In contrast to the LQR tracking approach, we only need a single MPC, as we can include constraints (5.1)–(5.3) directly in the optimization problem of the MPC. As weight matrices we choose $Q_{MPC} = \mathrm{diag}([24, 20, 120, 70])$ and $R_{MPC} = \mathrm{diag}([2, 3])$. By restricting $u_{MPC}(\,\cdot\,)$ to be piecewise constant, we are able to solve (5.27) fast with standard optimal control solvers, such as direct multiple shooting algorithms [3]. In order to reduce conservatism, we use a standard MPC formulation, which optimizes over the nominal dynamics, rather than a more restrictive robust MPC formulation. Therefore, it is possible that the actual, disturbed system violates the constraints. However, for this case, we have the safety net, and the safe controller would take over. It is also possible to tighten the nominal constraints a little bit to reduce the number of cases when the safety controller has to take over.

Results for Combining the Safety and Optimal Controllers

We illustrate the combination of safety controller and online controller for the *turn left* motion primitive. As described in Sec. 2.7, when concatenating two motion primitives, we know that the trajectories for the second motion primitive start inside the final set $\mathcal{X}_f$ of the first motion primitive. Therefore, we choose the set from which our simulated trajectories start to be the final set $\mathcal{X}_f$ shifted around the desired initial state $x^{(0)}$. We compute for both optimal controllers 100 trajectories which start at random states inside this initial set and show them in Fig. 5.7 for the LQR tracking controller and the MPC. For the 100 trajectories with 10 steps each, the safety controller takes over 2.2% of the time steps for the LQR controller and 0.6% for the MPC. As desired, we are able to apply the optimal controller most of the time; however, there are instances when the optimal controller becomes unsafe and the safety controller takes over. For both controllers, we choose the same maximum computation time of $t_c = 0.05\,s$, which is large enough that the safety controller never has to take over because the computation takes too long.

In Fig. 5.8, we show the inputs for the safety net controller with the LQR tracking controller and with the MPC, respectively. Aside from the normalized steering angle b for the MPC, the inputs are not very large and do not change drastically most of the time. This is the desired use of inputs, which of course also depends on the fact that the trajectories start in a small set around the initial state of the reference trajectory. The spikes we see in Fig. 5.8 occur at times when the safety controller takes over. As the safety controller aims to ensure safety for

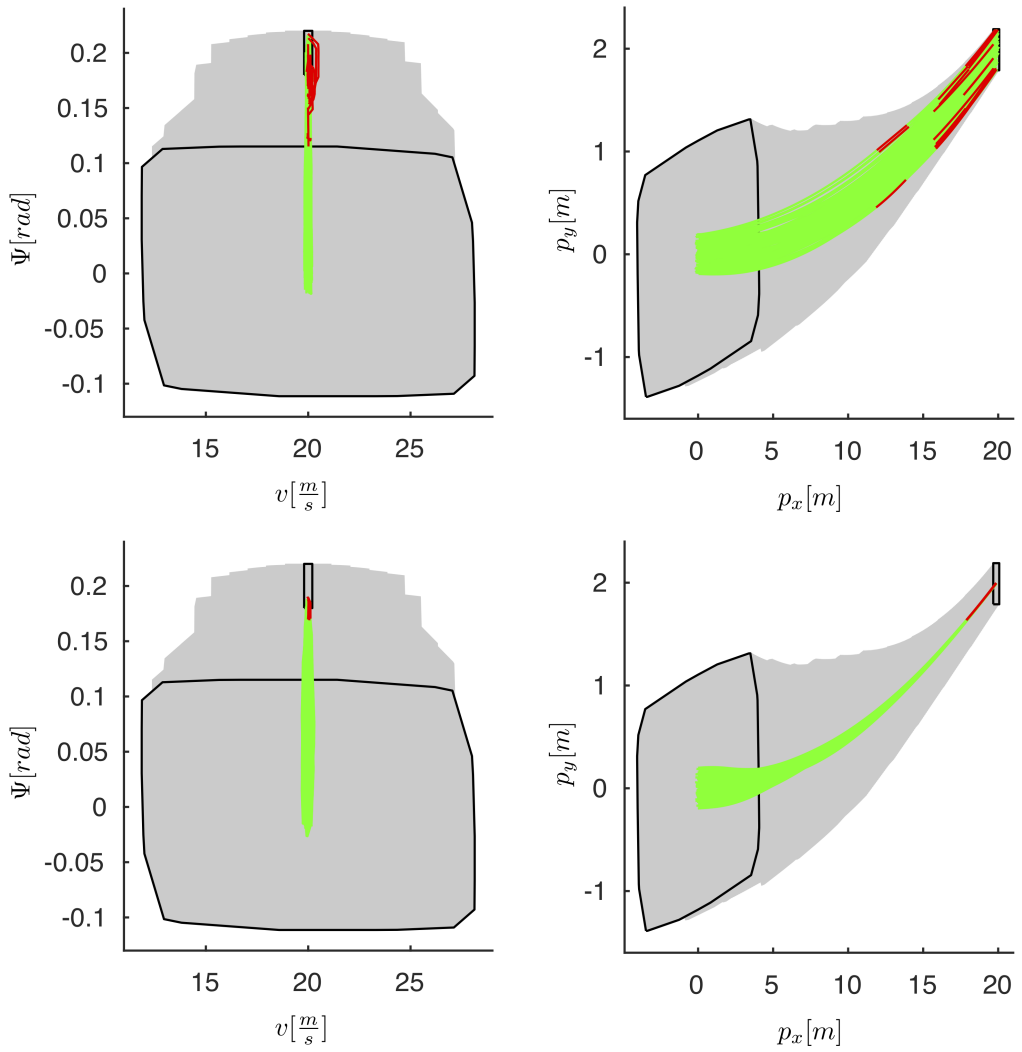


Figure 5.7: Illustration of 100 simulations of the LQR controllers (top) and MPC (bottom) for a *turn left* motion primitive projected onto the (v, Ψ) and the (p_x, p_y) planes. All simulations are randomly chosen from the final set shifted to the center of the backwards computed initial set. We show in green when the LQR controllers and MPC are active and in red when the safety controller takes over.

the largest possible set of states, it uses rather large inputs and switches quickly between them. Therefore, if the LQR controllers and MPC are safe, we benefit from their smoother inputs; if they would become unsafe, the safety controller steps in and brings the system back to a safe state.

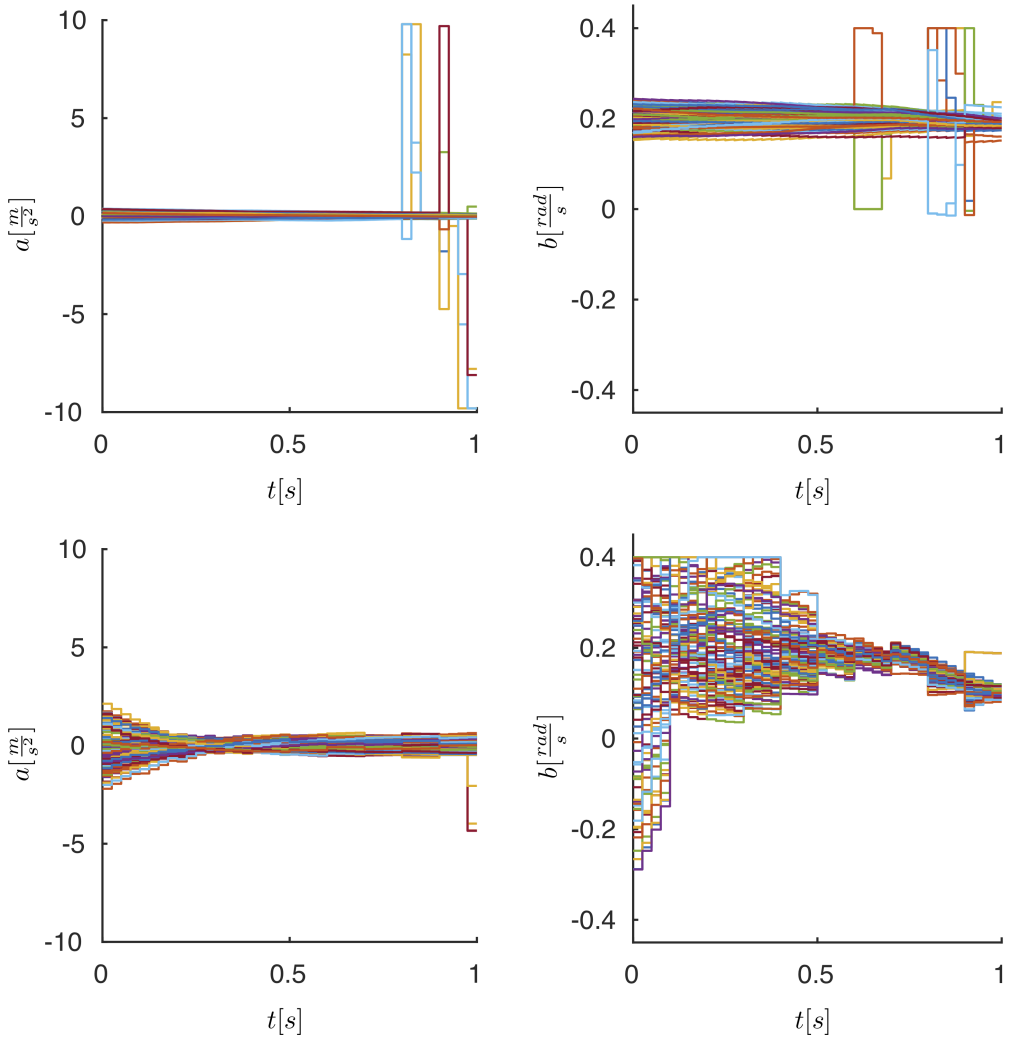


Figure 5.8: Plot of the inputs of the 100 simulations of the LQR controllers (top) and MPC (bottom) with safety net. The spikes occur at times when the safety controller takes over.

Comparison with Convex Interpolation Controller

Finally, we compare the new backward controller with our convex interpolation controller from Sec. 3.3 in Fig. 5.9. We consider the same *turn left* motion primitive as before, but start from a set with the size of the previous final set around the initial state $x^{(0)}$ (shown in red), i.e., the same initial set as in Sec. 3.3.7. We use the convex interpolation controller to compute the smallest forward reachable set (blue) around the desired final state $x^{(f)}$. Starting from this final set, we use our new backward algorithm to compute the maximum initial set (green) for which we can steer all states into the final set of the forward algorithm. We see that this backwards

computed initial set is much larger than the original initial set. This shows the advantage of the backward algorithm for computing a safety net controller, as the larger reachable set allows more freedom for the optimal controller to steer the system without the safety controller having to step in. For a fair comparison, we use the same parameters and algorithms for the reachable set computation for both approaches.

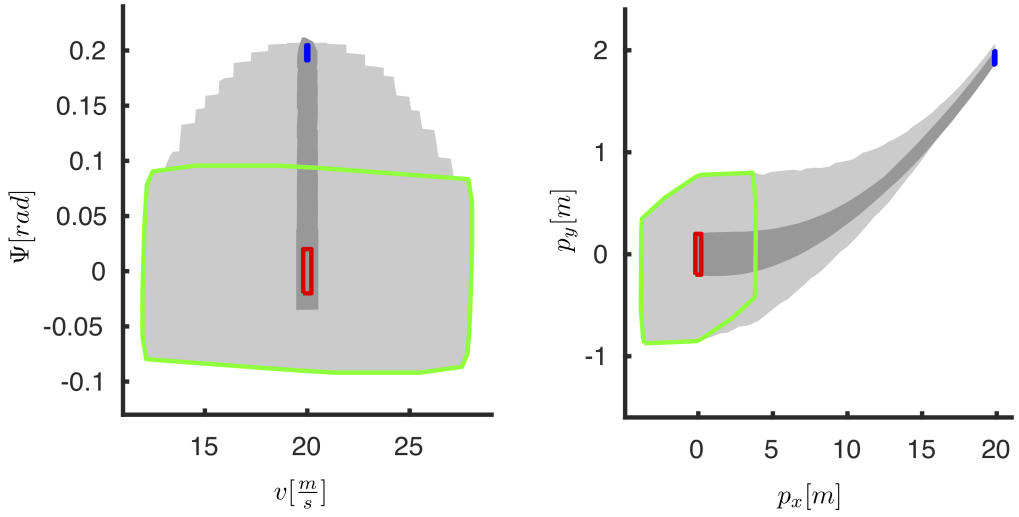


Figure 5.9: Comparison with the convex interpolation controller: Reachable set of the convex interpolation controller for a *turn left* motion primitive projected onto the (v, Ψ) and the (p_x, p_y) planes shown in dark gray. It starts from the red set and ends in the blue. Our new backward controller starts from the blue set and computes the backward reachable set (light gray), which ends in the green set. Since this set is much larger than the original initial set, it provides more flexibility for concatenating motion primitives, as well as freedom for the optimal controller.

5.7 Discussion

Online Computational Complexity

For the online complexity, we have to distinguish the three parts of our safety net framework: the optimal controller, the safety controller, and the switching logic. The complexity of the online controller depends on the applied control approach and is independent of the presented approach. For example, the LQR approach results in a simple matrix vector multiplication with computational complexity of $\mathcal{O}(n\,m)$, with n denoting the number of states and m denoting the number of inputs. Our safety controller requires the solution of a linear program to obtain the parameters $\hat{\alpha}(x)$ and then some matrix vector multiplications to obtain the input. As shown for example in [99], linear programs can be solved with polynomial time complexity in the number of optimization variables and constraints.

The last part is the switching logic, which requires the computation of the reachable set for

the current state. This is only performed for the short time horizon $[t_k - t_c, t_{k+1}]$, i.e., a single controller time step plus the allocated computation time t_c for online controller computation and reachable set computation, and only for a single initial state. Therefore, this can be computed fast. For this reachable set computation, we can use the reachability algorithm from the previous chapters, which has a computational complexity of $\mathcal{O}(n^3)$ and has been successfully applied for the online verification of an autonomous vehicle in [188].

This is also demonstrated in our numerical example, where all online computations are performed in the allocated time $t_c = 0.05\,s$. Note that this was done with a standard MATLAB implementation of the reachable set algorithm and the computation time could be significantly reduced with an optimized implementation and faster programming languages such as C++.

Offline Computational Complexity

The offline computation consists of several parts. As it involves the solution of nonlinear programming algorithms, it is not possible to provide a fixed bound on the computational complexity, as discussed for the offline algorithms in Sec. 3.7. Similar to the discussion in the previous chapters, we discuss the computational complexity of the different parts. The computation of the reference trajectory is done by solving a single optimal control problem. Even though it is a nonlinear optimization problem, this can be solved fast in practice, especially when restricting the inputs to be piecewise constant. The backward reachable set computation of the linearized dynamics involves only matrix multiplications, whose computational complexity is less than $\mathcal{O}(n^3)$, if $n \geq m$.

The forward reachability algorithm used in this chapter is computational more demanding than the one used in Ch. 3 and has a computational complexity of $\mathcal{O}(n^5)$ [76]. During the controller optimization, we solve a nonlinear program, where we cannot bound the number of iterations. In each iteration, however, we do not have to recompute the reachable set, only the approximation of the reachable set based on the scaling factors as optimization variables, which has a computational complexity of just $\mathcal{O}(n^2)$ [228]. This is less than the computational effort to compute the reachable set with the algorithm used in Ch. 3, which has a complexity of $\mathcal{O}(n^3)$ and which is computed in every iteration of the optimization algorithm for the continuous feedback controller from Sec. 3.5 and the combined controller from Sec. 3.6. Note that the simpler optimization of this chapter is possible for the backward controller synthesis since we have a fixed control law and the reachable set only depends on the size of the initial set, which is maximized. We cannot use this approach to optimize the continuous feedback controllers in Sec. 3.5 and Sec. 3.6, as changing the controller affects the closed-loop dynamics and therefore requires recomputing the reachable set in each iteration.

Optimality

During the application of our safety net framework, the optimality of the controlled system mainly results from the optimal controller. Consequently, it depends on the chosen controller type and what system effects are modeled, e.g., if constraints or disturbances are taken into account properly. As discussed before, the safety of the system is always ensured due to the

safety controller; however, the more effects the optimal controller takes into account, the less the safety controller has to step in and the more we can benefit from the advantages of the optimal controller. For a perfect optimal controller, which also never becomes unsafe, the only loss in optimal behavior would result from overly cautious switches to the safety controller if the reachability analysis falsely detects a potential safety risk due to over-approximation errors. As before, this does not lead to unsafe behavior, only possible performance reduction. Since it is not possible to compute exact reachable sets for most systems [80], this behavior cannot be avoided if one wants to ensure safety at all times.

Regarding the optimality of our safety controller, analogously to the approaches in Ch. 3, we cannot guarantee the convergence to a global optimal solution, as we still rely on solving nonlinear programming algorithms, which cannot provide such guarantees. As discussed in Sec. 3.7, this is also not possible for any existing approaches for constrained and disturbed nonlinear systems. With our new approach, however, we maximize the size of the initial set for which we can control all states under observance of constraints on states and inputs to the desired final set, despite the effect of disturbances and, if considered during the forward reachable set computation, sensor noise. This consequently maximizes the time during which we can apply the optimal controller.

Note that depending on the application, smaller reachable sets might make it easier to find feasible solutions during online planning. This would then lead to a trade-off between more freedom for the optimal controller versus easier online planning. If this is the case, we can always limit the size of the reachable sets during the backward controller synthesis of our new approach. Alternatively, we can also use the methods from [228] to compute subsets (online) of the reachable sets, either based on measurements of the actual states or by using the final sets of previous motion primitives during online planning.

5.8 Summary

We present a novel safety net control approach by combining an unverified optimal controller with a verified safety controller. Most of the time, the optimal controller is active and only if its behavior would become unsafe does the safety controller take over. We are thereby able to ensure safety of optimal controllers which cannot guarantee safety on their own. Since optimal controllers have a better control performance than safety controllers, which always have to consider the worst-case behavior, this combination leads to a better control outcome than pure safety controllers, while still maintaining safety guarantees.

We also present a novel way for computing the motion primitives of the safety controller, which is based on (i) backward reachable sets and (ii) circumventing the problem of computing under-approximations of reachable sets, and (iii) using polynomial zonotopes to obtain an analytical correlation between initial sets and reachable sets. As a result, our approach is the first that is able to formally safeguard an optimal controller for disturbed nonlinear systems under state and input constraints with the use of a set-based safety-net controller whose underlying reachable set computation scales polynomially. This drastically simplifies the computational effort for the optimization compared to existing approaches and allows an efficient computation

of large maneuver automata. Similar to our approaches from the previous chapters, since most computations can be performed offline in advance, the online control is fast and efficient. Since the resulting control law is simple to apply and the controller synthesis does not require a deep understanding of control theory or finding Lyapunov functions, the control approach is appealing for practical application. We demonstrate the effectiveness of our approach in the numerical example by safeguarding an LQR tracking controller and an MPC for an autonomous vehicle.

The results of this chapter therefore expand the state of the art in three directions:

(i) We present the first approach which is able to compute under-approximative backward reachable sets for general nonlinear systems while considering inputs and disturbances. While this is not possible for this general case with existing approaches like [156–167], it is required for synthesizing set-based controllers with safety guarantees. In addition, our novel reachability algorithm allows us to obtain analytical dependencies between the reachable set and its initial set and inputs acting on the system. These dependencies make it possible to substantially speed up the computation of the set-based controller, as the reachable set does not need to be recomputed in each iteration of the optimization. As shown in [228], this explicit dependency also simplifies many other verification problems.

(ii) We use the results from (i) to develop a new control algorithm based on backward reachable sets, which allows us to maximize the set of states that can be controlled to a desired final set while having formal guarantees for constraint satisfaction even for disturbed nonlinear systems. At the same time, it benefits from the same advantages as the approaches in Ch. 3, as it provides a control law by optimizing over all possible trajectories starting from an initial set under all possible disturbance realizations. Since we use reachable sets, we do not need to discretize the state space, as done in [50–71], or explicitly compute invariant sets, in contrast to, for example, [174, 175], nor do we need to know Lyapunov functions or rely on sums-of-squares programming, as in [44–49]. The resulting online controller again has a simple structure, which is easy to store and applicable to fast systems, and it is particularly suitable for the application as a safety net controller.

(iii) We provide a formal safety net control concept which allows us to formally safeguard an unverified optimal controller. By using reachability analysis based on the actual state of the real system, the safety controller predicts the behavior of the optimal controller, thereby ensuring it will always intervene if safety is at risk. Compared to approaches such as [171–175,177,180,181], we consider disturbed nonlinear systems. In contrast to the few other approaches which use reachable sets, such as [43], our reachable set algorithm scales polynomially, which allows for a much better scaling of the size of the considered system dimensions. As a result, this algorithm can even be used for checking the safety of the optimal controllers online. In contrast to most other approaches, we explicitly consider the computation time for our controller and reachable set computations during the online verification. We are even able to guarantee safety without requiring a worst-case execution time, since we can safely switch to the safety controller if the execution time exceeds the allocated time.

Chapter 6

Online Controller Synthesis

6.1 Introduction

In the last decades, model predictive control [12,13] has gained a lot of interest, both in academia as well as in industry. Advantages of this control method include its ability to provide optimized control trajectories and to handle constraints for both states and inputs. This, and its rather easy implementation, are the main reasons for its popularity in industry. With growing interest in safety-critical applications, current techniques for model predictive control have to be enhanced to provide formal guarantees of correct behavior despite complex dynamics, limited actuation capabilities, external disturbances, and sensor noise.

As discussed in the introduction to this thesis, there already exists a number of different approaches for robust MPC [22–28]. Yet, the question of how to obtain efficient and provably-safe MPC for constrained and disturbed nonlinear systems remains open. Due to the implicit nature of regular MPC and the limited online computation time, it is often not possible to apply formal verification tools like reachability analysis in order to prove safety. While some approaches compute reachable sets online [30, 31], though only for discrete-time systems, they face the problem that their reachability analysis methods produce large over-approximations. That is also the case for other techniques with implicit safety guarantees, e.g., the contraction-based approach [29], which easily becomes rather conservative, since the contraction set has to hold everywhere in the considered state space. Explicit MPC [14–21], on the other hand, does not encounter problems from online optimization. However, due to the division of the state space, the computation scales exponentially with the number of dimensions and constraints, so that only small-dimensional systems can be considered.

Another problem which most MPC approaches face is that most of them (with the exception of, e.g., [189]) do not consider computation time, therefore neglecting the fact that the time which is needed to perform the optimizations after obtaining a new measurement leads to delays and therefore possibly unsafe and unstable behavior. From a practical point of view, it is also often a problem that most techniques require a Lyapunov function in order to prove stability or to compute invariant sets. For complex, real-world systems, Lyapunov functions are often

hard to obtain.

The goal of this chapter[1] is to combine reachability analysis with regular MPC in order to obtain provably safe controllers for disturbed nonlinear systems with constraints on states and inputs. Our new approach aims at transferring safety guarantees from reachability analysis to model predictive control. As a result, users are not required to know any Lyapunov functions, nor have any other deep knowledge of control theory. This makes our approach particularly appealing for problems in practice. We are even able to consider the delay caused by the computation time of our approach. We also do not need to precompute a fixed tube size as required in many previously-mentioned approaches, which reduces conservatism. In addition, we are able to take continuous-time dynamics and measurement noise, which are neglected in many existing approaches but are critical to providing safety guarantees, into account.

6.2 Problem Formulation

We consider a very similar problem as in Sec. 3.2, with the main difference that here, instead of computing motion primitives offline and using them for online planning, we want to find an MPC algorithm which solves the problem online. Therefore, we consider a single initial state $x^{(0)} \in \mathcal{S}$ which should be steered in finite time into a goal set $\mathcal{X}_f \subseteq \mathcal{S}$ around a goal state $x^{(f)} \in \mathcal{X}_f$ while minimizing some cost function. We still consider the same disturbed nonlinear system (2.3) subject to state and input constraints (3.2) and (3.3), respectively.

As mentioned in the previous chapters, we could easily extend the approaches to uncertain measurements by including them as an additional disturbance affecting the feedback controller in the reachability analysis. We use the online control of this chapter as an opportunity to demonstrate how to consider them explicitly, which we do by using (2.4).

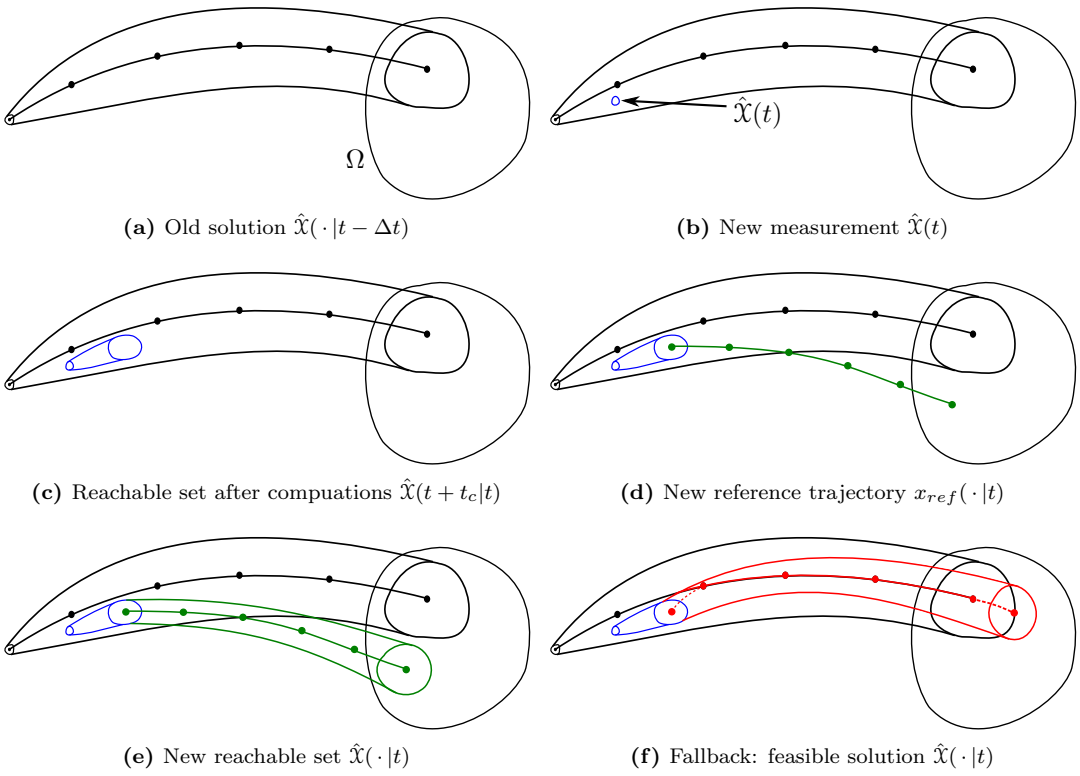
6.3 Reachset Model Predictive Control

6.3.1 Overview

The basic idea of our reachset MPC is illustrated in Fig. 6.1 and presented in more detail in Alg. 7. Starting from the solution of the previous step $t - \Delta t$ (Fig. 6.1(a)), we obtain a measurement $\hat{x}(t)$ at time t (Fig. 6.1(b)). As there might be measurement noise, we only know that we are in some uncertain set $\hat{\mathcal{X}}(t)$ (see (2.5)), which is a singleton when the state can be precisely measured. Based on this measurement, we aim at synthesizing the optimal controller which steers the system to the goal set $\mathcal{X}_f$. Since we cannot optimize for an infinite time horizon, we use a dual-mode MPC [190]. This means we consider a final prediction horizon of length t_N and require that the prediction ends in a terminal region Ω (defined formally later in Def. 18) for which we know a safe and stabilizing controller.

Based on the obtained measurement, we optimize a new reference trajectory $x_{ref}(\cdot\,|t)$, which is tracked with a predefined feedback controller. To solve the optimization problem and

[1] This chapter, including the figures, is based on [229] © 2018 IEEE. The implementation of the numerical example was done by Niklas Kochdumper.



(a) Old solution $\hat{\mathcal{X}}(\cdot\,|t-\Delta t)$ **(b)** New measurement $\hat{\mathcal{X}}(t)$

(c) Reachable set after compuations $\hat{\mathcal{X}}(t+t_c|t)$ **(d)** New reference trajectory $x_{ref}(\cdot\,|t)$

(e) New reachable set $\hat{\mathcal{X}}(\cdot\,|t)$ **(f)** Fallback: feasible solution $\hat{\mathcal{X}}(\cdot\,|t)$

Figure 6.1: Illustration of our reachset MPC approach: (a) Beginning with a feasible solution set $\hat{\mathcal{X}}(\cdot\,|t-\Delta t)$ from the previous time step, (b) we obtain the measurement of the (possibly uncertain) state at time t. (c) Based on this set of possible states, we compute the reachable set $\hat{\mathcal{X}}(t+t_c|t)$ (blue) for the time t_c which we need to solve the optimization problem. (d) Starting with the center of this reachable set, we optimize the reference trajectory $x_{ref}(\cdot\,|t)$ (green) for the time horizon t_N. (e) After the optimization, we compute the corresponding reachable set $\hat{\mathcal{X}}(\cdot\,|t)$ (green). If all constraints are satisfied for the reachable set, we use the new reference trajectory and continue with the next iteration at time $t+\Delta t$. (f) If the solution is not feasible or is not computed in time, we follow the feasible solution (red) from the previous time step, which is extended by the auxiliary controller in the terminal region Ω.

to compute the reachable set, we need some time $t_c \leq \Delta t$ analogous to Sec. 5.3, and we apply the controller from the previous prediction to the system during this time. Using reachability analysis, we predict where we will end after the optimization and computation of the reachable set and use this set $\hat{\mathcal{X}}(t+t_c|t)$ as the initial set for our optimization problem (Fig. 6.1(c)). As in the previous chapter, we use again the notation $\hat{\mathcal{X}}(t+t_c|t)$ to refer to the prediction for time $t+t_c$ made at time t.

For efficiency reasons, we solve the optimization problem for the reference trajectory only, but with tightened constraints (Fig. 6.1(d)). We then use reachability analysis to check if all

Algorithm 7 Reachset MPC Algorithm

Input: system dynamics $f(x,u,w)$, measurement function $h(x,\nu)$, cost function $J(\,\cdot\,)$, initial feasible solution $u_{feas,0}(\,\cdot\,)$, goal state $x^{(f)}$, goal set $\mathcal{X}_f$, terminal region Ω, terminal controller K_Ω, tracking controller K, time step size Δt, prediction horizon length t_N, maximum computation time t_c, constraint sets $\mathcal{S},\mathcal{U},\Phi,\bar{\mathcal{S}},\bar{\mathcal{U}},\bar{\Phi}$, disturbance and noise sets $\mathcal{W},\mathcal{V}$, parameters $\alpha,\rho,\bar{\alpha}$, state measurements $\hat{x}(t)$

Output: control inputs $u(\,\cdot\,)$

Initialize: $t \leftarrow 0,\ u_{ref}(\,\cdot\,|-\Delta t) \leftarrow u_{feas,0}(\,\cdot\,)$
while $\hat{\mathcal{X}}(t) \not\subseteq \Omega$ **do**
  $u(\tau) \leftarrow u_{ref}(\tau|t-\Delta t) + K(\hat{x}(\tau) - x_{ref}(\tau|t-\Delta t)),\ \forall \tau \in [t, t+t_c]$
  $u^{(f)}_{ref}(\,\cdot\,|t) \leftarrow$ feasible solution (6.3)
  $u^{*}_{ref}(\,\cdot\,|t) \leftarrow$ solution of optimization problem (6.6)
  if Optimization problem feasible & solved in time & (6.4), (6.13)–(6.15) satisfied **then**
    $u_{ref}(\,\cdot\,|t) \leftarrow u^{*}_{ref}(\,\cdot\,|t)$
  else
    $u_{ref}(\,\cdot\,|t) \leftarrow u^{(f)}_{ref}(\,\cdot\,|t)$
  end if
  $u(\tau) \leftarrow u_{ref}(\tau|t) + K\big(\hat{x}(\tau) - x_{ref}(\tau|t)\big), \quad \forall \tau \in [t+t_c, t+\Delta t]$
  $t \leftarrow t + \Delta t$
end while
$u(\tau) \leftarrow K_\Omega\big(\hat{x}(\tau) - x^{(f)}\big),\ \forall \tau \geq t$

possible solutions $\hat{\mathcal{X}}(\,\cdot\,|t)$ are guaranteed to satisfy all constraints (Fig. 6.1(e)). Only if this is the case, and if the computations finish in the allocated time t_c, do we apply the new, guaranteed-safe solution. If not, we use a feasible solution which consists of the solution from the previous step, extended by the safe controller from the terminal region (Fig. 6.1(f)). Therefore, under the common assumption that we know a feasible trajectory at the initial time, we always have a feasible solution. This solution is used as a backup if we cannot find a better feasible solution in the available time. We then apply the solution for time Δt before we start the next optimization problem based on the new measurement. The feasible solution is defined as:

Definition 17 (Feasible Solution)**.** *The feasible solution is a possible non-optimal input trajectory, which leads to trajectories* $\xi\big(x(t), u(\,\cdot\,), w(\,\cdot\,), \cdot\big)$ *satisfying the constraints* (3.2)–(3.3) *and ends in the terminal region* Ω *after time* t_N*:* $\xi\big(x(t), u(\,\cdot\,), w(\,\cdot\,), t+t_N\big) \in \Omega$.

In the following, we explain all steps of our approach in detail and discuss the guarantees at the end of this section.

6.3.2 Dual-Mode MPC

As is common in MPC, we use dual-mode MPC [190] to limit the prediction horizon. We use the control law

$$u_\Omega(\hat{x}(t)) = K_\Omega\big(\hat{x}(t) - x^{(f)}\big) \tag{6.1}$$

to stabilize a terminal region Ω around the goal state $x^{(f)}$ (Alg. 7, line 14) and the control law

$$u_{MPC}(\hat{x}(t)) = u_{ref}(t) + K\big(\hat{x}(t) - x_{ref}(t)\big), \tag{6.2}$$

which controls the system into the terminal region (Alg. 7, line 11). Here, $u_{ref}(\,\cdot\,)$ denotes the reference inputs, which is optimized online, and $x_{ref}(\,\cdot\,)$ refers to the corresponding state trajectory. The feedback matrices $K \in \mathbb{R}^{m\times n}$ and $K_\Omega \in \mathbb{R}^{m\times n}$ can be different from each other, and K can even be time-varying as discussed at the end of this section. As in previous chapters, we use linear controllers for faster computation times; however, all concepts presented also work for nonlinear controllers. This includes the case when the goal set $\mathcal{X}_f$ cannot be stabilized with a fixed feedback matrix, and we therefore need a more complex control law $u_\Omega(\,\cdot\,)$. The terminal region Ω is defined as a region of attraction in which the state and input constraints are satisfied despite the influence of disturbances and measurement noise:

Definition 18 (Terminal Region)**.** *Given a dynamical system of the form* (2.3) *and a terminal control law* (6.1)*. The terminal region* Ω*,* $\mathcal{X}_f \subseteq \Omega \subseteq \mathcal{S}$*, is defined as*

$$\begin{aligned}\Omega = \Big\{ x \in \mathbb{R}^n \;\Big|\; & \forall t \in \mathbb{R}_0^+,\ \forall w(t) \in \mathcal{W},\ \forall \hat{x}(t) \in \{h\big(\xi(x, u_\Omega(\hat{x}(\,\cdot\,)), w(\,\cdot\,), t), \nu(t)\big) \mid \nu(t) \in \mathcal{V}\},\\ & \exists t_f \in \mathbb{R}_0^+ :\\ & \xi(x, u_\Omega(\hat{x}(\,\cdot\,)), w(\,\cdot\,), t_f) \in \mathcal{X}_f,\\ & \xi(x, u_\Omega(\hat{x}(\,\cdot\,)), w(\,\cdot\,), t) \in \mathcal{S},\\ & u_\Omega(\hat{x}(t)) \in \mathcal{U} \Big\}.\end{aligned}$$

Using a terminal region is standard in many MPC approaches and is required to provide guarantees beyond the finite prediction horizon [190]. It is computed before the controller is applied online. There exist different ways to compute an approximation of an invariant set of a controller; many of them use Lyapunov functions, which might be hard to find in practice. While a region of attraction can also be computed using Lyapunov functions, there also exist methods to compute them automatically, and in many cases, more efficiently, using reachable sets [191]. The region of attraction is usually much larger than a positive invariant set, which provides more flexibility to our approach. In addition, by checking the satisfaction of the constraints during the execution of the algorithm from [191], we can automatically compute a safe region of attraction, i.e., a region of attraction for which the state and input constraints are satisfied despite disturbances.

As is common in dual-mode MPC, we also use this terminal region to obtain the feasible solution (Alg. 7, line 4) as a backup plan by using the remainder of the previous solution:

$$u_{ref}^{(f)}(\tau|t + \Delta t) := u_{ref}(\tau|t),\ \forall \tau \in [t + \Delta t + t_c, t + t_N]. \tag{6.3}$$

Once we reach the terminal region at time $t + t_N$, we switch to the terminal controller (6.1).

During operation, we compute future reachable sets $\hat{\mathcal{X}}(t+\tau|t) = \mathcal{R}_{\tau,u_{MPC},\mathcal{W}}(\hat{\mathcal{X}}(t))$, $\tau \in [0, t_N]$, based on the current input trajectory $u_{ref}(\cdot|t)$, with $\hat{\mathcal{X}}(t)$ containing the set of possible actual states based on the measured state $\hat{x}(t)$ (see (2.5)). Note that even though $\hat{\mathcal{X}}(t)$ might be partly outside of the reachable set, we know from the over-approximative nature of the reachability analysis that the real state $x(t)$ must lie inside the reachable set from the previous step, i.e., $x(t) \in \hat{\mathcal{X}}(t|t-\Delta t)$. Therefore, we only have to consider the intersection $\hat{\mathcal{X}}(t) \cap \hat{\mathcal{X}}(t|t-\Delta t)$ as the initial set for the next optimization. This is a common approach used in set-based observers [77–79].

6.3.3 Considering the Computation Time

When starting the optimization for a new measurement, we consider its computation time t_c, as done similarly for the online application of the safety net controller in Sec. 5.3. To be safe, we need to know the reachable set after t_c due to uncertainties and disturbances:

$$\hat{\mathcal{X}}(t + t_c|t) := \mathcal{R}_{t_c,u_{MPC},\mathcal{W}}(\hat{\mathcal{X}}(t)).$$

By applying the reference trajectory plus feedback controller from the previous optimization (Alg. 7, line 3), we know that the reachable set after the optimization time is inside the reachable set from the previous optimization.

The allowed computation time t_c for the optimization and reachability analysis is a user-defined design parameter. As discussed in Sec. 5.3, t_c can be estimated quite well and inappropriate values of t_c do not impede the desired properties in (3.2)–(3.3), as we can always go back to the feasible solution if t_c is not sufficient to find a new solution. We compute the reachable set for this allocated time (see Fig. 6.1(c)). If the optimization algorithm finishes before $t + t_c$, we keep following the previous solution until $t + t_c$, from which point on we apply the new solution. If we reach this point in time without a new feasible solution, we simply keep following the previous feasible solution and start a new optimization at $t + \Delta t$ (see Fig. 6.1(f)).

6.3.4 Contraction Constraint

An important consideration in MPC is to ensure the convergence to the goal set in a finite amount of time. While this could be done using Lyapunov functions, we use an approach similar to [31] which does not require a Lyapunov function. Through the construction of the terminal region using the approach from [191], we know that after reaching the terminal region, we converge in finite time to the desired goal set. Therefore, we only have to ensure that we converge in finite time to the terminal region. To do so, we introduce the distance operator from [31]:

Definition 19 (Distance Operator)**.** *Given sets* $\hat{\mathcal{X}}$ *and* $\Phi = \frac{1}{1+\alpha}\Omega$, *with* $\alpha \in \mathbb{R}^+$, $\|\hat{\mathcal{X}}\|_\Phi$ *is*

defined as

$$\begin{aligned} \|\hat{\mathfrak{X}}\|_\Phi = &\min \beta, \\ &s.t.\ \hat{\mathfrak{X}} \subseteq (1+\beta)\Phi, \\ &\beta \geq 0. \end{aligned}$$

As mentioned in [31], $\|\hat{\mathfrak{X}}\|_\Phi$ is equal to zero if and only if $\hat{\mathfrak{X}} \subseteq \Phi$, and if $x \notin \Omega$, it follows that $\|x\|_\Phi > \alpha$. The authors also show that if $\Phi = \langle C_\Phi, d_\Phi \rangle_H$, with $C_\Phi \in \mathbb{R}^{q \times n}$ and $d_\Phi \in \mathbb{R}^q$, is a polyhedron that contains the origin, i.e., $d_{\Phi,i} > 0,\ \forall i \in \{1, \ldots, q\}$, and if $\hat{\mathfrak{X}} = \langle c, G \rangle$ is a zonotope, then $\|\hat{\mathfrak{X}}\|_\Phi$ can be obtained from the equality

$$\|\hat{\mathfrak{X}}\|_\Phi = \max \left\{ 0, \max_{i \in \{1,\ldots,q\}} \frac{C_{\Phi,i}c - d_{\Phi,i} + \|C_{\Phi,i}G\|_1}{d_{\Phi,i}} \right\},$$

where $C_{\Phi,i}$ denotes the i-th row of C_Φ and $\|C_{\Phi,i}G\|_1$ denotes the sum of the absolute values of vector $C_{\Phi,i}G$.

We use this distance function to define the contraction constraint as

$$\sum_{k=1}^{N(t)-1} \|\hat{\mathfrak{X}}(t + k\Delta t|t)\|_\Phi - \sum_{k=1}^{N(t-\Delta t)-1} \|\hat{\mathfrak{X}}\big(t + (k-1)\Delta t|t - \Delta t\big)\|_\Phi < -\alpha, \tag{6.4}$$

where

$$\begin{aligned} N(t) = &\min k, \\ &\text{s.t. } \hat{\mathfrak{X}}(t + k\Delta t|t) \subseteq \Phi, \\ &k \in \mathbb{N}. \end{aligned} \tag{6.5}$$

This contraction constraint requires the sum of the distances of the reachable sets to get smaller in each time step and therefore ensures that the reachable sets converge to the terminal region Ω. By defining the distance with respect to the tighter set Φ, we ensure a desired contraction rate, as shown later in Thm. 5.

6.3.5 Optimal Control Problem

The optimization problem (Alg. 7, line 5) which is solved online at time t is given by

$$\min_{u_{ref}(\,\cdot\,|t)} J\big(\hat{\mathfrak{X}}(t+t_c|t), u_{ref}(\,\cdot\,|t)\big) \tag{6.6}$$

$$= \min_{u_{ref}(\,\cdot\,|t)} \int_{t+t_c}^{t+t_N} L\big(x_{ref}(\tau|t), u_{ref}(\tau|t)\big)d\tau + V\big(x_{ref}(t+t_N|t)\big)$$

$$\text{s.t. } \forall \tau \in [t+t_c, t+t_N]:$$

$$x_{ref}(t+t_c|t) = \text{center}(\hat{\mathfrak{X}}(t+t_c|t)), \tag{6.7}$$

$$\dot{x}_{ref}(t+\tau|t) = f(x_{ref}(t+\tau|t), u_{ref}(t+\tau|t), 0), \tag{6.8}$$

$$u_{ref}(\tau|t) \in \bar{\mathcal{U}}(\tau|t), \tag{6.9}$$

$$x_{ref}(\tau|t) \in \bar{\mathcal{S}}(\tau|t), \tag{6.10}$$

$$x_{ref}(t+t_N|t) \in \bar{\Phi}, \tag{6.11}$$

$$\sum_{k=1}^{\bar{N}(t)-1} \|x_{ref}(t+k\Delta t|t)\|_\Phi - \sum_{k=1}^{\bar{N}(t-\Delta t)-1} \|x_{ref}(t-\Delta t+k\Delta t|t-\Delta t)\|_\Phi < -\bar{\alpha}, \tag{6.12}$$

where $\text{center}(\hat{\mathfrak{X}}(t+t_c|t))$ refers to the center of the zonotope $\hat{\mathfrak{X}}(t+t_c|t)$ and

$$\begin{aligned} \bar{N}(t) = &\min k \\ &\text{s.t. } x_{ref}(t+k\Delta t|t) \in \bar{\Phi}, \\ &k \in \mathbb{N}. \end{aligned}$$

We minimize the cost function $J(\,\cdot\,)$ in (6.6), consisting of a positive definite state cost $L(\,\cdot\,)$ and a positive definite terminal cost $V(\,\cdot\,)$, with respect to the reference trajectory, which starts from the center of the reachable set (6.7) at time $t+t_c$. To ensure the satisfaction of the constraints for the disturbed, closed-loop dynamics, we use tightened time-dependent input (6.9) and state constraints (6.10), $\bar{\mathcal{U}}(\,\cdot\,)$ and $\bar{\mathcal{S}}(\,\cdot\,)$, respectively, as discussed in the next part. As is common in dual-mode MPC, we have a terminal constraint (6.11), which requires that the reference trajectory ends in a tightened terminal region $\bar{\Phi}$. Finally, we have a contraction constraint (6.12) with parameter $\bar{\alpha} \in \mathbb{R}^+$ (not necessarily equal to α), which ensures convergence to the terminal region Ω.

6.3.6 Tightened Constraints

To be able to apply our MPC approach online, we only optimize the reference trajectory without computing the reachable sets during this optimization. While it is possible to optimize over reachable sets as shown in Sec. 3.5 and Sec. 3.6, this is not possible in real time for fast systems. The authors of [31] propose optimizing over the reachable sets; however, they do not discuss the computation times and their approach is rather conservative as demonstrated later in Sec. 6.4. Instead, we optimize only the reference trajectory and tighten the constraint sets accordingly, such that state and input constraints are met. At the end of the optimization, we perform a

reachability analysis to check if all constraints are actually satisfied. If this is not the case, we always have the feasible solution as a safe fallback.

We initially guess the size of the reachable set and the resulting inputs from the controller based on the reachable set from the feasible solution and verify the solution later. This means that we take the size of the reachable set of the feasible solution at the corresponding time step, scaled by a factor $\rho \in \mathbb{R}_0^+$, and use this set to tighten the constraint sets. To do this in a set-based fashion, we use the Minkowski difference, see Def. 6. This allows us to write the tightened constraints as

$$\begin{aligned}
\bar{\mathcal{S}}(t+\tau) &= \mathcal{S} \ominus \rho\Big(\hat{\mathcal{X}}(t-\Delta t+\tau|t-\Delta t) \ominus x_{ref}(t-\Delta t+\tau|t-\Delta t)\Big),\ \forall \tau \in [t_c, t_N],\\
\bar{\mathcal{U}}(t+\tau) &= \mathcal{U} \ominus K\rho\Big(\hat{\mathcal{X}}(t-\Delta t+\tau|t-\Delta t) \ominus x_{ref}(t-\Delta t+\tau|t-\Delta t)\Big),\ \forall \tau \in [t_c, t_N],\\
\bar{\Phi} &= \Phi \ominus \rho\Big(\hat{\mathcal{X}}(t-\Delta t+t_N|t-\Delta t) \ominus x_{ref}(t-\Delta t+t_N|t-\Delta t)\Big).
\end{aligned}$$

As the reachable sets might change their size, the constraints become time dependent. If this guess is too conservative, we only obtain a sub-optimal solution; if it is too optimistic, we have to go back to the feasible solution. In any case, we have a safe solution in the end.

6.3.7 Guarantees Through Reachability Analysis

After obtaining the reference trajectory, we use the predefined feedback controller to compute the reachable set for the closed-loop dynamics. We start from the reachable set $\hat{\mathcal{X}}(t+t_c|t)$ and compute it for the remaining prediction horizon (see Fig. 6.1(e)). Afterwards, we check if the reachable set satisfies the state and input constraints at all times, if the final reachable set is completely inside the terminal region, and if the contraction constraint is also satisfied for the reachable sets. We do this by checking if $\forall \tau \in [t_c, t_N]$:

$$\hat{\mathcal{X}}(t+\tau|t) \subseteq \mathcal{S}, \tag{6.13}$$

$$u_{ref}(t+\tau|t) \oplus K\left(\hat{\mathcal{X}}(t+\tau|t) \ominus x_{ref}(t+\tau|t)\right) \subseteq \mathcal{U}, \tag{6.14}$$

$$\hat{\mathcal{X}}(t+t_N|t) \subseteq \Phi, \tag{6.15}$$

and checking (6.4). Since $\mathcal{S}$, $\mathcal{U}$, and Φ are all polyhedral sets and since the reachability analysis provides us with reachable sets for time intervals in the form of zonotopes, we can use Lemma 1 to efficiently check if constraints (6.13)–(6.15) are satisfied at all times. If this is the case, we apply the new control input to the system (Alg. 7, line 7) and start with a new iteration step. If the solution does not satisfy all those constraints or if the computation takes longer than the prespecified time, we apply the input from the feasible solution instead (Alg. 7, line 9) and keep the reachable sets from the previous time step.

6.3.8 Reachset MPC Theorem

Theorem 5. *If we know an initial feasible solution at $t = 0$, then Alg. 7 remains feasible for all times and the system robustly converges to the goal set $\mathcal{X}_f$ in finite time. During the whole*

time, the system satisfies the state and input constraints (3.2)–(3.3) *despite disturbances and uncertain measurements.*

Proof. We have to show three things: (i) The system remains recursively feasible, i.e., in each step we can find a feasible solution, (ii) the system reaches the goal set $\mathfrak{X}_f$ in finite time, and (iii) the constraints are satisfied at all times despite disturbances and measurement noise. We keep the proof concise, as many parts follow standard robust MPC techniques, as used in [31].

(i) Recursive feasibility can be shown by induction:
Base Case: For t=0, we know a feasible solution by assumption.
Induction Hypothesis: If we know a feasible solution at time t, then we can always get a feasible solution at $t + \Delta t$.
Induction Step: For every step at time $t + \Delta t$, we know from the over-approximative way of computing the reachable set that we start inside the reachable set of the previous step, i.e., $\hat{\mathfrak{X}}(t + \Delta t) \subseteq \hat{\mathfrak{X}}(t + \Delta t|t)$, for which we know the remainder of the solution from the previous step, i.e., $u_{ref}(t + \tau|t), \forall \tau \in [\Delta t, t_N]$. Since the solution at time t is feasible, it ends in the terminal region, where we know by construction that the terminal controller provides a feasible solution, see Def. 18. Therefore, the previous solution extended by the terminal controller, see (6.3), is always feasible and can be applied if we do not find a better solution in time.

(ii) The terminal region Ω is computed so that any state inside Ω robustly converges to the goal set $\mathfrak{X}_f$ in finite time despite disturbances and sensor noise. Therefore, we only have to ensure reaching the terminal region in finite time. From the contraction constraint (6.4), we enforce reaching the terminal region in at most $\frac{1}{\alpha}\sum_{k=1}^{N} \|\hat{\mathfrak{X}}(t+k\Delta t|t)\|_\Phi$ steps. If we find a new solution, we know from (6.4) that this new solution satisfies the rate of at least $-\alpha$. Let us now show that the feasible solution is also guaranteed to have this convergence rate:

$$\begin{aligned}&\sum_{k=1}^{N(t)-1} \|\hat{\mathfrak{X}}(t + k\Delta t|t - \Delta t)\|_\Phi - \sum_{k=1}^{N(t-\Delta t)-1} \|\hat{\mathfrak{X}}(t + (k-1)\Delta t|t - \Delta t)\|_\Phi \\ &= -\|\hat{\mathfrak{X}}(t|t - \Delta t)\|_\Phi < -\alpha,\end{aligned}$$

where we denote by $\hat{\mathfrak{X}}(t + k\Delta t|t - \Delta t)$ the resulting reachable set from the feasible solution $u_{ref}^{(f)}(\cdot\,|t)$. Since $\hat{\mathfrak{X}}(t + (N(t - \Delta t) - 1)\Delta t|t - \Delta t) \subseteq \Phi$, we know from (6.5) that $N(t - \Delta t) = N(t)+1$ and that $\|\hat{\mathfrak{X}}(t+(N(t-\Delta t)-1)\Delta t|t-\Delta t)\|_\Phi = 0$. Therefore, the difference is only the cost of $-\|\hat{\mathfrak{X}}(t|t - \Delta t)\|_\Phi$. Because $\hat{\mathfrak{X}}(t|t - \Delta t) \not\subseteq \Omega$, it follows from Def. 19 that $\|\hat{\mathfrak{X}}(t|t - \Delta t)\|_\Phi > \alpha$, and therefore the last inequality holds. As we can always revert to the feasible solution, the convergence in finite time is guaranteed.

(iii) Before we apply the new solution, we check the constraints for the over-approximated reachable set of the disturbed system in (6.4) and (6.13)–(6.15). If they are satisfied, then the new solution is safe and can be applied. If they are violated, we apply the safe feasible solution; see (i). □

6.3.9 Extension

As mentioned before, we cannot guarantee that the solution resulting from the reference trajectory which is computed with the tightened constraints (6.9)–(6.12) will satisfy the actual constraints (6.4) and (6.13)–(6.15). While we are always safe, this might make our approach unnecessarily conservative. One way to overcome the problem without getting too conservative is to compute several possible solutions in parallel. Using different estimations of reachable sets and inputs applied by the feedback controller results in several optimization problems with different constraints. As they are completely independent, we can utilize multi-core processors for solving these problems and using reachability analysis in parallel and thus choose the best feasible solution.

For a simpler notation, we did not explicitly consider a time-dependency for the feedback matrix K of the $u_{MPC}(\cdot)$ controller. While we purposefully do not consider optimizing the feedback controller online, as this might take too long, it is easily possible to use time-varying controllers and controllers which depend on the reference trajectory. A simple way, which we demonstrate in the numerical example, is the use of LQR controllers based on the linearized dynamics along the reference trajectory. In this case, we can adapt the controller with respect to the changing linearized dynamics, but keep the Q and R matrices constant to not have to optimize the feedback controller online.

If the feedback controller depends on the newly optimized reference trajectory, this introduces an additional uncertainty when tightening the constraints (6.9)–(6.12). In this case, we can further tighten the constraints to also include the uncertainty of the actual feedback control law or compute more solutions in parallel to reduce conservativeness as discussed before.

6.4 Numerical Example

To compare our reachset MPC control algorithm with the approach from [31], we use the same nonlinear continuous stirred tank reactor system for our numerical example. The model of the reactor for an exothermic, irreversible reaction A $\rightarrow$ B with constant liquid volume is given by [31]:

$$\frac{\mathrm{d}C_A}{\mathrm{d}t} = \frac{q}{V}\left(C_{Af} - C_A\right) - k_0\, e^{-\frac{E}{RT}}\, C_A + w_1, \tag{6.16}$$

$$\frac{\mathrm{d}T}{\mathrm{d}t} = \frac{q}{V}\left(T_f - T\right) - \frac{\Delta H\, k_0}{\rho\, C_p}\, e^{-\frac{E}{RT}}\, C_A + \frac{U\,A}{V\,\rho\, C_p}\left(T_c - T\right) + w_2, \tag{6.17}$$

where C_A is the concentration of A in the reactor, T is the temperature of the reactor and T_c is the coolant stream temperature. The system state is defined as $x = \left[\left(C_A - C_A^0\right), \left(T - T^0\right)\right]^T$, and the system input as $u = T_c - T_c^0$, with the steady state $C_A^0 = 0.5\,\frac{mol}{l}$, $T^0 = 350\,K$, $T_c^0 = 300\,K$. The model parameters can be found in [31].

The set of inputs is $\mathcal{U} = [-20, 70]\,K$ and the uncertainty $w = [w_1, w_2]^T$ is bounded by $w_1 \in [-0.1, 0.1]\,\frac{mol}{l\,min}$ and $w_2 \in [-2, 2]\,\frac{K}{min}$. The example does not consider state constraints and assumes that the state can be precisely measured.

In order to determine a terminal region Ω, we compute an LQR controller for the system linearized at the steady state and desired final state $x^{(f)} = [0, 0]^T$, which results in $K_\Omega = [66.65, -4.86]$. We then use the approach from [191] to calculate Ω as explained before. The time step size of $\Delta t = 1.8\,s$ and a prediction horizon of $t_N = 19.8\,s$, which is equal to $N = 11$ time steps, are the same as in [31]. We keep the reference inputs constant in each time step. The cost functions $L(x_{ref}, u_{ref}) = u_{ref}^T R_c\, u_{ref}$ and $V(x_{ref}) = x_{ref}^T Q_c\, x_{ref}$ are applied, with $R_c = 10^{-12}$, and $Q_c = \mathrm{diag}([100, 1])$. Since no cost function is provided in [31], we use these parameters to best approximate their trajectory. We use $\alpha = \bar{\alpha} = 0.1$ for the contraction parameter and $\bar{\mathcal{U}} = [-18, 68]\,K$ for the tightened input constraint.

For the control law $u_{MPC}(x)$, we apply a time-varying feedback matrix K, where at each time step k, we obtain a new K as an LQR controller for the system linearized at $x' = \frac{1}{2}\big(x_{ref}\,(t + k\Delta t|t) + x_{ref}\,(t + (k+1)\Delta t|t)\big)$ and $u' = u_{ref}\,(t + k\Delta t|t)$. We use the weighting matrices $R = 100$ for the inputs and Q as the identity for the states. In order to reproduce the behavior of the disturbed system during the execution of the algorithm, we simulate the model (6.16)–(6.17) with random values for the disturbances w. For the allocated optimization time, we use the value $t_c = 0.54\,s$.

Our algorithm is implemented again in MATLAB, and we use the ACADO toolbox to solve the optimal control problems with a multiple shooting algorithm. For the reachable set computation we use the CORA toolbox. All computations in this section are performed on a 2.9 GHz quad-core i7 processor with 32 GB memory and without using parallel computations.

The initial solution for the first numerical example with initial state $x^{(0)} = [-0.15\,\frac{mol}{l}, -45\,K]^T$ is displayed in Fig. 6.2. During Alg. 7, the maximum computation time for the optimization and reachability analysis is $0.51\,s < t_c$, which means that we are able to perform all computations in real time. Of this computation time, around 48% is needed for the optimal control algorithm and around 27% for the computation of the reachable sets. As a comparison to our algorithm, Fig. 6.3 shows the initial solution of the robust MPC (rMPC) approach from [31] for the same example. It is clearly visible from Fig. 6.2 and Fig. 6.3 that our reachable sets are smaller than the ones computed with rMPC. Small reachable sets are advantageous because they minimize the probability that the input or state constraints are violated. In addition, there is also a better chance that the sets are located inside the terminal region — even more so in our approach, as we obtain a larger terminal region than the rMPC approach.

Furthermore, the rMPC algorithm exhibits several major disadvantages that our approach is able to avoid: First, it does not provide formal safety guarantees for time-continuous systems, as it only considers time-discretized systems. Second, rMPC directly optimizes over the reachable sets, which leads to large computation times, because the reachable sets have to be calculated for each iteration of the optimization algorithm. To avoid this, we only optimize the reference trajectory and compute the reachable sets only once after the optimization. Third, the technique that rMPC uses for reachability analysis results in larger over-approximations of the real reachable set of the system, as their technique is more conservative than our approach.

In order to compare our approach with the rMPC algorithm, we use the same parameters and same initial point as the authors in [31]. However, the example is not really suited for a good comparison of control approaches, because to stabilize the system from this initial

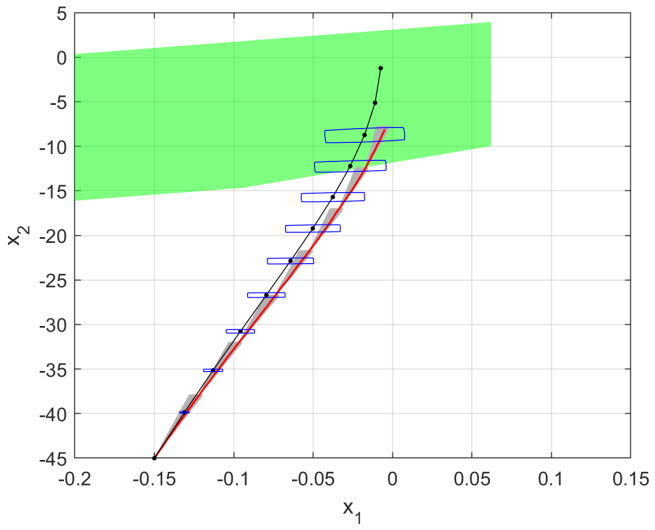


Figure 6.2: Reference trajectory (black) and reachable sets at discrete time points (blue) of the initial solution for our approach. A resulting trajectory of the real system is shown in red, its reachable set in gray, and the terminal region Ω in green.

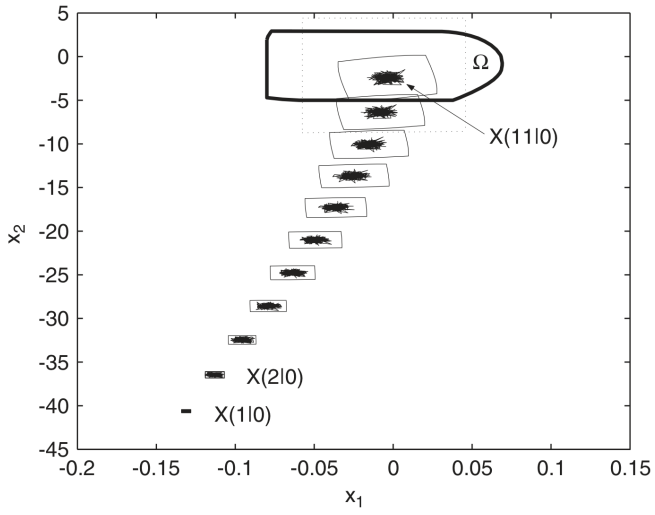


Figure 6.3: Initial solution with reachable sets for the rMPC approach, taken from [31].

point, the maximal available control input has to be applied for nearly the whole time horizon. This does not leave much room for the other objectives, like minimization of the cost function or counteracting disturbances. Therefore, we provide a second example for the initial point $x^{(0)} = [-0.3\,\frac{mol}{l}, -30\,K]$. Compared to the case above, we changed the final prediction horizon to $t_N = 9\,s$ and the input weighting matrix to $R_c = 0.9$. The results are displayed in Fig. 6.4. For this example, the maximum computation time for optimization and reachability analysis

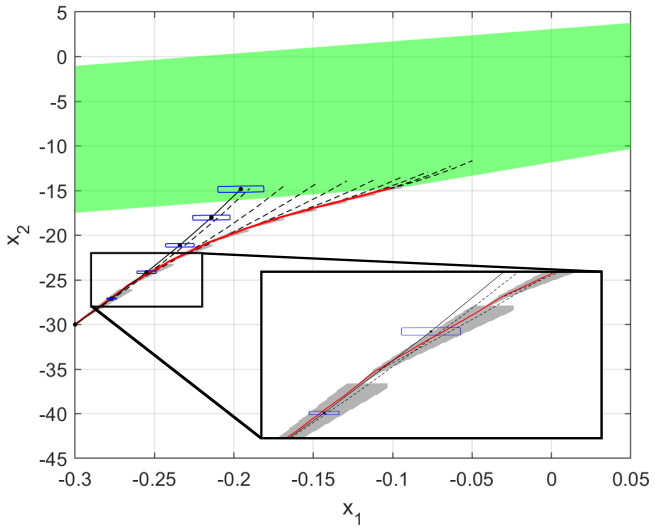


Figure 6.4: Our approach for a different initial point with terminal region Ω (green), reference trajectory (solid black) and reachable sets at discrete time points (blue) of the initial solution, reference trajectories for all iterations (dashed black), real system trajectory (red), and reachable set for the real system trajectory (gray). The resulting reachable sets can be seen better in the magnified section.

is $0.37\,s \leq t_c$ This example nicely demonstrates that our repeated optimization enables finding feasible trajectories that have a lower cost than the initial solution.

6.5 Discussion

The computational complexity for our optimization is the same as for regular MPC. During operation, we solve the optimal control problem (6.7)–(6.12). Since we only solve it for a single state, we can use the same solvers which are developed for solving nonlinear programs and which are used for existing MPC. The only additional computational effort is the reachability analysis [74], which has a complexity of $\mathcal{O}(n^3)$, with n denoting the dimension of the state space.

Because this computation only has to be performed once for the whole time horizon, we are able to do it in real time, as shown in the numerical example. Since we do not optimize over the reachable sets and therefore are not able to obtain a global optimal solution (which is not feasible for nonlinear programs in general), we save a lot of computation time while still guaranteeing safety.

An advantage of our approach is that any kind of feedback controller can be used to track the reference trajectory and counteract disturbances. It is also not necessary to compute the invariant set or some contraction set which has to hold everywhere in the state space. Instead, we compute the actual reachable set based on the predicted future situation, resulting in a less conservative solution.

6.6 Summary

We present a novel reachset MPC algorithm which combines reachability analysis with dual-mode MPC. This demonstrate how to use the proposed combination of reachability analysis and controller optimization in an online fashion to ensure safety for continuous-time, nonlinear systems with disturbances and uncertain measurements. Due to the online computation of reachable sets, we are not restricted to fixed-size tubes as often seen in literature (e.g., [22,25,28]) and therefore are less conservative. In addition, we directly take computation times into account and optimize the trajectory based on the set of initial states after the computation, rather than applying inputs that are computed for states which are measured before the optimization begins.

Compared to the few existing MPC techniques which use reachability analysis [30, 31], our approach has significant advantages, as we are able to provide guarantees for continuous-time systems and we are able to consider measurement noise and computation times, which are neglected by others. We illustrate the advantages of our approach compared to an existing approach in the numerical example and also show that the computations can be performed in real time.

The resulting controller has a simple structure, and it can be implemented using standard reachability tools and optimal control solvers. In addition, as we do not need to know Lyapunov functions, our approach is easy to use and therefore appealing in practice.

Chapter 7

Practical Application

After developing various set-based control algorithms in previous chapters, we now present their application to real systems. We focus on two examples, an autonomous car and a robotic manipulator. We show how the algorithms are applied to these systems and how they can be used for online planning.

7.1 Autonomous Car

Our first example[1] considers an autonomous car. For technical reasons explicated in Sec. 7.1.2, we are only able to consider scenarios without high positive acceleration. Therefore, we focus on obtaining fail-safe trajectories with formal guarantees on safety and driveability. These fail-safe trajectories can be used in addition to a normal trajectory planner and are applied in case this planner is unable to find a safe solution. As this is usually the case in emergency situations, it is crucial to have a solution which guarantees staying in certain safe sets in order to ensure that no collision happens.

We first use conformance checking, as described in Sec. 2.8, to obtain a model that captures the real behavior. Then, we apply the combined control approach to compute a maneuver automaton consisting of safe motion primitives. We use these motion primitives to match online-generated reference trajectories to ensure their safe execution. We demonstrate the combined approach in several simulations based on real traffic data.

[1] The results from this section are based on the master's thesis of Anna-Katharina Rettinger [192], which was supervised by the author, and subsequent collaboration.

7.1.1 Obtaining the Model

We model the car similarly to the numerical examples in previous sections as a kinematic single-track model [193], given as

$$\begin{aligned}\dot{v} &= a + w_a,\\ \dot{\Psi} &= v\,(\kappa + w_\kappa),\\ \dot{p}_x &= v\,\cos(\Psi),\\ \dot{p}_y &= v\,\sin(\Psi).\end{aligned}$$

Here, v denotes the vehicle's velocity, Ψ its orientation, and p_x as well as p_y its positions in the x and y directions. The inputs are the acceleration a and the vehicles curvature κ. The disturbances w_a and w_κ are additive on the corresponding inputs. We model the measurement noise as additive uncertainties for each state of the form

$$h(x, \nu) = x + \nu,$$

with $x = [v, \Psi, p_x, p_y]^T$ and $\nu = [\nu_v, \nu_\Psi, \nu_{p_x}, \nu_{p_y}]^T$.

To be more realistic, we consider a coupled input constraint $\mathcal{U}$

$$\begin{aligned}&\sqrt{\dot{v}^2 + (v\dot{\Psi})^2} \leq a_{max}\\ \Leftrightarrow\ &\sqrt{(a + w_a)^2 + (v^2\,(\kappa + w_\kappa))^2} \leq a_{max},\end{aligned} \tag{7.1}$$

with $a_{max} = 11.5\,\frac{m}{s^2}$, instead of individual input constraints as done in the previous numerical examples. The constraint results from the friction circle and is a coupled constraint on the acceleration and curvature based on the current velocity (see [193] for more details).

7.1.2 Obtaining Real Vehicle Data

To do the conformance checking for this model, we use a collection of real vehicle data which has been recorded on several test drives with an automated car. The test drives cover a number of different scenarios, at various velocities and places. The recorded data contains measured states as well as requested inputs. Due to safety regulations for the automated vehicle, the tracking controller of the automated car is not allowed to directly control the actuators, but rather sends the required inputs to some underlying safety and comfort controllers, which interact with the actuators. Therefore, we have to include the behavior of the underlying controllers in the model of the systems which the resulting controller acts on.

During high positive acceleration, the underlying controllers shift gears, which leads to significant delays and deviation from the requested acceleration. Since we neither have access to nor information about the underlying controllers, it is not possible to include this behavior into the conformant model without being overly conservative. Therefore, we restrict ourselves to driving cases where the vehicle drives with constant velocity or decelerates. Since we focus on fail-safe planning, this usually involves slowing down to avoid collisions, so this does not restrict us too much in this scenario.

7.1.3 Results from Conformance Checking

For the conformance checking, we consider over 15 minutes of driving data from various urban traffic scenarios and evaluate those parts which satisfy the previously discussed constraints. We divide the driving data into smaller test cases and perform the trace conformance checking as described in Sec. 2.8.1 on them.[1] We increase the disturbance and measurement noise sets until conformance can be shown for all test cases. The resulting values of the disturbances and measurement noises are given by

$$w_a \in [-0.75, 0.75]\,\frac{m}{s^2},\ w_\kappa \in [-0.005, 0.005]\,\frac{1}{m},$$
$$\nu_v \in [-0.06, 0.06]\,\frac{m}{s},\ \nu_\Psi = [-0.004, 0.004]\,rad,\ \nu_{p_x} = [-0.02, 0.02]\,m,\ \nu_{p_y} = [-0.02, 0.02]\,m.$$

We show the error traces from some sample test cases from the conformance checking in Figs. 7.1 and 7.2. For better visualization, we normalized the time duration in the plot to $[0, 1]$, as the original test cases have different durations. We present test cases which go close to the boundaries, but we see that no error trace exceeds the maximum error bounds indicated by the thick red lines. This shows conformance for these test cases.

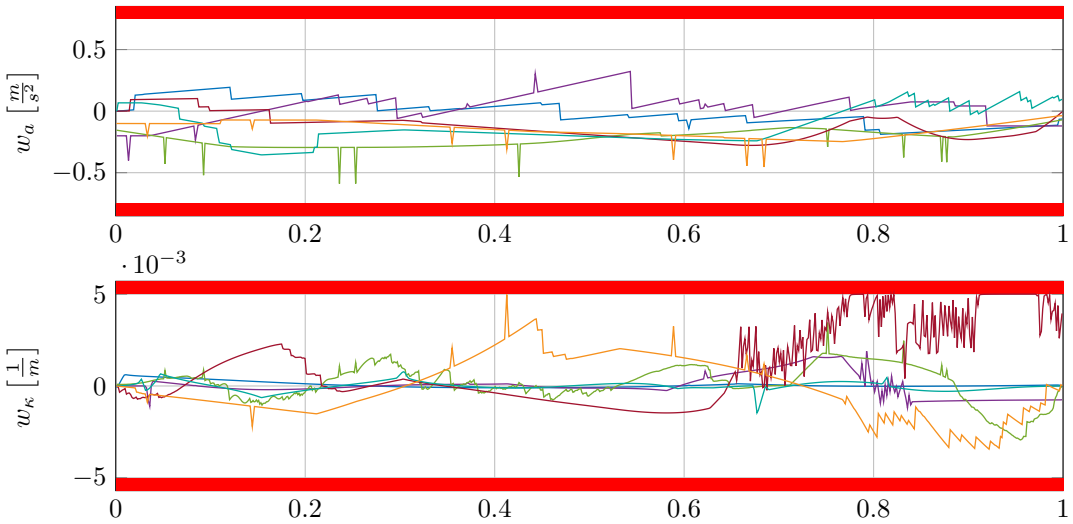


Figure 7.1: Resulting disturbance traces for six example cases from the conformance checking. The x-axes show the time normalized to $[0, 1]$.

7.1.4 Computation of the Maneuver Automaton

With the uncertain model obtained from the conformance checking, we use the combined control approach from Sec. 3.6 to automatically compute the motion primitives for a maneuver automaton offline. Due to positional and rotational invariance of the car dynamics, all

[1] We do this by using a MATLAB algorithm which was developed and implemented by Daniel Heß and which was part of the joint work in [215]. It uses MATLAB's `quadprog` function for solving the fitting problem.

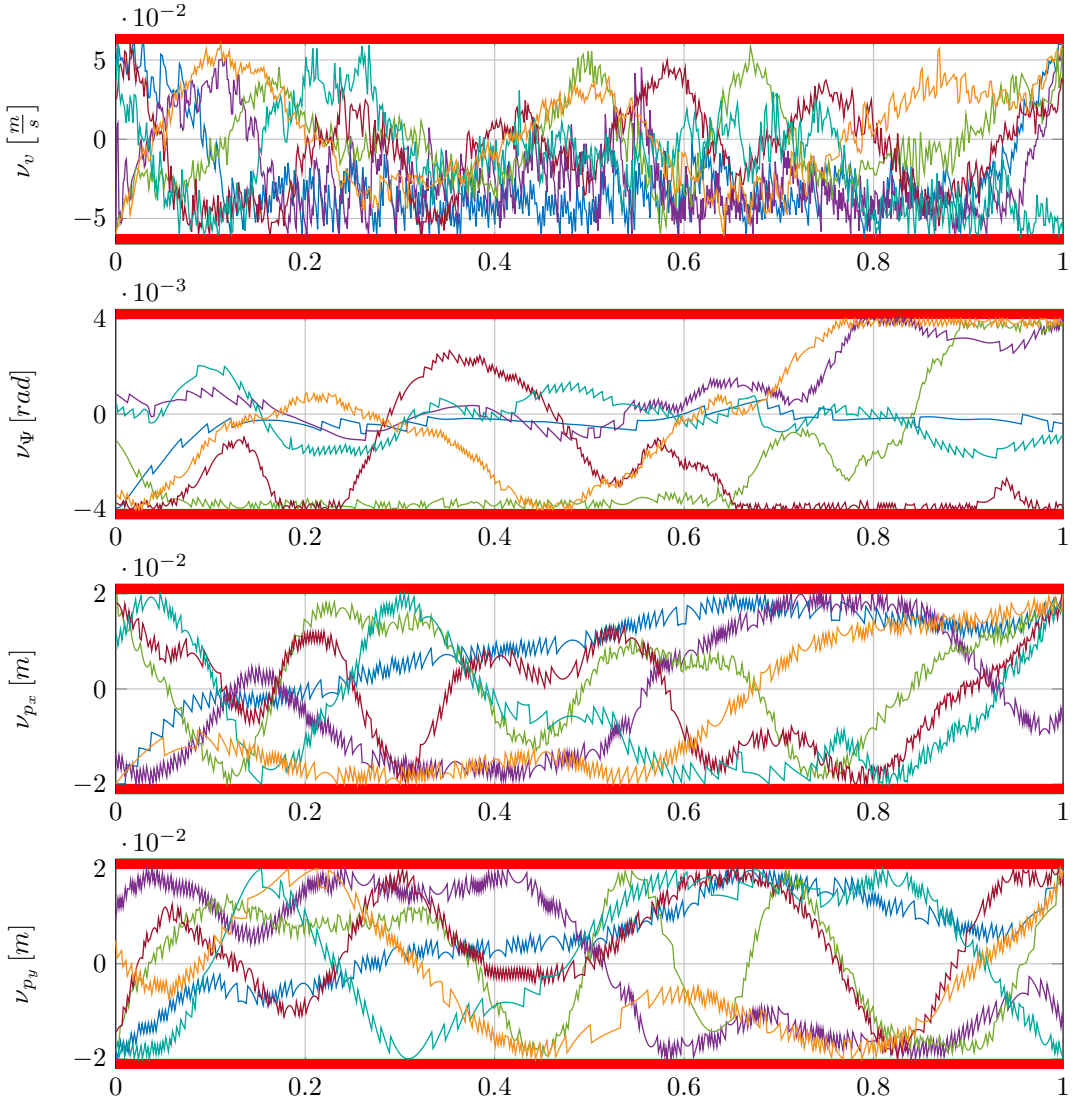


Figure 7.2: Resulting measurement error traces for six example cases from the conformance checking. The x-axes show the time normalized to $[0, 1]$.

reference trajectories of the motion primitives start at 0 for the position and orientation. We only have to sample different velocity ranges. To maximize flexibility, we choose the same size for the initial sets of all motion primitives. We choose the initial set as the box $[v_0 - 0.2, v_0 + 0.2] \frac{m}{s} \times [-0.02, 0.02] \, rad \times [-0.2, 0.2] \, m \times [-0.2, 0.2] \, m$, where v_0 refers to the initial velocity. This allows us to include the constraint that the shifted and rotated final set must be inside this initial set to ensure that we can concatenate any two motion primitives with matching final/initial reference velocities (see Fig. 1.2). To limit the number of considered velocities, we sample them in an interval $[1.2, 15.2] \frac{m}{s}$ with discretization of $\Delta v = 0.2 \frac{m}{s}$. For

velocities below $1.2\,\frac{m}{s}$, we compute specific motion primitives, which are explained later.

Instead of specifying the initial and final states and computing the reference trajectory $x_{ref}(\cdot)$ by solving an optimization problem, we fix the reference input $u_{ref}(\cdot)$ to be constant for the whole motion primitive. We compute $x_{ref}(\cdot)$ by simply integrating the inputs. The duration of each motion primitive is set to $t_f = 1\,s$. By computing all primitives for different input values, we are able to systematically cover the input space. We discretize the acceleration inputs in the interval $[-4, 0]\,\frac{m}{s^2}$ with discretization of $\Delta a_{ref} = 0.2\,\frac{m}{s^2}$. Since we only consider emergency braking maneuvers in this section, we restrict ourselves to nonpositive reference accelerations. The discretization size is chosen such that it matches up with the discretization size of the considered velocities: all acceleration values lead to a decrease of a multiple of $0.2\,\frac{m}{s}$ after $1\,s$, which matches the initial velocities of other motion primitives. For the curvature, we discretize the interval $[0, 0.06]\,\frac{1}{m}$ with discretization $\Delta\kappa = 0.001\,\frac{1}{m}$. For symmetry reasons, we only need to compute motion primitives for driving straight and turning left, as the turning right primitives can be obtained by simply changing the signs in the corresponding dimensions.

To compute the motion primitives, we divide the reference trajectory in $N = 5$ parts of duration $\Delta t = 0.2\,s$ for the state-dependent feedforward controller. Since the reference inputs are determined through our input sampling, we only have to choose $\mathcal{U}_{ff}$. We do this by including the size of U_{ff} into the linear optimization problem for the state-dependent feedforward controller, as noted in Sec. 3.6. Thus, we minimize the size of the reachable set of the nominal system and the required inputs at the same time. This allows us to maximize the remaining input capacities for the feedback controller. Since the optimization problem for the feedforward controller is still a convex problem, this can be solved very fast.

To check the input constraint for the closed-loop dynamics, we compute the zonotope of applied inputs $\mathcal{Z}_u$ for each time interval of the reachable set, as described in the implementation part of Sec. 3.6. The input constraint (7.1) is more involved than in the previous examples, as it is quadratic and depends on one state and several inputs and disturbances. Therefore, we compute the new set $\hat{\mathcal{Z}}_u = \mathcal{Z}_u \oplus \mathcal{W}$, i.e., we add the set of possible disturbances to each input. Lastly, we compute the maximum velocity v_{max} of the reachable set for this time interval and multiply the κ dimension of $\hat{\mathcal{Z}}_u$ by v_{max}^2. If the norm of this new zonotope is smaller than or equal to a_{max}, then the constraint (7.1) is satisfied. Here, we benefit from the fact that the norm of a zonotope can be computed exactly.

In addition to the regular motion primitives which have a minimal velocity of $1.2\,\frac{m}{s}$, we also consider full stop motion primitives for velocities down to $0\,\frac{m}{s}$. We do not consider velocities below $1.2\,\frac{m}{s}$ for regular motion primitives, as for velocities close to zero, the closed-loop model does not accurately capture the real behavior. Since we model braking as a negative acceleration, small disturbances and measurement noises result in negative velocities, which would not happen for the real vehicle. To accurately model this behavior, we use a hybrid model where the braking force always acts against the current velocity. We do this by applying a negative braking force to the system and over-approximate the reachable set until the velocities of all states are nonpositive. This over-approximated reachable set is then used as our braking motion primitive. For simpler computation and since we only want to brake, we do not use a controller here, but perform this open-loop by applying the desired curvature and the constant braking

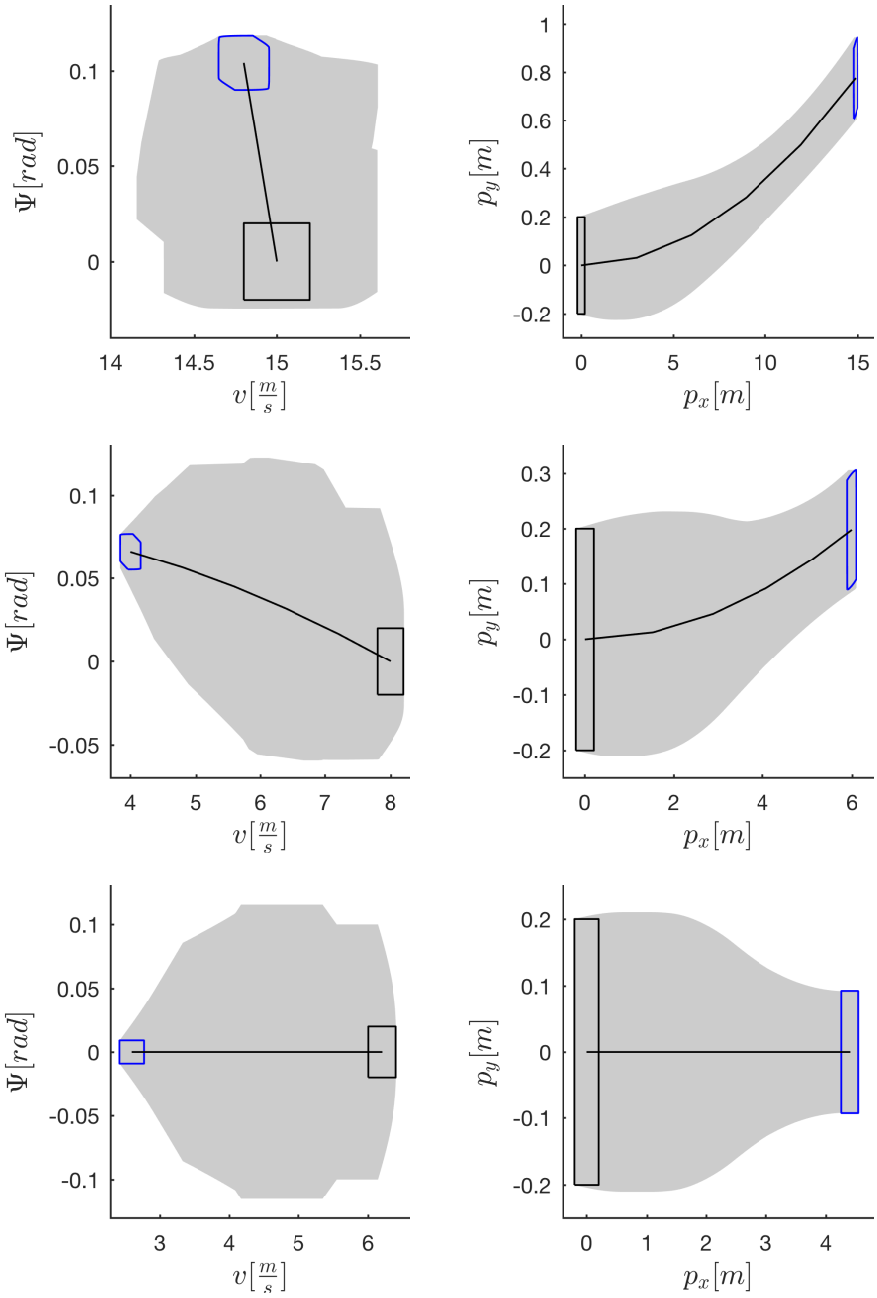


Figure 7.3: Reachable sets of motion primitives A (top), B (center), and C (bottom), projected onto the (v, Ψ) and the (p_x, p_y) planes. The initial sets are plotted in black, the final sets in blue, the reachable sets for all times in between in gray, and the center trajectories as black lines.

force of $-4\,\frac{m}{s}$ for all full stop primitives.

We systematically compute the motion primitives (regular and full stop) by solving the

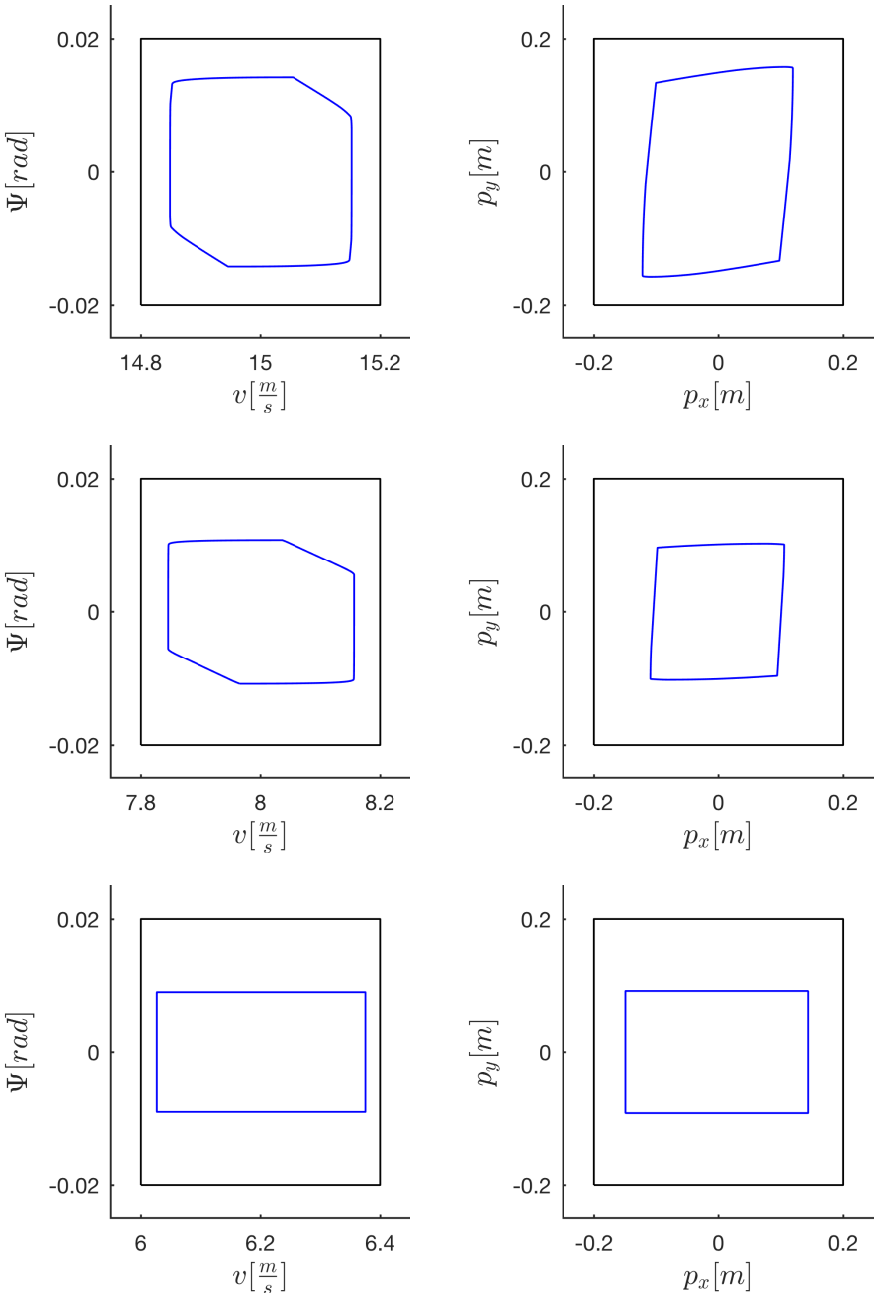
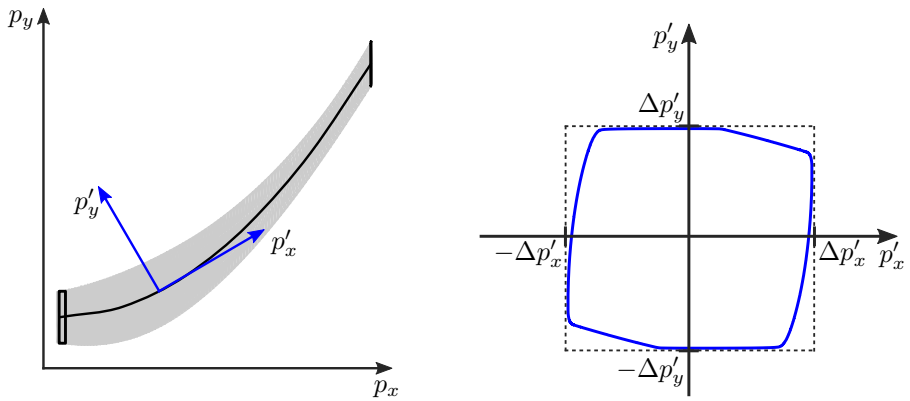


Figure 7.4: Initial (black) and shifted final sets (blue) of motion primitives A (top), B (center), and C (bottom), projected onto the (v, Ψ) and the (p_x, p_y) planes.

optimization problem for each discretized velocity and acceleration, and afterwards store the feasible motion primitives in the maneuver automaton. We use the MATLAB implementation with the toolboxes from Sec. 3.6.5 for the computations. In total, we compute around 16,000

motion primitives. Since we transform all turn left motion primitives into turn right, we obtain a maneuver automaton with over 30,000 motion primitives.

For illustration, we show three example motion primitives A, B, and C with different initial velocities and reference inputs. Motion Primitive A starts with an initial velocity of $v_{ref}(0) = 15\,\frac{m}{s}$ and has a reference input of $u_{ref}(\cdot) = \left[-0.2\,\frac{m}{s^2}, 0.007\,\frac{1}{m}\right]^T$. Motion Primitive B starts with an initial velocity of $v_{ref}(0) = 8\,\frac{m}{s}$ and has a reference input of $u_{ref}(\cdot) = \left[-4\,\frac{m}{s^2}, 0.011\,\frac{1}{m}\right]^T$. Motion Primitive C starts with an initial velocity of $v_{ref}(0) = 6.2\,\frac{m}{s}$ and has a reference input of $u_{ref}(\cdot) = \left[-3.6\,\frac{m}{s^2}, 0\,\frac{1}{m}\right]^T$. We show the reachable sets of these three motion primitives in Fig 7.3 and a comparison of their initial and shifted final sets in Fig. 7.4. By design, the shifted and rotated final sets are all contained inside the initial sets. As this is the case for all 30,000 motion primitives, we see that our algorithm is capable of successfully computing various types of motion primitives, even for systems affected by real-world disturbances and measurement noise.



(a) Relative vehicle coordinates **(b)** Bounds of the reachable set

Figure 7.5: For a simpler representation of motion primitives, we only store the maximum deviation of the reachable sets. (a) To be independent of the orientation, we define a coordinate system relative to the reference trajectory. (b) For each time step, the maximum deviation of the reachable set in these coordinates is computed.

For the online planning, we do not need to know the whole reachable set for each state. In fact, it is sufficient to know the maximum occupancy set in the positions' dimensions. To reduce the amount of stored data and to speed up the online planning, we are only interested in the maximum deviation from the reference trajectory for each motion primitive. For easier use, we compute them in the relative coordinate system p'_x and p'_y based on the orientation of the reference trajectory, see Fig. 7.5(a).

We begin by computing the maximum deviation of any state in the reachable set from the corresponding reference trajectory in dimensions p'_x, p'_y, and Ψ for each motion primitive. We denote these by $\Delta p'_x$, $\Delta p'_y$, and $\Delta\Psi$. For the positions, we do this by bounding the reachable set at each time step by an axis aligned box in the p'_x and p'_y coordinates (see Fig. 7.5(b)) and store the maximum values of the box dimensions of the whole motion primitive. This is the maximum

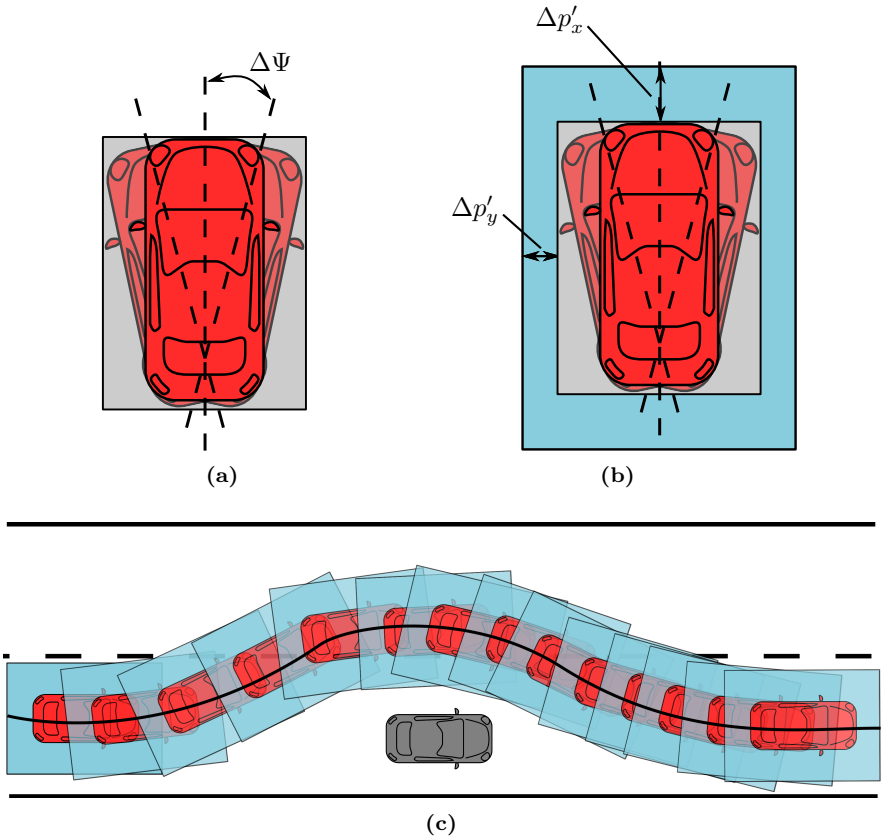


Figure 7.6: Over-approximation of the occupancy set for the car and its use for collision checking: (a) The occupancy set due to uncertain orientations is over-approximated by a box. (b) This box is enlarged in p_x, p_y dimensions for the maximum size of the reachable set. (c) The resulting occupancy set is used for collision checking during online planning.

uncertainty of the reference point of the car. To obtain the maximum occupancy set, we first over-approximate the occupancy resulting from the uncertain orientations, see Fig. 7.6(a), and then enlarge this set by the maximum size of the reachable set in the positions' dimensions, see Fig. 7.6(b). For simplicity's sake, we over-approximate both occupancy sets by boxes as shown in the illustration. We use this over-approximated occupancy set to check online for collisions with other traffic participants or road boundaries (Fig. 7.6(c)). For collision checking, we therefore only have to store for each motion primitive its reference trajectory and the size of its occupancy box in p'_x and p'_y dimensions.

7.1.5 Planning with Motion Primitives

In this subsection, we use the computed maneuver automaton for trajectory planning in different emergency situations. For the online motion planning, we use the approach of matching a

reference trajectory with our motion primitives, as illustrated in Fig. 1.4. To compute the reference trajectory for the fail-safe scenarios, we use the planner from [194]. The planner uses convex optimization to find a trajectory based on a simplified vehicle model without disturbances which takes the occupancies of other vehicles into account.

In our examples, we consider different emergency situations. The scenarios considered are all based on real-world situations which are obtained from real vehicle data from various test drives. They cover both braking and evasive maneuvers. The matching algorithm for the motion primitive is implemented in Python on a computer with an Intel i3 2.2 GHz processor and 8 GB memory.

Given the reference trajectory, the planning algorithm has to select the motion primitives from the maneuver automaton which best match the reference trajectory and do not lead to a collision. The number of possible combinations is M^H, assuming we have M motion primitives for each initial velocity and a horizon of H. Since it grows exponentially with the prediction horizon, it is not feasible to consider all possible combinations for real-time planning. Instead, we need a heuristic which limits the number of checked cases.

The simplest option would be to set $H = 1$ and match one primitive at a time, i.e., find the primitive which best matches the first part of the reference trajectory, then find the best primitive which starts from the first one for the second part of the primitive, and so on. While this limits the number of combinations to consider, the short prediction horizon would make the planning too shortsighted, and it becomes hard to track the reference trajectory.

Therefore, we present a heuristic which limits the number of considered primitives and therefore allows us to predict a horizon larger than one. The basic idea is to only consider motion primitives for which at least one of their reference inputs is close to the average inputs of the fail-safe trajectory. To do so, we divide the reference trajectory from the fail-safe-trajectory planner into intervals of length $t_f = 1\,s$, i.e., the duration of our motion primitives. For each of these intervals, we compute the average acceleration $\bar{a}$ and average curvature $\bar{\kappa}$. For the matching, we now only consider motion primitives whose reference inputs $u_{ref}(\,\cdot\,) = [a_{ref}(\,\cdot\,), \kappa_{ref}(\,\cdot\,)]^T$ during the respective time interval satisfy at least one of the following conditions:

$$a_{ref} \in [\bar{a} - \Delta a, \bar{a} + \Delta a],$$
$$\kappa_{ref} \in [\bar{\kappa} - \Delta\kappa, \bar{\kappa} + \Delta\kappa].$$

Since we compute the motion primitives based on the discretized inputs, we can order them such that the potential primitives can be found very fast. We then check which of these potential motion primitives best matches the reference trajectory. For computational efficiency, we only check this at five discrete points for each primitive, and use a quadratic cost function for the difference between the states of the motion primitive and the reference trajectory at each of these time points. We weight the individual dimensions by $[1, 500, 1, 10]$, with an additional weighting factor of 4 for the final state of each motion primitive. These weights are manually chosen with larger weights for the orientation, as the values are generally smaller, and for the p_y dimension, as these errors are harder to counteract.

We perform the matching in a receding horizon fashion. This means, we are looking for a concatenation of $H = 2$ motion primitives with the lowest costs and which are collision free.

After having found them, we keep the first motion primitive and move the prediction horizon by one and start again from the end of the selected primitive. We continue this until we have found a collision free combination for the whole length of the reference trajectory and start applying the chosen sequence of motion primitives. If we ever reach an iteration where every combination of motion primitives leads to a collision, we use backtracking, i.e., we go back one step and choose the next best motion primitive combination and continue from there.

Note that the presented matching heuristic serves merely as an example for a fast matching algorithm. Since it reduces the base M of the planning complexity M^H, it is possible to plan faster than when considering all possible motion primitives, especially for longer horizons. When comparing the heuristic with a planner which considers all motion primitives, the planning results are quite similar, but the heuristic needs far less computation time, see [192] for a detailed comparison. There exists a large variety of different planning algorithms and heuristics, all with their own advantages and disadvantages. The focus of this thesis is on providing tools to simply compute safe maneuver automata which can be used with any applicable planning algorithm. Therefore, a general comparison of these planning algorithms is outside the scope of this work, and we only use this one heuristic for our tests.

In the following, we consider four different situations for which an emergency plan is necessary: Two with a static obstacle in front of the car and two with another car in front. For each of the two cases, we have one where we brake to stop behind the obstacle or car and one where we change to a different lane.

Scenario 1: Braking Maneuver with Static Obstacle

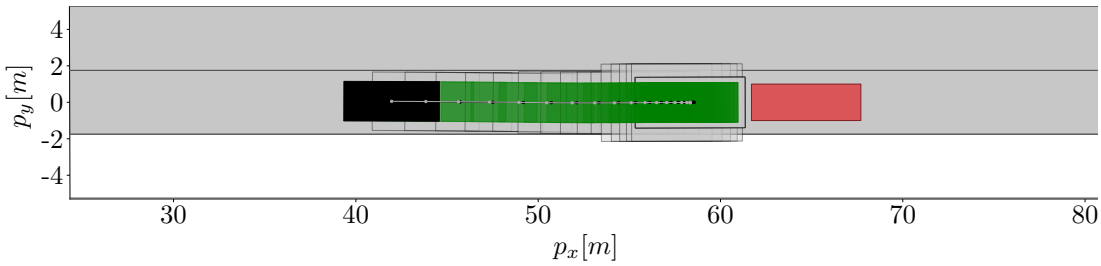


Figure 7.7: Results of our safe motion planning for the first scenario: The ego vehicle (black box) drives towards a static obstacle (red box) and the fail-safe planner plans for emergency braking. The resulting reference trajectory (black line) is matched by the motion primitives. Their centers are shown as the gray line, the occupancy of the car without uncertainty as green boxes, and the resulting occupancies increased by the reachable sets due to uncertainties as black rectangles.

Our first scenario is a braking maneuver, where the ego vehicle, i.e., the vehicle which we control, drives with an initial velocity of $9.6\,\frac{m}{s}$ and has a static obstacle in a distance of $17.1\,m$ in front of it. The results of the scenario are shown in Fig. 7.7. The reference trajectory of length $t_r = 4\,s$ from the planner [194] only needs to brake while staying on the same lane in order to avoid a collision.

Our planning algorithm is able to match this reference trajectory quite accurately, especially considering the sampled velocities and accelerations. In the final step, there is only a small deviation of $0.1\,m$ in p_x dimension and since no steering is necessary, no deviation in p_y or Ψ dimensions. As we come to a full stop, clearly, the final velocity is zero, as is the final velocity of the reference trajectory. Matching this two second scenario takes $1.19\,s$. These computation times can be significantly improved by using a better implementation with a faster language like C++, by further utilizing parallelization of computations, and by changing to a faster computer.

We see that the occupancy sets become larger for small velocities at the end due to the previously discussed increase of uncertainty in the model at small velocities. To be less conservative, one would need to do a second conformance checking to obtain another model which better describes the dynamics and, especially, disturbances for very low velocities.

Scenario 2: Evasive Maneuver with Static Obstacle

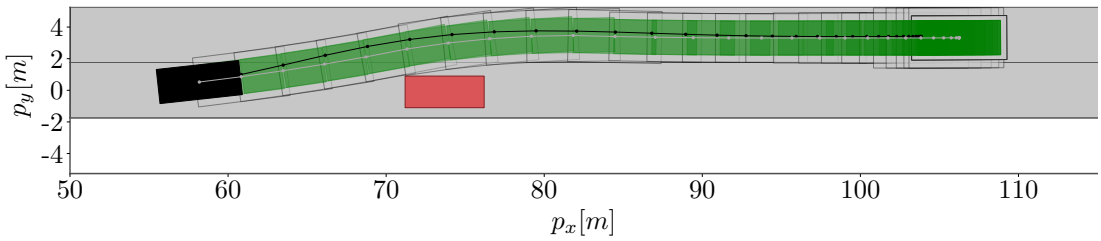


Figure 7.8: Results of our safe motion planning for the second scenario: The ego vehicle (black box) drives towards a static obstacle (red box) and the fail-safe planner plans for an emergency evading maneuver to the left lane. The resulting reference trajectory (black line) is matched by the motion primitives. Their centers are shown as the gray line, the occupancy of the car without uncertainty as green boxes, and the resulting occupancies increased by the reachable sets due to uncertainties as black rectangles.

In the second scenario, we again consider the case in which our ego vehicle drives with a static obstacle in a distance of $10.4\,m$ in front of it. This time, the ego vehicle has a higher initial velocity of $13.4\,\frac{m}{s}$, and the obstacle is closer. Therefore, it is not possible to simply brake, but the ego vehicle has to change the lane and then slow down. This is done with a reference trajectory of length $t_r = 6.2\,s$, which we use again as a guidance for finding a collision-free sequence of motion primitives. We show the results in Fig. 7.8.

We see that the motion primitives are able to avoid the obstacle and change the lane without any overshooting to the side. It does not follow the curvature of the reference trajectory exactly, though. The reason for this behavior is that the motion planner for the fail-safe reference trajectory does not take the limits of the input capacities in combination with uncertainties into account. Therefore, it is possible that the reference trajectory is not drivable when taking disturbances into account, as the occupancy sets would leave the left lane. To avoid this, the discrete planner chooses motion primitives which follow the direction of the reference trajectory

while ensuring safety even in case of disturbances. Therefore, by using the reference trajectory as a guidance for the search of our motion primitives, we obtain the best of both worlds: We do not have to exhaustively search the whole street, but only try to follow the reference trajectory. And on the other hand, when computing the reference trajectory, we only have to find an approximative solution without any safety guarantees under uncertainties, as we obtain them by the concatenated motion primitives.

At the end of the maneuver, there is an overshoot in the p_x direction of $2.4\,m$. Though, since there are no obstacles in the front, this does not matter. In the p_y direction, the distance to the reference trajectory is less than $0.1\,m$. Matching the reference trajectory over the whole $t_r = 6.2\,s$ horizon takes only $0.56\,s$.

Scenario 3: Braking Maneuver with Other Vehicle

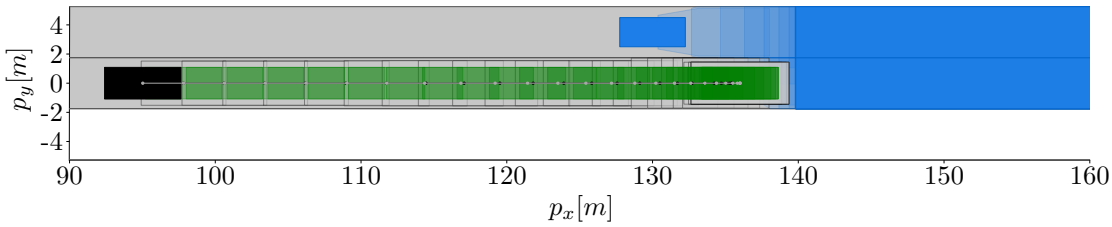


Figure 7.9: Results of our safe motion planning for the third scenario: The ego vehicle (black box) drives with another vehicle (blue box) in front of it. The predicted occupancy sets of all possible behaviors of the other vehicle are also shown in blue. The fail-safe planner plans for emergency braking and the resulting reference trajectory (black line) is matched by the motion primitives. Their centers are shown as the gray line, the occupancy of the car without uncertainty as green boxes, and the resulting occupancies increased by the reachable sets due to uncertainties as black rectangles.

In the third scenario, shown in Fig. 7.9, we consider another traffic participant as a moving obstacle. The other vehicle, illustrated by a blue box, is located $30.1\,m$ ahead in the lane to the left from the ego vehicle. The ego and the other vehicle have a similar initial velocity of $14.0\,\frac{m}{s}$ and $13.9\,\frac{m}{s}$, respectively. We use SPOT [5] to over-approximately predict the possible occupancies of the other vehicle at different time intervals (blue sets) and plan the reference trajectory for the worst case behavior: cut into the ego vehicle's lane and execute an emergency stop.

To avoid colliding with the other vehicle, the trajectory planner computes a braking maneuver, which is matched with motion primitives. The reference trajectory has a duration of $t_r = 5\,s$, and our algorithm needs $1.6\,s$ to match it. The matching algorithm is able to track the reference trajectory very closely with the motion primitives. Similar to the braking situation in Scenario 1, we only have a small final overshoot in p_x direction of $0.1\,m$ and no deviation in the other dimensions.

Scenario 4: Evasive Maneuver with Other Vehicle

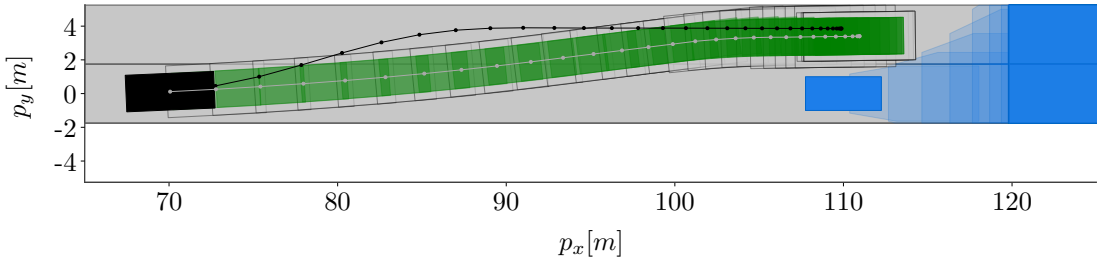


Figure 7.10: Results of our safe motion planning for the fourth scenario: The ego vehicle (black box) drives with another vehicle (blue box) in front of it. The predicted occupancy sets of all possible behaviors of the other vehicle are shown in blue as well. The fail-safe planner plans an emergency evading maneuver to the left lane including braking to a full stop. The resulting reference trajectory (black line) is matched by the motion primitives. Their centers are shown as the gray line, the occupancy of the car without uncertainty as green boxes, and the resulting occupancies increased by the reachable sets due to uncertainties as black rectangles.

In the last scenario, the ego vehicle and another vehicle are both driving in the right lane. The scenario is shown in Fig. 7.10. Both vehicles have again similar velocities of $13.8\,\frac{m}{s}$ for the ego vehicle and $13.9\,\frac{m}{s}$ for the other vehicle. The distance between both vehicles is $35.1\,m$. We again use SPOT to predict the possible occupancy set of the other vehicle and plan a fail-safe trajectory which drives the ego vehicle to the left lane, where it comes to a full stop, so as not to intersect with the possible occupancy set of the other vehicle. We match the reference trajectory with a duration $t_r = 6.2\,s$ in $9.5\,s$. As discussed before, this time can be further reduced by using a different implementation and/or parallel computations.

The concatenated motion primitives successfully change the lane without leaving the street and bring the ego vehicle to a safe stop behind the possible occupancy set of the vehicle in front. We see that our planning algorithm chooses the motion primitives such that they do not match the curve of the reference trajectory exactly. This is likely for similar reasons as in Scenario 2, as the reference trajectory is not exactly centered in the left lane. Without uncertainties, this does not matter; however, if we consider the possible effects of uncertainties, being too far left in the left lane might lead the car out of the street. By choosing the motion primitives more in the center of the left lane, our planning algorithm ensures that the ego vehicle stays on the street despite disturbances. In this particular situation, having the fail-safe trajectory going very close to the outer boundary of the street and changing curvatures in between, makes it especially hard to better match it using motion primitives with constant inputs.

Since it is not possible to safely track the fail-safe trajectory exactly, it is unavoidable that the center trajectories of the motion primitives have an offset of $0.45\,m$ to the reference trajectory in p_y direction at the end of the scenario. The additional offset of $1.06\,m$ in p_x direction likely results from the discretized motion primitives and was safely chosen, as there is enough space before the occupancy set of the other vehicle.

We conclude that our approach is able to find safe concatenations of motion primitives for all four scenarios. Since we consider uncertainties when computing the motion primitives, the results are much safer than only using the fail-safe planner from [194], which does not consider these. Further work is needed to improve the implementation of the matching algorithm to obtain faster computation times. In addition, the overall approach would benefit from an improved planner for the reference trajectories which includes the estimated size of the occupancy set of the motion primitives. This would result in closer matches which reduce backtracking and therefore significantly speed up the matching process.

The demonstrated feasibility of ensuring safety of fail-safe trajectories is a crucial base step for general safe motion planning. In a normal situation, a trajectory planner would plan for the most likely behavior. Therefore, if other vehicles behave as expected, there is no need to perform an emergency maneuver, and the fail-safe maneuver is not used. However, if there ever is an emergency situation, we can rely on having a safe solution. With a different vehicle model and new motion primitives, the approach can also be used for general trajectory planning, not just for emergency situations.

7.2 Robot Manipulator

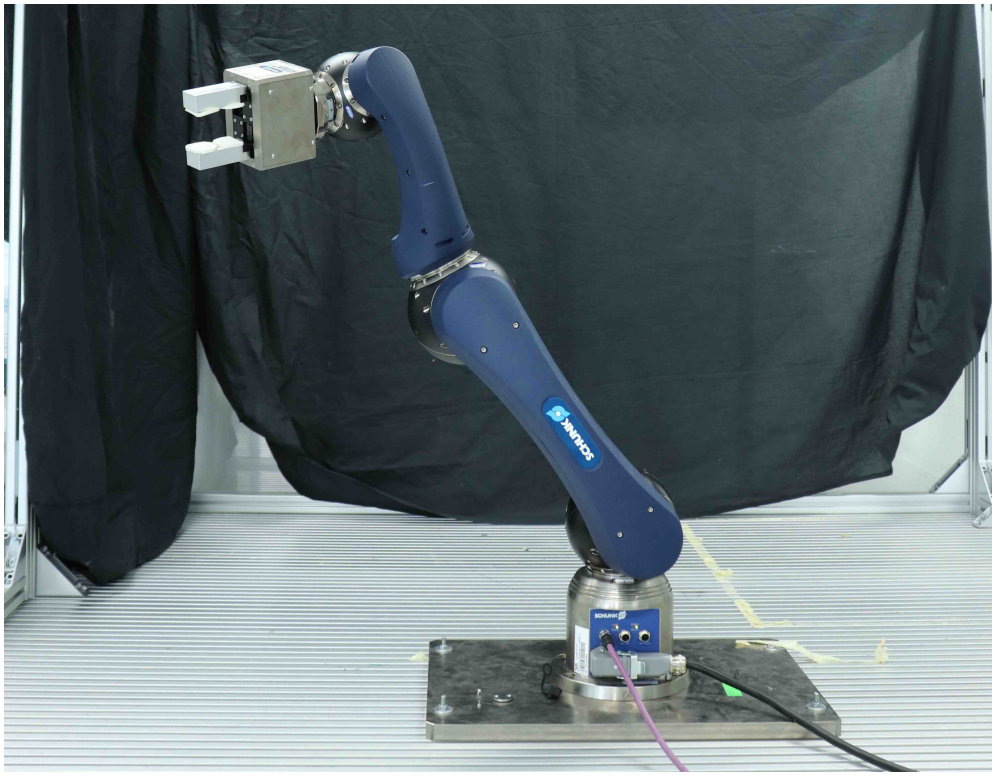


Figure 7.11: Schunk LWA 4P robot for which we design a safe controller.

In the second application[1], we test our algorithms on a robotic manipulator. For the experiments, we use the Schunk LWA 4P robot shown in Fig. 7.11, a lightweight robotic manipulator with six degrees of freedom. The robot is controlled using a centralized controller on a Speedgoat real-time target machine which uses a CANopen fieldbus system to send currents commands and receive positions measurements. This type of robot has high potential for flexible manufacturing in collaboration with human workers. To ensure safety at all times, we must guarantee that the robot stays in a safe set around the desired trajectory, such that it cannot collide with the human despite external disturbances, sensor noise, or model mismatch. Therefore, we aim at computing a safe tracking controller with a corresponding reachable set, which can be used for real-time planning.

The rigid-body dynamics of the robot manipulator has the form

$$\tau = M(q)\ddot{q} + c(q, \dot{q}) + g(q),$$

where $q, \dot{q}, \ddot{q} \in \mathbb{R}^6$ are the joint positions, velocities, and accelerations, respectively, and $\tau \in \mathbb{R}^6$ the motor torques at each joint. The matrix $M(q)$ contains the masses and inertias of the robot,

[1]The results of this section are based on a joint work [230] with Stefan Liu who also performed the tests on the real robot.

the vector $c(q, \dot{q})$ the Coriolis and centrifugal forces, and the vector $g(q)$ the gravitational forces. Despite this simple form, the actual dynamics are highly nonlinear and can become even more complicated if friction and similar disturbance effects are modeled. If we want to express the system as a differential equation with $\ddot{q}$ on one side, we have to invert the inertia matrix to obtain

$$\ddot{q} = M^{-1}(q)\left(\tau - c(q, \dot{q}) - g(q)\right). \tag{7.2}$$

Computing the matrix $M^{-1}(q)$ analytically becomes extremely challenging for higher dimensional systems, and it is no longer feasible for our robot with six degrees of freedom.

We could control the robot the same way as previously described for the car, by computing motion primitives offline and combining them in a maneuver automaton for online application. This would be more challenging than the previous case due to the very long expression of (7.2) and the fact that the dynamics are not independent of the initial position. On the other hand, there exists a number of control approaches for robotic manipulators which have proven to work well in practice but are not formally verified. An example is feedback linearization and dynamic decoupling control [195, Ch. 8.5], where an underlying controller counteracts the nonlinear behavior and aims at changing the closed-loop dynamics to be close to a linear system of the form

$$\dot{x}_i = x_{i+6}, \qquad \dot{x}_{i+6} = u_i,$$

where $x_i := q_i$ and $x_{i+6} := \dot{q}_i$ for all $i \in \{1, \dots, 6\}$. From the new input to our feedback-linearized system u, the actual input to the robot τ is then obtained by

$$\tau = M(q)u + c(q, \dot{q}) + g(q).$$

By additionally using disturbance observers, this approach works well in practice and linear feedback controllers can be used with this simplified dynamics to track a reference trajectory. However, since the actual disturbed nonlinear system can never be modeled accurately, the feedback linearization cannot achieve an exact linear behavior. Therefore, this approach alone is not able to provide formal safety guarantees. With this example, we demonstrate how to use our set-based control methods in combination with conformance checking to design a formally safe controller for the linearized dynamics, which also ensures safety of the actual nonlinear dynamics. This allows us to use the established practical approaches with our formal approaches to combine efficiency with safety.

To account for the imperfections from the feedback linearization, we model the resulting dynamics of the actual robot with the feedback linearization as a disturbed linear system with sensor noise of the form

$$\dot{x}_i = x_{i+6} + w_i, \qquad \dot{x}_{i+6} = u_i + w_{i+6},$$

with measurement function

$$h(x, \nu) = x + \nu,$$

where $\nu \in \mathbb{R}^{12}$. To further improve the accuracy and performance, we also use a linear high-gain observer from [196] to obtain the velocity states and include the delay caused by the discrete communication between the sensors, the central computer, and the actuators over a bus network to our model. Using reachset conformance checking, we want to find sets of disturbances and measurement noises such that the actual behavior of the feedback-linearized real system is contained inside the set of possible behavior of our uncertain linear model.

For the conformance checking, we drive a large number of different trajectories covering a wide range of scenarios with the real robot. These scenarios include polynomial and trapezoidal trajectories where the target positions, velocities, and accelerations are chosen randomly such that they cover large parts of the available workspace, of the velocities, and of the accelerations. In addition, we have a special focus for test cases on the most critical scenarios, where the model mismatch and errors are expected to be the highest. To model the friction in the feedback linearization, we used a model from [197] which considers Coulomb and linear viscous friction. This simplified model has the largest errors in situations where the velocities are close to the maximum possible values and where the velocities change their sign, see [198]. The generated trajectories are filtered to exclude any which would lead to self-collisions or exceed the maximum motor currents. The remaining trajectories are tracked with a manually tuned controller. By not including this controller in the conformance model, but only recording the system's in- and outputs, i.e., the out- and inputs of the auxiliary controller, we can perform the standard conformance checking from Sec. 2.8.2, which is independent of the controller used during the test cases. We record a total of over 33 minutes of tests. To maximize the number of tests, we consider each sampling instant as the start point of a new test case, which results in 497,880 test cases for each joint.

The resulting uncertain model from the conformance checking contains the real behavior of the robot in combination with the observers, communication delay, and feedback linearization. Next, we use our set-based controller synthesis to obtain a safe controller for it. Since the dynamics of each joint are independent of the other joints due to the decoupling from the feedback linearization, we can compute the controllers for each joint individually. Therefore, we only have to consider a two-dimensional system with a single input for each joint, which drastically simplifies the computational complexity. We consider again a linear tracking controller of the form $u_{ctrl}(t_k) = u_{ref}(t_k) + K(\hat{x}(t_k) - x_{ref}(t_k))$. As we run it on a digital controller, we directly consider the discrete control inputs corresponding to the controller cycle of $\Delta t = 4ms$. Since we consider linear dynamics, we only need to compute a single time-invariant feedback matrix K which is independent of the online-computed reference trajectories.

The inputs of the real system are limited by the maximum torques of the motors $\tau_{max} = [75.5, 75.5, 75.5, 75.5, 20, 20]^T Nm$. They result from the fact that the robot has four larger motors and two smaller motors for the last two joints as they have to move less weight from attached segments. The total torques requested from the motors result from the combination of the feedback linearizing controller, the reference inputs, and the feedback controllers of the joints. While the inputs of the feedback linearizing controllers depend on the actual position of the robot due to gravity effects for example, the linear system is independently of the absolute robot position. To be able to compute the feedback controllers independently of the

absolute position, we solve an optimization problem which aims for the largest possible input set $\mathcal{U}$ for the linear system for which the maximum torques are never exceeded for any robot configuration. This results in a maximum input set of $\mathcal{U}_2 = [-7.27, 7.27]\frac{rad}{s^2}$ for the second joint and $\mathcal{U}_i = [-20, 20]\frac{rad}{s^2},\ \forall i \in \{1, 3, 4, 5, 6\}$, for all other joints. The reason for the discrepancy here is the fact that the second joint has to mainly compensate the force of gravity acting on the robot. Since the robot is designed with identical motors for the first four joints, the remaining torque for the second joint is quite limited. We divide the available input capacities $\mathcal{U}$ into two parts, $\mathcal{U}_{ref}$ and $\mathcal{U}_{fb}$, with $\mathcal{U}_{ref} \oplus \mathcal{U}_{fb} \subseteq \mathcal{U}$. This allows us to compute the feedback matrix K independent of the reference input $u_{ref}(\cdot)$. We reserve input capacities of $\mathcal{U}_{ref,i} = [-3, 3]\frac{rad}{s^2},\ \forall i \in \{1, \dots, 6\}$ for the reference trajectory. Therefore, the remaining input capacities for the feedback controllers are $\mathcal{U}_{fb,2} = [-4.27, 4.27]\frac{rad}{s^2}$ for the second joint and $\mathcal{U}_{fb,i} = [-17, 17]\frac{rad}{s^2},\ \forall i \in \{1, 3, 4, 5, 6\}$, for the other joints.

We obtain the values of the feedback matrix K by optimizing over the reachable sets as described in Sec. 3.5. However, we not only have the feedback matrix as optimization parameters, we also optimize at the same time over the size of the set of disturbances $\mathcal{W}$ and sensor noise $\mathcal{V}$. In [199], a conformance checking approach for robotic manipulators is presented where the uncertainty sets for reachset conformance are obtained in an optimization problem. The authors present a way in which approximations can be used to simplify the optimization to a linear program. The idea behind this approach is the fact that the choice of uncertainty sets which achieve reachset conformance are not unique. By increasing a certain dimension of the disturbance, for example, one can reduce the size of the disturbance in a different dimension or the size of the sensor noise set, while still establishing conformance. The approach from [199] uses these degrees of freedom to minimize the size of the reachable set that is used to check reachset conformance.

In our case, we use the optimization problem from Sec. 3.5 and simply add the linear constraints from [199], which are used to ensure that conformance can be shown for the resulting uncertainty sets $\mathcal{W}$ and $\mathcal{V}$. By optimizing the feedback controller and uncertainty values all together in a single optimization problem, we obtain the combination which minimizes the volume of the reachable set of the closed-loop system and thereby leads to the best control performance.

In contrast to the optimization problem from Sec. 3.5 where we have a fixed final time, we are now interested in the smallest reachable set in which the tracking error remains for all times. We obtain it by starting from a small initial set $\mathcal{X}_0$ around the origin and then computing the reachable set until it converges, i.e., until time $t_\infty \in \mathbb{R}_0^+$ for which it holds that

$$\mathcal{R}_{t_\infty+\tau, K\hat{x}, \mathcal{W}}(\mathcal{X}_0) \subseteq \mathcal{R}_{t_\infty, K\hat{x}, \mathcal{W}}(\mathcal{X}_0),\ \forall \tau \in \mathbb{R}_0^+.$$

To deal with numerical effects which might prevent an exact convergence to happen, we use a convergence tolerance criterion similar to [200] to terminate the computation. The resulting

optimization problem is given by

$$\begin{aligned} &\min_{K,\mathcal{W},\mathcal{V}} \|\mathcal{R}_{[0,t_\infty],K\hat{x},\mathcal{W}}(\mathcal{X}_0)\|_1 \\ &\text{s.t. } \forall t_k \in \{0, t_\infty\} : \; Kh\left(\mathcal{R}_{t_k,K\hat{x},\mathcal{W}}(\mathcal{X}_0), \mathcal{V}\right) \subseteq \mathcal{U}_{fb}, \\ &\quad \forall i \in \{1, \dots, I\}, \; \forall t_k \in T_i, \; \forall \hat{x}(t_k) \in \hat{X}_i : \; \hat{x}(t_k) \in h\left(\mathcal{R}_{t_k,K\hat{x},\mathcal{W}}(\mathcal{X}_0), \mathcal{V}\right), \end{aligned} \tag{7.3}$$

where $\hat{X}_1, \dots, \hat{X}_I$ denote the previously measured test cases for the included conformance checking, see Sec. 2.8.2. In our optimization, we only value the size of the reachable set, i.e., the tracking error, and we do not have any state constraints, as obstacles are considered only during online planning. Similarly to the approaches in Ch. 3, we could easily add a cost term for the applied inputs and consider polyhedral state constraints, if desired.

After solving optimization problem (7.3) once for each joint, we obtain the feedback matrix

$$K = \text{diag}([371.73, 24.18, 348.64, 434.89, 371.73, 472.95, 38.56, 9.83, 37.34, 41.71, 38.56, 43.49])$$

and the corresponding reachable set $\mathcal{R}_{[0,t_\infty],K\hat{x},\mathcal{W}}(\mathcal{X}_0)$, which is shown in Fig. 7.12 in the relative coordinates $\Delta x_i(t) := x_i(t) - x_{ref}(t), \; \forall i \in \{1, \dots 12\}$. We see that we get rather tight bounds for joints 1 and 3 and a small increase in the bounds for joints 4 through 6. The reason for this increase lies already in the model from the conformance checking, where we obtained larger uncertainties for the later joints. This is most likely due to the fact that the feedback linearizing controller does not work as well for the last joints as for the ones closer to the base. A possible explanation might be that there are more oscillations closer to the end of the robot. The biggest exception is joint 2 (note the different scaling in the corresponding plot in Fig. 7.12), which is much larger than the others. This results from the limited input capacities which we have left after computing the linearizing controller, as it has to compensate the force of gravity, which mainly acts on the second joint. As a consequence, the tracking controller for the second joint cannot act as aggressively as the controllers for the other joints, as it has to compensate any disturbances with less available input, thus ending up with a larger reachable set. Many other robots solve this problem by using larger motors for those joints which have to compensate the force of gravity, which is not the case in the modular setup of the used Schunk robot.

Due to the linearized dynamics, we can superpose the reference input and the feedback controller. Therefore, we only have to compute the feedback controllers offline once, and during the online planning, we solely optimize the reference trajectory. We restrict its inputs to the previously chosen set $\mathcal{U}_{ref}$, and we require that $\left(x_{ref}(\cdot) \oplus \mathcal{R}_{[0,t_\infty],K\hat{x},\mathcal{W}}(\mathcal{X}_0)\right) \cap \mathcal{O} = \emptyset$, where $\mathcal{O}$ denotes the set of obstacles, which might be time-varying and non-convex. Therefore, this optimization problem can be solved with the same tools as for tube-based MPC.

To demonstrate the safety of our developed controller, we test it on the real robot and check whether the closed-loop controller actually satisfies the guaranteed constraints. The considered scenario has a duration of 25 seconds, and we repeat it ten times. We show the recorded trajectories in Cartesian space in a 3-D plot in Fig. 7.13. Similar to our scenarios from conformance checking, we choose a scenario which includes a number of different accelerations and velocities, goes through different parts of the state space, and also includes slow zero velocity crossings to capture friction effects [198].

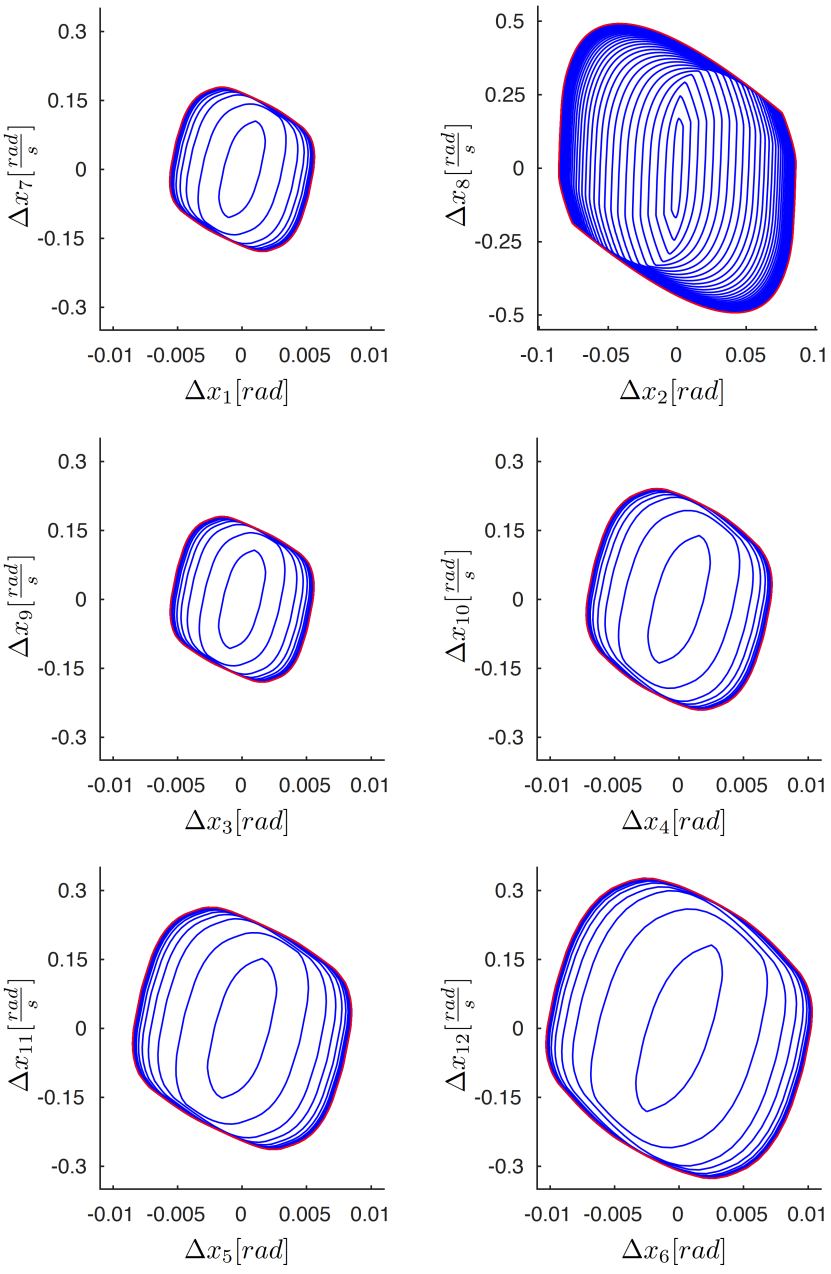


Figure 7.12: Illustration of the reachable sets of our feedback controller for the six joints. Reachable sets at different points in time are shown in blue; the final set which we use for the collision checking is shown in red. Please note the different axes scalings in the plot of the $(\Delta x_2, \Delta x_8)$ plane.

In Fig. 7.14 and Fig. 7.15, we show the recorded trajectories for the joint positions and joint velocities, respectively. In addition, we show the tube in which the safety controller guarantees

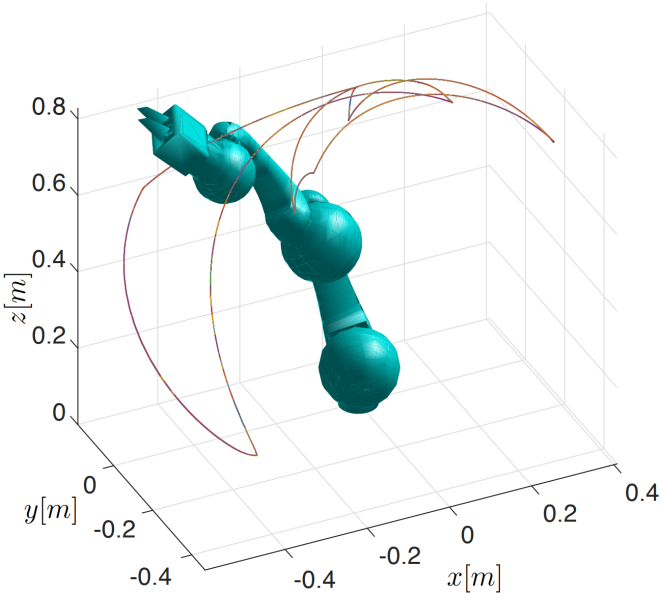


Figure 7.13: 3-dimensional visualization of the ten repetitions of the test trajectory. The lines show the movement of the robot flange for the end effector.

that the trajectories remain around the reference trajectory. We see that the trajectories in fact always remain in this precomputed reachable set as desired. Due to higher uncertainties when estimating the velocities, both the reachable sets as well as the uncertainties of the joint velocities are larger compared to the tighter bounds for the joint positions. We also see again the increased reachable set for the second joint, both in position and in velocity, where we have less input capacity for the feedback controller. Finally, in Fig. 7.16, we plot, for each joint, the deviations of the recorded trajectories from the reference trajectory projected on the position and velocity planes. We do this for each time point of all ten replications of our scenario which results in a total of around 62,500 measurements for each joint. For comparison, we also show the bounds of the reachable sets for each joint, i.e., the tube in which the tracking controller guarantees that the trajectories remain and see again that this guarantee holds in all cases.

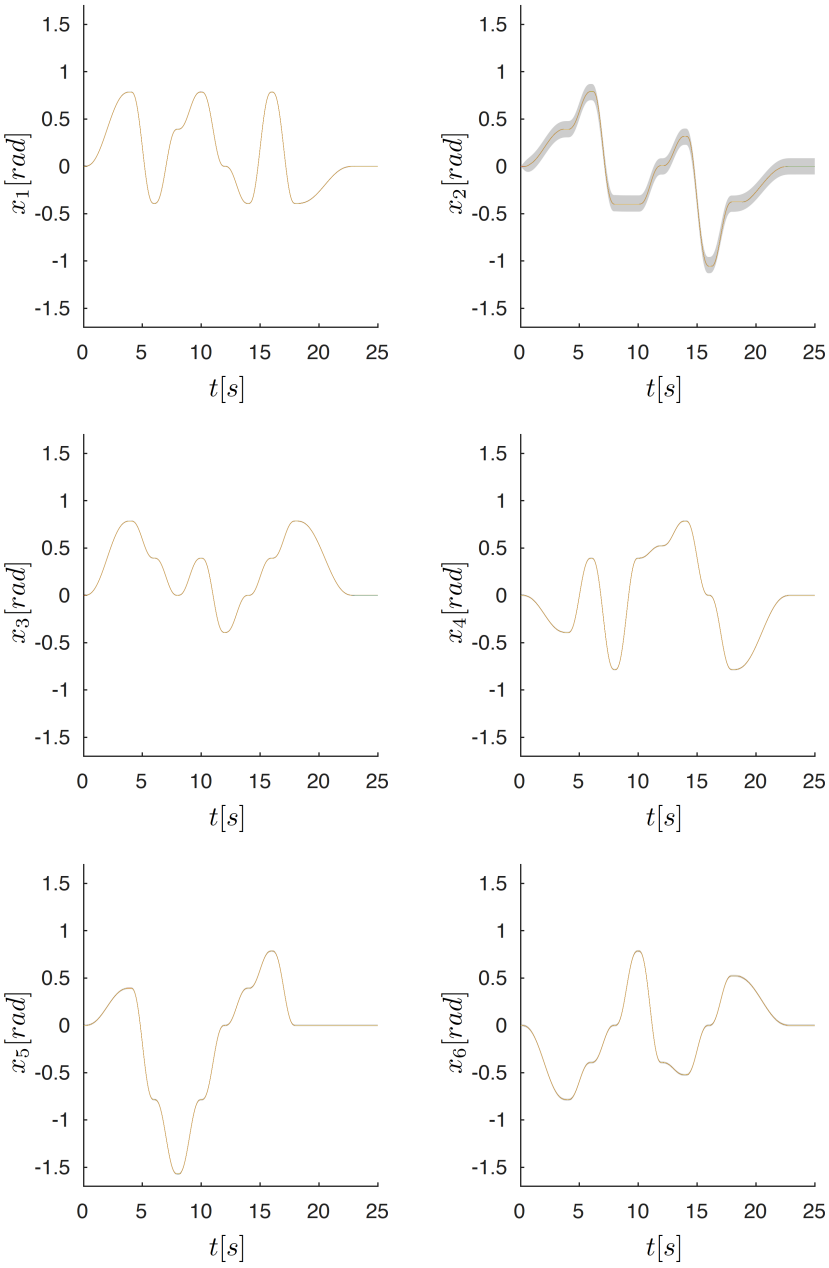


Figure 7.14: Measurements of the joint positions of all six joints for the ten repetitions of the test trajectory. In addition, we show in gray the tube corresponding to the precomputed reachable set.

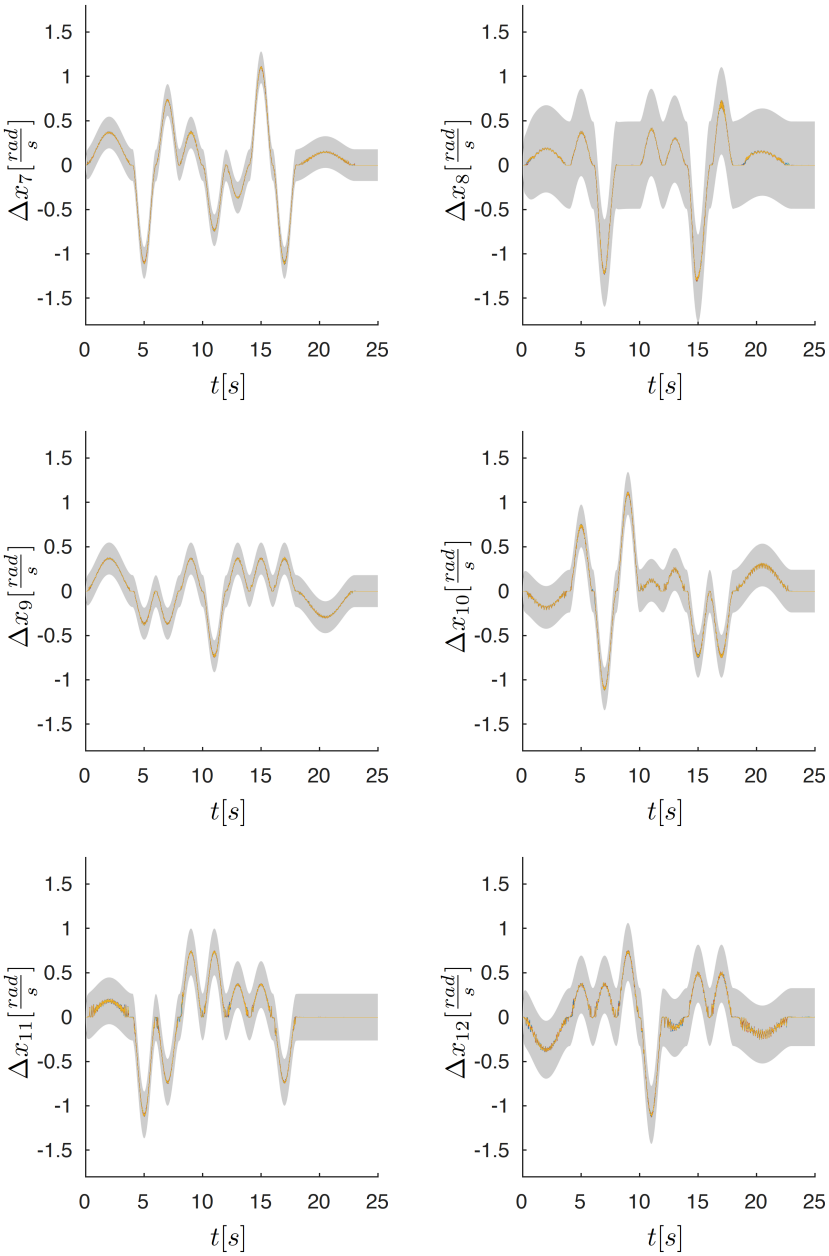


Figure 7.15: Measurements of the joint velocities of all six joints for the ten repetitions of the test trajectory. In addition, we show in gray the tube corresponding to the precomputed reachable set.

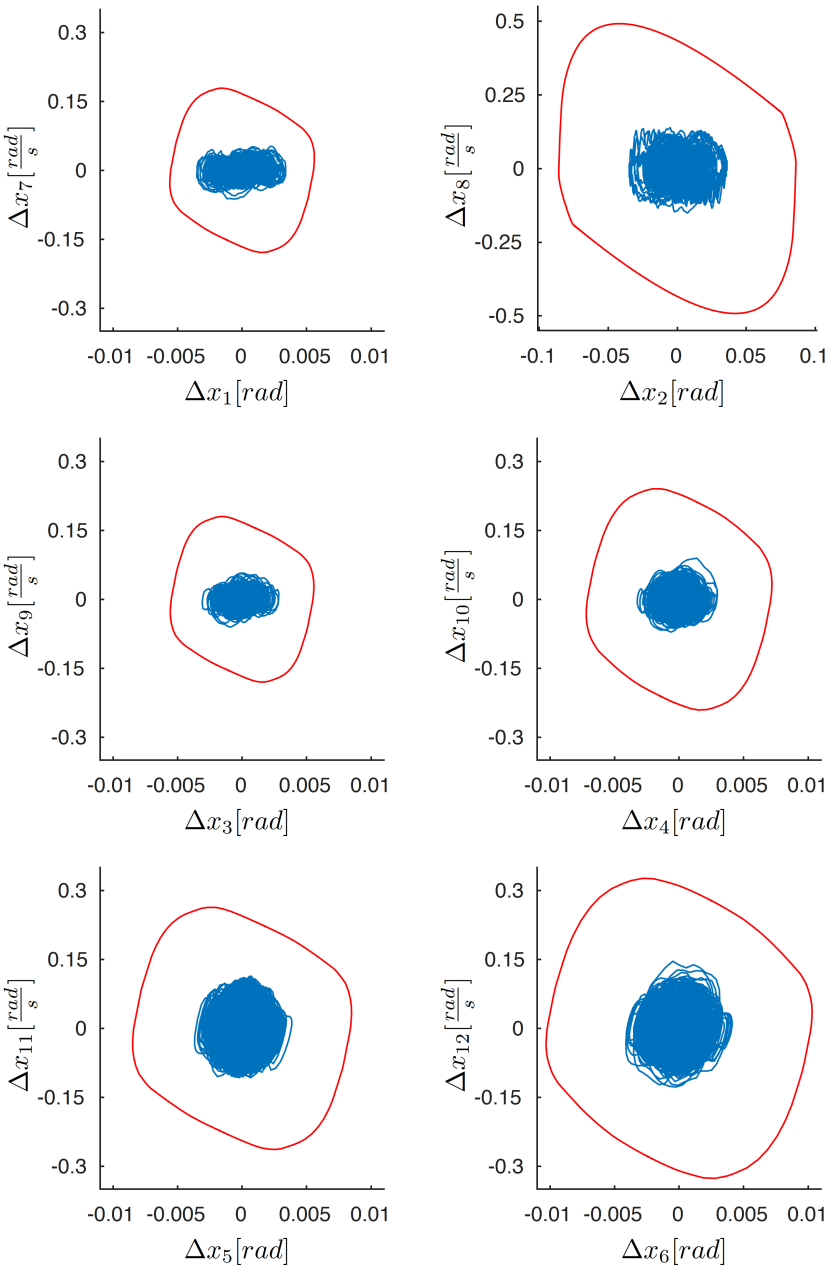


Figure 7.16: All measurements (blue) from the ten repetitions of the test trajectory of the closed loop system for all six joints. The reachable set (red) of our safety controller is shown for comparison. As desired, all measurements lie in the precomputed reachable set. Please note the different axes scalings in the plot of the $(\Delta x_2, \Delta x_8)$ plane.

7.3 Summary

In this chapter, we apply our set-based controllers to two practical systems: a self-driving car and a robotic manipulator suitable for human-robot interaction. For the car, we demonstrate how the combined control approach can be used to automatically compute a large maneuver automaton with more than 30,000 motion primitives for planning fail-safe maneuvers. We use a conformant model based on data from test-drives and thereby include real-world disturbances and measurement errors. We successfully test the resulting maneuver automaton for online planning of emergency trajectories in four different scenarios from recorded traffic data. We demonstrate that we can ensure safety and driveability of the emergency trajectories even for the worst-case behavior of other traffic participants.

In the second application example, we combine our continuous feedback controller with an existing feedback linearizing controller to control a robotic manipulator. The combination with the existing controller not only demonstrates how we can use our approaches together with established heuristics or domain specific controllers, it also allows us to decouple the robot's dynamics and apply the superposition principle. By using conformance checking for the closed-loop dynamics, we ensure that our uncertain model includes all possible errors due to the feedback linearization, in addition to normal disturbances and measurement errors.

We develop an optimization approach which includes reachset conformance as an additional constraint, thereby optimizing not only the control parameters, but also the uncertainty sets from the conformance checking, to obtain the smallest possible reachable sets. Due to the superposition principle of linear systems, the resulting controllers and reachable sets only have to be computed offline once and can be applied to track any online computed reference trajectory. Similar to tube-based MPC, we can use the offline-computed reachable sets to avoid collisions during online planning. We show in a test on the real robot that even for maneuvers which challenge the feedback linearization controller, our computed bounds hold and our controller satisfies all constraints. These two application examples demonstrate the practical applicability and usefulness of the developed methods.

Chapter 8

Conclusion and Future Directions

8.1 Conclusion

In this thesis, we presented various controller synthesis approaches with formal guarantees for the constraint satisfaction of complex, safety-critical systems. The developed methods combined elements from control theory, formal verification in the form of reachability analysis, and optimization. The key idea was the integration of the formal verification into the controller synthesis to obtain controllers with optimized performance and safety guarantees. For many future applications, such as autonomous driving, human-robot collaboration, or surgical robots, both parts are crucial for success.

In contrast to many existing formal approaches, we took advantage of established simplifications and heuristics such that the computational effort for our methods scaled well, while still being able to guarantee safety. A special focus was on the simple use and application of these methods. All developed methods can be implemented as "push-button" approaches that do not require a deep understanding of control theory to use. The resulting controllers all have simple structures which can be easily and efficiently implemented on actual systems.

These novel ideas of interweaving reachability algorithms with control theory have led to a new way of designing controllers with unique advantages over the state-of-the-art approaches. There exists no other control approach which is able to provide formal guarantees for constrained nonlinear systems affected by external disturbances and sensor noise while also being efficient to compute and easy to apply online. While there exist many controllers which are simple to use and to apply online such as LQR [37] and other classical control approaches, these approaches do not provide any kind of formal guarantees. Formal control approaches such as abstraction-based control [50–71] or approaches using sum-of-squares programming [34–36] are able to provide formal guarantees, although only few approaches explicitly consider disturbances and sensor noise. However, the main challenge for these kinds of approaches is that their computational effort scales poorly for higher-dimensional systems and, in the case of sum-of-squares programming, also for the considered polynomial degree. Similarly, model predictive control approaches have the problem of either having to solve complex optimizations online

[12, 22–30], which is not feasible in real time for higher-dimensional, complex systems, or having to store complex divisions of the state space with individual controllers [14–21].

For the offline algorithms, we took advantage of the fact that we could perform all optimization and verification steps before the online application and store the results as motion primitives in maneuver automata. These maneuver automata can be used for online planning and make the whole process of finding and verifying a safe trajectory very efficient.

We used different ideas for the offline synthesis of controllers for nonlinear systems with constraints and subject to disturbances and sensor noise: Since the basic problem was how to obtain inputs for a continuous set of possible states, we first presented two approaches which interpolate between finitely many solutions, either using extreme points or the generators of a zonotope. By repeatedly applying these steps together with reachable set computations, we achieved robust feedback. Another idea was the direct optimization of continuous feedback controllers. There, we were able to express the optimization as a nonlinear program, where we directly optimized over the resulting reachable sets and therefore included reachable set computations into the cost and constraint functions.

We then combined the ideas of the interpolation-based controllers and the continuous feedback controller to benefit from the advantages of both approaches while overcoming their weaknesses. The resulting combined approach uses an interpolation-based controller to compute individual reference trajectories for each of the infinitely many states of the initial set by solving a single linear program and tracks these individual reference trajectories using a feedback controller, which is obtained by optimizing over the reachable sets.

We extended the idea of the offline algorithms to hybrid dynamics through the example of piecewise affine systems. Here, the changing dynamics based on the region of the state space made the controller synthesis, as well as reachable set computation, hard. We tackled this problem by considering the distances to the boundaries of the region during the controller computation. This reduced the times where the reachable set would cover multiple regions with different dynamics.

A second extension was the use of backward reachable set computation. All previous approaches were computed forward and tried to minimize the final reachable set for a given initial set. In the backward case, we maximized the initial set for which a controller could ensure that all trajectories starting from this initial set ended in a given final set. Since formally computing backward reachable sets is very challenging, we circumvented this problem by obtaining an analytical correlation between initial and final sets of formal forwards reachable sets and used them to scale an approximation of the backward reachable set until it satisfied the constraints for the actual dynamics.

In addition to using the offline controllers on their own, we also showed that they could be used as safety nets for unverified optimal controllers. This combination allowed us to ensure safety for optimal controllers which might not be able to be formally verified otherwise, but which might have advantages by being optimized for improved performance, comfort, or energy savings. Typically, the optimal controller is applied while predicting the immediate future behavior using reachable sets. Only if this becomes unsafe does our formal controller step in, thereby always ensuring safety while benefiting from the advantages of the optimal

controller. The controllers based on backward reachable sets are particularly suitable as safety net controllers, as their maximized reachable sets lead to an increased flexibility for the optimal controller.

After presenting offline-computed controllers, we considered online controllers as well. We developed a novel robust model predictive control algorithm which uses reachability analysis to ensure safety despite disturbances and sensor noise. Including the reachable set computation into this MPC framework allowed more flexibility than fixed tubes from tube-based MPC. In addition, we were able to explicitly consider the computation time which our algorithm needed and ensured that nothing unsafe happened due to these delays, an effect which is usually neglected.

In addition to a number of different numerical examples, where our new controllers outperformed existing approaches, we also demonstrated their applicability on real-world systems. The first application example was autonomous driving, where we used real driving data to obtain a model using conformance checking. Based on this model, we used the combined control approach to compute a maneuver automaton with over 30,000 motion primitives covering a wide variety of velocities and turn radii offline. We showed the applicability of the safe online planner for a number of simulations based on real traffic data. By combining the discrete planning with a reference trajectory, we were able to obtain fast results while guaranteeing the driveability and collision-freeness of the obtained trajectory.

In a second example, we applied the developed set-based control approaches to a robotic manipulator. Here, we used an existing feedback linearizing controller in combination with conformance checking to obtain a reliable model for the feedback linearized system. Using the continuous feedback approach, we computed a controller for each of the robot's joints which minimized the reachable set while satisfying the input constraints despite disturbances and sensor noise. Due to the feedback linearizing controller, we could directly apply this new controller and ensure collision-freeness, even during online planning, without having to precompute motion primitives. The tests on the real robot demonstrated that the controlled system actually stays in the precomputed safety bounds.

These are just two of many possible examples. The approaches developed for this thesis are applicable to a wide variety of practically relevant systems. We have not only proven their benefits in theory but also demonstrated their advantages both in simulation and in practice. By combining control, formal verification, and optimization, we have laid the basis for a new field of control theory, one which is easy to apply, scales well, and provides formal guarantees for safety critical systems.

8.2 Future Directions

This thesis already covers a wide variety of system classes and application scenarios for the combination of control, formal verification, and optimization. Still, there are a number of directions where the research can be continued. New system classes are one such direction. For example, one can use the presented ideas which we used to extend our approaches to piecewise affine systems to further generalize the approach to other hybrid dynamics. For hybrid systems

with nonlinear continuous dynamics, one can combine the ideas from Ch. 4 with those from Ch. 3.

A second, straightforward extension is on the algorithmic side. Since we heavily rely on algorithms for reachable set computation and numerical optimization, our approaches directly benefit from any improvements in these areas. While our approaches are not restricted to specific reachability or optimization algorithms, it could still be beneficial to adapt some of our approaches depending on new algorithms. An example could be the use of different set representations and therefore possibly new forms of the resulting control laws.

Another research direction is the online application of set-based control approaches. This goes hand in hand with improvements in reachability and optimization algorithms. So far, we demonstrated a model predictive controller based on reachable sets for slower systems. Using better implementations, faster programming languages, and faster hardware, the computation times could be significantly improved and be made applicable for faster systems. However, the actual online optimization of the feedback control law (not only optimizing the reference trajectory) for high-dimensional, complex, nonlinear systems for fast applications is not yet feasible. With future improvements in reachable set computations, this opens an interesting direction for further research in the future.

A large field with vast potential is the application to real systems. We demonstrated the applicability for two example systems, but the application to a wide variety of systems has not yet been done. Together with domain knowledge in particular, there is a lot of potential to combine existing controllers and heuristics with our formal methods, as done with the feedback linearization for the robot manipulator. Using established controllers with our safety-net approach, for example, is a straightforward way in which to benefit from the combination of controllers tuned by year-long domain experience with the formal safety guarantees of our approaches. Alternatively, one can also use proven controller structures and improve their control parameters by optimizing over the reachable sets of the closed-loop system, as done for the continuous feedback controller.

Another direction which builds up on our developed algorithms is the controller synthesis for systems with complex formal specifications, for example, in temporal logic. Many existing approaches use tools from automata theory to obtain control laws with formal guarantees for satisfying these specifications. However, these tools often require discrete abstractions of the underlying dynamics, which are hard to obtain, and often struggle dealing with disturbances. Therefore, our safe maneuver automata offer a good way to abstract the underlying dynamics, and the corresponding controllers take care of possible disturbances and sensor noise. The tools from automata theory then only need to consider the motion primitives as abstractions without having to take the actual dynamics into account. We successfully demonstrated this for formally verified traffic planning in [221] and thereby illustrated the potential of this research direction.

Similar to the last two points, further research would be useful in the area of planning with motion primitives. While the precomputation of motion primitives drastically reduces the online effort, we still need algorithms and heuristics for real-time planning and checking collisions with moving obstacles. We demonstrated for the autonomous vehicle example in Sec. 7.1 that we

can successfully plan in real traffic scenarios, laying the foundation for further research.

Safe planning using motion primitives in combination with other tools also opens interesting opportunities in new fields. For example in [220], we showed how planning with our motion primitives can be used in combination with probabilistic models of human drivers to enable interactive motion planning which takes human behavior into account. Such new concepts are only possible with the availability of safe motion primitives which allow fast online planning with safety guarantees. Many more of these kinds of combinations and research fields are waiting to be explored in the future.

Our approaches offer a number of new possibilities for the safe control of complex systems. This thesis provides the basis for the combination of controller synthesis, reachability analysis, and optimization. The next steps are to build on this basis by improving the approaches, applying them to different domains, and advancing algorithms which use the newly developed controllers and motion primitives.

Appendix A

Closed-Form Expressions of Convex Combinations

Convex combinations are used in many different applications in which all points in a set must be expressed as combinations of extreme points. In Sec. 3.3, we present a control approach which uses convex combinations to interpolate the input trajectories of the extreme states of a set in order to obtain inputs for any interior state of this set. But convex combinations are also useful for other control approaches without formal guarantees, for example robust controllers of linear parameter-varying systems where the uncertain parameters vary in a convex set [201]. Rather than computing a control matrix for each possible parameter, the authors compute a control matrix for each extreme point of the parameter set. Online, they interpolate these control matrices based on the actual, measured parameter at that time point.

Similar to our convex interpolation controller, the approach from [201] therefore requires us to determine the $\lambda_i(x)$ parameters describing a state x inside a polytope $\mathcal{P}$ as a convex combination of the q extreme points $\breve{x}^{(i)}$ of $\mathcal{P}$ in real time. This means one has to solve the following system of inequalities at every sampling time:

$$\sum_{i=1}^{q} \lambda_i(x)\breve{x}^{(i)} = x, \tag{A.1}$$

$$\lambda_i(x) \geq 0, \ \forall i \in \{1, \ldots, q\}, \tag{A.2}$$

$$\sum_{i=1}^{q} \lambda_i(x) = 1. \tag{A.3}$$

These are $n+1$ equalities and q inequalities for q unknowns. In particular, for high-dimensional polytopes with many extreme points, this leads to a large system of inequalities which are hard to solve. Moreover, with the exception of simplices, this system of inequalities might have non-unique solutions.

In this appendix[1], we present methods to compute closed-form expressions of convex combinations for different types of convex sets. In the literature, closed-form expressions are

[1]This appendix, including the figures, is based on [224] © IEEE 2016.

mainly considered for special cases only, for example for two-dimensional, axis-aligned boxes in [201–205] or axis-aligned boxes of arbitrary dimension in [206, 207]. However, none of these papers provide proofs of the formulas. All of these sources, even [206, 207], who provide a formula for n-dimensional, axis-aligned boxes, discuss their theory for general polytopes but are only able to provide formulas for special forms of boxes. Other papers, for example [208], circumvent the problem of obtaining closed-form expressions by simply assuming that the convex combinations can be obtained in real time.

More general approaches, though mostly limited to two-dimensional sets, can be found in [209] and the references therein. One of the few papers which considers general convex sets in higher dimensions is [210]. However, even the proposed algorithm is numerically unstable in dimensions higher than two. Therefore, we present algorithms which work even in high dimensions and for general polytopes. Our algorithms reduce online computation times and therefore enable the use of the previously discussed control approaches for fast dynamical systems. In addition, our algorithms allow the use of formal verification tools, such as reachability analysis, which require closed-form expressions of the closed-loop dynamics.

We refer to an expression which can be written as an explicit mathematical formula, i.e., without any loops or iterations, as analytical closed-form expressions. For general closed-form expressions, we allow that there are finitely many case distinctions, but each case must only consist of an analytical closed-form expression. While it is not possible to provide an analytical closed-form expression for every type of set, we discuss how such an expression is possible for certain classes of sets, more precisely simplices and parallelotopes. Moreover, we show how non-analytical closed-form expressions can be obtained even for arbitrary polytopes with finitely many extreme points. The latter case involves solving point location problems, which makes it non-analytical.

A.1 Simplices

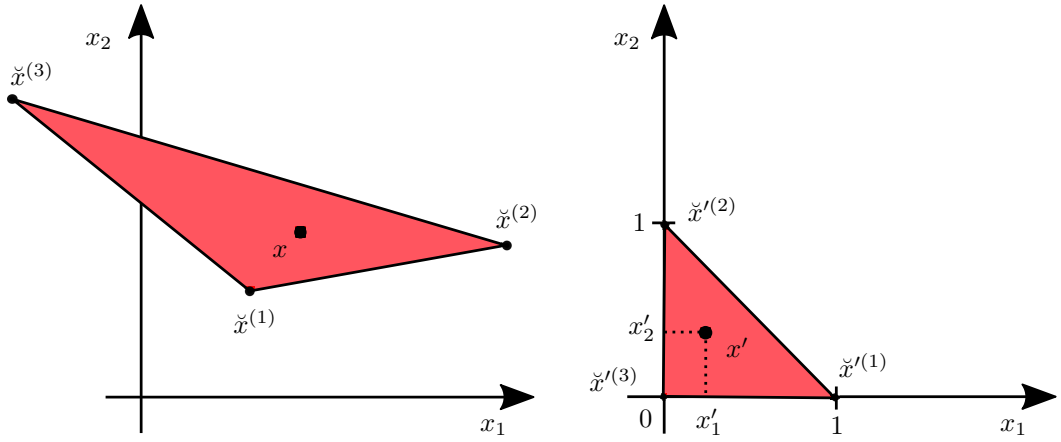
Simplices are polytopes with the least number of extreme points which still span a full-dimensional subset in $\mathbb{R}^n$. The convex combination of the extreme points for a given point inside a simplex is unique. We show that any simplex can be transformed to the unit simplex, i.e., the simplex with one extreme point equal to the origin and the others equal to the unit vectors $e^{(i)}$. This is achieved using an affine transformation, i.e., a linear transformation together with a translation. Neither operation changes any convexity properties [73].

Let us consider a simplex $\mathcal{S} \subset \mathbb{R}^n$ described by its $n+1$ extreme points $\breve{x}^{(1)}, \ldots, \breve{x}^{(n+1)}$, for which we define the matrix

$$G = \left[\breve{x}^{(1)} - \breve{x}^{(n+1)}, \ldots, \breve{x}^{(n)} - \breve{x}^{(n+1)} \right].$$

We use this matrix to transform $\mathcal{S}$ to the unit simplex and we denote the affine transformation of a point $x \in \mathcal{S}$ by x' :

$$x' = G^{-1} \left(x - \breve{x}^{(n+1)} \right).$$



(a) Original simplex **(b)** Simplex after transformation

Figure A.1: Any simplex (a) can be transformed to the unit simplex (b), for which the closed-form decomposition problem can easily be solved.

The new coordinates x' are known as barycentric coordinates [211]. By subtracting $\breve{x}^{(n+1)}$, we shift the simplex such that the transformed extreme point $\breve{x}'^{(n+1)}$ lies in the origin. By multiplying $\left(x - \breve{x}^{(n+1)}\right)$ with G^{-1}, we transform the other extreme points to the unit vectors, i.e., $\breve{x}'^{(i)} = e^{(i)}$, $\forall i \in \{1, \ldots, n\}$. Note that since $\breve{x}^{(i)}$ are all extreme points, the vectors $\breve{x}^{(i)} - \breve{x}^{(n+1)}$, $\forall i \in \{1, \ldots, n\}$, are all linearly independent, and therefore G^{-1} exists.

For this special simplex, we show how a unique, analytical closed-form expression can be obtained. The advantage of this result is that the convex combination obtained in this way can be used for the original, non-unit simplex. This is stated in the following theorem and is illustrated in Fig. A.1:

Theorem 6. *We consider an arbitrary simplex $\mathcal{S} \subset \mathbb{R}^n$, described by its $n+1$ extreme points $\breve{x}^{(1)}, \ldots, \breve{x}^{(n+1)}$. Given a point $x \in \mathcal{S}$, this point can be expressed as a convex combination of the extreme points as*

$$x = \sum_{i=1}^{n+1} \lambda_i(x) \breve{x}^{(i)}, \tag{A.4}$$

where the parameters $\lambda_i(x)$ are given by the following closed-form expression:

$$\lambda_i(x) = \left(G^{-1}\left(x - \breve{x}^{(n+1)}\right)\right)_i, \ \forall i \in \{1, \ldots, n\}, \tag{A.5}$$

$$\lambda_{n+1}(x) = 1 - \sum_{i=1}^{n} \lambda_i(x). \tag{A.6}$$

Proof. We have to show that the two statements of the theorem hold: (A.4) is a convex combination, and (A.4) with $\lambda_i(x)$ resulting from (A.5)–(A.6) actually describes the point x. We

begin by showing that $\lambda_i(x')$ sum up to 1, then that they are nonnegative, and finally that using these $\lambda_i(x')$ defined for x' also describe x in $\mathcal{S}$.

Since the transformed extreme points $\breve{x}'^{(i)}$ are the unit vectors $e^{(i)}$, any transformed x', with $x \in \mathcal{S}$, can be uniquely written as

$$x' = \sum_{i=1}^{n} x'_i e^{(i)} = \sum_{i=1}^{n} x'_i \breve{x}'^{(i)},$$

see also Fig. A.1. Adding the last transformed extreme point $\breve{x}'^{(n+1)} = \mathbf{0}$ does not change the result, since it is the origin. From the definition of the simplex as the convex hull of the extreme points, it follows that for any $x \in \mathcal{S}$, x' is inside the unit simplex and therefore

$$\sum_{i=1}^{n} \lambda_i(x') = \sum_{i=1}^{n} x'_i \leq 1.$$

Since $\sum_{i=1}^{n} \lambda_i(x') \leq 1$ is not necessarily a convex combination, as the sum might be smaller than one, we define $\lambda_{n+1}(x') = 1 - \sum_{i=1}^{n} \lambda_i(x')$. We use this weight to add $\breve{x}'^{(n+1)} = \mathbf{0}$, which does not change the result of the original decomposition, i.e.,

$$x' = \sum_{i=1}^{n+1} \lambda_i(x') \breve{x}'^{(i)} = \sum_{i=1}^{n} \lambda_i(x') \breve{x}'^{(i)} + \lambda_{n+1}(x')\mathbf{0} = \sum_{i=1}^{n} \lambda_i(x') \breve{x}'^{(i)}.$$

However, this definition of $\lambda_{n+1}(x')$ ensures that $\sum_{i=1}^{n+1} \lambda_i(x') = 1$.

Because $x \in \mathcal{S}$ and $\mathcal{S}$ is transformed to the unit simplex, all points are mapped to a nonnegative set. Therefore, it holds that $\lambda_i(x') = x'_i \geq 0, \ \forall i \in \{1, \ldots, n\}$. From

$$\sum_{i=1}^{n} \lambda_i(x') = \sum_{i=1}^{n} x'_i \leq 1,$$

it follows that

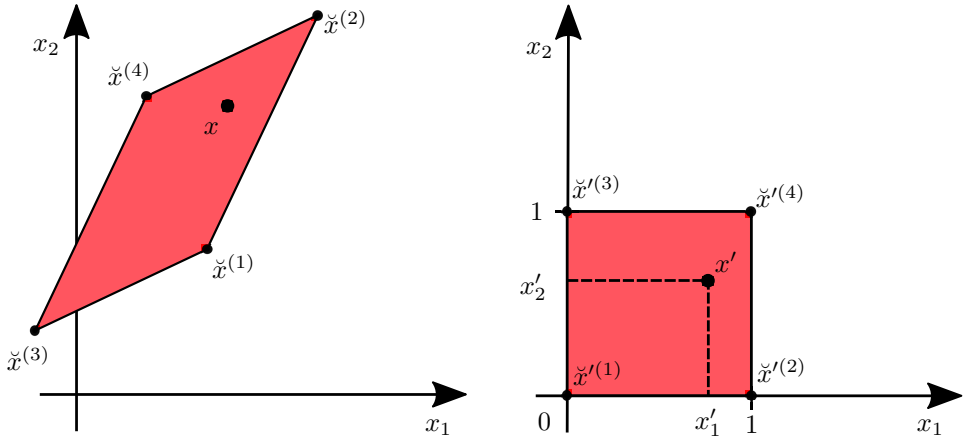
$$\lambda_{n+1}(x') = 1 - \sum_{i=1}^{n} \lambda_i(x') \geq 0.$$

Therefore, the $\lambda_i(x')$ are a convex combination and define any point $x' \in \mathcal{S}'$ in terms of the transformed extreme points $\breve{x}'^{(i)}$.

In the last step, we show that we can use the parameters of the transformed states $\lambda_i(x')$ for the original state x as well, by transforming the point x' back into the original coordinates. Since $x' = G^{-1}\left(x - \breve{x}^{(n+1)}\right)$ by definition, the following holds:

$$\begin{aligned} x = Gx' + \breve{x}^{(n+1)} &= G\left(\sum_{i=1}^{n+1} \lambda_i(x') \breve{x}'^{(i)}\right) + \underbrace{\sum_{i=1}^{n+1} \lambda_i(x')}_{=1} \breve{x}^{(n+1)} \\ &= \sum_{i=1}^{n+1} \lambda_i(x') \left(G\breve{x}'^{(i)} + \breve{x}^{(n+1)}\right) = \sum_{i=1}^{n+1} \lambda_i(x') \breve{x}^{(i)}, \end{aligned} \tag{A.7}$$

which is a direct result from the fact that the convexity properties are not changed by affine transformations. Therefore, x can be written as a convex combination of the extreme points of $\mathcal{S}$, and it has the same parameters $\lambda_i(x')$ as the convex combination of x'. □



(a) Original parallelotope **(b)** Parallelotope after transformation

Figure A.2: Any parallelotope (a) can be transformed to the unit hypercube (b), for which the closed-form decomposition problem can easily be solved.

A.2 Parallelotopes

In the previous section, our solution involved transforming simplices to unit simplices, for which the analytical closed-form expression is easy to obtain. In this section, we present a similar approach for parallelotopes. We first transform a general parallelotope $\mathcal{P}$ to the unit hypercube $[0,1] \times [0,1] \times \dots [0,1]$ and then present a way to obtain a closed-form expression for this special class of parallelotopes. The advantage is that, in the case of boxes whose edges are parallel to the axes, we can consider each dimension independently. Although there are infinitely many different combinations for choosing $\lambda_i(x)$, except for points on the boundary, we reduce the degrees of freedom such that we obtain a unique solution. We do this by defining for any point $x \in \mathcal{P} = \langle c_\mathcal{P}, G_\mathcal{P} \rangle$:

$$x' = \frac{1}{2} G_\mathcal{P}^{-1} \left(x - c_\mathcal{P} \right) + \frac{1}{2} \mathbf{1}. \tag{A.8}$$

This affine transformation maps the parallelotope $\mathcal{P}$ to

$$\begin{aligned}
\mathcal{P}' &= \frac{1}{2} G_\mathcal{P}^{-1} \left(\mathcal{P} - c_\mathcal{P} \right) + \frac{1}{2} \mathbf{1} \\
&= \{ x' \in \mathbb{R}^n \mid x' = \frac{1}{2} G_\mathcal{P}^{-1} \left(c_\mathcal{P} - c_\mathcal{P} \right) + \frac{1}{2} G_\mathcal{P}^{-1} G_\mathcal{P} \alpha + \frac{1}{2} \mathbf{1},\ \alpha_i \in [-1,1] \} \\
&= \{ x' \in \mathbb{R}^n \mid x' = \frac{1}{2} \mathbf{1} + \frac{1}{2} I \alpha,\ \alpha_i \in [-1,1] \},
\end{aligned}$$

which is the unit hypercube. Note that $G_\mathcal{P}^{-1}$ always exists since $G_\mathcal{P}$ is quadratic and has full rank by definition of a parallelotope. The transformation for an example in $\mathbb{R}^2$ is visualized in Fig. A.2. In the following, we state Thm. 2 again and prove it.

Theorem 2. *We consider a parallelotope $\mathcal{P} \subset \mathbb{R}^n$ given by*

$$\mathcal{P} = \{x \in \mathbb{R}^n \mid x = c_\mathcal{P} + G_\mathcal{P}\alpha,\ \alpha_i \in [-1, 1]\},$$

with $c_\mathcal{P} \in \mathbb{R}^n$ and $G_\mathcal{P} \in \mathbb{R}^{n\times n}$, and which has 2^n extreme points $\breve{x}^{(1)}, \dots, \breve{x}^{(2^n)}$. Given a point $x \in \mathcal{P}$, this point can be expressed as a convex combination of the extreme points as

$$x = \sum_{i=1}^{2^n} \lambda_i(x)\breve{x}^{(i)}, \tag{A.9}$$

where the parameters $\lambda_i(x),\ i \in \{1, \dots, 2^n\}$, are given by the following closed-form expression

$$\lambda_i(x) = \prod_{j=1}^{n} \mu_{i,j},$$

where

$$\mu_{i,j} = \begin{cases} x'_j, & \text{if } \alpha_j(\breve{x}^{(i)}) = 1 \\ 1 - x'_j, & \text{if } \alpha_j(\breve{x}^{(i)}) = -1. \end{cases} \tag{A.10}$$

Here, x' is the transformed point of x under the affine transformation (A.8).

Note that for a point $\breve{x}^{(i)} \in \mathcal{P}$ to be an extreme point, the entries in the corresponding parameter vector $\alpha(\breve{x}^{(i)})$ must all be ± 1; therefore, one of the cases in (A.10) is always satisfied.

Proof. We first show that all $\lambda_i(x)$ are nonnegative, then we show that they sum up to 1, and finally that this convex combination (A.9) actually results in the point x.

Since $\mathcal{P}'$ is the unit hypercube, it follows that for any $x' \in \mathcal{P}' : x'_i \in [0, 1],\ \forall i \in \{1, \dots, n\}$. Because $\lambda_i(x)$ is a product of the entries x'_j or $(1 - x'_j)$, i.e., a product of n numbers between 0 and 1, its value must be between 0 and 1, too, which proves the first part.

Let us now show the second part by induction over the number of states n. We state that

$$\sum_{i=1}^{2^n} \lambda_i = \sum_{i=1}^{2^n} \prod_{j=1}^{n} \mu_{i,j} = 1. \tag{A.11}$$

Before we begin, note that it follows from the fact that the $\alpha(\breve{x}^{(i)})$ consist of all 2^n possible combinations of ± 1 entries, i.e.,

$$\alpha(\breve{x}^{(1)}) = \begin{bmatrix} -1 \\ -1 \\ \vdots \\ -1 \end{bmatrix}, \alpha(\breve{x}^{(2)}) = \begin{bmatrix} 1 \\ -1 \\ \vdots \\ -1 \end{bmatrix}, \dots, \alpha(\breve{x}^{(2^n)}) = \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix},$$

that we can find for any given $\lambda_i(x)$ and an arbitrary $l \in \{1, \dots, n\}$ another $\lambda_k(x)$ for which $\mu_{i,j} = \mu_{k,j}, \forall j \neq l$ and $\mu_{i,l} = 1 - \mu_{k,l}$. For an easier notation, we order the extreme points such that

$$\mu_{1,n} = \mu_{2,n} = \dots = \mu_{2^{n-1},n}$$

and

$$\mu_{2^{n-1}+1,n} = \mu_{2^{n-1}+2,n} = \cdots = \mu_{2^n,n} = 1 - \mu_{1,n}.$$

Let us now begin by showing that (A.11) holds for $n = 1$:

$$\sum_{i=1}^{2^1} \prod_{j=1}^{1} \mu_{i,j} = \mu_{1,1} + \mu_{2,1} = \mu_{1,1} + (1 - \mu_{1,1}) = 1,$$

which follows from (A.10) and from the fact that we have only two extreme points in the one-dimensional case.

The induction step is done by showing that we can reduce (A.11) from $n = N+1$ to $n = N$, for any $N \geq 1$:

$$\sum_{i=1}^{2^{N+1}} \prod_{j=1}^{N+1} \mu_{i,j} = \sum_{i=1}^{2^{N+1}} \mu_{i,N+1} \prod_{j=1}^{N} \mu_{i,j} = \mu_{1,N+1} \sum_{i=1}^{2^N} \prod_{j=1}^{N} \mu_{i,j} + \underbrace{\mu_{2^N+1,N+1}}_{=(1-\mu_{1,N+1})} \sum_{i=1}^{2^N} \prod_{j=1}^{N} \mu_{i,j} \tag{A.12}$$

$$= \sum_{i=1}^{2^N} \prod_{j=1}^{N} \mu_{i,j}. \tag{A.13}$$

Since we can order the $\lambda_i(x)$ such that the same is possible for the $\mu_{i,j}$ for $n = N$, we can reduce this sum down to $n = 1$, for which we know that $\sum_{i=1}^{2^1} \prod_{j=1}^{1} \mu_{i,j} = 1$. This shows that $\sum_{i=1}^{n} \lambda_i(x) = 1$.

The only remaining part to show is that (A.9)–(A.10) actually describes x correctly. We can do so by using the previous results from the induction proof. First note that because of the transformation (A.8), the extreme points of $\mathcal{P}'$ are the corner points of the unit hypercube, i.e.,

$$\breve{x}'^{(1)} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \breve{x}'^{(2)} = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \ldots, \breve{x}'^{(2^n)} = \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix},$$

and that $\breve{x}_j'^{(i)} = 1$ if and only if $\alpha_j(\breve{x}^{(i)}) = 1, \forall i \in \{1, \ldots, 2^n\}, \forall j \in \{1, \ldots, n\}$, see also Fig. A.2. From this structure it follows that $\lambda_i(x)$ contains the factor x'_j if and only if $\breve{x}_j'^{(i)} = 1$. Let us now consider the k-th entry of x' if computed by (A.9)–(A.10). For an easier notation and without loss of generality, we assume that we ordered the extreme points such that $\breve{x}_k'^{(1)} = \cdots = \breve{x}_k'^{(2^{n-1})} = 1$ and $\breve{x}_k'^{(2^{n-1}+1)} = \cdots = \breve{x}_k'^{(2^n)} = 0$. Then the following holds:

$$\sum_{i=1}^{2^n} \lambda_i(x) \breve{x}_k'^{(i)} = \sum_{i=1}^{2^n} \prod_{j=1}^{n} \mu_{i,j} \breve{x}_k'^{(i)} = \sum_{i=1}^{2^n} \mu_{i,k} \prod_{j=1}^{k-1} \mu_{i,j} \prod_{j=k+1}^{n} \mu_{i,j} \breve{x}_k'^{(i)} \tag{A.14}$$

$$\overset{(A.10)}{=} x'_k \underbrace{\sum_{i=1}^{2^{n-1}} \prod_{j=1}^{k-1} \mu_{i,j} \prod_{j=k+1}^{n} \mu_{i,j}}_{=1 \text{ (see explanation below)}} 1 + (1 - x'_k) \sum_{i=2^{n-1}+1}^{2^n} \prod_{j=1}^{k-1} \mu_{i,j} \prod_{j=k+1}^{n} \mu_{i,j} 0 \tag{A.15}$$

$$= x'_k 1 = x'_k, \tag{A.16}$$

where the sum over the remaining $n-1$ factors in (A.15) is equal to 1 because of the same arguments as in the induction proof (A.12)–(A.13). Since this holds for every $k \in \{1,\dots,n\}$, this convex combination actually results in x'.

The affine transformation (A.8) does not change any convexity properties [73]. Therefore, analogous to (A.7), the state x can be represented by the convex combination of the extreme points $\breve{x}^{(i)}$ by using the parameters for the transformed state $\lambda_i(x')$, which concludes the proof. □

For the special case of axis-aligned boxes, the presented approach can be simplified to the closed-form expression which was stated without a proof in [206] and [207].

Note that while we describe the approaches for full-dimensional simplices and parallelotopes only, they can be adapted for lower-dimensional simplices and parallelotopes. One simply has to project the points on a lower-dimensional subspace in which they become full-dimensional simplices or parallelotopes. Then our presented techniques can be applied again.

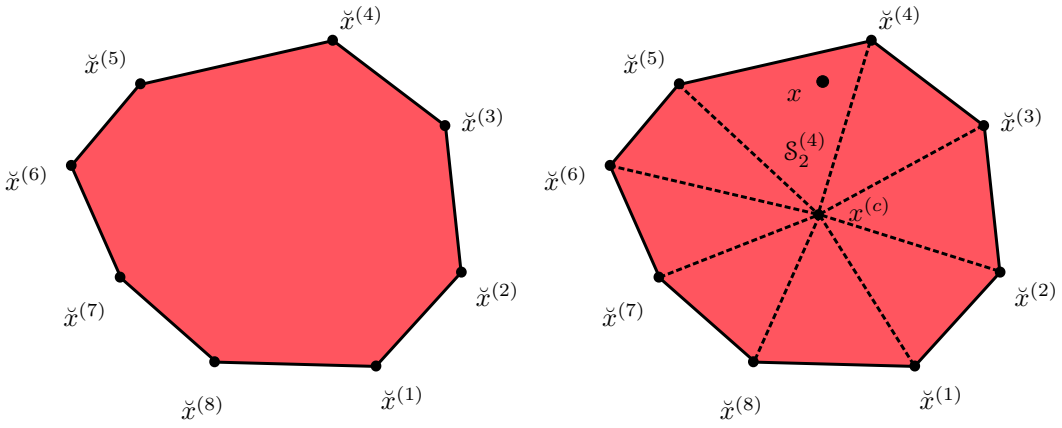
A.3 General Polytopes

In the previous two sections, we discussed how to find analytical closed-form expressions for simplices and parallelotopes. In the case of simplices, the resulting convex combination is unique. This is not necessarily the case with parallelotopes and general polytopes. For parallelotopes, we presented an approach which always results in the same convex combination. This is possible since we are able to uniquely transform simplices and parallelotopes to axis-aligned simplices and axis-aligned boxes. Since the corresponding mapping matrices are square, they are invertible. For other shapes, like general polytopes, it is not possible to find such a map.

While it is therefore not possible to provide an analytical closed-form expression for general polytopes, we are nevertheless always able to obtain a closed-form expression which can be solved without using any iterations. Thus, this approach can be used online in real time without the disadvantages of using linear programming approaches.

The basic idea for the general polytope is the fact that any polytope with finitely many extreme points can be expressed by the union of finitely many non-overlapping simplices [212]. An analytical closed-form expression can be found for each of these simplices. This process is summarized in Algorithm 8 and visualized in Fig. A.3.

The boundary of the polytope $\mathcal{P} \subset \mathbb{R}^n$, denoted by $\mathcal{B}(\mathcal{P})$, defines an $n-1$ dimensional subspace of $\mathbb{R}^n$. We divide this space into s simplices $\mathcal{S}_{n-1}^{(i)}, i \in \{1,\dots,s\}$, each of them defined by n extreme points of $\mathcal{P}$ such that $\bigcup_{i=1}^{s} \mathcal{S}_{n-1}^{(i)} = \mathcal{B}(\mathcal{P})$, and $\forall i \neq j : \mathcal{I}(\mathcal{S}_{n-1}^{(i)}) \cap \mathcal{I}(\mathcal{S}_{n-1}^{(j)}) = \emptyset$ (Alg. 8, line 1). We use this division of the boundary of $\mathcal{P}$ to divide $\mathcal{P}$ itself into s simplices. We use the fact that if we choose any point $\tilde{x} \in \mathcal{I}(\mathcal{P})$ in the interior of $\mathcal{P}$ and compute the convex hull of this $\tilde{x}$ and the extreme points of a simplex $\mathcal{S}_{n-1}^{(i)}$ on the boundary of $\mathcal{P}$, we obtain an n-dimensional simplex $\mathcal{S}_n^{(i)} \subset \mathcal{P}$. By doing this for every simplex on the boundary, we obtain s simplices $\mathcal{S}_n^{(1)},\dots,\mathcal{S}_n^{(s)}$ such that $\bigcup_{i=1}^{s} \mathcal{S}_n^{(i)} = \mathcal{P}$ and $\forall i \neq j : \mathcal{I}(\mathcal{S}_n^{(i)}) \cap \mathcal{I}(\mathcal{S}_n^{(j)}) = \emptyset$ (Alg. 8, line 3). For each of the simplices $\mathcal{S}_n^{(i)}$, $i \in \{1,\dots,n\}$, we use Thm. 6 to obtain the closed-form expression for the convex combination of all points inside it and store them for



(a) Original polytope **(b)** Point inside divided polytope

Figure A.3: (a) Given is a general polytope $\mathcal{P} \subset \mathbb{R}^2$ defined as the convex hull of its $q = 8$ extreme points $\breve{x}^{(i)}$. (b) The boundary $\mathcal{B}(\mathcal{P})$ is divided into $s = 8$ simplices $\mathcal{S}_1^{(i)}$ of dimension $n-1$ (here: line segments). Together with a center point $x^{(c)}$, the boundary simplices define s simplices $\mathcal{S}_2^{(i)}$ of dimension n (here: triangles). Any point x inside the polytope can be expressed by the closed-form convex combination of the simplex in which it is contained.

Algorithm 8 Convex Decomposition Algorithm for General Polytopes – Offline Part

Input: polytope $\mathcal{P}$ with q extreme points $\breve{x}^{(1)}, \ldots, \breve{x}^{(q)}$

Output: simplices $\mathcal{S}_n^1, \ldots, \mathcal{S}_n^s$ with closed form convex expressions

1: Connect each extreme point with its neighboring extreme points along the edges of the polytope $\rightarrow$ results in s simplices $\mathcal{S}_{n-1}^{(1)}, \ldots, \mathcal{S}_{n-1}^{(s)}$ in $\mathbb{R}^{n-1}$
2: Define $x^{(c)} \leftarrow \sum_{i=1}^{q} \frac{1}{q} \breve{x}^{(i)}$
3: Connect $x^{(c)}$ with all extreme points $\breve{x}^{(i)} \rightarrow$ results in s simplices $\mathcal{S}_n^{(1)}, \ldots, \mathcal{S}_n^{(s)}$ in $\mathbb{R}^n$
4: **for** $i = 1, \ldots, s$ **do**
5: Use Theorem 6 to obtain the closed-form for the convex combination of all points inside simplex $\mathcal{S}_n^{(i)}$ and store them
6: **end for**

Algorithm 9 Convex Decomposition Algorithm for General Polytopes – Online Part

Input: state $x \in \mathcal{P}$, simplices $\mathcal{S}_n^1, \ldots, \mathcal{S}_n^s$ with closed form convex expressions

Output: parameters $\lambda_1(x), \ldots, \lambda_q(x)$

1: Find i, such that $x \in \mathcal{S}_n^{(i)}$
2: Use the closed-form convex expression of simplex $\mathcal{S}_n^{(i)}$ to describe x as the convex combination of its extreme points

later online use (Alg. 8, line 5). Note that this works with any point in the interior of $\mathcal{P}$. We choose $\tilde{x} = x^{(c)} = \sum_{i=1}^{q} \frac{1}{q} \breve{x}^{(i)}$ (Alg. 8, line 2) because we then immediately know that $\lambda_i(x^{(c)}) = \frac{1}{q}$, $\forall i \in \{1, \ldots, q\}$, are the corresponding parameters of the convex combination for $x^{(c)}$. Since $x^{(c)}$ is in many cases around the center, the simplices will have similar sizes.

When this approach is used to obtain the convex combination for a given point $x \in \mathcal{P}$ online as described in Alg. 9, we check which simplex contains x (Alg. 9, line 1). We then simply have to use the offline-computed closed-form expression for the convex combinations in this simplex in the same way as in Section A.1 (Alg. 9, line 2). Checking which simplex contains x is a point location problem for which efficient solutions exist, e.g., [213]. As we show in the next section, it is even feasible to check all simplices.

A.4 Numerical Example

In this section, we compare the previously presented closed-form approaches to obtain convex combinations with existing approaches which solve (A.1)–(A.3) using linear programming. All computations are performed using MATLAB on the same computer as in Sec. 3.3.7 and without using parallel computations. We performed all precomputations which do not depend on the actual point inside the polytope offline in advance.

Simplices and Parallelotopes

We randomly generate 100 simplices and 100 parallelotopes each in 2, 3, 5, 8, 10, 12, and 15 dimensions. We also generate 100 simplices in 50, 100, 200, and 1,000 dimensions. Then we randomly generate points inside these simplices and parallelotopes and solve the convex decomposition problem first with our approach and then by solving a linear program using the `linprog` function with the default settings, the cost function set to zero, and the solution space bounded to $[0, 1]$. The results are presented in Table A.1.

We see that our approach is significantly faster in any of the presented dimensions for both simplices and parallelotopes. For simplices we are faster with factors between 1,400 and up to almost 20,000. The computation times for parallelotopes are between 160 and 1,900 times faster than using linear programming. We can compute the simplices for much higher dimensions than parallelotopes, since the number of extreme points increases linearly with the dimensions for simplices, while the number of extreme points for parallelotopes increases exponentially. Therefore, the computation times almost do not increase for small dimensions for simplices. For high dimensions, the closed-form expressions for parallelotopes become quite large, thus requiring a significant time to load the functions in MATLAB before the online computations can begin.

General Polytopes

In this subsection, we compare our closed-form approach to the linear programming approach for general polytopes. During the implementation of our closed-form algorithm, we must decide how to check which simplex the point is contained in. There are many ways which can reduce

Table A.1: Average run times for simplices and parallelotopes (in ms)

Dimension	Simplices		Parallelotopes	
	Cl.-Form	Lin. Progr.	Cl.-Form	Lin. Progr.
2	0.0052	7.3922	0.0049	8.3247
3	0.0053	7.8886	0.0052	9.7929
5	0.0053	8.4302	0.0071	10.5306
8	0.0055	8.5203	0.0438	16.9912
10	0.0057	8.5928	0.1243	30.3809
12	0.0058	8.6812	0.4810	77.1852
15	0.0058	8.9057	3.5609	666.9693
50	0.0077	11.4315	–	–
100	0.0123	20.5933	–	–
200	0.0254	75.5463	–	–
1,000	0.5471	10,902	–	–

the number of simplices checked, for example the techniques in [213], or by computing the distance of the point to the center points of the s simplices $\mathcal{S}_{\mathbb{R}^n}^{(i)}$ and starting the computation with the nearest simplex. We choose the simplest strategy, which is to check all simplices. Since it is also the worst-case scenario, it provides a lower bound for the performance of our approach. Checking all simplices is feasible, since for each simplex only a few operations have to be performed. We compute polytopes in 3, 4, and 5 dimensions with 20, 50, 100, and 150 vertices. For each of these combinations we randomly compute 10 polytopes, and for each polytope we randomly choose 10 points for which we solve the convex decomposition problem. The results are summarized in Table A.2. We see that our approach is much faster in all of these cases, even though we used the easiest method of checking all simplices. The results can be further improved by either using better heuristics for choosing the order in which the simplices are checked or by using multi-core processors, as these tests can be performed in parallel without overhead since each computation is independent.

Note that we consider only the online computation times, since these are the critical times in control. This is, of course, only feasible if the polytopes are known in advance. For simplices and parallelotopes, our approach is still significantly faster than the linear programming approach, even if everything has to be computed online (i.e., mainly matrix inverses).

A.5 Summary

Convex combinations are used for many applications in control, such as for our convex interpolation controller or for robust gain-scheduling control approaches for polytopic linear parameter-

Table A.2: Average run times for general polytopes (in ms)

# Extreme Points	Closed-Form Approach	Linear Programming
Dimension $n = 3$		
20	0.1585	9.5866
50	0.3234	10.1925
100	0.5317	11.9193
150	0.8613	13.2450
Dimension $n = 4$		
20	0.2888	9.9935
50	0.6673	10.1101
100	1.3337	11.2544
150	2.1542	12.7434
Dimension $n = 5$		
20	0.5205	10.1305
50	1.6159	10.2715
100	4.0592	11.2688
150	5.4805	12.0558

varying systems. In many cases it is necessary to have closed-form expressions of convex combinations, for example for formal verification of safety or for faster online computation times for fast dynamical systems. We provide analytical closed-form expressions for simplices and polytopes and prove their validity. Furthermore, we present a technique for obtaining closed-form expressions for general polytopes. We compare the computation times of our approaches to linear programming algorithms and show that our approaches provide significantly faster computation times, therefore making them suitable for the control of fast dynamical systems.

References

[1] Matthias Althoff, Stanley Bak, Marcelo Forets, Goran Frehse, Niklas Kochdumper, Rajarshi Ray, Christian Schilling, and Stefan Schupp. **ARCH-COMP19 Category Report: Continuous and Hybrid Systems with Linear Continuous Dynamics**. In *Proc. of the International Workshop on Applied Verification of Continuous and Hybrid Systems*, pages 14–40, 2019. 3, 41, 92, 118

[2] Fabian Immler, Matthias Althoff, Luis Benet, Alexandre Chapoutot, Xin Chen, Marcelo Forets, Luca Geretti, Niklas Kochdumper, David P. Sanders, and Christian Schilling. **ARCH-COMP19 Category Report: Continuous and Hybrid Systems with Nonlinear Dynamics**. In *Proc. of the International Workshop on Applied Verification of Continuous and Hybrid Systems*, pages 41–61, 2019. 3, 22, 41, 92, 118

[3] Hans Georg Bock and Karl-Josef Plitt. **A multiple shooting algorithm for direct solution of optimal control problems**. *IFAC Proceedings Volumes*, **17**(2):1603–1608, 1984. 3, 36, 137

[4] D. Hess, M. Althoff, and T. Sattel. **Formal Verification of Maneuver Automata for Parameterized Motion Primitives**. In *Proc. of the International Conference on Intelligent Robots and Systems*, pages 1474–1481, 2014. 4

[5] Markus Koschi and Matthias Althoff. **SPOT: A tool for set-based prediction of traffic participants**. In *Proc. of the IEEE Intelligent Vehicles Symposium*, pages 1686–1693, 2017. 4, 173

[6] Emilio Frazzoli. *Robust hybrid control for autonomous vehicle motion planning*. PhD thesis, Massachusetts Institute of Technology, 2001. 5

[7] S. Magdici and M. Althoff. **Fail-Safe Motion Planning of Autonomous Vehicles**. In *Proc. of the International Conference on Intelligent Transportation Systems*, pages 452–458, 2016. 5

[8] Dimitri P Bertsekas. *Dynamic Programming and Optimal Control*. Athena Scientific Belmont, MA, 3rd edition, 2005. 8, 87

[9] Franco Blanchini and Stefano Miani. *Set-Theoretic Methods in Control*. Springer, 2008. 8

[10] John Lygeros, Claire Tomlin, and Shankar Sastry. **Controllers for Reachability Specifications for Hybrid Systems**. *Automatica*, **35**(3):349–370, 1999. 8, 92

[11] A. B. Kurzhanski, I. M. Mitchell, and P. Varaiya. **Optimization Techniques for State-Constrained Control and Obstacle Problems**. *Journal of Optimization Theory and Applications*, **128**(3):499–521, 2006. 8

[12] James B Rawlings and David Q. Mayne. *Model Predictive Control: Theory and Design*. Nob Hill, 2009. 8, 43, 90, 117, 145, 188

[13] Alberto Bemporad and Manfred Morari. **Robust model predictive control: A survey**. In A. Garulli and A. Tesi, editors, *Robustness in identification and control*, pages 207–226. Springer, 1999. 8, 145

[14] Francesco Borrelli. *Constrained Optimal Control of Linear and Hybrid Systems*. Springer, 2003. 8, 90, 145, 188

[15] Francesco Borrelli, Alberto Bemporad, and Manfred Morari. *Predictive Control for linear and hybrid systems*. Cambridge University Press, 2015. 8, 90, 145, 188

[16] Munoz de la Pena, Alberto Bemporad, and Carlo Filippi. **Robust explicit MPC based on approximate multi-parametric convex programming**. In *Proc. of the Conference on Decision and Control*, pages 2491–2496, 2004. 8, 90, 145, 188

[17] Alessandro Alessio and Alberto Bemporad. **A Survey on Explicit Model Predictive Control**. In Lalo Magni, Davide Martino Raimondo, and Frank Allgöwer, editors, *Nonlinear Model Predictive Control: Towards New Challenging Applications*, pages 345–369. Springer, 2009. 8, 90, 92, 115, 145, 188

[18] Eric C Kerrigan and Jan M Maciejowski. **Feedback min-max model predictive control using a single linear program: robust stability and the explicit solution**. *International Journal of Robust and Nonlinear Control*, **14**:395–413, 2004. 8, 90, 145, 188

[19] Alexandra Grancharova, Tor A. Johansen, and Petter Tøndel. **Computational Aspects of Approximate Explicit Nonlinear Model Predictive Control**. In Rolf Findeisen, Frank Allgöwer, and Lorenz T. Biegler, editors, *Assessment and Future Directions of Nonlinear Model Predictive Control*, pages 181–192. Springer, 2007. 8, 90, 145, 188

[20] E. N. Pistikopoulos. **Perspectives in multiparametric programming and explicit model predictive control**. *AIChE Journal*, **55**(8):1918–1925, 2009. 8, 90, 145, 188

[21] D.M. Raimondo, S. Riverso, C.N. Jones, and M. Morari. **A robust explicit nonlinear MPC controller with input-to-state stability guarantees**. *IFAC Proceedings Volumes*, **44**(1):9284 – 9289, 2011. 8, 90, 145, 188

[22] D. Q. Mayne, M. M. Seron, and S. V. Raković. **Robust model predictive control of constrained linear systems with bounded disturbances**. *Automatica*, **41**(2):219 – 224, 2005. 8, 72, 90, 93, 145, 159, 188

[23] Saša V Raković, Basil Kouvaritakis, Mark Cannon, Christos Panos, and Rolf Findeisen. **Parameterized tube model predictive control**. *IEEE Transactions on Automatic Control*, **57**(11):2746–2761, 2012. 8, 31, 90, 145, 188

[24] Saša V Raković, Basil Kouvaritakis, Rolf Findeisen, and Mark Cannon. **Homothetic Tube Model Predictive Control**. *Automatica*, **48**(8):1631–1638, 2012. 8, 90, 145, 188

[25] W. Langson, I. Chryssochoos, S.V. Raković, and D.Q. Mayne. **Robust model predictive control using tubes**. *Automatica*, **40**(1):125–133, 2004. 8, 90, 145, 159, 188

[26] Matteo Rubagotti, Davide Martino Raimondo, Antonella Ferrara, and Lalo Magni. **Robust Model Predictive Control With Integral Sliding Mode in Continuous-Time Sampled-Data Nonlinear Systems**. *IEEE Transactions on Automatic Control*, **56**(3):556–570, 2011. 8, 90, 145, 188

[27] L. Magni, G. De Nicolao, R. Scattolini, and F. Allgöwer. **Robust model predictive control for nonlinear discrete-time systems**. *International Journal of Robust and Nonlinear Control*, **13**(3-4):229–246, 2003. 8, 90, 145, 188

[28] D. Q. Mayne, E. C. Kerrigan, E. J. van Wyk, and P. Falugi. **Tube-based robust nonlinear model predictive control**. *International Journal of Robust and Nonlinear Control*, **21**(11):1341–1353, 2011. 8, 90, 145, 159, 188

[29] Sumeet Singh, Anirudha Majumdar, Jean-Jacques Slotine, and Marco Pavone. **Robust online motion planning via contraction theory and convex optimization**. In *Proc. of the IEEE International Conference on Robotics and Automation*, pages 5883–5890, 2017. 8, 90, 145, 188

[30] D. Limon, T. Alamo, J. M. Bravo, E. F. Camacho, D. R. Ramirez, D. Muñoz de la Peña, I. Alvarado, and M. R. Arahal. **Interval Arithmetic in Robust Nonlinear MPC**. In Rolf Findeisen, Frank Allgöwer, and Lorenz T. Biegler, editors, *Assessment and Future Directions of Nonlinear Model Predictive Control*, pages 317–326. Springer, 2007. 8, 90, 145, 159, 188

[31] J.M. Bravo, T. Alamo, and E.F. Camacho. **Robust MPC of constrained discrete-time nonlinear systems based on approximated reachable sets**. *Automatica*, **42**(10):1745–1751, 2006. 8, 145, 150, 151, 152, 154, 155, 156, 157, 159

[32] A. Agung Julius and Andrew K Winn. **Safety Controller Synthesis using Human Generated Trajectories: Nonlinear Dynamics with Feedback Linearization and Differential Flatness**. In *Proc. of the American Control Conference*, pages 709–714, 2012. 8

[33] Andrew K Winn and A Agung Julius. **Feedback control law generation for safety controller synthesis**. In *Proc. of the Conference on Decision and Control*, pages 3912–3917, 2013. 9

[34] Russ Tedrake, Ian R Manchester, Mark Tobenkin, and John W Roberts. **LQR-trees: Feedback Motion Planning via Sums-Of-Squares Verification**. *The International Journal of Robotics Research*, **29**(8):1038–1052, 2010. 9, 90, 187

[35] Anirudha Majumdar and Russ Tedrake. **Funnel Libraries for Real-Time Robust Feedback Motion Planning**. *The International Journal of Robotics Research*, **36**(8):947–982, 2017. 9, 90, 187

[36] Philipp Reist and Russ Tedrake. **Simulation-based LQR-trees with input and state constraints**. In *Proc. of the International Conference on Robotics and Automation*, pages 5504–5510, 2010. 9, 90, 187

[37] Huibert Kwakernaak and Raphael Sivan. *Linear optimal control systems*. Wiley-interscience New York, 1972. 9, 46, 134, 187

[38] Anirudha Majumdar and Russ Tedrake. **Robust Online Motion Planning with Regions of Finite Time Invariance**. In *Algorithmic Foundations of Robotics X*, **86** of *Springer Tracts in Advanced Robotics*, pages 543–558. Springer, 2013. 9

[39] Avishai Weiss, Claus Danielson, Karl Berntorp, Ilya Kolmanovsky, and Stefano Di Cairano. **Motion planning with invariant set trees**. In *Proc. of the Conference on Control Technology and Applications*, pages 1625–1630, 2017. 9

[40] Indranil Saha, Rattanachai Ramaithitima, Vijay Kumar, George J Pappas, and Sanjit A Seshia. **Automated composition of motion primitives for multi-robot systems from safe LTL specifications**. In *Proc. of the International Conference on Intelligent Robots and Systems*, pages 1525–1532, 2014. 9

[41] Ricardo G Sanfelice and Emilio Frazzoli. **A hybrid control framework for robust maneuver-based motion planning**. In *Proc. of the American Control Conference*, pages 2254–2259, 2008. 9

[42] Jeremy H Gillula, Gabriel M Hoffmann, Haomiao Huang, Michael P Vitus, and Claire J Tomlin. **Applications of hybrid reachability analysis to robotic aerial vehicles**. *The International Journal of Robotics Research*, **30**(3):335–354, 2011. 9

[43] S. L. Herbert, M. Chen, S. Han, S. Bansal, J. F. Fisac, and C. J. Tomlin. **FaSTrack: A modular framework for fast and guaranteed safe motion planning**. In *Proc. of the Conference on Decision and Control*, pages 1517–1522, 2017. 9, 119, 143

[44] Aaron D Ames, Xiangru Xu, Jessy W Grizzle, and Paulo Tabuada. **Control barrier function based quadratic programs for safety critical systems**. *IEEE Transactions on Automatic Control*, **62**(8):3861–3876, 2016. 9, 90, 119, 143

[45] Xiangru Xu, Jessy W Grizzle, Paulo Tabuada, and Aaron D Ames. **Correctness guarantees for the composition of lane keeping and adaptive cruise control**. *IEEE Transactions on Automation Science and Engineering*, **15**(3):1216–1229, 2017. 9, 90, 119, 143

[46] Peter Wieland and Frank Allgöwer. **Constructive safety using control barrier functions**. *IFAC Proceedings Volumes*, **40**(12):462–467, 2007. 9, 90, 119, 143

[47] Manuel Rauscher, Melanie Kimmel, and Sandra Hirche. **Constrained robot control using control barrier functions**. In *International Conference on Intelligent Robots and Systems*, pages 279–285, 2016. 9, 90, 119, 143

[48] Yuxiao Chen, Huei Peng, and Jessy Grizzle. **Obstacle avoidance for low-speed autonomous vehicles with barrier function**. *IEEE Transactions on Control Systems Technology*, **26**(1):194–206, 2017. 9, 90, 119, 143

[49] Xiangru Xu, Paulo Tabuada, Jessy W Grizzle, and Aaron D Ames. **Robustness of control barrier functions for safety critical control**. *IFAC-PapersOnLine*, **48**(27):54–61, 2015. 9, 90, 119, 143

[50] Marius Kloetzer and Calin Belta. **A Fully Automated Framework for Control of Linear Systems from Temporal Logic Specifications**. *IEEE Transactions on Automatic Control*, **53**(1):287–297, 2008. 9, 90, 143, 187

[51] Majid Zamani, Giordano Pola, Manuel Mazo Jr., and Paulo Tabuada. **Symbolic Models for Nonlinear Control Systems without Stability Assumptions**. *IEEE Transactions on Automatic Control*, **57**(7):1804–1809, 2012. 9, 90, 143, 187

[52] Jonathan A DeCastro and Hadas Kress-Gazit. **Synthesis of nonlinear continuous controllers for verifiably correct high-level, reactive behaviors**. *The International Journal of Robotics Research*, **34**(3):378–394, 2015. 9, 90, 143, 187

[53] Georgios E. Fainekos, Antoine Girard, Hadas Kress-Gazit, and George J. Pappas. **Temporal Logic Motion Planning for Dynamic Robots**. *Automatica*, **45**(2):343–352, 2009. 9, 90, 143, 187

[54] Antoine Girard and Samuel Martin. **Motion planning for nonlinear systems using hybridizations and robust controllers on simplices**. In *Proc. of the Conference on Decision and Control*, pages 239–244, 2008. 9, 90, 143, 187

[55] Antoine Girard. **Controller synthesis for safety and reachability via approximate bisimulation**. *Automatica*, **48**(5):947–953, 2012. 9, 90, 143, 187

[56] Hadas Kress-Gazit, Georgios E Fainekos, and George J Pappas. **Temporal-logic-based reactive mission and motion planning**. *Transactions on Robotics*, **25**(6):1370–1381, 2009. 9, 90, 143, 187

[57] Yinan Li, Jun Liu, and Necmiye Ozay. **Computing finite abstractions with robustness margins via local reachable set over-approximation**. *IFAC-PapersOnLine*, **48**(27):1–6, 2015. 9, 90, 143, 187

[58] Jun Liu, Necmiye Ozay, Ufuk Topcu, and Richard M. Murray. **Synthesis of Reactive Switching Protocols from Temporal Logic Specifications**. *IEEE Transactions on Automatic Control*, **58**(7):1771–1785, 2013. 9, 90, 143, 187

[59] Jun Liu and Necmiye Ozay. **Finite abstractions with robustness margins for temporal logic-based control synthesis**. *Nonlinear Analysis: Hybrid Systems*, **22**:1 – 15, 2016. 9, 90, 143, 187

[60] Petter Nilsson and Necmiye Ozay. **Incremental synthesis of switching protocols via abstraction refinement**. In *Proc. of the Conference on Decision and Control*, pages 6246–6253, 2014. 9, 90, 143, 187

[61] Necmiye Ozay, Jun Liu, Pavithra Prabhakar, and Richard M Murray. **Computing augmented finite transition systems to synthesize switching protocols for polynomial switched systems**. In *Proc. of the American Control Conference*, pages 6237–6244, 2013. 9, 90, 143, 187

[62] Giordano Pola, Antoine Girard, and Paulo Tabuada. **Symbolic models for nonlinear control systems using approximate bisimulation**. In *Proc. of the Conference on Decision and Control*, pages 4656–4661, 2007. 9, 90, 143, 187

[63] Vasumathi Raman, Alexandre Donzé, Dorsa Sadigh, Richard M Murray, and Sanjit A Seshia. **Reactive synthesis from signal temporal logic specifications**. In *Proc. of the International Conference on Hybrid Systems: Computation and Control*, pages 239–248, 2015. 9, 90, 143, 187

[64] Matthias Rungger, Manuel Mazo Jr., and Paulo Tabuada. **Specification-Guided Controller Synthesis for Linear Systems and Safe Linear-Time Temporal Logic**. In *Proc. of the International Conference on Hybrid Systems: Computation and Control*, pages 333–342, 2013. 9, 90, 143, 187

[65] Sadra Sadraddini and Calin Belta. **Safety control of monotone systems with bounded uncertainties**. In *Proc. of the Conference on Decision and Control*, pages 4874–4879, 2016. 9, 90, 143, 187

[66] Tichakorn Wongpiromsarn, Ufuk Topcu, and Richard M Murray. **Receding horizon temporal logic planning**. *Transactions on Automatic Control*, **57**(11):2817–2830, 2012. 9, 90, 143, 187

[67] Majid Zamani, Alessandro Abate, and Antoine Girard. **Symbolic models for stochastic switched systems: A discretization and a discretization-free approach**. *Automatica*, **55**:183–196, 2015. 9, 90, 143, 187

[68] Eric M Wolff, Ufuk Topcu, and Richard M Murray. **Automaton-guided controller synthesis for nonlinear systems with temporal logic**. In *Proc. of the International Conference on Intelligent Robots and Systems*, pages 4332–4339, 2013. 9, 90, 143, 187

[69] Eric M Wolff and Richard M Murray. **Optimal control of nonlinear systems with temporal logic specifications**. In *Robotics Research*, pages 21–37. Springer, 2016. 9, 90, 143, 187

[70] Jonathan A. DeCastro and Hadas Kress-Gazit. **Nonlinear Controller Synthesis and Automatic Workspace Partitioning for Reactive High-Level Behaviors**. In *Proc. of the International Conference on Hybrid Systems: Computation and Control*, pages 225–234, 2016. 9, 90, 143, 187

[71] Ivan Papusha, Jie Fu, Ufuk Topcu, and Richard M Murray. **Automata theory meets approximate dynamic programming: Optimal control with temporal logic constraints**. In *Proc. of the Conference on Decision and Control*, pages 434–440, 2016. 9, 90, 143, 187

[72] C.F. Verdier and Manuel Mazo Jr. **Formal Controller Synthesis via Genetic Programming**. *IFAC-PapersOnLine*, **50**(1):7205 – 7210, 2017. IFAC World Congress. 9

[73] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004. 15, 39, 194, 200

[74] Matthias Althoff. *Reachability Analysis and its Application to the Safety Assessment of Autonomous Cars*. PhD thesis, Technische Universität München, 2010. 18, 22, 23, 24, 39, 40, 85, 92, 127, 158

[75] N. Kochdumper and M. Althoff. **Sparse Polynomial Zonotopes: A Novel Set Representation for Reachability Analysis**. *IEEE Transactions on Automatic Control*, **66**(9):4043–4058, 2020. 18, 19, 132

[76] M. Althoff. **Reachability Analysis of Nonlinear Systems using Conservative Polynomialization and Non-Convex Sets**. In *Proc. of the International Conference on Hybrid Systems: Computation and Control*, pages 173–182, 2013. 18, 24, 118, 130, 141

[77] C. Combastel. **A state bounding observer based on zonotopes**. In *Proc. of the European Control Conference*, pages 2589–2594, 2003. 20, 150

[78] V. T. H. Le, C. Stoica, T. Alamo, E. F. Camacho, and D. Dumur. **Zonotopic Guaranteed State Estimation for Uncertain Systems**. *Automatica*, **49**(11):3418–3424, 2013. 20, 150

[79] Matthias Althoff and Jagat Jyoti Rath. **Comparison of guaranteed state estimators for linear time-invariant systems**. *Automatica*, **130**:109662, 2021. 20, 150

[80] André Platzer and Edmund M Clarke. **The Image Computation Problem in Hybrid Systems Model Checking**. In *Proc. of the International Conference on Hybrid Systems: Computation and Control*, pages 473–486, 2007. 21, 142

[81] Matthias Rungger and Majid Zamani. **Accurate Reachability Analysis of Uncertain Nonlinear Systems**. In *Proc. of the International Conference on Hybrid Systems: Computation and Control*, pages 61–70, 2018. 21

[82] M. Althoff. **An Introduction to CORA 2015**. In *Proc. of the Workshop on Applied Verification for Continuous and Hybrid Systems*, pages 120–151, 2015. 22, 45

[83] A. Girard. **Reachability of uncertain linear systems using zonotopes**. In *Proc. of the International Conference on Hybrid Systems: Computation and Control*, pages 291–305, 2005. 24, 127

[84] Gerrit Jan Tretmans. *A formal approach to conformance testing*. PhD thesis, Universiteit Twente, 1992. 25, 26

[85] Thao Dang, Oded Maler, and Romain Testylier. **Accurate Hybridization of Nonlinear Systems**. In *Proc. of the International Conference on Hybrid Systems: Computation and Control*, pages 11–20, 2010. 26, 91

[86] Hendrik Roehm, Jens Oehlerking, Matthias Woehrle, and Matthias Althoff. **Reachset Conformance Testing of Hybrid Automata**. In *Proc. of the International Conference on Hybrid Systems: Computation and Control*, pages 277–286, 2016. 27

[87] Panos J. Antsaklis and Anthony N. Michel. *Linear Systems*. Birkhäuser, 2nd edition, 2006. 31

[88] Thi Xuan Thao Dang. *Vérification Et Synthèse Des Systèmes Hybrides*. PhD thesis, Institut National Polytechnique de Grenoble, 2000. 31

[89] Alexander Weber and Gunther Reissig. **Classical and Strong Convexity of Sublevel Sets and Application to Attainable Sets of Nonlinear Systems**. *SIAM Journal on Control and Optimization*, **52**(5):2857–2876, 2014. 34

[90] Rajesh Rajamani. *Vehicle Dynamics and Control*. Springer, 2012. 44

[91] B. Houska, H.J. Ferreau, and M. Diehl. **ACADO Toolkit – An Open Source Framework for Automatic Control and Dynamic Optimization**. *Optimal Control Applications and Methods*, **32**(3):298–312, 2011. 45

[92] George B Dantzig and Mukund N Thapa. *Linear Programming 1: Introduction*. Springer, 2006. 53, 100

[93] Michael Grant, Stephen Boyd, and Yinyu Ye. **Disciplined Convex Programming**. In *Global Optimization: from Theory to Implementation*, pages 155–210. Springer, 2006. 53, 100

[94] Michael Grant and Stephen Boyd. **CVX: Matlab Software for Disciplined Convex Programming, version 2.1**. http://cvxr.com/cvx, March 2014. 58

[95] Karl Johan Aström and Richard M Murray. *Feedback Systems: An Introduction for Scientists and Engineers*. Princeton University Press, 2010. 61

[96] Ibtissem Ben Makhlouf and Stefan Kowalewski. **Networked Cooperative Platoon of Vehicles for Testing Methods and Verification Tools**. In *Proc. of the International Workshop on Applied veRification for Continuous and Hybrid Systems*, pages 37–42, 2015. 67

[97] Shuyou Yu, Hong Chen, and Frank Allgöwer. **Tube MPC scheme based on robust control invariant set with application to Lipschitz nonlinear systems**. In *Proc. of the Conference on Decision and Control*, pages 2650–2655, 2011. 79, 82, 83

[98] John T Betts. *Practical Methods for Optimal Control and Estimation Using Nonlinear Programming*. Siam, 2010. 85

[99] N. Karmarkar. **A new polynomial-time algorithm for linear programming**. *Combinatorica*, **4**(4):373–395, 1984. 86, 113, 140

[100] O. Schütze. *Set Oriented Methods for Global Optimization*. PhD thesis, Univ. Paderborn, 2004. 87

[101] Frank L. Lewis. *Optimal Control*. Wiley, 1986. 87

[102] Ernest Bruce Lee and Lawrence Markus. *Foundations of Optimal Control Theory*. Wiley, 1967. 87

[103] Hassan K. Khalil. *Nonlinear Systems*. Pearson Education International, 3rd edition, 1996. 90

[104] Eugene Asarin, Thao Dang, and Antoine Girard. **Hybridization methods for the analysis of nonlinear systems**. *Acta Informatica*, **43**(7):451–476, 2007. 91

[105] A. Girard and S. Martin. **Synthesis for Constrained Nonlinear Systems Using Hybridization and Robust Controllers on Simplices**. *IEEE Transactions on Automatic Control*, **57**(4):1046–1051, 2012. 91

[106] Stanley Bak, Sergiy Bogomolov, Thomas A. Henzinger, Taylor T. Johnson, and Pradyot Prakash. **Scalable Static Hybridization Methods for Analysis of Nonlinear Systems**. In *Proc. of the International Conference on Hybrid Systems: Computation and Control*, pages 155–164, 2016. 91

[107] Tobias Geyer, Fabio D. Torrisi, and Manfred Morari. **Optimal complexity reduction of polyhedral piecewise affine systems**. *Automatica*, **44**(7):1728–1740, 2008. 91

[108] Alessandro Vittorio Papadopoulos and Maria Prandini. **Model reduction of switched affine systems**. *Automatica*, **70**:57–65, 2016. 91

[109] A. Bemporad, G. Ferrari-Trecate, and M. Morari. **Observability and Controllability of Piecewise Affine and Hybrid Systems**. *IEEE Transactions on Automatic Control*, **45**(10):1864–1876, 2000. 91, 97, 98

[110] Giancarlo Ferrari-Trecate, Marco Muselli, Diego Liberati, and Manfred Morari. **A clustering technique for the identification of piecewise affine systems**. *Automatica*, **39**(2):205–217, 2003. 91

[111] Jacob Roll, Alberto Bemporad, and Lennart Ljung. **Identification of piecewise affine systems via mixed-integer programming**. *Automatica*, **40**(1):37–50, 2004. 91

[112] A. Bemporad, A. Garulli, S. Paoletti, and A. Vicino. **A bounded-error approach to piecewise affine system identification**. *IEEE Transactions on Automatic Control*, **50**(10):1567–1580, 2005. 91

[113] M. Rubagotti, S. Trimboli, and A. Bemporad. **Stability and Invariance Analysis of Uncertain Discrete-Time Piecewise Affine Systems**. *IEEE Transactions on Automatic Control*, **58**(9):2359–2365, 2013. 92

[114] M. Gholami, V. Cocquempot, H. Schiøler, and T. Bak. **Active fault tolerant control of piecewise affine systems with reference tracking and input constraints**. *International Journal of Adaptive Control and Signal Processing*, **28**(11):1240–1265, 2014. 92

[115] J. Xu and L. Xie. *Control and Estimation of Piecewise Affine Systems*. Elsevier Science, 2014. 92

[116] A. Bemporad, W. P. M. H. Heemels, and B. De Schutter. **On hybrid systems and closed-loop MPC systems**. *IEEE Transactions on Automatic Control*, **47**(5):863–869, 2002. 92

[117] M. Lazar, W. P. M. H. Heemels, S. Weiland, and A. Bemporad. **Stabilizing Model Predictive Control of Hybrid Systems**. *IEEE Transactions on Automatic Control*, **51**(11):1813–1818, 2006. 92

[118] Jan Lunze and Françoise Lamnabhi-Lagarrigue. *Handbook of hybrid systems control: theory, tools, applications*. Cambridge University Press, 2009. 92

[119] A. Bemporad and M. Morari. **Control of systems integrating logic, dynamics, and constraints**. *Automatica*, **35**(3):407–427, 1999. 92, 97, 98

[120] Alongkrit Chutinan and Bruce H. Krogh. **Verification of Polyhedral-Invariant Hybrid Automata Using Polygonal Flow Pipe Approximations**. In *Proc. of the International Workshop on Hybrid Systems: Computation and Control*, pages 76–90, 1999. 92

[121] Goran Frehse and Rajarshi Ray. **Flowpipe-Guard Intersection for Reachability Computations with Support Functions**. In *Analysis and Design of Hybrid Systems*, pages 94–101, 2012. 92

[122] M. Althoff, C. Le Guernic, and B. H. Krogh. **Reachable Set Computation for Uncertain Time-Varying Linear Systems**. In *Proc. of the International Conference on Hybrid Systems: Computation and Control*, pages 93–102, 2011. 92

[123] M. Althoff and B. H. Krogh. **Avoiding Geometric Intersection Operations in Reachability Analysis of Hybrid Systems**. In *Proc. of the International Conference on Hybrid Systems: Computation and Control*, pages 45–54, 2012. 92

[124] Jochen Till, Sebastian Engell, Sebastian Panek, and Olaf Stursberg. **Applied hybrid system optimization: An empirical investigation of complexity**. *Control Engineering Practice*, **12**(10):1291–1303, 2004. 92, 114

[125] M. S. Branicky, V. S. Borkar, and S. K. Mitter. **A unified framework for hybrid control: model and optimal control theory**. *IEEE Transactions on Automatic Control*, **43**(1):31–45, 1998. 92, 114

[126] M. S. Branicky and S. K. Mitter. **Algorithms for optimal hybrid control**. In *Proc. of the Conference on Decision and Control*, pages 2661–2666, 1995. 92, 114

[127] M Shahid Shaikh and Peter E Caines. **On the optimal control of hybrid systems: Optimization of trajectories, switching times, and location schedules**. *Lecture Notes in Computer Science*, **2623**:466–481, 2003. 92, 114

[128] M Shahid Shaikh and Peter E Caines. **On the hybrid optimal control problem: theory and algorithms**. *IEEE Transactions on Automatic Control*, **52**(9):1587–1603, 2007. 92, 114

[129] Alberto Bemporad, Francesco Borrelli, and Manfred Morari. **Optimal controllers for hybrid systems: Stability and piecewise linear explicit form**. In *Proc. of the Conference on Decision and Control*, **2**, pages 1810–1815, 2000. 92, 114

[130] Alberto Bemporad, Fabio Danilo Torrisi, and Manfred Morari. **Optimization-based verification and stability characterization of piecewise affine and hybrid systems**. In *Proc. of the International Conference on Hybrid Systems: Computation and Control*, pages 45–58, 2000. 92, 114

[131] A. Bemporad, F. Borrelli, and M. Morari. **Piecewise linear optimal controllers for hybrid systems**. In *Proc. of the American Control Conference*, pages 1190–1194, 2000. 92, 114

[132] Francesco Borrelli, Mato Baotić, Alberto Bemporad, and Manfred Morari. **Dynamic programming for constrained optimal control of discrete-time linear hybrid systems**. *Automatica*, **41**(10):1709–1721, 2005. 92, 114

[133] Benjamin Passenberg and Olaf Stursberg. **Graph search for optimizing the discrete location sequence in hybrid optimal control**. *IFAC Proceedings Volumes*, **42**(17):304–309, 2009. 92

[134] Miguel Pedro Silva, L Pina, A Bemporad, M Ayala Botto, and J Sá da Costa. **Robust optimal control of linear hybrid systems: an MLD approach**. In *Proc. of the Portuguese Conference on Automatic Control*, pages 208–213, 2004. 92

[135] Miguel Pedro Silva, Alberto Bemporad, M Ayala Botto, and J Sá da Costa. **Optimal control of uncertain piecewise affine/mixed logical dynamical systems**. In *Proc. of the European Control Conference*, pages 1573–1578. IEEE, 2003. 92

[136] Stefano Di Cairano, WP Maurice H Heemels, Mircea Lazar, and Alberto Bemporad. **Stabilizing dynamic controllers for hybrid systems: a hybrid control Lyapunov function approach**. *IEEE Transactions on Automatic Control*, **59**(10):2629–2643, 2014. 92

[137] L. C. G. J. M. Habets, Pieter J Collins, and Jan H van Schuppen. **Reachability and control synthesis for piecewise-affine hybrid systems on simplices**. *IEEE Transactions on Automatic Control*, **51**(6):938–948, 2006. 92

[138] L. C. G. J. M. Habets, P. J. Collins, and J. H. van Schuppen. **Control to Facet by Piecewise-Affine Output Feedback**. *IEEE Transactions on Automatic Control*, **57**(11):2831–2843, 2012. 92

[139] David Q Mayne and S Raković. **Model predictive control of constrained piecewise affine discrete-time systems**. *International Journal of Robust and Nonlinear Control*, **13**(3-4):261–279, 2003. 92, 115

[140] E.F. Camacho, D.R. Ramirez, D. Limon, D. Muñoz de la Peña, and T. Alamo. **Model predictive control techniques for hybrid systems**. *Annual Reviews in Control*, **34**(1):21–31, 2010. 92, 115

[141] Masakazu Mukai, Takehito Azuma, and Masayuki Fujita. **Robust receding horizon control for piecewise linear systems based on constrained positively invariant**. In *Proc. of the American Control Conference*, pages 2348–2353, 2002. 92, 115

[142] Eric C Kerrigan and David Q Mayne. **Optimal control of constrained, piecewise affine systems with bounded disturbances**. In *Proc. of the Conference on Decision and Control*, pages 1552–1557, 2002. 92, 115

[143] I Necoara, B De Schutter, Ton JJ van den Boom, and J Hellendoorn. **Model predictive control for perturbed continuous piecewise affine systems with bounded disturbances**. In *Proc. of the Conference on Decision and Control*, pages 1848–1853, 2004. 92, 115

[144] Yuanyuan Zou and Shaoyuan Li. **Robust constrained model predictive control of piecewise linear systems with bounded additive disturbances**. In *Proc. of the Conference on Decision and Control*, pages 6339–6344, 2009. 93

[145] K. Hariprasad and Sharad Bhartiya. **A computationally efficient robust tube based MPC for linear switched systems**. *Nonlinear Analysis: Hybrid Systems*, **19**:60–76, 2016. 93, 115

[146] Mohammad S. Ghasemi and Ali A. Afzalian. **Robust tube-based MPC of constrained piecewise affine systems with bounded additive disturbances**. *Nonlinear Analysis: Hybrid Systems*, **26**:86–100, 2017. 93

[147] S Raković, P Grieder, Michal Kvasnica, DQ Mayne, and Manfred Morari. **Computation of invariant sets for piecewise affine discrete time systems subject to bounded disturbances**. In *Proc. of the Conference on Decision and Control*, pages 1418–1423, 2004. 93, 115

[148] Paul J Goulart, Eric C Kerrigan, and Jan M Maciejowski. **Optimization over state feedback policies for robust control with constraints**. *Automatica*, **42**:523–533, 2006. 108

[149] Karl Henrik Johansson. **The quadruple-tank process: A multivariable laboratory process with an adjustable zero**. *Control Systems Technology, IEEE Transactions on*, **8**(3):456–465, 2000. 108

[150] Manuel Mazo, Anna Davitian, and Paulo Tabuada. **PESSOA: A Tool for Embedded Controller Synthesis**. In *Proc. of the International Conference on Computer Aided Verification*, pages 566–569. Springer, 2010. 112, 115

[151] R. M. Vignali, D. Ioli, and M. Prandini. **A data-driven approach to stochastic constrained control of piecewise affine systems**. In *Proc. of the American Control Conference*, pages 1424–1429, 2018. 112, 114, 115

[152] Dimitris Bertsimas and John H. Tsitsiklis. *Introduction to Linear Optimization*. Dynamic Ideas and Athena Scientific, 2008. 114

[153] Dimitris Bertsimas and Robert Weismantel. *Optimization over Integers*. Dynamic Ideas, 2005. 114

[154] Ian M Mitchell. **Comparing forward and backward reachability as tools for safety analysis**. In *Proc. of the International Conference on Hybrid Systems: Computation and Control*, pages 428–443. Springer, 2007. 118

[155] D.P. Bertsekas and I.B. Rhodes. **On the minimax reachability of target sets and target tubes**. *Automatica*, **7**(2):233–247, 1971. 118

[156] Alexander B Kurzhanski and Pravin Varaiya. **Ellipsoidal techniques for reachability analysis: internal approximation**. *Systems & control letters*, **41**(3):201–211, 2000. 118, 143

[157] Antoine Girard, Colas Le Guernic, and Oded Maler. **Efficient computation of reachable sets of linear time-invariant systems with inputs**. In *Proc. of the International Conference on Hybrid Systems: Computation and Control*, pages 257–271, 2006. 118, 143

[158] Abdullah Hamadeh and Jorge Goncalves. **Reachability analysis of continuous-time piecewise affine systems**. *Automatica*, **44**(12):3189–3194, 2008. 118, 143

[159] Bai Xue, Martin Fränzle, and Naijun Zhan. **Inner-approximating reachable sets for polynomial systems with time-varying uncertainties**. *IEEE Transactions on Automatic Control*, **65**(4):1468–1483, 2019. 118, 143

[160] Milan Korda, Didier Henrion, and Colin N. Jones. **Inner approximations of the region of attraction for polynomial dynamical systems**. *IFAC Proceedings Volumes*, **46**(23):534–539, 2013. 118, 143

[161] Bai Xue, Martin Fränzle, and Naijun Zhan. **Under-Approximating Reach Sets for Polynomial Continuous Systems**. In *Proc. of the International Conference on Hybrid Systems: Computation and Control*, pages 51–60, 2018. 118, 143

[162] Meilun Li, Peter N Mosaad, Martin Fränzle, Zhikun She, and Bai Xue. **Safe over-and under-approximation of reachable sets for autonomous dynamical systems**. In *Proc. of the International Conference on Formal Modeling and Analysis of Timed Systems*, pages 252–270. Springer, 2018. 118, 143

[163] B. Xue, Z. She, and A. Easwaran. **Underapproximating Backward Reachable Sets by Semialgebraic Sets**. *IEEE Transactions on Automatic Control*, **62**(10):5185–5197, 2017. 118, 143

[164] Bai Xue, Zhikun She, and Arvind Easwaran. **Under-approximating backward reachable sets by polytopes**. In *Proc. of the International Conference on Computer Aided Verification*, pages 457–476. Springer, 2016. 118, 143

[165] Xin Chen, Sriram Sankaranarayanan, and Erika Ábrahám. **Under-approximate flowpipes for non-linear continuous systems**. In *Proc. of the Conference on Formal Methods in Computer-Aided Design*, pages 59–66, 2014. 118, 143

[166] Eric Goubault and Sylvie Putot. **Forward inner-approximated reachability of non-linear continuous systems**. In *Proc. of the International Conference on Hybrid Systems: Computation and Control*, pages 1–10, 2017. 118, 143

[167] Eric Goubault and Sylvie Putot. **Inner and Outer Reachability for the Verification of Control Systems**. In *Proc. of the International Conference on Hybrid Systems: Computation and Control*, pages 11–22, 2019. 118, 143

[168] D. Han, A. Rizaldi, A. El-Guindy, and M. Althoff. **On enlarging backward reachable sets via Zonotopic set membership**. In *Proc. of the International Symposium on Intelligent Control*, pages 1–8, 2016. 118

[169] Mo Chen, Sylvia L Herbert, Mahesh S Vashishtha, Somil Bansal, and Claire J Tomlin. **Decomposition of reachable sets and tubes for a class of nonlinear systems**. *IEEE Transactions on Automatic Control*, **63**(11):3675–3688, 2018. 118

[170] Lui Sha. **A Software Architecture for Dependable and Evolvable Industrial Computing Systems**. Technical report, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, 1995. 119

[171] Lui Sha. **Using simplicity to control complexity**. *IEEE Software*, **18**(4):20–28, 2001. 119, 143

[172] Stanley Bak, Taylor T. Johnson, Marco Caccamo, and Lui Sha. **Real-Time Reachability for Verified Simplex Design**. In *Proc. of the Real-Time Systems Symposium*, pages 138–148, 2014. 119, 143

[173] Stanley Bak, Karthik Manamcheri, Sayan Mitra, and Marco Caccamo. **Sandboxing Controllers for Cyber-Physical Systems**. In *Proc. of the International Conference on Cyber-Physical Systems*, pages 3–12, 2011. 119, 143

[174] J. Wolff and M. Buss. **Invariance Control Design for Constrained Nonlinear Systems**. *IFAC Proceedings Volumes*, **38**(1):37–42, 2005. IFAC World Congress. 119, 143

[175] Melanie Kimmel and Sandra Hirche. **Invariance control with chattering reduction**. In *Proc. of the Conference on Decision and Control*, pages 68–74, 2014. 119, 143

[176] Elmer Gilbert and Ilya Kolmanovsky. **Nonlinear tracking control in the presence of state and control constraints: a generalized reference governor**. *Automatica*, **38**(12):2063–2073, 2002. 119

[177] A. Bemporad. **Reference governor for constrained nonlinear systems**. *IEEE Transactions on Automatic Control*, **43**(3):415–419, 1998. 119, 143

[178] Emanuele Garone, Stefano Di Cairano, and Ilya Kolmanovsky. **Reference and command governors for systems with constraints: A survey on theory and applications**. *Automatica*, **75**:306–328, 2017. 119

[179] Jaime F Fisac, Anayo K Akametalu, Melanie N Zeilinger, Shahab Kaynama, Jeremy Gillula, and Claire J Tomlin. **A general safety framework for learning-based control in uncertain robotic systems**. *IEEE Transactions on Automatic Control*, **64**(7):2737–2752, 2018. 119

[180] Bettina Könighofer, Mohammed Alshiekh, Roderick Bloem, Laura Humphrey, Robert Könighofer, Ufuk Topcu, and Chao Wang. **Shield synthesis**. *Formal Methods in System Design*, **51**(2):332–361, 2017. 119, 143

[181] Mohammed Alshiekh, Roderick Bloem, Rüdiger Ehlers, Bettina Könighofer, Scott Niekum, and Ufuk Topcu. **Safe Reinforcement Learning via Shielding**. In *Proc. of the Conference on Artificial Intelligence*, pages 2669–2678, 2018. 119, 143

[182] Somil Bansal, Mo Chen, Sylvia Herbert, and Claire J Tomlin. **Hamilton-Jacobi reachability: A brief overview and recent advances**. In *Proc. of the Conference on Decision and Control*, pages 2242–2253, 2017. 119

[183] Edward Barry Saff and Arthur David Snider. *Fundamentals of Matrix Analysis with Applications*. John Wiley & Sons, 2015. 122

[184] S. Sadraddini and R. Tedrake. **Linear Encodings for Polytope Containment Problems**. In *Proc. of the Conference on Decision and Control*, pages 4367–4372, 2019. 127, 133

[185] Adrian Kulmburg and Matthias Althoff. **On the co-NP-completeness of the zonotope containment problem**. *European Journal of Control*, 2021. DOI https://doi.org/10.1016/j.ejcon.2021.06.028. 127, 133

[186] G. C. Shephard. **Combinatorial Properties of Associated Zonotopes**. *Canadian Journal of Mathematics*, **26**(2):302–321, 1974. 132

[187] M. Maïga, N. Ramdani, L. Travé-Massuyès, and C. Combastel. **A Comprehensive Method for Reachability Analysis of Uncertain Nonlinear Hybrid Systems**. *IEEE Transactions on Automatic Control*, **61**(9):2341–2356, 2016. 132

[188] M. Althoff and J. M. Dolan. **Online Verification of Automated Road Vehicles Using Reachability Analysis**. *IEEE Transactions on Robotics*, **30**(4):903–918, 2014. 141

[189] Victor M. Zavala and Lorenz T. Biegler. **The advanced-step NMPC controller: Optimality, stability and robustness**. *Automatica*, **45**(1):86–93, 2009. 145

[190] H. Michalska and D. Q. Mayne. **Robust Receding Horizon Control of Constrained Nonlinear Systems**. *IEEE Transactions on Automatic Control*, **38**(11):1623–1633, 1993. 146, 149

[191] Ahmed El-Guindy, Dongkun Han, and Matthias Althoff. **Estimating the region of attraction via forward reachable sets**. In *Proc. of the American Control Conference*, pages 1263–1270, 2017. 149, 150, 156

[192] Anna-Katharina Rettinger. *Ensuring Drivability of Fail-safe Trajectories for Autonomous Vehicles using Set-based Control Techniques*. Master's thesis, Technische Universität München, 2018. 161, 171

[193] M. Althoff, M. Koschi, and S. Manzinger. **CommonRoad: Composable benchmarks for motion planning on roads**. In *Proc. of the Intelligent Vehicles Symposium*, pages 719–726, 2017. 162

[194] Christian Pek and Matthias Althoff. **Computationally Efficient Fail-safe Trajectory Planning for Self-driving Vehicles Using Convex Optimization**. In *Proc. of the International Conference on Intelligent Transportation Systems*, pages 1447–1454, 2018. 170, 171, 175

[195] Bruno Siciliano, Lorenzo Sciavicco, Luigi Villani, and Giuseppe Oriolo. *Robotics: modelling, planning and control*. Springer Science & Business Media, 2010. 177

[196] Salvatore Nicosia, A. Tornambè, and P. Valigi. **State estimation in robotic manipulators: Some experimental results**. *Journal of Intelligent & Robotic Systems*, **7**(3):321–351, 1993. 178

[197] Brian Armstrong-Hélouvry, Pierre Dupont, and Carlos Canudas De Wit. **A survey of models, analysis tools and compensation methods for the control of machines with friction**. *Automatica*, **30**(7):1083–1138, 1994. 178

[198] Timothy D Tuttle and Warren P Seering. **A nonlinear model of a harmonic drive gear transmission**. *IEEE Transactions on Robotics and Automation*, **12**(3):368–374, 1996. 178, 180

[199] Stefan B. Liu and Matthias Althoff. **Reachset Conformance of Forward Dynamic Models for the Formal Analysis of Robots**. In *Proc. of the International Conference on Intelligent Robots and Systems*, pages 370–376, 2018. 179

[200] Felix Gruber and Matthias Althoff. **Computing Safe Sets of Linear Sampled-Data Systems**. *IEEE Control Systems Letters*, **5**(2):385–390, 2021. 179

[201] Pierre Apkarian, Pascal Gahinet, and Greg Becker. **Self-scheduled H-infinity Control of Linear Parameter-varying Systems: a Design Example**. *Automatica*, **31**(9):1251–1261, 1995. 193, 194

[202] Zhongwei Yu, Huitang Chen, and Peng-yung Woo. **Gain Scheduled LPV H-infinity Control Based on LMI Approach for a Robotic Manipulator**. *Journal of Robotic Systems*, **19**(12):585–593, 2002. 194

[203] Hakan Koc, Dominique Knittel, Michel De Mathelin, and Gabriel Abba. **Modeling and Robust Control of Winding Systems for Elastic Webs**. *IEEE Transactions on Control Systems Technology*, **10**(2):197–208, 2002. 194

[204] Xing-Jia Yao, Chang-Chun Guo, and Yan Li. **LPV H-infinity controller design for variable-pitch variable-speed wind turbine**. In *International Power Electronics and Motion Control Conference*, pages 2222–2227, 2009. 194

[205] Zhilin Liu, Qidan Zhu, and Lihui Wang. **Predictive Control for Multi-joint Manipulator with Polytopic Model**. In *Proc. of the International Conference on Robotics and Biomimetics*, pages 2130–2133, 2009. 194

[206] He Chaofan, Yang Lingyu, Wang Zhenchao, Sun Bin, and Zhang Jing. **Linear Parameter-Varying Attitude Controller Design for a Reusable Launch Vehicle during Reentry**. In *Proc. of the Chinese Guidance, Navigation and Control Conference*, pages 2723–2728, 2014. 194, 200

[207] Dengying and Zhoujie. **LPV H-infinity Controller Design for a Wind Power Generator**. In *Proc. of the Conference on Robotics, Automation and Mechatronics*, pages 873–878, 2008. 194, 200

[208] V F Montagner, R C L F Oliveira, Valter J S Leite, and Pedro Luis Dias Peres. **Gain scheduled state feedback control of discrete-time systems with time-varying uncertainties: an LMI approach**. In *Proc. of the Conference on Decision and Control and European Control Conference*, pages 4305–4310, 2005. 194

[209] Michael S. Floater. **Generalized barycentric coordinates and applications**. *Acta Numerica*, **24**:161–214, 2015. 194

[210] Joe Warren, Scott Schaefer, Anil N. Hirani, and Mathieu Desbrun. **Barycentric coordinates for convex sets**. *Advances in Computational Mathematics*, **27**(3):319–338, 2007. 194

[211] August Ferdinand Möbius. *Der barycentrische Calcul*. Verlag von Johann Ambrosius Barth, 1827. 195

[212] Carl W. Lee. **Subdivisions and Triangulations of Polytopes**. In Jacob E. Goodman and Joseph O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, pages 271–290. Chapman and Hall/CRC, 2nd edition, 2004. 200

[213] Petter Tøndel, Tor Arne Johansen, and Alberto Bemporad. **Evaluation of Piecewise Affine Control via Binary Search Tree**. *Automatica*, **39**(5):945–950, 2003. 202, 203

Author's Publications

[214] Bastian Schürmann, Moritz Klischat, Niklas Kochdumper, and Matthias Althoff. **Formal Safety Net Control Using Backward Reachability Analysis**. *IEEE Transactions on Automatic Control*, 2022. In press. 4, 12, 117

[215] Bastian Schürmann, Daniel Hess, Jan Eilbrecht, Olaf Stursberg, Frank Köster, and Matthias Althoff. **Ensuring Drivability of Planned Motions Using Formal Methods**. In *Proc. of the Intelligent Transportation Systems Conference*, pages 1661–1668, 2017. 5, 10, 13, 26, 163

[216] Bastian Schürmann and Matthias Althoff. **Convex Interpolation Control with Formal Guarantees for Disturbed and Constrained Nonlinear Systems**. In *Proc. of the International Conference on Hybrid Systems: Computation and Control*, pages 121–130, 2017. DOI 10.1145/3049797.3049800. 10, 31

[217] Bastian Schürmann and Matthias Althoff. **Optimal Control of Sets of Solutions to Formally Guarantee Constraints of Disturbed Linear Systems**. In *Proc. of the American Control Conference*, pages 2522–2529, 2017. 10, 61

[218] Bastian Schürmann and Matthias Althoff. **Guaranteeing Constraints of Disturbed Nonlinear Systems Using Set-Based Optimal Control in Generator Space**. In *Proc. of the IFAC World Congress*, pages 12020–12027, 2017. DOI 10.1016/j.ifacol.2017.08.1617. 10, 49

[219] Bastian Schürmann and Matthias Althoff. **Optimizing Sets of Solutions for Controlling Constrained Nonlinear Systems**. *IEEE Transactions on Automatic Control*, **66**(3):981–994, 2021. 10, 30, 71, 89

[220] Jin Ge, Bastian Schürmann, Richard M. Murray, and Matthias Althoff. **Risk-aware motion planning for automated vehicle among human-driven cars**. In *Proc. of the American Control Conference*, pages 3987–3993, 2019. 10, 191

[221] Tung Phan, Steve Guo, Bastian Schürmann, Matthias Althoff, and Richard M. Murray. **A modal interface contract theory for guarded input/output automata with an application in traffic system design**. In *Proc. of the American Control Conference*, pages 1704–1711, 2019. 10, 190

[222] Albert Rizaldi, Fabian Immler, Bastian Schürmann, and Matthias Althoff. **A Formally Verified Motion Planner for Autonomous Vehicles**. In *Proc. of the International Symposium on Automated Technology for Verification and Analysis*, pages 75–90, 2018. 10

[223] Niklas Kochdumper, Felix Gruber, Bastian Schürmann, Victor Gassmann, Moritz Klischat, and Matthias Althoff. **AROC: A Toolbox for Automated Reachset Optimal Controller Synthesis**. In 1-6, editor, *Proc. of the International Conference on Hybrid Systems: Computation and Control*, 2021. 11

[224] Bastian Schürmann, Ahmed El-Guindy, and Matthias Althoff. **Closed-Form Expressions of Convex Combinations**. In *Proc. of the American Control Conference*, pages 2795–2801, 2016. 11, 193

[225] Ahmed El-Guindy, Konstantin Schaab, Bastian Schürmann, Dongkun Han, Olaf Stursberg, and Matthias Althoff. **Formal LPV Control for Transient Stability of Power Systems**. In *Proc. of the IEEE Power and Energy Society General Meeting*, pages 1–5, 2017. 11

[226] Anna-Kathrin Kopetzki, Bastian Schürmann, and Matthias Althoff. **Methods for Order Reduction of Zonotopes**. In *Proc. of the Conference on Decision and Control*, pages 5626–5633, 2017. 11, 24, 36, 113, 127

[227] Bastian Schürmann, Riccardo Vignali, Maria Prandini, and Matthias Althoff. **Set-Based Control for Disturbed Piecewise Affine Systems with State and Actuation Constraints**. *Nonlinear Analysis: Hybrid Systems*, **36**:100826, 2020. DOI 10.1016/j.nahs.2019.100826. 11, 91

[228] Niklas Kochdumper, Bastian Schürmann, and Matthias Althoff. **Utilizing Dependencies to Obtain Subsets of Reachable Sets**. In *Proc. of the International Conference on Hybrid Systems: Computation and Control*, pages 1–10, 2020. 12, 129, 133, 141, 142, 143

[229] Bastian Schürmann, Niklas Kochdumper, and Matthias Althoff. **Reachset Model Predictive Control of Disturbed Nonlinear Systems**. In *Proc. of the Conference on Decision and Control*, pages 3463–3470, 2018. 13, 146

[230] Stefan B. Liu, Bastian Schürmann, and Matthias Althoff. **Guarantees for Real Robotic Systems: Unifying Formal Controller Synthesis and Reachset-Conformant Identification**. Submitted to IEEE Transactions on Robotics. Preliminary version available at arXiv.org under arXiv:2103.01626. 13, 176