Diss. ETH Zurich No. 25139

Path Planning and Control for Autonomous Racing

A thesis submitted to attain the degree of

DOCTOR OF SCIENCES of ETH ZURICH

(Dr. sc. ETH Zurich)

presented by

ALEXANDER LINIGER

MSc ETH ME, ETH Zürich
born on 31. Dec. 1987
citizen of Wohlen bei Bern, Switzerland

accepted on the recommendation of

Prof. Dr. John Lygeros, examiner (ETH Zurich, Switzerland)
Prof. Dr. Manfred Morari, co-examiner (University of Pennsylvania, USA)
Prof. Dr. Emilio Frazzoli, co-examiner (ETH Zurich, Switzerland)
2018

Alexander Liniger
All Rights Reserved

ISBN 978-3-906916-35-4
DOI 10.3929/ethz-b-000302942

To my family

Acknowledgements

First and foremost, I would like to express my gratitude to Professor Dr. John Lygeros for giving me the possibility to pursue my PhD at the Automatic Control Laboratory (IfA). I appreciated the freedom I had in my research, being able to pursue my ideas while receiving guidance and advice when needed. I am really thankful that he shared his vast knowledge with me and helped me with all the problems I had during my PhD.

Next, I want to thank Professor Dr. Manfred Morari, for being my co-examiner, but also for the advice he gave me during my PhD. Finally, I would also like to thank him for making this PhD possible by starting the ORCA project back in 2009, and for his constant interest in the project. I am also very thankful to Professor Dr. Emilio Frazzoli to agree to be a co-examiner and for his interest in my research.

I would also like to thank the people I collaborated with over the past years, mainly George Zhang for getting me interested in randomized MPC and dual of optimization problems, especially when they can be used for collision avoidance. I would also like to thank him for a very fruitful collaboration which is lasting for years. Further, my gratitude also goes to Tommaso Novi for teaching me about vehicle dynamics and all the interesting discussions we had. Damian Frick I would like to thank for being always there and helping me with strange theory question whenever I showed up in the office. Finally, I would like to thank Alex Domahidi for helping me write my first paper and getting the ORCA project up to speed before I started my PhD.

I also want to thank my office mate Sandro Merkli, and my former office mates Bart van Parys, David Ochsenbein, and Tyler Summers, for all the interesting discussions and joining me for coffees. I would also like to thank, Tony Wood, Damian Frick, and Sandro Merkli for helping me at the “Maturandentage” year in year out.

I would also thank P. Beuchat, M. Colombino, A. Eichler, B. Flamm, C. Fischer, D. Frick, A. Hauswirth, A Hemple, N. Kariotoglou, S. Merkli, D. Paccagnan, N. Pagan, M. Quack, F. Rey, M. Schmitt, Y. Stürz, T. Sutter, G. Torrisi, B. van Parys, T. Wood, and X. Zhang for making the time at IfA so enjoyable. Also, I would like to thank the administrative staff Sabrina Bauman, Tanja Turner, Alain Bolle and Martine Wassmer for running IfA so smooth.

Last but not least I would like to thank my family who has always been there for me: My Parents Edith and Daniel and my sister Jeannette for supporting me and believing in me, without you I would have never achieved what I did. And finally, I want to thank Christina for your love, support, and being always there for me, you are awesome.

Abstract

We investigate autonomous racing of cars and derive control approaches replacing the decision-making process of a human race car driver. The task has three main challenges. First, the controller needs to be able to control the car at the limit of handling. Second, the car has to stay safe on the track and finally, the controller needs to able to interact with other race cars and dynamically adapt to the driving of the other cars. We derive a control approach that can tackle all three goals and validate the proposed approach on the miniature race car set-up hosted at the Automatic Control Lab at ETH Zürich. The validation is performed either on the experimental set-up, or in a simulation environment replicating the experimental set-up.

In a first step, we propose a novel path planning model which is based on a dynamic bicycle model with nonlinear tire forces. The proposed model is able to generate a rich set of possible trajectories that approximate the capabilities of the dynamic bicycle model. The path planning model is optimized for decision making, with only a few decisions and a long discretization time which allows for long term predictions with a few prediction steps. Based on the path planning model, in a second step, we propose a hierarchical racing controller, which uses the strength of the path planning model in an upper level to generate a set of finite look-ahead trajectories out of which the trajectory that maximizes the progress and stays within the track is selected. This trajectory is then tracked using an MPC controller in the lower level subject to the dynamic bicycle model and track constraints.

We propose to use the viability kernel in the path planning step, to guarantee that all generated finite look-ahead trajectories are recursively feasible with respect to static obstacles (e.g., stay inside the track). This formally adds safety to the path planning step and at the same time speeds up the computation time and increases the closed-loop performance. The viability kernel computation is based on space discretization. To make the calculation robust against discretization errors, we propose a novel numerical scheme based on game theoretical methods, in particular the discriminating kernel. We show that the resulting algorithm provides an inner approximation of the viability kernel and guarantees that, for all states in the cell surrounding a viable grid point, there exists an input that keeps the system within the kernel. The performance of the proposed hierarchical racing controller with viability constraints is studied in simulation where we determine the effects of various design choices and parameters and in experiments in the autonomous racing set-up. Both simulation and experimental results suggest that the more conservative approximation using the discriminating kernel results in a safer driving style which starts braking earlier into curves, but this changed driving style comes at the cost of a small increase in lap time.

The proposed hierarchical racing controller with viability constraints is able to tackle the first two challenges in autonomous racing. For the third task where racing against other cars is of interest, we propose an approach to formulate racing decisions as a non-cooperative non-zero-sum game. We design three different games where the players aim to fulfill static track constraints as well as avoid collision with each other; the latter constraint depends on the combined actions of the two players. The difference between the games are the collision constraints and the payoff. In the first game collision avoidance is only considered by the follower, and each player maximizes their own progress towards the finish line. We show that, thanks to the sequential structure of this game, equilibria can be computed through an efficient sequential maximization approach. Further, we show that these actions, if feasible, are also a Stackelberg and Nash equilibrium in pure strategies of our second game where both players consider the collision constraints. The payoff of our third game is designed to promote blocking, by additionally rewarding the cars for staying ahead at the end of the horizon. We show that this changes the Stackelberg equilibrium, but has a minor influence on the Nash equilibria. For online implementation, we propose to play the games in a moving horizon fashion, and discuss two methods for guaranteeing feasibility of the resulting coupled repeated games. Finally, we study the performance of the proposed approaches in simulation for the miniature race car set-up. The simulation study shows that the presented games can successfully model different racing behaviors and generate interesting racing situations.

Zusammenfassung

Wir untersuchen autonomes rennfahren von Autos und entwickeln Regelalgorithmen die den Entscheidungsprozess eines menschlichen Rennfahrers ersetzen können. Die Aufgabe hat drei Hauptherausforderungen. Erstens muss der Regler fähig sein das Auto im Grenzbereich zu beherrschen. Zweitens, sollte das Auto immer sicher innerhalb der Strecke bleiben und letztlich sollte der Regler mit anderen Autos interagieren können und dynamisch auf das Fahrverhalten der anderen Autos reagieren. In dieser Doktorarbeit entwickelten wir einen Regelalgorithmus der alle diese Herausforderungen lösen kann und validierten den vorgestellten Ansatz mithilfe des Miniatur-Rennauto Versuchsaufbaus, des Institutes für Automatik an der ETH Zürich. Die Validierung wurde entweder auf dem experimentalen Versuchsaufbau durchgeführt oder in einer virtuellen Simulationsumgebung, die den experimentalen Versuchsaufbau wiederspiegelt.

In einem ersten Schritt präsentieren wir ein neuartiges Pfadplanungsmodel, wobei das Model auf einem dynamischen Einspurmodel mit nichtlinearen Reifenkräften basiert. Das vorgeschlagene Model kann ein umfangreiche Menge von Trajektorien generieren, welche die Fähigkeiten des dynamischen Einspurmodels gut approximieren. Das Pfadplanungsmodel ist optimiert für Entscheidungsprozesse, mit nur wenigen möglichen Entscheidungen und einer langen Diskretisierungszeit die es ermöglicht trotz eines kurzen Prädiktionshorizonten mit nur wenigen Zeitpunkten, Langzeit Entscheidungen zu treffen. Basierend auf diesem Pfadplanungsmodel, schlagen wir in einem zweiten Schritt einen hierarchischen Regler für autonomes Rennfahren vor, der in einer oberen Stufe die Stärken des Pfadplanungsmodels ausnutzt um eine Menge von prädiktiven Trajektorien zu generieren aus dieser Menge von Trajektorien wird die Trajektorie ausgewählt die den Fortschritt entlang der Strecke maximiert und innerhalb der Strecke bleibt. Diese Trajektorie wird dann, in der zweiten Stufe, von einem MPC Regler gefolgt, unter den Nebenbedingungen das die MPC Trajektorie das dynamische Einspurmodel erfüllt und innerhalb der Strecke bleibt.

Wir schlagen vor den Viabilitätskern (viability kernel) im Pfadplanungsschritt zu verwenden um zu garantieren das alle generierten prädiktiven Trajektorien rekursiv zulässig (recursive feasible) sind mit Respekt zu statischen Hindernissen (z.B. innerhalb der Strecke bleiben). Das garantiert, dass der Pfadplanungsschritt formell sicher ist und führt gleichzeitig dazu, dass die Rechenzeiten reduziert werden und verbessert das Verhalten des geschlossenen Regelkreises. Die Berechnung des Viabilitätskernes basiert auf der Diskretisierung des Zustandsraums. Um die Berechnung robust gegen diese Diskretisierungsfehler zu machen, schlagen wir einen neuen spieltheoretischen numerischen Algorithmus vor, welcher auf dem “discriminating kernel” Algorithmus basiert. Wir Zeigen, dass der resultierende Algorithmus eine innere Approximation des Viabilitätskernes generiert und garantiert, dass für alle Zustände in einer Zelle um einen “viable” Gitterpunkt auch eine Regelgrösse existiert, die das System innerhalb des Kerns behält. Die Leistungsfähigkeit des vorgeschlagenen hierarchischen Regler mit Viabiliätsnebenbedingungen wurde in einer Simulationsstudie untersucht, wo die Effekte von verschiedenen Designausführungen und Parametern bestimmt wurden, sowie in dem experimentellen autonomen Rennauto Versuchsaufbau. Beide die Simulations- und Experimentellen Resultate suggerieren, dass die konservativere Approximation basierend auf dem “discriminating kernel” zu einem sichereren Fahrstil führt, wobei sich die Rundenzeit nur leicht erhöht.

Der vorgeschlagene hierarchische Regler mit Viabilitätsnebenbedingungen, kann die ersten zwei Herausforderungen im autonomen Rennfahren lösen. Für die dritte Aufgabe, wo das Rennfahren gegen andere Autos von Interesse ist, schlagen wir eine Methode vor, die den Entscheidungsprozess als ein Kein-Nullsummenspiel formuliert. Genauer haben wir drei verschiedene Spiele hergeleitet, und in allen Spielen müssen die Spieler Nebenbedingungen erfüllen die Verlangen, dass die Autos innerhalb der Streck bleiben, sowie dass die Autos keine Kollisionen haben, was die Entscheidungen der zwei Spieler verbindet. Der Unterschied zwischen den Spielen sind die Kollisionsnebenbedingungen und der Gewinn der Spieler. Im ersten Spiel werden die Kollisionsnebenbedingungen nur vom Follower berücksichtigt und jeder Spieler maximiert sein Fortschritt in Richtung der Ziellinie. Durch die sequentielle Struktur im Spiel kann die optimale Lösung, durch einen sequentiellen Maximierungsansatz berechnet werden. Zusätzlich können wir zeigen, dass die gleichen Entscheidungen, wenn zulässig für die Nebenbedingungen, auch ein Stackelberg und Nash Equilibrium in reinen Strategien von unserem zweiten Spiel sind, wo beide Spieler die Kollisionsnebenbedingungen berücksichtigen. Der Gewinn der Spieler im dritten Spiel ist konstruiert um Verteidigungsverhalten zu belohnen, dass ist erreicht indem das Auto was am Ende des Horizontes in Führung ist einen zusätzlichen Bonus Gewinn bekommt. Wir zeigen, dass diese geänderte Gewinnstruktur die Stackelberg Equilibria ändert aber nur eine minimalen Einfluss auf die Nash Equilibria hat. Für eine Online Implementierung schlagen wir vor das Spiel in einer “Moving Horizon” Art zu Spielen, und wir präsentieren zwei Methoden die garantieren das alle Entscheidungen der verknüpften repetitiven Spielen zulässig mit den Nebenbedingungen sind. Schlussendlich, untersuchen wir die Leistungsfähigkeit der vorgeschlagenen Methoden in einer Simulationsstudie für den autonomen Rennauto Versuchsaufbau. Die Simulationsstudie zeigt, dass die präsentierten Spiele erfolgreich verschiedene Rennfahrer Verhalten modellieren kann, und interessante Rennsituationen generieren können.

Contents

Acknowledgments **i**

Abstract **iii**

Zusammenfassung **v**

Contents **vii**

1 Introduction **1**
- 1.1 Outline and contribution 2
  - 1.1.1 Experimental set-up 2
  - 1.1.2 Hierarchical racing controller 2
  - 1.1.3 Viability constraints 4
  - 1.1.4 Racing games 5
  - 1.1.5 Conclusion and Outlook 7
- 1.2 Publications 7

2 Experimental set-up **9**
- 2.1 Control loop 9
  - 2.1.1 Race track 9
  - 2.1.2 Miniature RC cars 9
  - 2.1.3 Vision based localization system 10
  - 2.1.4 Controller PC 11
  - 2.1.5 Control loop and system delays 11
- 2.2 Projects 12
  - 2.2.1 Model predictive contouring control 13
  - 2.2.2 Other projects 15

3 Vehicle model **17**
- 3.1 Bicycle model 17
  - 3.1.1 Constant velocity analysis 20
- 3.2 Path planning model 22

3.2.1 Hybrid system reformulation 24
3.2.2 Path planning model as an hybrid automaton 24
3.2.3 Reformulation as a difference equation 26
3.3 Comparison of models 27
3.3.1 Bicycle vs 4-wheel model 27
3.3.2 Bicycle vs path planning vs kinematic bicycle model 28
3.4 Conclusion 30
3.5 Appendix 30
3.5.1 System identification 30
3.5.2 Stationary velocity grid 34

4 Hierarchical Racing Controller 37
4.1 Path planning step 37
4.1.1 Implementation details 40
4.2 Tracking MPC step 42
4.3 Simulation results 45
4.4 Conclusion 47

5 Real-Time Control for Autonomous Racing Based on Viability Theory 49
5.1 Viability kernel and discriminating kernel 50
5.1.1 Viability kernel algorithm 51
5.1.2 Discriminating kernel algorithm 52
5.2 Robustifying the viability kernel algorithm against space discretization errors 53
5.2.1 Errors due to the space discretization 53
5.2.2 Inner approximation of viability kernel 54
5.2.3 Reconstructing viable controls 55
5.3 Upper-semicontinuity of the path planning model 56
5.4 Viability kernel for the track 58
5.4.1 Track constraints 58
5.4.2 Viability vs. discriminating kernel 58
5.5 Simulation results and sensitivity analysis 60
5.5.1 Viability constraints and grid size 61
5.5.2 Number of modes, N_m 63
5.5.3 Discretization time, T_{pp} 63
5.5.4 Number of constant velocity segments, N_S 64
5.5.5 Concluding remarks 64
5.5.6 Closed-loop behavior of $\mathcal{C}_c^v$ and $\mathcal{C}_c^d$ 65

5.5.7 Obstacle avoidance 67
5.6 Experimental results 68
5.7 Conclusion 69
5.8 Appendix 71
5.8.1 Finite inner approximation 71

6 A Non-Cooperative Game Approach to Autonomous Racing 79
6.1 Game ingredients 80
6.2 Game formulation 81
6.2.1 Sequential game 82
6.2.2 Cooperative game 84
6.2.3 Blocking game 85
6.3 Characterization of equilibria 86
6.3.1 Equilibrium concepts and feasibility 86
6.3.2 Equilibrium of the sequential game 88
6.3.3 Equilibrium of the cooperative game 89
6.3.4 Equilibrium of the blocking game 92
6.4 Moving horizon games 95
6.4.1 Terminal viability constraints 95
6.4.2 Soft constraints 96
6.5 Simulation results 97
6.5.1 Simulation set-up 97
6.5.2 Results 99
6.5.3 Computation time 102
6.6 Conclusion 103
6.7 Appendix 103
6.7.1 Terminal viability constraints 103
6.7.2 Soft constraints 105
6.7.3 Sequential iterative best response 106

7 Conclusion and Outlook 109

Bibliography 113

Curriculum Vitae 123

CHAPTER 1

Introduction

Control design for autonomous driving has attracted considerable attention from the research community and has been successfully demonstrated on several occasions, for example in the context of Autonomous Highway Systems (AHS) in 1997 [1] within the California PATH program, or in contests such as the DARPA Grand Challenge in 2005 [2] and the Urban Challenge in 2007 [3] by numerous research groups. Autonomous driving stayed an active research field ever since, see [4–6] for three recent surveys. In these projects, the main goal was to develop autonomous driving systems for public roads. In contrast to this stands autonomous racing, where the goal is to drive as fast as possible around a predefined track, which requires driving the car at the limit of the handling. Autonomous racing has attracted far less attention than autonomous driving nevertheless several approaches do exist for autonomous racing [7–9].

In this thesis, we mainly focus on optimization-based controllers for autonomous racing, more specifically we present a hierarchical racing controller. For the given controller we tackle two problems, which are also generally arising in optimization-based control for autonomous driving. The first problem is caused by the requirement for the car to stay on the track which is encoded through state constraints. However, without appropriate modifications, these constraints can lead to a loss of feasibility of the underlying optimization problem when the control algorithm is implemented in a receding horizon fashion, which may lead to accidents. In receding horizon control, recursive feasibility can be achieved by imposing terminal set constraints [10]. In autonomous driving, however, this issue is often neglected because terminal set constraints can be difficult to compute. One approach to incorporate terminal set constraints is by using the method proposed in [9], where the repetitive nature of autonomous racing is used to construct a terminal set from past observations. In this thesis, we propose an approach which uses viability theory to derive a safe set which is incorporated using viability constraints that guarantee recursive feasibility of the controller. The second problem we tackle is the interaction with other cars. This is a problem not limited to autonomous racing and is one of the biggest challenges for self-driving cars on public roads as well. The fundamental problem comes from the fact that the intentions of the other cars are not known.

Autonomous racing allows for a more controlled environment where the rival cars generate a dynamically changing environment. However, in racing the interactions between the vehicles are governed by less strict rules than on public roads; for example, there are no lanes and lane changes are not indicated. This implies that while in autonomous driving one can develop methods that assume a minimum degree of cooperation from neighboring vehicles [11–13], such an approach is harder for autonomous racing. Therefore, methods for autonomous racing often ignore obstacle avoidance altogether [7–9], or only consider the avoidance of static obstacles [14, 15]. In this thesis, we propose a game theoretical approach to deal with the uncertainty in the driving behavior of the opponent cars resulting in an autonomous racing game.

1.1 Outline and contribution

In the following, the outline of this thesis is presented, and the contributions are put into the context of the existing literature.

1.1.1 Experimental set-up

In Chapter 2 the experimental set-up is introduced. The set-up is used in subsequent chapters either directly to validate the controller or as a basis for our simulation environment. Furthermore, we introduced some additional work conducted on the experimental set-up during this thesis.

1.1.2 Hierarchical racing controller

In Chapters 3 and 4 we discuss the vehicle models used in this thesis. This includes a newly developed vehicle model which is tailored for path planning and decision making. In Chapter 4 the model is then incorporated into a hierarchical control structure wherein the higher level plans a finite horizon path using this model, and in the lower level a Model Predictive Controller (MPC) tracks the planned trajectory. In the autonomous driving literature, hierarchical control structures are a popular technique. Often the lower level incorporates a detailed model of the vehicle, but the higher level is based on simplified models. Using an MPC tracking controller in the lower level is common due to the systematic way constraints can be incorporated. The main difference between the different approaches is the way the nonlinear vehicle dynamics are considered in the MPC problem. The most popular techniques are to either directly use the nonlinear dynamics and rely on a nonlinear programming (NLP) solver [16], or to use approximation techniques, for example, linear time-varying (LTV) approximations [17] or piece-wise affine (PWA) approximations [18], resulting in a convex quadratic program (QP) or a mixed-integer QP, respectively. Our approach follows the same ideas and uses an LTV approximation, mainly due to computational benefits. Note that there exists a significant amount of non-MPC based low-level controllers, see for example [7,19,20], however, these methods have the drawback that state constraints are hard to incorporate.

When designing a hierarchical controller, one of the main challenges is the interplay between the two levels. The main problem is to guarantee that the decision of the higher level path planner can be executed by the lower level tracking controller. For example, in [21] a simple point mass model is used in the high-level path planner to limit the computational complexity. This, however, can be problematic if the generated trajectory is infeasible with respect to the car's physical dynamics, which can lead to preventable collisions [22]. The latter reference suggests to use a library of steady-state movements (trims) with parametrizable length, which are generated from a more complex dynamical model, and to connect them by maneuvers which allow the transition from one trim to another, similar to the idea of [23]. This path planning methodology has the advantage that it generates feasible reference trajectories for the low-level tracking controller. However, the resulting optimization problem is a mixed-integer program that is too complex to be solved within the typical sampling times. In [24], this problem is overcome by incorporating the lower level steering controller in the higher level path planner, which allows one to guarantee that the planned path is drivable by the car.

In our approach, the model used in the higher level path planner is designed such that the path generated is a good approximation for the vehicle model in the lower level controller. Our modeling approach is similar to the trim and maneuver based approach used in [22,23]. However, by simplifying the model and only considering trims and by modifying the task from a standard point-to-point path planning problem to a finite horizon optimal control problem with a well-chosen objective function the computation times can be reduced. These modifications allow implementing the higher level path planner in real-time, at least for short prediction horizons.

Alternatively to hierarchical controllers, to avoid non-driveable references by the path planner, one-level approaches have been investigated, for example, based on nonlinear optimization [8,14,21,25,26] or using rapidly exploring random trees (RRT) in [27,28]. However, in [21], where obstacle avoidance is incorporated by using a cost term that penalizes coming too close to the obstacle, the controller is reported to perform worse than its two-level counterpart, especially if the car has to diverge from the reference trajectory. The second disadvantage of these methods are often long computation times that prohibit real-time implementations. Exceptions include [8,14] where the NLP is approximated using the real-time iterations [29] to achieve low computation times needed for a real-time implementation.

1.1.3 Viability constraints

In Chapter 5 we show how viability theory can be used to guarantee recursive feasibility of the high-level path planner in the hierarchical controller presented in Chapter 4. Recursive feasibility is a well-known and well studied issue with receding horizon controllers [10, 30] and can be tackled by appropriately chosen terminal constraints. In autonomous driving, however, it is often neglected since it is hard to compute the necessary invariant sets. In this work, we use the particular structure given by an autonomous racing set-up to compute the viability kernel which, by enforcing the predictive path planner to stay inside, guarantees recursive feasibility.

Viability theory was developed to characterize states of a dynamical system for which solutions exist that satisfy its state constraints [31]. Though the use of viability theory in model predictive control has been considered [32], most applications of the theory limit themselves to establishing safe/viable regions in the state space [33–37] and reconstructing safe feedback controls [38]. At the heart of all these applications is the computation of the viability kernel, the largest subset of states for which the state constraints can be satisfied indefinitely. The viability kernel is typically approximated numerically using either the viability kernel algorithm introduced in [39], the discriminating kernel algorithm in the case of an additional disturbance input [40], or by exploiting the link to optimal control through viscosity solutions for Hamilton-Jacobi partial differential equations [41, 42]. All these numerical methods are based on gridding the state space, and hence their computational complexity grows exponentially in the dimension of the state space. To reduce the computational load the link to reachable set calculation has been exploited in [43], that allows for the development of efficient algorithms for linear systems [44–47]. Similarly, if the dynamical system is polynomial and the constraints are semi-algebraic sets, gridding can be avoided by using methods based on linear matrix inequality (LMI) hierarchies [48].

The contributions in this chapter are threefold: We show that apart from guaranteeing safety, resorting to viability constraints also reduces computation times, which enables the use of longer prediction horizons. A related approach was proposed in [49], where the authors exhaustively generate trajectories and then discard the infeasible ones a posteriori. Viability theory was also used in [35, 50] to construct safe trajectories and speed up trajectory planning processes, respectively. Similarly, viability and reachability analysis have also been used in autonomous driving to guarantee safety [11, 37, 51]. The main difference between those approaches and our method is that we use the viability kernel in a receding horizon fashion to generate viable trajectories only, which not only ensures recursive feasibility but also speeds up the computation.

The second contribution of this chapter concerns the computation of the viability kernel. In particular, we propose a method which takes into account the discretization error due to the gridding in the viability kernel algorithm. More specifically, we propose to model the discretization error as an additive uncertainty, and then formulate the viability computation as a dynamic game between the uncertainty and the control input. The victory domain of this game is then computed using the discriminating kernel algorithm [40]. This stands in contrast to other algorithms for computing the viability kernel, which mainly establish inner or outer approximations thereof. For example, the effects due to the discretization are considered in [39, 40] by "extending" the set-valued dynamics, which results in an outer approximation of the viability kernel; an error bound for this approximation is given in [52]. A different approach was proposed in [53], where interval analysis is used to find inner and outer approximations of the viability kernel. Compared to those methods our game theoretical approach, which is an inner approximation of the viability kernel, does not only guarantee viability of the grid points, but also viability of points "close" to viable grid points. This property is of special interest in real applications, where the state of a system rarely coincides with a grid point.

As a third contribution of this chapter, we examine the performance of the proposed controller in an extensive simulation study where we perform a detailed sensitivity analysis with respect to parameters of our viability-based controller. We demonstrate that the proposed controller scheme not only improves the overall performance of the race cars but also dramatically decreases the online computation time compared to the original controller presented in Chapter 4; the price to pay for this reduction in online computation is a significant increase in offline computations. Finally, we verify the controller's performance experimentally and demonstrate that our viability-based controller is indeed suited for autonomous racing.

1.1.4 Racing games

In Chapter 6 we introduce a game theoretical approach for dealing with the uncertainty in the driving behavior of the opponent cars in autonomous racing. The approach builds on the high-level path planner introduced in Chapter 4 by adding game theoretical decision making. The use of game theory is a popular method for dealing with uncertain actions of an opponent agent. For example, zero-sum game methods have been investigated for related problems in air traffic [54] and autonomous driving [55]. However, since such games assume the worst-case behavior of the opponent, special care would be needed to ensure that the resulting driving style is not too defensive for autonomous racing. Game theory has also been used to derive driver models to simulate and verify autonomous driving algorithms [56, 57]. Those approaches do not assume a worst-case behavior and are typically investigated by means of Stackelberg equilibria.

Adopting some of the assumptions of these game theory based driver models, we assume that in a race the competing car has no benefit from causing a collision. We propose to model the interactions between race cars as a non-cooperative non-zero-sum game, where the players only get rewarded if they do not cause a collision. By restricting our attention to finite horizon two-player games, we derive three different games that, under the additional hypothesis that the action set of the two players is finite are formulated as bimatrix games. In the first game, both players maximize their progress, but only the follower is concerned with avoiding collisions. For this game, we show that the Stackelberg and Nash equilibria can be computed by a sequential maximization approach, where the leader determines his trajectory independently of the follower, and the follower plays his best response. In the second game, both players consider collisions. This allows to prove that the players always choose a collision-free trajectory pair if possible. Further, we show that if the sequential maximization approach leads to a collision-free trajectory pair, this trajectory pair is also a Stackelberg and Nash equilibrium of the second game. In the third game, we propose a modified payoff that promotes blocking behavior, by rewarding actions that do not necessarily maximize progress towards the finishing line but instead aim at preventing the opponent from overtaking. For the third game, we show that the Stackelberg equilibrium is a blocking trajectory pair if one exists, which stands in contrast to the Nash equilibrium which only results in blocking trajectories in particular cases.

The approach proposed here is related to [58], where a similar game theoretic approach to autonomous racing is proposed. Also related is the approach proposed in [59], which formulates a zero-sum game for drone racing, which is played in a receding horizon fashion. However, in both [58] and [59] the equilibria of the games are not analyzed and only best-response dynamics are consider to solve the game. One could in principle also consider extending autonomous racing approaches with static obstacle avoidance, such as those developed in [14, 15], to incorporate dynamical obstacles by predicting the opponent movement and using time-varying constraints. This, however, would not allow the flexibility of our game theoretic approach, where both players influence each other's decisions.

For online implementation, we propose repeating the game in a receding horizon fashion, similar to MPC, giving rise to a sequence of coupled games. The use of moving horizon games has been proposed in other applications [60–62], as a method to tackle higher dimensional problems not tractable by dynamic programming. We investigate the sequence of coupled games and propose modified constraints based on viability theory which guarantee recursive feasibility and introduce an exact soft constraint reformulation that guarantees feasibility at all times. Compared to Chapter 5 where viability theory is used to guarantee recursive feasibility only with respect to track constraints, here recursive feasibility is additionally guaranteed with respect to the actions of the opponent player.

Finally, we investigate the game formulations in a simulation study, replicating the miniature race car set-up, and investigate the influence of the game formulation on different performance indicators such as the number of overtaking maneuvers and the collision probability.

1.1.5 Conclusion and Outlook

Concluding remarks and a brief summary of the thesis are given in Chapter 7, where we also provide an outlook to possible directions for future research.

1.2 Publications

The work presented in this thesis was obtained in close collaboration with colleagues, and is documented in submitted or published articles which are listed below:

- **A. Liniger**, A. Domahidi and M. Morari, "Optimization-Based Autonomous Racing of 1:43 Scale RC Cars", *Optimal Control Applications and Methods*, vol 36, pp 628–647, September 2015, [14].
- **A. Liniger** and J. Lygeros, "A viability approach for fast recursive feasible finite horizon path planning of autonomous RC cars", *Hybrid Systems: Computation and Control (HSCC)*, Seattle (USA), April 2015, [63].
- **A. Liniger** and J. Lygeros,, "Real-Time Control for Autonomous Racing Based on Viability Theory", *IEEE Transactions on Control System Technology*, [64].
- **A. Liniger** and J. Lygeros,, "A Non-Cooperative Game Approach to Autonomous Racing", *IEEE Transactions on Control System Technology*, Submitted, [65].

The following papers have been prepared during my doctorate. Most are related to autonomous racing, but in the interest of space their contributions have not been included in this thesis (listed in chronological order).

- M. Tranzatto, **A. Liniger**, S. Grammatico, and A. Landi, "The debut of Aeolus, the autonomous model sailboat of ETH Zurich", *OCEANS*, Genova (Italy), May, 2015, [66].
- J. Carrau*, **A. Liniger***, X. Zhang* and J. Lygeros, "Efficient Implementation of Randomized MPC for Miniature Race Cars", *European Control Conference (ECC)*, Aalborg (Denmark), June 2016, [67].
- **A. Liniger***,X. Zhang*, P. Aeschbach*, A. Georghiou and J. Lygeros, "Racing miniature cars: Enhancing performance using Stochastic MPC and disturbance feedback", *American Control Conference (ACC)*, Seattle (USA), May 2017, [68].

- X. Zhang*, **A. Liniger***, and F. Borrelli, “Optimization-Based Collision Avoidance”, *IEEE Transactions on Control System Technology*, Submitted, [69].
- T. Novi, **A. Liniger**, R. Capitani, M. Fainello, G. Danisi and C. Annicchiarico, “The influence of autonomous driving on passive vehicle dynamics”, *SAE World Congress Experience (WCX)*, Detroit (USA), 2018, [70].
- L. Hewing, **A. Liniger**, and M. Zeilinger, “Cautious NMPC with Gaussian Process Dynamics for Autonomous Miniature Race Cars”, *European Control Conference (ECC)*, Limassol (Cyprus), 2018, [71].
- T. Novi, **A. Liniger**, R. Capitani, M. Fainello, G. Danisi and C. Annicchiarico, “The influence of autonomous driving on passive vehicle dynamics”, *SAE International Journal of Vehicle Dynamics, Stability, and NVH*, accepted, 2018, [72].
- X. Zhang*, **A. Liniger***, A. Sakai and F. Borrelli, “Autonomous Parking using Optimization-Based Collision Avoidance”, *Conference on Decision and Control (CDC)*, Miami Beach (USA), accepted, [73].
- T. Novi, **A. Liniger**, R. Capitani and C. Annicchiarico, “Control of vehicle at at-limite on a predefined path”, *Vehicle System Dynamics*, submitted, [74].

*: These authors contributed equally to the respective works.

CHAPTER 2

Experimental set-up

The experimental set-up used in this thesis consists of remote controlled (RC) miniature race cars driven around a race track. In this chapter the set-up is introduced and discussed. Furthermore, additional projects conducted on the experimental set-up are briefly discussed. The set-up was build through a series of M.Sc. student projects, which are referred to throughout the discussion.

2.1 Control loop

The control loop shown in Figure 2.1 consists of four main parts, the race track, the miniature race cars, the vision system and the controller PC. These four parts are discussed next, and the section is finished with a short discussion on the control loop and the resulting delays and how they are handled.

2.1.1 Race track

The track is built on a 4x4 meter area, and consists of 13 turns with a resulting length of 18.43 meters. The track is 37 cm wide which is approximately six car widths, similar to the width of full-scale race tracks. The material used for the track is a fine-granulated rubber material which together with the used soft rubber tires results in a high level of traction. For more details on the race track layout and construction, we refer to [75].

2.1.2 Miniature RC cars

The used miniature race cars are Kyosho *dnano* 1:43 scale cars, which are about 12×5 cm in size and weigh 41 grams. The cars are rear wheel driven by a DC electric motor and have a proportional front wheel steering. The cars do not have separated breaks, but the motor can break the rear axle. Furthermore, the cars are equipped with front and rear suspension and a rear axle differential.

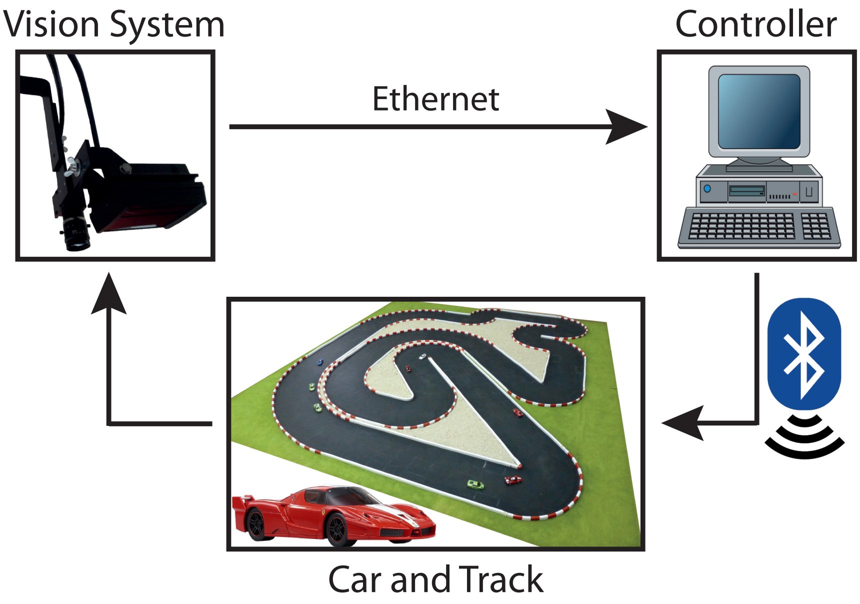

Figure 2.1: Schematic drawing of the control loop in the experimental set-up.

The original electronics of the cars are replaced with a custom electronics that allow a custom made communication protocol, onboard sensing, as well as low-level control loops. In particular the custom electronics feature a Bluetooth chip, current and voltage sensors, H-bridges for DC motor actuation, a steering angle sensor, a six degree of freedom internal measurement unit and an ARM Cortex M4 microcontroller. For the presented experiments, only the basic functionality of the board is used, mainly the Bluetooth communication link, as well as a steering angle controller. For more details on the custom electronics see [76, 77].

2.1.3 Vision based localization system

The localization of the cars, is based on an infrared vision-based system. Since the cars are moving in a plane one camera is enough to measure the position and orientation, this stands in contrast to commercial systems which are usually designed for 3D applications and use multiple cameras [78]. In the following, we briefly discuss the used camera and software implementation.

Camera

The camera used is a Point Grey Flea3 which runs at 100 frames per second. A wide angle objective is used to fit the whole track on the image, and an infrared filter is mounted in front of the camera to filter out unwanted wavelengths. Besides the camera, an infrared light source is mounted which emits light at the desired wavelength. The cars are marked with reflecting tape, which reflects the infrared light better than all the other materials used on the track. The camera is then connected via USB3 to a host PC which runs the vision software, see [79] for more details.

Vision software

The goal of the vision software is to extract the position and orientation of all the cars on the track and send this information to the controller PC. The software uses OpenCV [80] and can be separated into the following steps. First, the background is subtracted from the image, second, the marker placed on the cars are detected using a valley filling algorithm. Note that the markers on the cars are placed in predefined patterns, see Figure 2.2, so that in the next step one can identify the different cars based on the detected markers. From the identified cars it is possible to extract the position and the orientation. This information is further processed in a simple Extended Kalman Filter (EKF) using a kinematic model, which can additionally be used to estimate the yaw rate and velocity in the inertial frame. This information of all cars is then sent via Ethernet and TC/IP to all controller PCs. For more detail on the vision software we refer to [81], and for the state estimation and marker pattern design to [82].

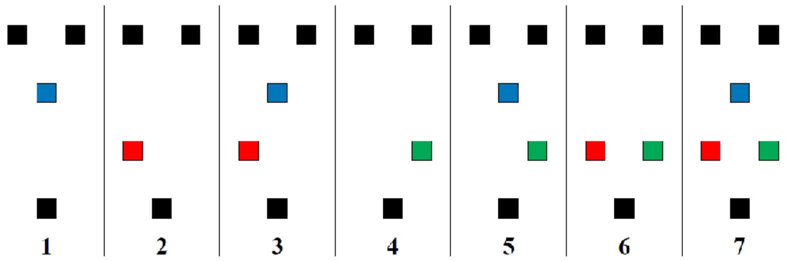

Figure 2.2: Schematic drawing of the seven different marker patterns used to identify the different cars [82].

2.1.4 Controller PC

The controller PC runs Ubuntu as an operating system with the control algorithm running in a soft-real-time system. The sampling time is restricted to a multiple of 10 ms, due to the update rate of the camera system.

2.1.5 Control loop and system delays

The resulting control loop can be summarized as follows; the controller sends commands to the car via Bluetooth which drives along the track, the camera then takes a new image which is transmitted via USB3 to the vision PC. The image is processed, and the new position of the car is detected and the velocities are estimated. This information is then sent via Ethernet to the controller PC which computes new commands for the car.

One fundamental problem of such complicated feedback loops are time delays. In our case the delays are caused by communication latencies in all three communication links, as well as computation delays in the vision software and the controller. To estimate the delay a simple experiment is conducted. An infrared LED is mounted on the car which can be remotely triggered by the Bluetooth link. Additionally, the vision software is modified to detect the blinking LED. For the loop delay measurement the vision PC sends a signal to the controller PC to trigger the LED and waits until the LED is detected; the elapsed time is then the loop delay. The left plot in Figure 2.3 shows the histogram of such an experiment. Notice that due to the sampling time of the camera system we only have a resolution of 10 ms in the measurement. It is clearly visible that the loop delay is in the order of 30 to 50 ms with most of the measurements between 30 and 40 ms. When investigating the delay further, we can split it into individual components. First, the runtime of the vision software can be measured and is around 4 ms, see the right plot in Figure 2.3 where the computation delay is subtracted. Furthermore, from separate roundtrip time measurements we know that the Bluetooth adds a delay of approximately 10 ms. Resulting in a remaining delay of around 20 ms caused by communication delays in the Ethernet network, USB3 communication and asynchronous effects between the vision and controller PC. Note that the measurements presented here do not include computation delays of the controller PC, which in our implementation adds one sampling period of delay.

To deal with these delays, we implemented an EKF with delay compensation, which compensates for the computation delay of the controller as well as the previously discussed loop delay. The EKF is running on the controller PC and is using the nonlinear bicycle model presented in Chapter 3. For simplicity, it is assumed that the delay is constant and a multiple of the sampling time. Given this assumption the best results are achieved with a loop delay of one sampling period of 20 ms plus 20 ms computation delay, see [83].

2.2 Projects

Besides the results presented in the rest of the thesis, the experimental set-up was also used in several other projects. The projects conducted can be roughly separated into two groups, the first group is based on the so called model predictive contouring control formulation, and the second group are projects using the experimental set-up for proof of concept studies of a “challenging task”.

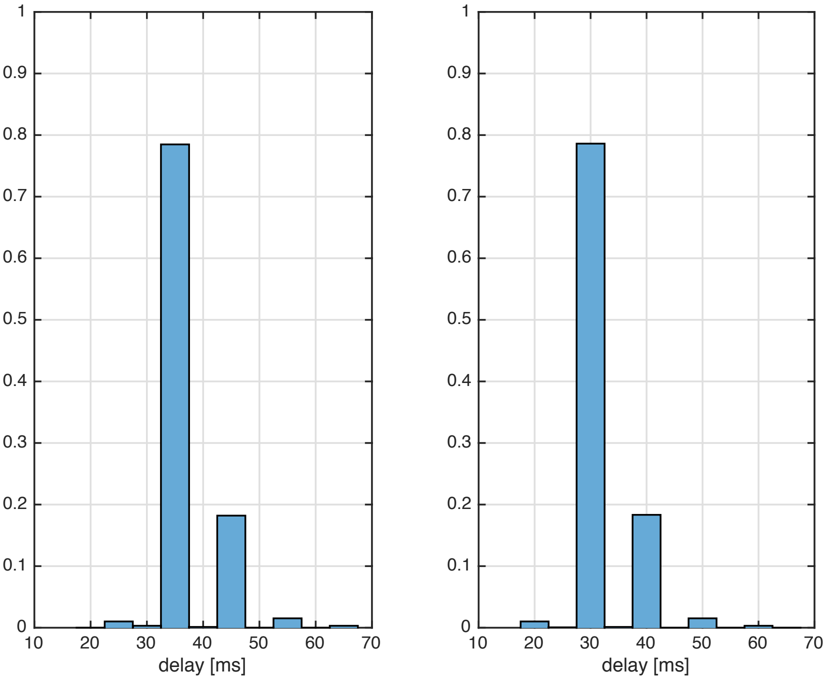

Figure 2.3: Histogram of the loop delay, in the left plot including the vision software computation time, and in the right plot without the computational delay of the vision system.

2.2.1 Model predictive contouring control

Model predictive contouring control (MPCC) presented in [84], is a path following control [85], where the goal is to follow a spatial path as fast as possible without a priory defining a velocity profile. The MPCC formulation was originally designed for machine tools, where the two main goals are to follow a path with an as small as possible contouring error while follow the path as fast as possible. This can be achieved by trading-off the tasks. The MPCC formulation can be used for autonomous racing by weighting the contouring task low and focusing on the task of following the path as fast as possible, which in terms is equivalent to maximizing the progress. Compared to the original formulation we additionally impose track constraints limiting the position of the car within the track, which is of particular importance if the contouring cost is chosen small. Note that the MPCC formulation has since been used for other robotics application such as control of drones [86] and for parallel autonomy for self-driving cars [87].

Real-time iteration vs nonlinear programming

The resulting MPCC problem is a Nonlinear Program (NLP) due to several reasons, (*i*) the cost is nonconvex, (*ii*) the dynamics are nonlinear, and (*iii*) the track constraints are nonconvex. Two approaches to tackle these problems are tested on the experimental set-up, where the main difference is how the nonlinear dynamics are handled. Both methods use a time-varying quadratic approximation of the cost function and local convex track constraints, where the local time-varying approximations are based on the last optimal solution. However, in the first approach, the nonlinear dynamics are handled by linearizing and discretizing the dynamics around the last optimal solution, which results combined with the other approximations in a quadratic program (QP) which can be efficiently solved using tailored solvers [88]. This approach of approximating the NLP as a QP is often referred to as real-time iteration [29] or linear-time-varying (LTV) MPC. The second method [89] does not approximate the nonlinear dynamics and directly solves the problem using an NLP solver [90]. Note that the cost function and track constraints are still approximated in the presented formulation, even though alternative nonconvex formulation would be possible. In summary, the LTV approximation has the advantage that the resulting QP can be solved more efficiently which allows implementing the MPCC on an ARM A15 microcontroller [14]. On the other hand the NLP implementation improves the driving performance and allows for additional non-convex constraints such as friction ellipse constraints of the rear wheel forces [89]. The NLP implementation of the MPCC was also successfully tested on the autonomous *Formula Student* race car developed by AMZ (Akademischer Motorsportverein Zürich), see `https://youtu.be/FbKLE7uar9Y`.

Obstacle avoidance

Both approaches allow obstacle avoidance, in the LTV scheme a high-level dynamic programming algorithm is used to find the “best” avoidance corridor, this new avoidance corridor is then used to alter the convex track constraint approximation [14]. In the NLP setting the same approach would be possible; alternatively, we tested to impose the obstacle avoidance constraints directly as ellipses which should be avoided [89].

Efficient randomized MPCC

One problem in MPC is that the model is typically uncertain. However, if the deterministic MPC problem is solved this knowledge is not considered which results in over-optimistic decisions. In [67] and [68], we investigated how the model uncertainties can be incorporating into the MPCC design using the so called scenario approach [91]. To implement the randomized MPCC in real-time a sparse formulation was derived which separates the problem in a pre-processing step, which computes a time-varying constraint tightening factor, and a second step which solves the MPCC LTV approximation including the scenario-based constraint tightening.

When incorporating the uncertainty in the MPC problem, it is usually required to include feedback policies to not end up with over-conservative results. Ideally the controller should optimize over the feedback policies, unfortunately this is a nonconvex problem. However, there are two popular approaches to achieve a tractable problem. First, pre-defined feedback policies, and second affine disturbance feedback (ADF). The first technique, including a heuristic mimicking the behavior of the state feedback controller was used in [67]. However, it can be challenging to tune the feedback controller. Thus we investigated if ADF can improve the performance of the controller as well as reduce the tuning effort, and showed that it indeed is beneficial for the performance, [68]. Since the computational complexity of the ADF approach does not allow for a direct real-time implementation, we further proposed a 3-sample heuristic to achieve an MPCC implementation with ADF, which can run in real-time while considering the uncertainty and optimize over the feedback policies.

In summary, the two sampling based MPCC implementations achieve a reduced number of constraint violation while at the same time achieving lower mean lap times.

Modified tasks

Based on the MPCC formulation we also tested modified tasks, such as driving backward [92], or driving with hard plastic tires with significantly lower traction [89].

2.2.2 Other projects

Besides projects using the MPCC formulation, the experimental set-up was also used as a proof of concept for other projects, either using the complete set-up including the miniature cars or only the feedback loop. For example, an approximate dynamic programming based reach-avoid controller was verified on the set-up in [93]. The set-up was also used to test if the miniature RC cars can be used to jump, and what is necessary to jump over a specially designed ramp automatically [94]. Finally, the existing feedback loop was used to test the MPC controller for the Autonomous Solar Vehicle developed at IfA, thus not using the miniature cars and the track, but only the positioning system and the communication links [95].

CHAPTER 3

Vehicle model

In this chapter, we derive the vehicle models used in this thesis, which are fundamental for the controller design. Furthermore, we compare the derived models with alternatives that can be found in the literature [96–98]. This chapter is organized as follows, first a bicycle model with nonlinear tire forces is derived and analyzed in Section 3.1. Based on the analysis of the bicycle model, in Section 3.2 a new model is derived which is based on constant velocity motion primitives, which is especially suited for path planning and therefore called “path planning model”. The model is formally defined in terms of a hybrid system with a slightly changed hybrid automaton, and further a discrete time system approximation is introduced. In Section 3.3, the bicycle and path planning model used in this thesis are compared with a 4-wheel model and a kinematic bicycle model. In the Appendix 3.5.1, the process to identify the bicycle model is briefly discussed, and in Appendix 3.5.2 the constant velocity motion primitives used within this thesis are described.

3.1 Bicycle model

The miniature cars are modeled using a bicycle model as done in [99,100], where the car is modeled as one rigid body with a mass m and an inertia I_z, and the four-wheeled car is reduced to a bicycle. At each of the two wheels of the resulting bicycle, a lateral and longitudinal tire force is acting, and the steering angle δ changes the angle of the front wheel.

To derive the differential equation of the car, we make the following simplifications: first we only consider in-plane motions, i.e. the pitch and roll dynamics as well as load changes are neglected. Second, as the cars used are rear wheel driven and do not have active brakes, the longitudinal force on the front wheel is neglected. The schematic drawing of the resulting model is shown in Figure 3.1 and the corresponding differential equations are given by,

$$\begin{aligned}
\dot{X} &= v_x \cos(\varphi) - v_y \sin(\varphi)\,, \\
\dot{Y} &= v_x \sin(\varphi) + v_y \cos(\varphi)\,, \\
\dot{\varphi} &= \omega\,, \\
\dot{v}_x &= \frac{1}{m}\left(F_{r,x} - F_{f,y} \sin\delta + m v_y \omega\right), \\
\dot{v}_y &= \frac{1}{m}\left(F_{r,y} + F_{f,y} \cos\delta - m v_x \omega\right), \\
\dot{\omega} &= \frac{1}{I_z}\left(F_{f,y} l_f \cos\delta - F_{r,y} l_r\right).
\end{aligned} \tag{3.1}$$

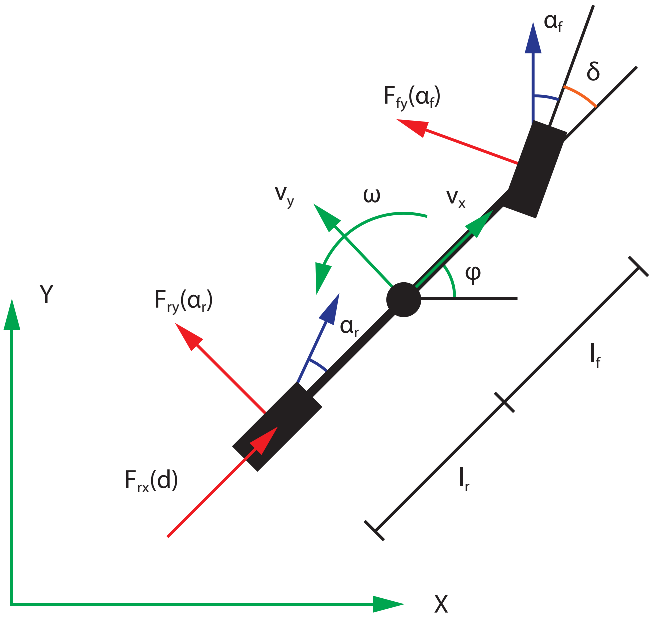

Figure 3.1: Schematic drawing of the car model.

The equation of motion is derived around the center of gravity (CoG), where the states are the position X, Y of the CoG in the inertial frame and the heading angle of the car relative to the inertial frame, φ. This is the kinematic part of the model, and just contains the integration of the velocities. The dynamic part of the model is derived around a body-fixed frame centered at the CoG, where the states are the longitudinal and lateral velocity of the car, v_x and v_y, and the yaw rate ω. The control inputs are the Pulse Width Modulation (PWM) duty cycle d of the electric drivetrain motor and the steering angle δ. The subscripts x and y indicate longitudinal and lateral forces or velocities, while r and f refer to the front and rear tires, respectively. Finally, l_f and l_r are the distance from the CoG to the front and the rear wheel. In the following the state is denoted by $\zeta = [X, Y, \varphi, v_x, v_y, \omega]$, and the inputs by $\nu = [d, \delta]$.

The two most important parts of the dynamics are the lateral tire forces $F_{f,y}$ and $F_{r,y}$ and the longitudinal force $F_{r,x}$. The lateral tire forces model the interaction between the car and the road and since the goal is for the cars to race, the model of the tire forces has to be realistic enough to represent the car at high speeds and its handling limits. Therefore, the lateral forces $F_{f,y}$ and $F_{r,y}$ are modeled using a simplified Pacejka tire model [101], see (3.2a) and (3.2b), which achieves a good approximation of the measured friction curves. The longitudinal force of the rear wheel $F_{r,x}$ is modeled using a motor model for the DC electric motor as well as a friction model for the rolling resistance and the drag, see (3.2c),

$$F_{f,y} = D_f \sin(C_f \arctan(B_f \alpha_f))\,, \quad \text{where} \quad \alpha_f = -\arctan\left(\frac{\omega l_f + v_y}{v_x}\right) + \delta\,, \tag{3.2a}$$

$$F_{r,y} = D_r \sin(C_r \arctan(B_r \alpha_r))\,, \quad \text{where} \quad \alpha_r = \arctan\left(\frac{\omega l_r - v_y}{v_x}\right)\,, \tag{3.2b}$$

$$F_{r,x} = (C_{m1} - C_{m2} v_x) d - C_{rr} - C_d v_x^2\,. \tag{3.2c}$$

The parameters $B_{f/r}$, $C_{f/r}$ and $D_{f/r}$ define the exact shape of the semi-empirical tire force curve, C_{m1} and C_{m2} define the motor model, C_{rr} the rolling resistance and C_d the drag. Notice that due to the size of the cars, it is currently not feasible to measure the wheel speed, which makes it impossible to use combined slip models such as done in [25, 99].

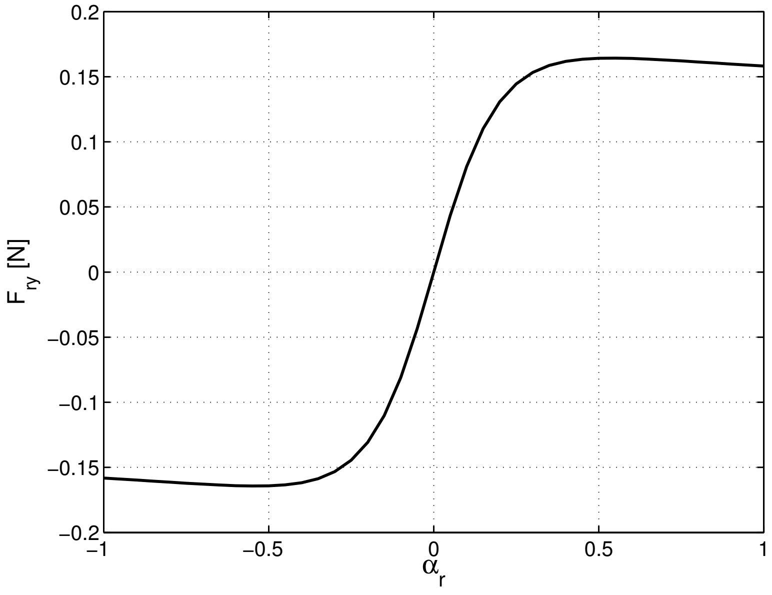

Figure 3.2: Shape of the simplified Pacejka tire model.

The model is identified using a combination of stationary and dynamic identification experiments, using the methodology reported in [100]. The idea is to identify the rear

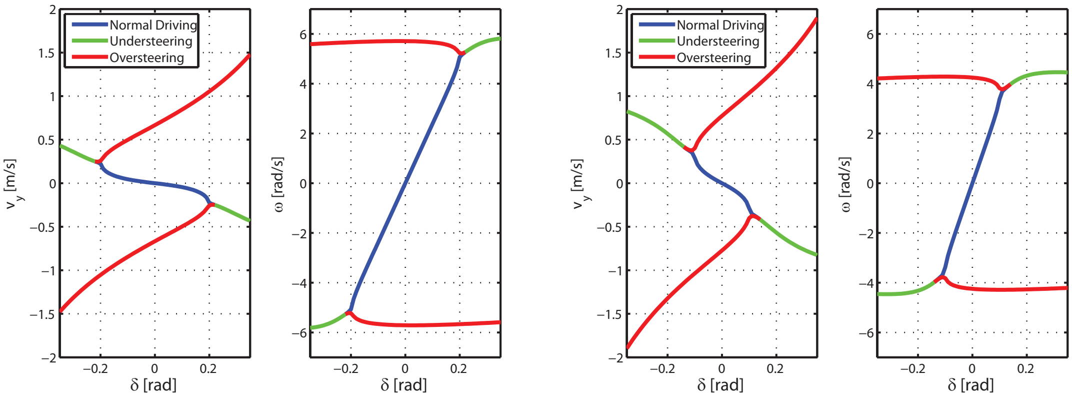

Figure 3.3: Stationary velocities for $\hat{v}_x = 1.5\,\mathrm{m/s}$ (plots on the left) and $\hat{v}_x = 2\,\mathrm{m/s}$ (plots on the right), parameterized by the steering angle $\hat{\delta}$.

wheel combined slip effects into the lateral tire friction model (3.2b), without considering them explicitly. Thus, the identified model is suitable to represent the car also at its handling limits, when the tire forces are saturated or close to saturation. The identification procedure is further discussed in Appendix 3.5.1.

3.1.1 Constant velocity analysis

For a better understanding of the model we investigate the points where all the velocities are constant. Note that a similar, more elaborate analysis is presented in [99, 100]. The objective of this analysis is to find points in the model where all accelerations are zero. This is done for different constant forward velocities $\hat{v}_x$ and constant steering angles $\hat{\delta}$, thus the problem becomes a nonlinear algebraic system of equations, with two equations ($\dot{v}_y = 0$, $\dot{\omega} = 0$) and two unknowns ($\hat{v}_y$, $\hat{\omega}$). By finding a solution to the algebraic equations for different steering angles $\hat{\delta}$ and different initial conditions, the constant velocities ($\hat{v}_x$, $\hat{v}_y$, $\hat{\omega}$) can be calculated.

The resulting constant velocities are shown for $\hat{v}_x = 1.5\,\mathrm{m/s}$ and $\hat{v}_x = 2\,\mathrm{m/s}$ in Figure 3.3 where the constant velocities are categorized into three different regions. The normal driving region highlighted in blue, which is characterized by the linear dependency of the yaw rate on the steering angle, as well as small lateral velocities. The other two regions in red and green correspond to over- and understeering points in the model. In the understeering region (green curves), the saturated front tire prevents the car from achieving larger yaw rates, and thus the car cannot take a sharper turn. In the oversteering region (red curves), the rear tire is fully saturated, and the car drives with a high lateral velocity which is usually called drifting.

The behavior of the model in the three regions is different, for example in the oversteering region where the car is drifting, the steering angle can be of opposite sign than the curvature the car is driving, known as countersteering, which does not occur in the normal and the understeering region. A stability analysis of the linearized lateral velocity model (v_y and ω), where the different stationary velocity points are used as linearization points, also shows the difference: While the linearized models in the normal driving and understeering regions are stable, the linearized models in the drifting region are unstable [100]. Thus, to track a constant velocity point in the drifting region requires active steering and throttle control [99, 100].

The second interesting outcome of this analysis can best be observed when looking at the constant velocity manifold, which can be computed by finding the constant velocities for the full range of longitudinal velocities, see Figure 3.4. From the constant velocity manifold, it is visible that the normal steering region is getting smaller when the car is driving faster. Thus the maximal and minimal steering angle for which the tire forces are not saturated is reduced with increased forward velocity and at the same time the maximal yaw rate decreases.

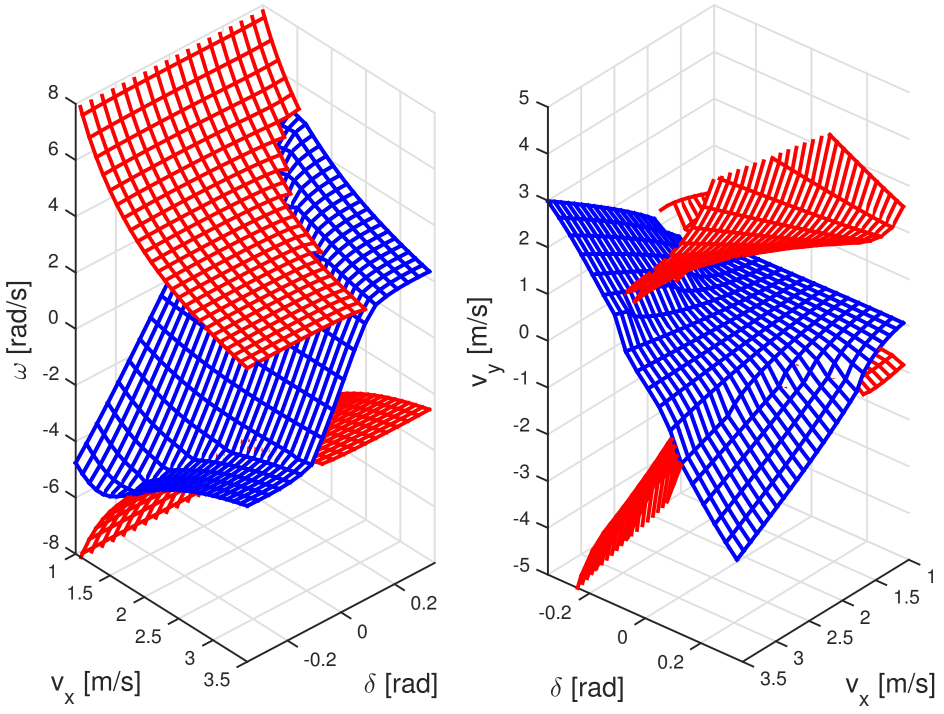

Figure 3.4: Stationary velocity manifold of the bicycle model. Rendered in red the parts corresponding to oversteering, in blue neutral and understeering.

When all velocities are constant, the movement of the car is a uniform circular movement, for which the relationship between the yaw rate, the radius R and the curvature

κ is known,

$$|\omega| = \frac{v}{R} = v\kappa \quad \text{where} \quad v = \sqrt{v_x^2 + v_y^2}\,.$$

Consequently, all stationary velocity points correspond to the car driving in a circle with a constant radius, i.e., the model can predict drifting along a circle. This insight motivates the derivation of the path planning model, introduced next.

3.2 Path planning model

Since the bicycle model (3.1) is complex and has fast-changing dynamics, we propose a new model especially suited for path and motion planning. The proposed model simplification is motivated by the time scale separation present in the system, using the fact that the system's velocity changes significantly faster than its position and orientation. A similar approach was adopted in [33,36,102] to reduce the number of states of the model. The core idea is that based on the constant velocity analysis, we know the "steady state" velocities. Thus, by considering a finite number of constant velocity points, we can formulate the "path planning model" which only considers trajectories corresponding to driving at these constant velocity points for a fix duration T_{pp}. In vehicle dynamics, such trajectories are known as *steady state cornering*, whereas in aeronautics they are referred to as *trimmed flight*. The trajectory segments are then directly linked under the assumption that the new constant velocity can be reached immediately. To ensure that these velocity discontinuities are "realistic" constraints are imposed on the segments that can be concatenated. Therefore, we only allow for transitions that are achievable by the nonlinear dynamical model (v_x, v_y and ω) in a certain small time step $T_t \approx T_{pp}/2$. The problem can be posed as a feasibility problem (3.3) where the goal is to find controls which allow a transition from a constant velocity point i defined as $\hat{v}(i) = (\hat{v}_x(i), \hat{v}_y(i), \hat{\omega}(i))$ to another point j in at most T_t seconds, subject to the vehicle dynamics and constraints on the control inputs,

$$\begin{aligned}
\text{find} \quad & d, \delta, t_f\,, \\
\text{s.t.} \quad & v(0) \;= (\hat{v}_x(i), \hat{v}_y(i), \hat{\omega}(i)) = \hat{v}(i)\,, \\
& v(t_f) = (\hat{v}_x(j), \hat{v}_y(j), \hat{\omega}(j)) = \hat{v}(j)\,, \\
& \dot{v}(t) = \begin{cases} \dot{v}_x & = \frac{1}{m}(F_{r,x} - F_{f,y}\sin\delta + m v_y \omega)\,, \\ \dot{v}_y & = \frac{1}{m}(F_{r,y} + F_{f,y}\cos\delta - m v_x \omega)\,, \\ \dot{\omega} & = \frac{1}{I_z}(F_{f,y} l_f \cos\delta - F_{r,y} l_r)\,, \end{cases} \\
& t_f \in [0, T_t]\,, \\
& d \in [\underline{d}, \bar{d}] \quad \delta \in [\underline{\delta}, \bar{\delta}]\,.
\end{aligned} \tag{3.3}$$

Here, $\underline{d}$ and $\bar{d}$ are the lower and upper bound on the duty cycle d and similarly $\underline{\delta}$ and $\bar{\delta}$ the bounds on the steering angle δ. This problem allows classifying all the transitions from one stationary point to another as admissible or inadmissible transitions. An illustrative example of such allowed transitions is shown in Figure 3.5, with nine constant velocity points for three grid points in v_x and δ. The allowed transitions from one constant velocity point $\bar{v}(4)$ are shown by arrows.

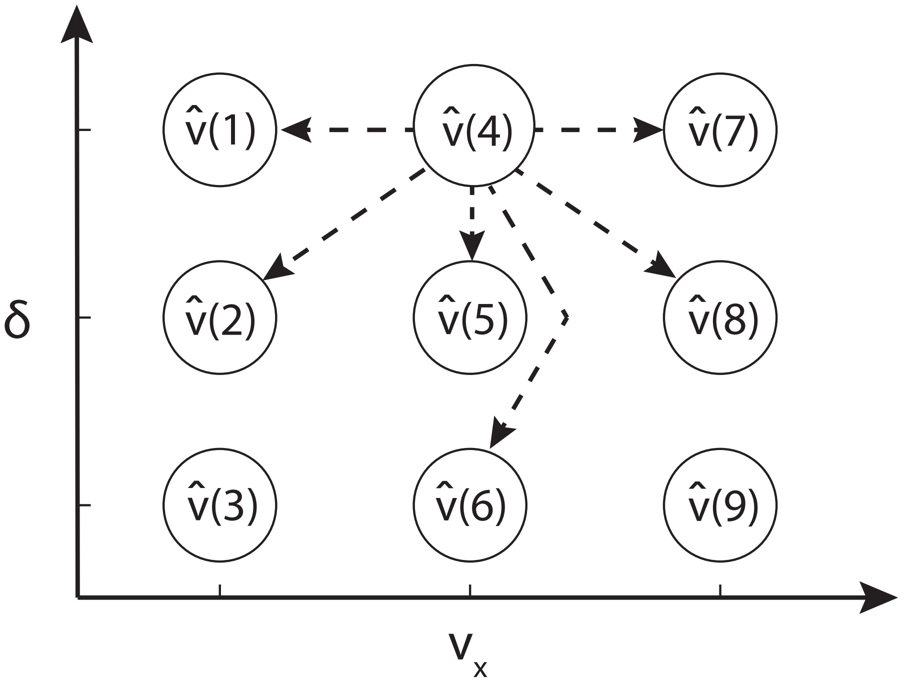

Figure 3.5: Schematic drawing of allowed transitions from $\hat{v}(4)$ to other constant velocity points.

In practice, problem (3.3) is solved using the backward reachability algorithm of the multi-parametric toolbox 3.0 [103] using a model linearized around the target velocity and bounds on the controls. The allowed time to reach the new stationary velocity is $T_t \approx 0.1$s.

Note that in robotics a similar model was derived in [23] and applied to autonomously driving cars in [22]. However, our proposed model is different since the different trims are not linked with pre-computed maneuvers, but we propose to directly link trims and limit the concatenation of trims. In our approach, this leads to the fact that each segment has the same duration T_{pp}, which simplifies the following analysis.

The constant velocity points used in the path planning model are obtained by gridding the constant velocity manifold (Figure 3.4), resulting in a grid of N_m constant velocity points. While most of the points are gridded within the normal steering region, several additional points from the oversteering region are also considered, to enrich the approximation of the model by incorporating drifting trajectories into the set of motion primitives. This allows, if the model is used for path planning, to consider drifting trajectories if this is beneficial. For more details on the different grids used within this thesis, see Appendix 3.5.2.

3.2.1 Hybrid system reformulation

The proposed path planning model is similar to a control system with zero order hold and quantized control inputs, where additionally the controls are restricted based on the last applied control input. Thus, the path planning fits well into a hybrid model framework.

3.2.2 Path planning model as an hybrid automaton

A hybrid model is characterized by its interplay between continuous and discrete states, where the continuous state changes with respect to a dynamic system, and the changes in the discrete state happen at specific events and are determined by some automaton. In the path planning model, the discrete mode q of the system is the current constant velocity and the continuous evolution is given by the kinematic model with the corresponding constant velocity point. Furthermore, a time state is needed to ensure that at every T_{pp} seconds a jump to a different discrete mode/constant velocity is possible. Thus the continuous evolution is given by,

$$\dot{X} = \hat{v}_x(q)\cos(\varphi) - \hat{v}_y(q)\sin(\varphi)\,, \tag{3.4a}$$

$$\dot{Y} = \hat{v}_x(q)\sin(\varphi) + \hat{v}_y(q)\cos(\varphi)\,, \tag{3.4b}$$

$$\dot{\varphi} = \hat{\omega}(q)\,, \tag{3.4c}$$

$$\dot{T} = 1\,, \tag{3.4d}$$

where $\hat{v}_x(q), \hat{v}_y(q), \hat{\omega}(q)$ are the constant velocities at the discrete mode q. The discrete transition takes place every T_{pp} seconds, and a discrete control input determines the discrete mode after the transition. To capture this behavior we consider the following modification of the hybrid automaton considered in [104].

Definition 3.1. *A* ***hybrid automaton*** *H is a collection $H = (M, Z, \Upsilon, \eta, Dom, \mathcal{E}, \mathcal{G}, \rho)$ comprising*

- *discrete state variables $\mu \in M$,*
- *continuous state variables $z \in Z$,*
- *discrete control inputs $\upsilon \in \Upsilon$,*
- *vector field $\eta(\cdot,\cdot) : M \times Z \to Z$,*
- *domain set, $Dom(\cdot) : M \to 2^Z$,*

- *edges*, $\mathcal{E} \subset M \times M$,
- *guard*, $\mathcal{G}(\cdot,\cdot) : \mathcal{E} \times \Upsilon \to 2^Z$,
- *reset map*, $\rho(\cdot,\cdot) : \mathcal{E} \times Z \to 2^Z$.

The modifications are kept as small as possible to keep the definition as general as possible. The main difference to the definitions of [104, 105] is that a discrete input appears in the discrete transition guards. To avoid technical difficulties with the definition of executions of the hybrid automaton given below we introduce the following assumption (adapted from [104]).

Assumption 3.1. *The cardinality of* M *is finite.* $Z = \mathbb{R}^{n_z}$, *for some* $n_z \geq 0$. *For all* $\mu \in M$, *the vector field* $\eta(z,\mu)$ *is globally Lipschitz continuous in* z. *For all* $e \in \mathcal{E}$, *there exist* υ *such that* $\mathcal{G}(e,\upsilon) \neq \emptyset$, *and for all,* $z \in \mathcal{G}(e,\upsilon)$, $\rho(e,z) \neq \emptyset$.

Definition 3.2 ([105]). *A* ***hybrid time*** *set* $\tau = \{I_i\}_{i=0}^N$ *is a finite or infinite sequence of intervals of the real line, such that*

- $I_i = [\tau_i, \tau_i']$, *for all* $i < N$,
- *if* $N < \infty$, *then either* $I_N = \{\tau_N, \tau_N'\}$ *or* $I_N = \{\tau_N, \tau_N')$,
- $\tau_i \leq \tau_i' = \tau_{i+1}$ *for all* i.

We are now in a position to define the execution that the hybrid automaton H accepts. The following definition is adapted from [105].

Definition 3.3 ([105]). *Let* $\tau = \{I_i\}_{i=0}^N$ *be a hybrid time set and consider the sequence of functions* $\{\mu_i(\cdot)\}_{i=0}^N$, $\{z_i(\cdot)\}_{i=0}^N$, $\{\upsilon(\cdot)\}_{i=0}^N$, *with* $\mu_i(\cdot) : I_i \to M$, $z_i(\cdot) : I_i \to Z$, $\upsilon(\cdot) : I_i \to \Upsilon$. *The collection of these sequences of functions is called* ***execution of the hybrid automaton*** H *starting from initial condition* $(\mu_0(\tau_0), z_0(\tau_0))$, *if and only if it satisfies the following conditions:*

- Discrete evolution: *For all* $i < N$,
  1. $(\mu_i(\tau_i'), \mu_{i+1}(\tau_{i+1})) \in \mathcal{E}$,
  2. $z_i(\tau_i') \in \mathcal{G}(\mu_i(\tau_i'), \mu_{i+1}(\tau_{i+1}), \upsilon_{i+1}(\tau_{i+1}))$,
  3. $z_{i+1}(\tau_{i+1}) = \rho(\mu_i(\tau_i'), \mu_{i+1}(\tau_{i+1}), z_i(\tau_i'))$.
- Continuous evolution: *For all* i *with* $\tau_i < \tau_i'$,
  1. $(\mu_i(t) = \mu_i(\tau_i))$ *and* $(\upsilon(t) = \upsilon(\tau_i))$ *for all* $t \in I_i$,
  2. $z_i(\cdot) : I_i \to Z$ *is the solution of the differential equation*

$$\dot{z}_i(t) = \eta(\mu_i(t), z_i(t)),$$

  over the interval I_i *with the initial condition* $z_i(\tau_i)$, *3.* $z_i(t) \in Dom(\mu_i(t))$ *for all* $t \in [\tau_i, \tau_i')$.

The main difference to the corresponding definition of [105] is point 2 in the discrete evolution: The guard depends on $\upsilon_{i+1}(\tau_{i+1})$ which allows to determine the discrete state after a transition by choosing an appropriate discrete control input υ_{i+1}.

The path planner outlined above can be captured by a hybrid automaton H with,

- $M = \{1, 2, 3, \ldots, N_m\}$,
- $z = (X, Y, \varphi, T) \in \mathbb{R}^4 = Z$,
- $\Upsilon \subset \mathbb{Z}$,
- $\eta(\mu, z)$ given in Eq (3.4),
- $\mathrm{Dom}(\mu) = \{(X, Y, \varphi, T) \in \mathbb{R}^4 | T \leq T_{pp}\}$,
- $\mathcal{E}$, all edges where a transition is possible,
- special guard, with two input arguments

$$\mathcal{G}(e, \upsilon) = \begin{cases} \{z \in \mathbb{R}^4 | T \geq T_{pp}\} & \text{if } \mathcal{G}(\mu_i, \mu_{i+1}, \upsilon) \neq \emptyset \\ \emptyset & \text{else} \end{cases},$$

- $\rho(e, (X, Y, \varphi, T)) = \{(\hat{X}, \hat{Y}, \hat{\varphi}, \hat{T}) \in \mathbb{R}^4 |$
  $\hat{X} = X, \hat{Y} = Y, \hat{\varphi} = \varphi, \hat{T} = 0\}$.

Note that the constant velocities and the allowed transitions can be always designed such that the path planner automaton satisfies Assumption 3.1. The Lipschitz condition is easy to verify as the vector field in (3.4) is differentiable with bounded derivatives.

3.2.3 Reformulation as a difference equation

From a computational point of view, hybrid systems can be challenging to tackle, due to both the continuous and discrete state and their potentially complex interactions. For example, if a gridding based algorithm is used for the proposed path planning model, it is required to grid the four-dimensional continuous space for each of the N_m discrete modes, a task that is computationally very demanding [105].

However, since the discrete transitions occur at fixed time intervals, the system can be interpreted as a sampled data system, with constant inputs over the time interval T_{pp}. Therefore, the system can be approximated by a discrete-time system with a sampling time of T_{pp}. The discrete state q of the hybrid system is considered by embedding it in the real numbers; hence the discrete dynamics can be encoded as a transition relation that depends on the current mode. This renders the time state redundant and reduces the state space dimension by one.

We denote the state of the difference equation at time step k by $x_k = [X_k, Y_k, \varphi_k, q_k]$, and the control by u_k, which determines the next mode $q_{k+1} = u_k$. As the control input depends on the current mode q_k, we abstractly write $u_k \in U(q_k)$, where $U(q_k)$ encodes the concatenation constraints in terms of an automaton, which given q_k returns all allowed next modes q_{k+1}, or based on the guard of the hybrid system, $U(q_k) = \{u_k \in M | \ \exists \upsilon \in \Upsilon \ \exists \mathcal{G}(q_k, u_k, \upsilon) \neq \emptyset\}$. Let $X(x_k, u_k, t)$, $Y(x_k, u_k, t)$, $\varphi(x_k, u_k, t)$ denote the solution of (3.4a)-(3.4c) at time t, with $q = u_k$ while starting at $x_k = [X_k, Y_k, \varphi_k, q_k]$. Note that, since the velocities are constant the solution can be explicitly computed as a function of time. Thus, we can define the path planner as the following discrete-time model,

$$x_{k+1} = f(x_k, u_k) = \begin{bmatrix} X(x_k, u_k, T_{pp}) \\ Y(x_k, u_k, T_{pp}) \\ \varphi(x_k, u_k, T_{pp}) \\ u_k \end{bmatrix}, \tag{3.5}$$

where T_{pp} is the path planning discretization time.

3.3 Comparison of models

To evaluate the two proposed models, we compare them with two alternative models, a 4-wheel vehicle model including static load changes [98] and a kinematic bicycle model [106]. Compared to the bicycle model the 4-wheel model includes all four wheels as well as the varying loads on the wheels depending on the driving situation. This should result in a more accurate representation of the forces and moments. The kinematic model, on the other hand, assumes that the lateral velocity at the two wheels of the bicycle are zero which allows eliminating the states v_y and ω.

For our comparison we look at a simple scenario where the car is driving straight with a fixed velocity, and at time zero we instantaneously change both control inputs (d and δ) and keep them constant for 0.1 s. The controls are determined by gridding the two inputs with eleven points in each direction, uniformly spaced between the input bounds. For the path planning model, we investigate the trajectories for all allowed modes.

3.3.1 Bicycle vs 4-wheel model

When looking at the predicted position of the bicycle and 4-wheel model after 0.1 s there is nearly no difference between the two models, even at high forward velocities of 3 m/s, where the differences should be the largest, see Figure 3.6.

From a vehicle dynamics point of view, this makes sense since the used 4-wheel model neglects combined slip effects as well as the stiffness of the tires. Under the assumption that the stiffness of the tires can be neglected, which is the case for the miniature cars, the lumping of the left and right wheels of the bicycle model only causes a negligible error. Thus, the main difference between the two models is the longitudinal load change

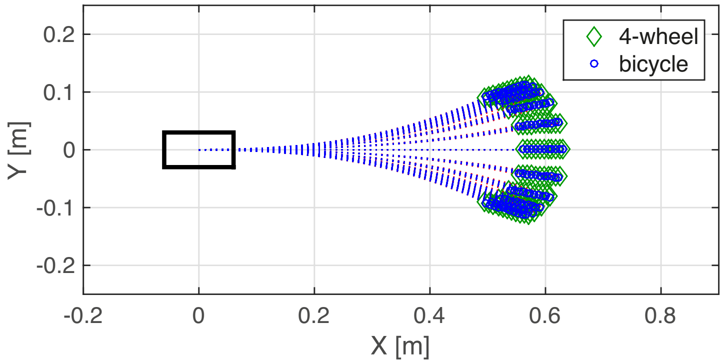

Figure 3.6: Trajectories of the bicycle and the 4-wheel model with an initial forward velocity of 3 m/s.

which is also negligible due to the low center of gravity of the car. This observation can also be verified by the good cross validation results of the identified bicycle model compared to the real car in Figure 3.10 in the Appendix 3.5.1.

3.3.2 Bicycle vs path planning vs kinematic bicycle model

From the previous comparison we know that the bicycle model compares well against more sophisticated vehicle models. However, we should still investigate the path planning model. Therefore, we compare the path planning model with the dynamic bicycle model and the kinematic bicycle model. The kinematic bicycle model is an attractive approach to model cars in slow driving situations. Since the no-slip assumption is fulfilled, it is not necessary to identify the tires, and the model has only four states. In Figure 3.7 the trajectories of the three models for different initial forward velocities of 0.6 m/s (top left), 1 m/s (top right), 2 m/s (bottom left), and 3 m/s (bottom right) are shown. From Figure 3.7 we can see that, as expected, the kinematic model is a good approximation at low velocities of around 0.6 m/s. On the other hand, at higher velocities the no-slip assumption leads to an overestimation of the turning rate. Thus, the kinematic model is not suited for our application where the goal is high-speed driving at the limit. On the other hand, the path planning model is significantly better in estimating the driveable envelope of the bicycle model. For low forward velocities the path planning model slightly over approximates the driveable envelope, and for higher forward velocities the path planning model is a conservative approximation of the driveable envelope. Furthermore, using more constant velocity points should improve the approximation.

Similarly, the good approximation of the path planning model can also be observed in the velocity states. Therefore, in Figure 3.8 the velocities of the bicycle model after 0.1 s are compared to the maximal constant lateral velocity and yaw rate in the normal driving region (bifraction point in Figure 3.3). On the left in Figure 3.8 the case for an

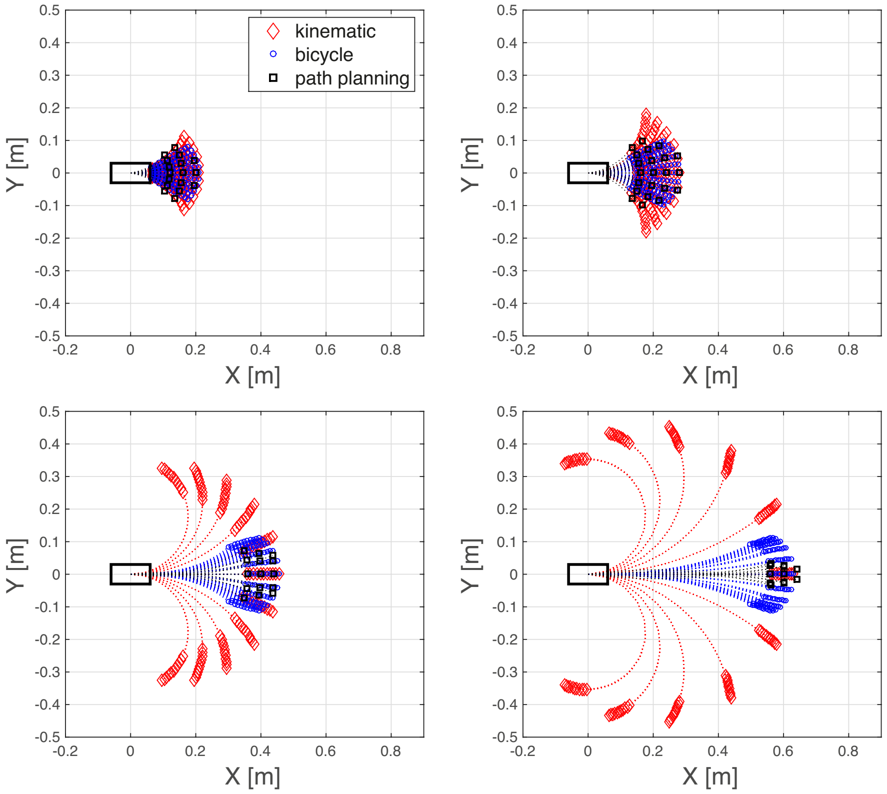

Figure 3.7: Trajectories of the bicycle and kinematic model compared to the proposed path planning model with $N_m = 129$ at different initial forward velocity, 0.6 m/s (top left), 1 m/s (top right), 2 m/s (bottom left), and 3 m/s (bottom right).

initial velocity of 1 m/s is shown and on the right for an initial velocity of 2 m/s. It can be seen that for an initial velocity of 1 m/s, the constant velocity points approximate the bicycle model well, but for an initial velocity of 2 m/s the constant velocities slightly underestimate the dynamic capabilities of the bicycle model.

Therefore, we can conclude that the dynamic bicycle model performs equally well as more complex vehicle models. Thus we claim that it is also a good approximation of the miniatures cars. This claim is also backed up by the good cross validation results presented in the Appendix 3.5.1. Furthermore, the path planning model approximates the dynamic bicycle model well, even at high forward velocities. The slight underestimation at high velocities can be even seen as a feature since these are states where the model is most uncertain. Thus, both models should be good in approximating the real *dnano* cars over a wide range of operation points.

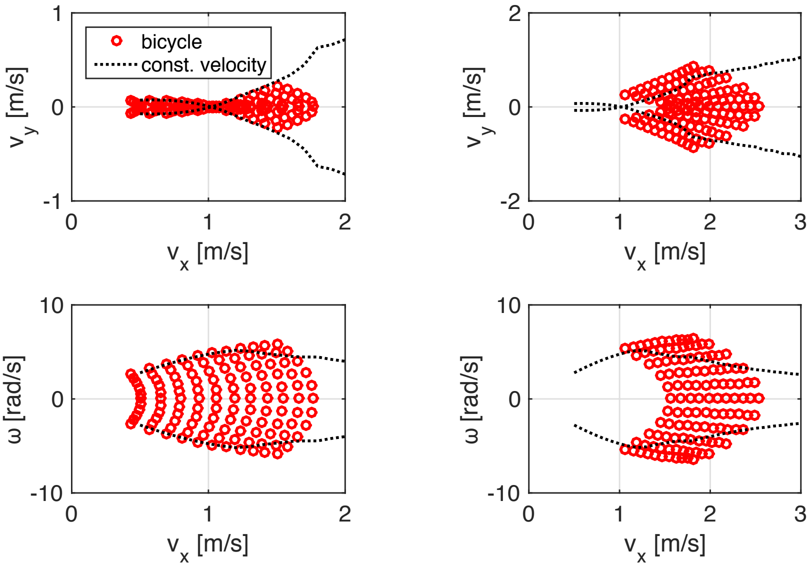

Figure 3.8: Velocities after 0.1 s for different inputs, for an initial velocity of 1 m/s on the left and 2 m/s on the right, as well as the maximal constant velocity $\bar{v}_y$ and $\bar{\omega}$ within the normal driving region.

3.4 Conclusion

In this chapter we introduced the two vehicle models used in this thesis: first a bicycle model with nonlinear tire forces which is popular in the literature, second a newly derived model based on constant velocity motion primitives which we formally modeled as a hybrid system using a slightly modified hybrid automaton definition. Furthermore, the path planning model is approximated as a discrete time system which gives computational advantages and a more condensed system description. Finally we compared the two proposed models with other popular vehicle models and showed that the bicycle and path planning model are good approximations. Therefore, the two models are well suited to derive an autonomous racing controller.

3.5 Appendix

3.5.1 System identification

In this section, the technique used to identify the bicycle model is briefly summarized. For more details, the interested reader is referred to [100] for the original methodology and [107] for a more in-depth discussion.

The approach is based on a three step procedure, where in a first step static parameters such as the mass are determined. Secondly, ramp steer experiments are performed to obtain a first guess of the tire force laws. Finally, the tire forces and the drivetrain model parameters are identified by fitting the model response to measurements in the time-domain. Note that the identification of the miniature race car has several challenges, first the goal is to identify a model which also represents the car at the handling limit, second, there is limited information, especially there is no wheel speed and no body roll and pitch information. Finally, it is only possible to identify the tires using driving experiments, since no tire measurement test-benches exist.

Static parameters

Since Kyosho does not publish exact vehicle data, i.e., mass, center of gravity, moment of inertia and steering limits, these parameters have to be measured or identified. The mass can be measured using a scale, and the center of gravity can be determined by finding the balancing point of the car in longitudinal direction, in lateral direction we assume the center of gravity is in the middle of the car. The moment of inertia was determined by a CAD model, using an assembly of the four main parts of the car (main chassis, battery, rear axle and hull). Finally, the steering angle limit can be determined by the turning radius. We found the following parameters for the Lamborghini Murcielago model,

- mass: $m = 41$ g
- CoG: $l_f = 29$ mm / $l_r = 33$ mm
- moment of inertia: $I_z = 27.8 \cdot 10^{-6}$ kg m^2
- steering limit: $\bar{\delta} = 0.35$ rad.

Note that these parameters are only valid for the Lamborghini Murcielago, which has the widest track and longest wheelbase of all *dnano* cars.

Steady state identification

An initial guess of the tire friction parameters can be obtained using the assumption that the velocities are in steady-state, or in other words $\dot{v}_y = 0$ and $\dot{\omega} = 0$. Thus, based on (3.1) we get the following system of nonlinear equations describing the lateral forces,

$$\begin{aligned} F_{r,y} &= \frac{m l_f}{l_f + l_r} \, v_x \omega \ , \\ F_{f,y} &= \frac{m l_r}{l_f + l_r} \, \frac{v_x \omega}{\cos(\delta)} \, . \end{aligned} \tag{3.6}$$

Therefore, the lateral forces are completely defined given the velocities (v_x, v_y, ω) and the steering angle δ. However, only few type of maneuvers fulfill the assumption. The simplest are steady state experiments, where the inputs are fixed and the velocities are in steady state, but such experiments can be very time consuming. An alternative are ramp steer inputs with a constant velocity, which result in low accelerations. In the case of the miniature RC cars the steering angle is changed as slowly as possible, while guaranteeing that the car stays within the given training area of about 3x3 m, and the duty cycle is kept constant to guarantee as little interference of the drivetrain as possible. These experiments are repeated for different duty cycles and for left and right turns.

Thus, given the velocities and the steering angle of all the experiments, it is possible to compute α_r, α_f, $F_{r,y}$ and $F_{f,y}$ for each time point. The result is a point cloud of forces at the corresponding slip angles. Given this data a nonlinear least-square problem can be solved where the parameters of the Pacejka tire model are optimized to minimize the 2-norm between the forces computed using (3.6) and the tire model.

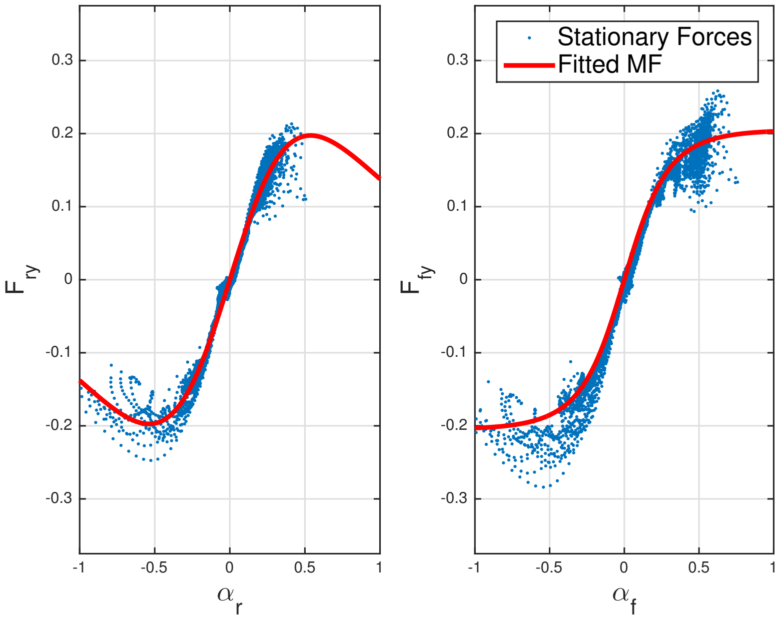

Figure 3.9: Steady-state tire force identification

In Figure 3.9 the forces computed using (3.6) and the identified model are shown. From the figure we can draw two main conclusions, first, the car is not completely symmetric which can be seen especially at high slip angles. Second, similar to the observation in [100], the experiments excited the saturated region of the front tires but not the one of the rear tire. Therefore, the fit of the front tire model is reasonably good, but the rear tire model seems not correct in the saturated region. Note that the relative spread of the estimated forces in Figure 3.9 is comparable to the results in [100] where a full-sized car is used.

Time-domain system identification

In the time-domain system identification step, the goal is to find parameters of the model such that the measured state evolution fits best the simulated state evolution. In the case of the bicycle model, the parameters are the Pacejka tire model parameters of the front and the rear wheel $B_f, C_f, D_f, B_r, C_r, D_r$, as well as the drivetrain and friction model parameters $C_{m1}, C_{m2}, C_{rr}, C_d$. There are two factors which can be changed, first, which of the parameters are assumed fix and which are identified and second what kind of maneuvers are used to identify the parameters.

We used two types of maneuvers to find the parameters, first ramp steer maneuvers with high longitudinal velocities, and second maneuvers generated by an autonomous racing controller. The advantage of the first type of maneuvers is that no working controller is needed and that no noise is fed back by the controller, but the maneuvers may not represent the operation range of interest. On the other hand, the second type of maneuvers, the closed-loop trajectories have the advantage that the model is identified in the operation range of interest and that the data is widely available once a controller is working. However, the input trajectory is influenced by the measurement noise which can lead to problems and the controller potentially does not excite enough of the dynamics, which results in an overfitted model which does not generalize. In practice, the first type of experiments is suited to identify the model and is mainly used. The identification based on closed-loop data seems to work for the autonomous racing application, where the model is constantly excited even in the region where the tires are saturated. However, the approach is mainly of interest to slightly adapt the model due to model changes such as slow tire wear, or in the case where it is not possible to perform ramp steer or other identification maneuvers since there exists no open space using the same track material.

Regarding which parameters can be fixed, two approaches to fix parameters and therefore simplify the task are possible. First, the drivetrain model can be separately identified from the tire forces, by using the measured forward velocity to identify the tire forces and given the tire forces identify the drivetrain by only investigating the longitudinal dynamics. Secondly, instead of identifying both tire models, the front tire model can be fixed to the one obtained by the steady-state experiments. Both ideas allow to reduce the number of parameters, and we observed that this helps the convergence of the optimization problem.

Model validation

We conclude this appendix on the identification approach by showing model validation results. More precisely we show how the identified model compares against the real car driving around the track using an automatic racing controller. When looking at Figure 3.10 we can see that the measured velocities can be accurately predicted over a long time span, in this case around two laps. The largest errors occur in the forward velocity state at high velocities and in the lateral velocity at hard to characterize places. The latter errors are most probably caused by combined slip effects which are not considered in the model (3.1).

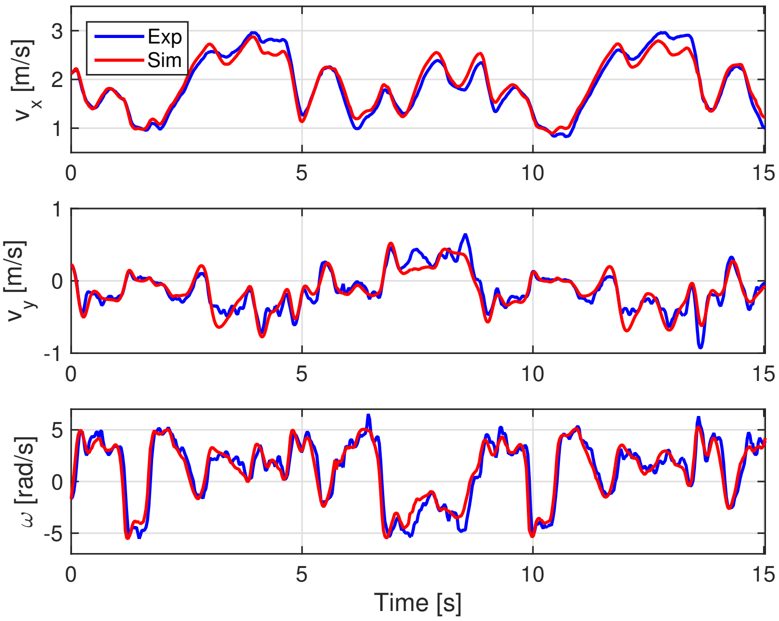

Figure 3.10: Model validation of the velocity states of the idendified bicycle model

If we look at the cross validation of the X-Y position and the yaw angle in Figure 3.11, we can see that after approximately 5 s the measured and predicted states start to diverge significantly. However, since these states are the integration of the velocity states, all errors accumulate over time, which makes accurate predictions over longer time spans nearly impossible. Finally, note that in our predictive controllers we only consider look ahead predictions of around 0.5 to 1 s which is well within the time span where the predictions are accurate.

3.5.2 Stationary velocity grid

The stationary velocity manifold is gridded in v_x and δ coordinates. In the v_x direction the space is uniformly gridded between a lower and an upper velocity, with a fixed spacing, see Table 3.1. For every v_x grid point, δ is gridded within the normal driving region with a fixed number of grid points, see Table 3.1. Additionally, in all cases the same 12 constant velocity points corresponding to drifting are included.

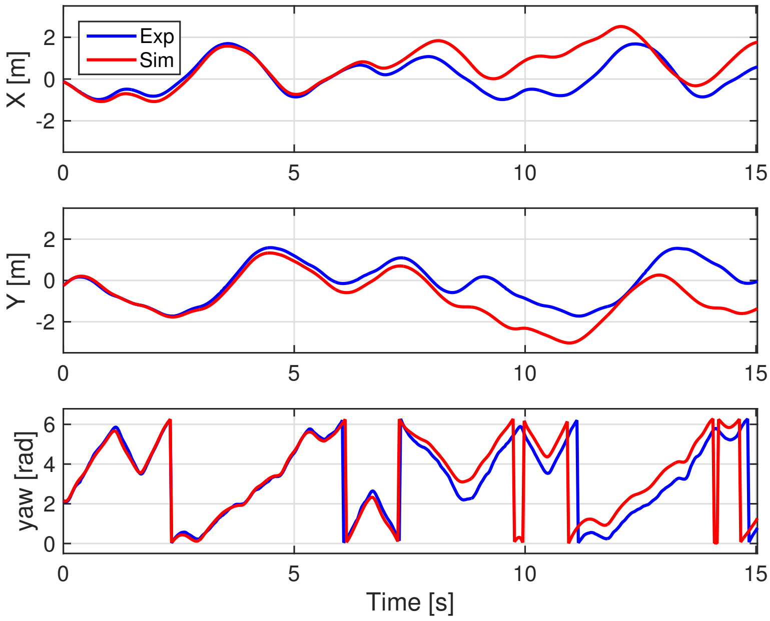

Figure 3.11: Model validation of the kinematic states of the idendified bicycle model

Table 3.1: Used gridding variables, for corresponding N_m

# modes	v_x range	v_x spacing	δ grid points	drifting modes
89	$[0.5, 3.5]$	0.25	5	12
99	$[0.6, 3.4]$	0.2	5	12
115	$[0.5, 3.5]$	0.25	7	12
129	$[0.6, 3.4]$	0.2	7	12

CHAPTER 4

Hierarchical Racing Controller

Based on the previous Chapter 3, we propose a hierarchical receding horizon racing controller. The controller is designed to race around a track with known boundaries and uses both the bicycle as well as the path planning model. The two level hierarchical structure is designed such that, at the higher level, many possible finite horizon trajectories are generated based on the current state, the track layout, and the path planning model. In the lower level, the trajectory with the largest progress from the higher level is tracked using an MPC subject to the bicycle model dynamics. In the following, we will refer to the higher level as the “path planning” step and the lower level as the “tracking MPC” step. The chapter is structured as follows, in Section 4.1 and 4.2 the path planning and tracking MPC step are formulated and discussed. The controller is then studied in simulation in Section 4.3 and the chapter is concluded in Section 4.4.

4.1 Path planning step

In the path planning step all trajectories consisting of N_S constant velocity segments and fulfilling the path planning model are generated. Since the admissible input set $U(q_k)$ is finite there only exist a finite number of state-input trajectories and once all these trajectories have been computed the path planning algorithm discards those that violate the state constraints (i.e., leave the track). Among the remaining feasible trajectories, the path planner selects the trajectory with the greatest progress or in other words the trajectory which maximizes the driven distance relative to the track.

To properly formulate the problem we need several ingredients. First, as mentioned above we use the path planning model, where the state is $x = [X, Y, \varphi, q]$ and the dynamics is given by the discrete time system $x_{k+1} = f(x_k, u_k)$ together with input constraints $u_k \in U(q_k)$ as described in Chapter 3. Second, the state constraints are used to discard trajectories where the car leaves the track and are given by $x_k \in \mathcal{X}_{\mathrm{T}}$, with,

$$\mathcal{X}_{\mathrm{T}} = \begin{cases} (X, Y) \in \mathcal{X}_{\mathrm{Track}}\,, \\ \varphi \in \mathbb{R}\,, \\ q \in \{1, ..., N_m\}\,, \end{cases} \tag{4.1}$$

where, $\mathcal{X}_{\mathrm{Track}}$ represents the track in global X-Y coordinates, φ is unconstrained since it is periodic and the mode q needs to be one of the N_m constant velocity points.

Finally, the objective function is the progress of the last point in the horizon. Formally, we define the progress by considering the center line of the track as a parametrized curve $cl\colon [0, L) \to \mathbb{R}^2$, where 0 represents the "starting line"; note that the argument of the function cl loops from L back to 0 every time the car completes a lap. Given a state $x \in \mathbb{R}^4$, we define its projection $\hat{p} : \mathbb{R}^4 \to [0, L)$ onto the center line using the first two components of the state

$$\hat{p}(x) = \arg\min_{l \in [0,L)} \|cl(l) - (X, Y)\|_2\,. \tag{4.2}$$

The projection (4.2) is computed by first generating off line a piecewise affine approximation of the center line, comprising 488 pieces in our case with endpoints regularly distributed on the center line. Online the projection can then be computed by finding the closest affine piece and taking an inner product.

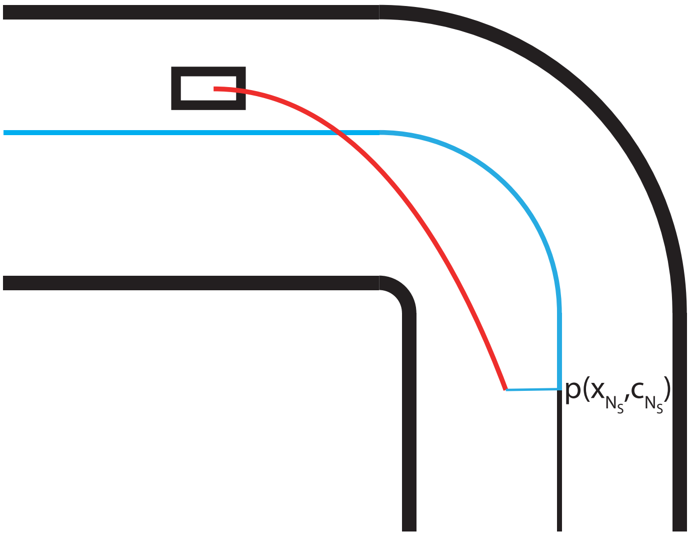

Figure 4.1: Visualization of the final state progress of a trajectory, shown in light blue.

To compute the progress we further consider a lap counter c which is initialized with $c = 0$ and is incremented whenever the car crosses the starting line. Under the reasonable assumption that the track length is such that the car can only cover a fraction of one lap in a single sampling time, we then define the update equation of the lap counter by

$$c_{k+1} = \mathcal{C}(x_k, u_k, c_k) = \begin{cases} c_k + 1 & \text{if } \hat{p}(x_k) > \hat{p}(f(x_k, u_k)) \\ c_k & \text{otherwise}\,. \end{cases}$$

Finally, given a state x_k and the lap counter c_k we define the progress function

$$p(x_k, c_k) = \hat{p}(x_k) + c_k L\,, \tag{4.3}$$

with Figure 4.1 visualizing the concept. Note that the lap counter allows to deal with the discontinuity introduced by the periodic track, and is especially needed when two cars race wheel-to-wheel in Chapter 6.

Therefore, the path planning step can be formulated as the following optimization problem,

$$\begin{aligned} \min_{\mathbf{u},\mathbf{x}} \quad & p(x_{N_S}, c_{N_S})\,, \\ \text{s.t.} \quad & x_0 = x, \quad c_0 = c\,, \\ & x_{k+1} = f(x_k, u_k)\,, \quad u_k \in U(q_k)\,, \\ & c_{k+1} = \mathcal{C}(x_k, u_k, c_k)\,, \\ & x_{k+1} \in \mathcal{X}_{\mathrm{T}}\,, \quad k = 0, ..., N_S - 1\,, \end{aligned} \tag{4.4}$$

where, $\mathbf{u} = [u_0, ..., u_{N_S-1}]$ is the input sequence, $\mathbf{x} = [x_0, ..., x_{N_S}]$ the corresponding state sequence, x the measured state and c the current lap counter. Note, the optimization problem (4.4) can be solved by enumeration, since there only exist a finite number of possible trajectories. However, the number of trajectories grows exponentially in the horizon length N_S. See Figure 4.2 for an illustration of all the positions of the path planning model which fulfill the state constraints.

Compared to similar methods performing path planning with motion primitives such as [22, 23], our approach is different since we do not plan a path from a starting point to a target set, but a finite horizon trajectory.

Alternatively, it is also possible to define the progress by extending the state with the lap counter c, leading to $\bar{x} = [x, c] \in \mathbb{R}^5$ as well as append the lap counter update equation to the dynamics $f(x, u)$ which gives rise to the dynamics of the combined state $\bar{x}_{k+1} = \bar{f}(\bar{x}_k, u_k)$. Note that the lap counter takes values in the discrete set of integers, but similar to the discrete mode q we again can embed in the real line, since its update equation is such that if initialized in this set it remains in this set. However, we only adapt this formulation in Chapter 6 where it leads to a significantly more readable notation.

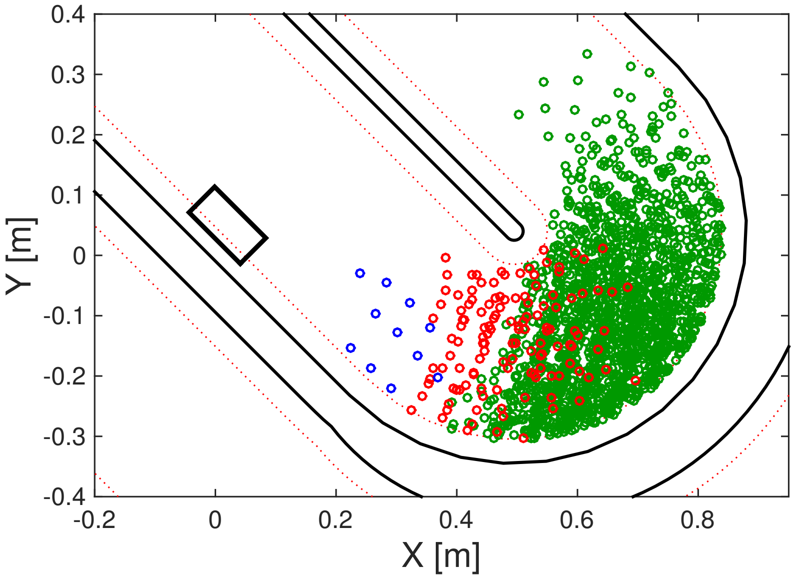

Figure 4.2: Visualization of the all the positions, which fulfill the path planning model and stay within the track constraints, x_1 shown in blue, x_2 shown in red and x_3 shown in green.

4.1.1 Implementation details

There are two main implementation issues with the above described path planner. First, the state of the path planning model cannot be measured, and secondly only checking collision every T_{pp} seconds potentially allows to "jump" over constraints, see Figure 4.3.

Since it is not possible to measure the entire state of the path planning model, we have to rely on estimation. We assume that the state of the bicycle model $\zeta = [X, Y, \phi, v_x, v_y, \omega]$ is available. In simulation this is given, since we use the bicycle model for simulation, and in the experimental set-up the state of the bicycle model is given by the vision system and the EKF state estimator mentioned in Chapter 2. Thus, to estimate the state of the path planning model $x = [X, Y, \varphi, q]$, it is only necessary to find the "closest" constant velocity mode q. We propose a two-step approach, first finding the closest stationary longitudinal velocity $\hat{v}_x$, and second from all the constant velocity points with this longitudinal velocity the weighted least square point is determined. The weighting is introduced so as to scale the lateral velocity and yaw rate to plus minus one. Note that this two-step approach is only possible since the constant velocity points are gridded with fixed longitudinal velocities. If this is not the case other methods would be necessary for example directly finding the weighted least square point.

The second problem arises from the fact that T_{pp} is large. Therefore, guaranteeing

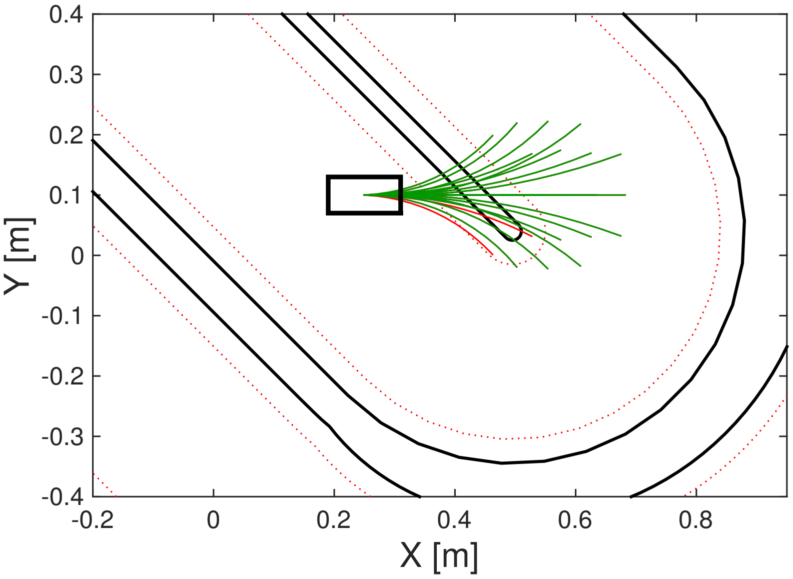

Figure 4.3: Visualization that only checking the switching points x_k is not enough to verify that the continuous trajectory does not leave the track. The real track boundaries are visualized in black, and the constraints the trajectory of the CoG needs to fulfill are marked in dotted red and finally the car illustrated by a box. Green trajectories indicate that the endpoint is inside the red dotted track constraints and red trajectories show that the constraints are violated.

that x_k is inside the track does not guarantee that the continuous trajectory does not leave the track and then reenters again. For example in Figure 4.3 where real track boundaries are visualized in black, the constraints the trajectory of the CoG needs to fulfill are marked with a dotted red line, and the car is illustrated by a box, the car stands clearly at a position where all trajectories are infeasible. However, when only the endpoints of the constant velocity trajectories, or in other words x_1 are checked for collision most of the trajectories are feasible. This is a common problem in sampled-data systems and needs to be tackled appropriately [108]. Theoretically, this can be avoided by checking that the continuous trajectory stays inside the track constraints. Practically, this is achieved by sampling the continuous trajectory and checking that all the sampled points stay inside the track constraints. In our implementation, we sample the continuous trajectory every 20 ms, which is about ten times smaller than T_{pp} and is small enough that the remaining violations of the continuous trajectory can be neglected (given the used track layout).

Finally, from a computational point of view, it makes no sense to generate all trajectories and then discard the ones which leave the track. It is more efficient to prune the tree of possible trajectories at every step. Thus, the path planning step for $N_S = 3$, can be summarized by the four stages in Figure 4.4. In the first three stages, the trajectories which stay inside the track are sequentially generated, where red trajectories leave the track, green trajectories stay inside the track, and blue trajectories are the feasible trajectories from the last stage. In the fourth stage, the progress of all trajectories is computed, and the one with the largest progress is selected, marked in red.

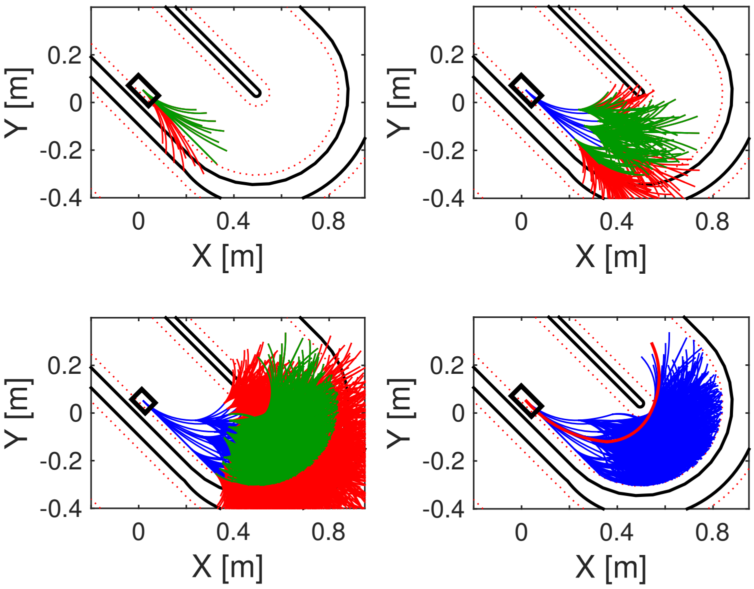

Figure 4.4: Visualization of the four stages in the path planning step, for $N_S = 3$ and $T_{pp} = 0.24\,\mathrm{s}$. In the first three stages, the trajectories which stay inside the track are sequentially generated, where red trajectories leave the track, green trajectories stay inside the track, and blue trajectories are the feasible trajectories from the last stage. In the fourth stage, the progress of all trajectories is computed and the one with the largest progress is selected, marked in red.

4.2 Tracking MPC step

The trajectory determined by the path planning step cannot be directly used to control the car since the model is not sophisticated enough. Therefore, a lower level controller is used to follow the trajectory of the higher level path planner. We propose to use a model predictive trajectory tracking controller. The main advantage of using an MPC in this setting is the natural way the trajectory of the path planner can be included, and that it is straightforward to add input as well as state constraints. Additionally, it is possible to handle the nonlinear dynamics by approximating them as a linear time-varying system.

For the reference tracking cost, note that the path planning model does not only give a trajectory in space, but given the constant velocity mode the velocities are known. Further, it is possible to sample the continuous trajectory of the path planning model at any sampling time, similar to the constraint checking in the path planning step. Thus, we can run the MPC at any sampling time T_s as long as T_{pp} is a multiple of T_s. See Figure 4.5, for an example where the optimal trajectory from Figure 4.4 is sampled with $T_s = 0.02\,\mathrm{s}$. Thus the following MPC problem, with a horzion of $N_{\mathrm{MPC}} = N_s T_{pp}/T_s$ can be formulated,

$$\begin{aligned}
\min_{\zeta,\nu}\ & (\zeta_{N_{\mathrm{MPC}}} - \zeta^{\mathrm{ref}}_{N_{\mathrm{MPC}}})^T P (\zeta_{N_{\mathrm{MPC}}} - \zeta^{\mathrm{ref}}_{N_{\mathrm{MPC}}}) \\
& + \sum_{k=0}^{N_{\mathrm{MPC}}-1} (\zeta_k - \zeta^{\mathrm{ref}}_k)^T Q (\zeta_k - \zeta^{\mathrm{ref}}_k) + \Delta\nu_k^T R \Delta\nu_k \,, \\
\text{s.t. } & \zeta_0 = \zeta \,, \\
& \zeta_{k+1} = f_b(\zeta_k, \nu_k) \,, \\
& \zeta_{k+1} \in \mathcal{Z} \,, \quad \nu_k \in \mathcal{N} \,, \\
& \Delta\nu_k = \nu_{k+1} - \nu_k \,, \quad k = 0, ..., N_{\mathrm{MPC}} - 1 \,.
\end{aligned} \tag{4.5}$$

Where, ζ is the initial state, $\zeta = [\zeta_0, ..., \zeta_{N_{\mathrm{MPC}}}]$ and $\nu = [\nu_0, ...\nu_{N_{\mathrm{MPC}}-1}]$ the state and input trajectory, and $Q = Q^T \succeq 0$, $P = P^T \succeq 0$, $R = R^T \succeq 0$ are tuning matrices. Furthermore, ζ^{ref}_k denotes the reference from the path planning step at time step k sampled with a discretization time of T_s and $\Delta\nu_k$ is the change in inputs. The discretized version of the nonlinear bicycle model (3.1) is given by $f_b(\cdot,\cdot)$ and $\mathcal{Z}$ represents the state constraints including the track constraints and $\mathcal{N}$ represents the input constraints.

The MPC problem (4.5) cannot be solved efficiently, mainly due to the nonlinear dynamics and the nonconvex track constraints. However, since we have a realistic reference trajectory, we can locally approximate these non-convex constraints as convex constraints. The nonlinear dynamic can be approximated by linearizing the continuous time bicycle model around the reference trajectory and then by discretizing it using zero order hold. This results in a discrete time linear time-varying system $\zeta_{k+1} = A_k\zeta_k + B_k\nu_k + g_k$. The state constraints are tackled by first separating the track constraints from the bounds on the state, given by $\underline{\zeta} \leq \zeta_{k+1} \leq \bar{\zeta}$. A local convex approximation of the track constraints is achieved by limiting each point in the horizon to lie within two parallel half spaces, which leads to two affine inequality constraints per time step (one for the right and one for the left border), resulting in time varying linear constraints $F_k\zeta_k \leq f_k$. Figure 4.6 depicts these border constraints which we formulate as soft constraints to avoid infeasibility problems in practice. By using an exact penalty function (an infinity norm in our case), we ensure that the resulting optimization problem is always feasible and that the original solution of the hard constraint problem is recovered in case it is

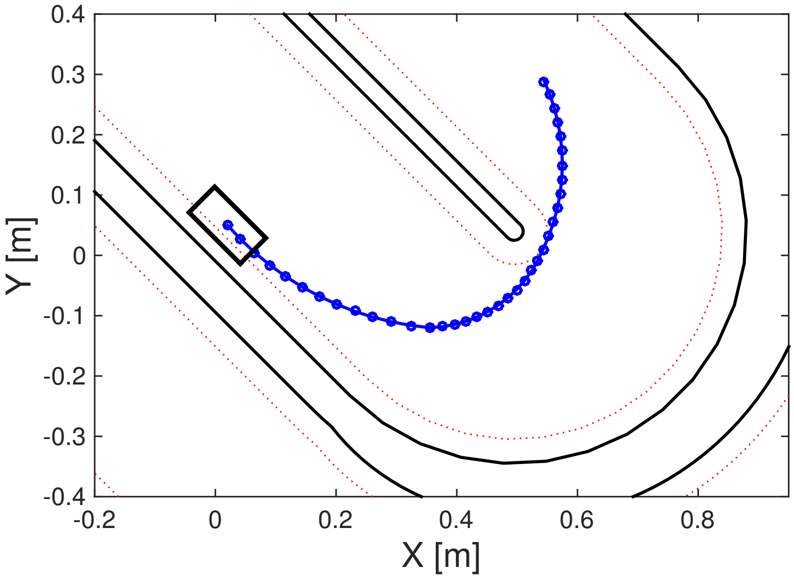

Figure 4.5: Trajectroy from the path planner, which can be used as the MPC reference. The trajectory from the path planner has a discretization time of $T_{pp} = 0.24\,\mathrm{s}$ and is sampled with $T_s = 0.02\,\mathrm{s}$.

feasible [109, 110]. Finally, the input constraints $\nu_k \in \mathcal{N}$ are already convex since we only consider bounds on the inputs $\underline{\nu} \leq \nu_k \leq \bar{\nu}$. Using this convexification approach the resulting optimization problem is given by,

$$
\begin{aligned}
\min_{\zeta,\nu,\mathbf{s}} \; & (\zeta_{N_{\mathrm{MPC}}} - \zeta^{\mathrm{ref}}_{N_{\mathrm{MPC}}})^T P (\zeta_{N_{\mathrm{MPC}}} - \zeta^{\mathrm{ref}}_{N_{\mathrm{MPC}}}) + q \|s_{N_{\mathrm{MPC}}}\|_\infty + \\
& \sum_{k=0}^{N_{\mathrm{MPC}}-1} (\zeta_k - \zeta^{\mathrm{ref}}_k)^T Q (\zeta_k - \zeta^{\mathrm{ref}}_k) + q \|s_k\|_\infty + \Delta\nu_k^T R \Delta\nu_k \,, \\
\text{s.t.} \; & \zeta_0 = \zeta \,, \\
& \zeta_{k+1} = A_k \zeta_k + B_k \nu_k + g_k \,, \\
& F_k \zeta_k \leq f_k + s_k \,, \quad s_k \geq 0 \,, \\
& \underline{\zeta} \leq \zeta_{k+1} \leq \bar{\zeta} \,, \quad \underline{\nu} \leq \nu_k \leq \bar{\nu} \,, \\
& \Delta\nu_k = \nu_{k+1} - \nu_k \,, \quad k = 0, ..., N_{\mathrm{MPC}} - 1 \,.
\end{aligned}
\tag{4.6}
$$

Here, $\mathbf{s} = [s_1, ...s_N]$ are the positive slack variables used in the soft constraint formulation and $q > 0$ is the weight which is chosen large enough to guarantee an exact reformulation. The resulting optimization problem is a convex multi-stage quadratic program which can be solved efficiently within milliseconds if tailored QP solvers such as FORCES Pro [111] are used.

In practice, the matrices Q, R and P are chosen diagonal, and it is important to

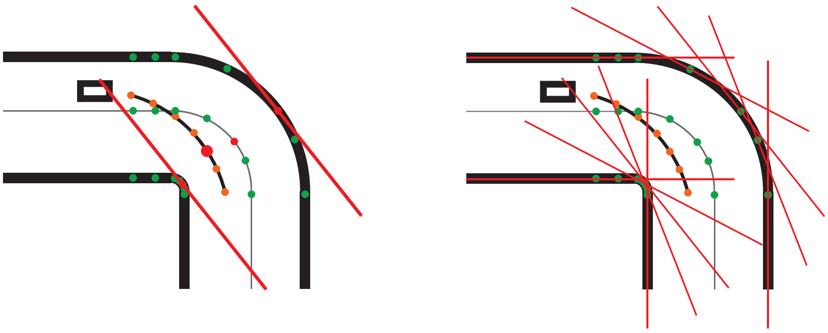

Figure 4.6: Drawing of the two half spaces approximating the track constraints. On the left for one point in the horizon and on the right shown for all points in the horizon.

properly tune them to achieve a good performance. Especially important is the terminal cost P which is chosen significantly larger than Q. This results in an MPC which tracks the given path well, while not being to aggressive. However, the terminal cost is still a tuning variable since a systematic approach to determine a terminal cost for the given setup is difficult to determine.

4.3 Simulation results

In this section we perform a simulation study, where we vary the prediction horizon N_S from 2 to 4, to show the performance of the controller and to highlight some of the drawbacks. For the simulation study, we use the full controller introduced in this chapter, which includes the path planner (4.4) and the MPC tracking controller (4.6). The controls are applied to the continuous time bicycle model (3.1), including control quantization motivated by the communication link in the real set-up, where both control inputs are represented using 8 bits each. The path planning model uses a discretization time of $T_{pp} = 0.24\,\mathrm{s}$ and the MPC controller uses a discretization time of $T_s = 0.02\,\mathrm{s}$. The path planning model uses $N_m = 129$ different modes, which are generated as described in Appendix 3.5.2. The whole controller is run at the sampling time of the MPC, and both steps are repeated at this sampling time.

To measure a controller's performance, we consider the mean lap time, the number of constraint violations (i.e., the number of time steps the car is outside the track constraints) and the computation time required by the path planner, as well as the computation time to solve the QP with FORCES Pro. Each controller is simulated for 10'000 time steps; with a sampling time of 20 ms, this corresponds to 200 s or roughly 23 laps. The simulations are performed on a MacBook Pro with a 2.4 GHz Intel Core i7 processor, implemented in `C`, using `gcc` as a compiler and `−O3` code optimization.

Table 4.1: Influence of N_S

N_S	mean lap time [s]	# constr. violations	PP comp. time [ms] median	PP comp. time [ms] max	QP comp. time [ms] median	QP comp. time [ms] max
2	9.03	13	4.826	15.083	2.237	6.332
3	9.17	4	23.986	121.694	3.539	9.581
4	9.02	0	87.686	1305.868	4.679	9.311

The simulation results are summarized in Table 4.1, and the driven trajectory, as well as the velocity profile of the fastest lap, is shown in Figure 4.7. When looking at the quantitative results in Table 4.1, we first investigate the computation time, where we can see that the median as well as maximum computation time for the path planning step (PP comp. time) increases exponentially, which is expected since the number of trajectories increases exponentially. The median computation time to solve the MPC problem, on the other hand, only increases linearly with the prediction horizon, as expected by FORCES Pro [88]. The maximum computation time for the MPC problem is caused by a few cases where the solver needs significantly more iterations to converge. It is also visible that only for a horizon of $N_S = 2$ the problem is real-time feasible. Note that even though the sum of the two max computation times is slightly above 20 ms, the maximal computation time measured for both tasks is 16.8 ms.

For the second part of the discussion we look at the driving performance indicators. When looking at the mean lap time, we can see that $N_S = 3$ performs worst. This is surprising as one would expect longer prediction horizons to perform better. However, the number of constraint violations is ordered as expected and $N_S = 2$ has the most violations. To explain this behavior it is necessary to investigate the driven trajectories in Figure 4.7, which shows the fastest lap driven for the three different horizon lengths, in solid blue $N_S = 2$, in dash-dotted red $N_S = 3$ and in dashed green $N_S = 4$. The beginning of the lap is in the top left corner marked with a line perpendicular to the track, and the car race counterclockwise.

In the bottom plot the forward velocity of the three controllers is shown relative to the center line. Showing the velocity relative to the center line allows us to compare the velocity at a given point on the track; the projection on the center line is performed as discussed in Section 4.1. We see that for most of the track, the controllers drive similarly and even have the same velocity. However, at some places around the track, marked in Figure 4.7 in light green, the difference in the look-ahead leads to a different driving style. For example at 2 m along the center line in the first curve, $N_S = 2$ approaches the apex very directly, and as visible in the velocity plot, needs to brake harder to get around the curve. This is caused by a too short prediction horizon, and a similar behavior can also be seen at 7, 9, and 16 m along the center line.

However, due to the short horizon, the controller is also over-optimistic on straights, driving faster in between curves than the other two controllers. This short-sighted driving style results in a lap time similar to $N_S = 4$ and even outperforms $N_S = 3$ regarding lap time. However, the price to pay is a higher number of constraint violations (possible crashes in reality) and a driving style which results in nearly three times as many feasibility issues, where each infeasible path planning problem triggers an emergency braking maneuver until the problem is feasible again.

Thus, in summary for the controller to perform well, at least a horizon of $N_S = 3$ is needed, and an even longer horizon is preferred. However, this comes with computation times that are not real-time feasible. The only way to implement the hierarchical racing controller as it is presented here on current computers is with a horizon of one or two. This, on the other hand, results in a controller which is short-sighted and has a nondesirable driving style, which inherently limits the performance.

4.4 Conclusion

In this chapter we introduced a hierarchical racing controller, which uses the strengths of the path planning and bicycle model. The path planning model allows to plan realistic reference paths, considering complex state constraints and objective functions. On the other hand the bicycle model incorporated into an MPC allows to find suited inputs to the car necessary for autonomous racing. Furthermore, the reference path from the higher level allows to drastically simplify the MPC problem by finding a local convex approximation around the reference. The proposed hierarchical racing controller allows to race the cars around a predefined track, but the controller is only real-time feasible for short prediction horizons. These short prediction horizons intrinsically limit the performance of the controller which needs a certain look-ahead in the path planning step, in order for the resulting closed-loop behavior to be suited for racing.

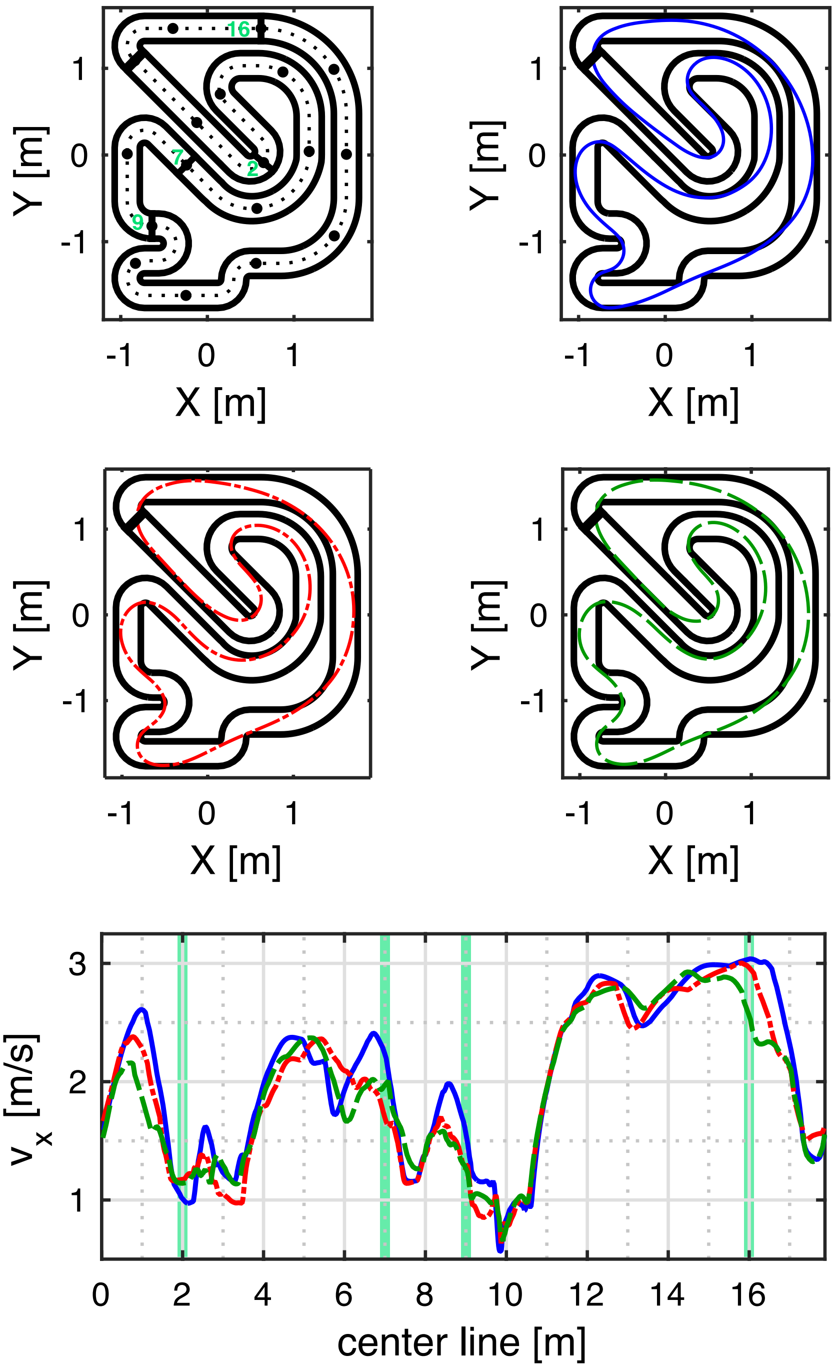

Figure 4.7: Top left, relative position to the center line, every meter along the center line is marked with a point, whereas points of interest are marked with a line across the track and the length of the center line in green. Top right, controller with $N_S = 2$ in solid blue, middle left, controller with $N_S = 3$ in dash-dotted red and middle right, controller with $N_S = 4$ in dashed green. Bottom: Velocity profile relative to the center line of the fastest lap for all three controllers, points of interest from the top left are marked with a green bar.

CHAPTER 5

Real-Time Control for Autonomous Racing Based on Viability Theory

While the hierarchical racing controller outlined in the previous chapter enables us to perform real-time racing of the RC cars, it faces two major drawbacks that limit its performance. The first drawback is that the number of trajectories grows exponentially in the prediction horizon N_S. The second drawback is that feasibility does not imply recursive feasibility. Indeed the maximal progress trajectory is often not recursively feasible when only feasibility is enforced. This manifests itself in the non-desirable driving style of the controller with a horizon of $N_S = 2$.

To tackle the second drawback, in this chapter we propose to restrict the path planner to trajectories that keep the system within the viability kernel (instead of just the track). Thereby, recursive feasibility of the path planning step is guaranteed under the assumption of no model mismatch. Moreover, the use of the viability kernel has two computational advantages that can dramatically reduce the computation time. First, it allows one to consider fewer trajectories (only those that remain in the viability kernel instead of all trajectories that stay on the track) and second, it suffices to verify that at any switching point remains within the viability kernel instead of checking that the whole trajectory stays on the track. Note that using the viability kernel does not fundamentally tackle the exponential growth of trajectories in the path planning step with increased horizon length. Empirically, however, this additional pruning of candidate trajectories leads to a significant reduction in the online computation times. This, in turn, allows extending the considered paths by more segments, resulting in a better closed-loop performance.

The benefit of using the viability kernel in the path planning step can be seen in Figure 5.1, where on the left the trajectories generated by the standard path planning step discussed in Chapter 4 are shown, and on the right, all non-viable trajectories are eliminated. Note that this helps to reduce the number of trajectories by approximately 25% and only removes trajectories which are doomed to leave the track in the future.

However, since in our setting it is only possible to approximate the viability kernel

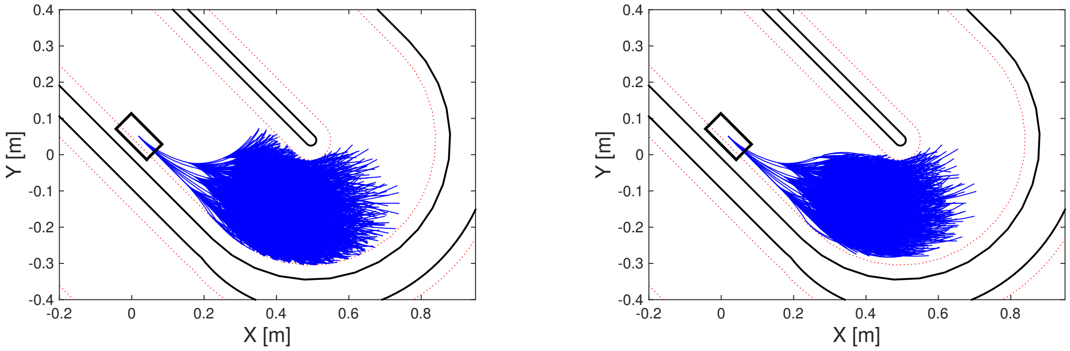

Figure 5.1: Left: Trajectories generated by the path planning step where only feasibility w.r.t. the track is enforced. Right: Trajectories generated by the path planning step with viability constraints. For $N_S = 3$, $T_{pp} = 0.16$ s and $N_m = 129$.

numerically by gridding the state space, the performance is also influenced by the space discretization. Therefore, it is also worth studying how the gridding affects the viability kernel. Based on these observations we propose a game theoretical approach to compute the viability kernel. More specifically we use the discriminating kernel algorithm which considers the discretization errors as an opponent player. The resulting discriminating kernel inner approximates the viability kernel and guarantees that also states in the "cell" around a grid point are viable.

This chapter is structured as follows: Section 5.1 summarizes the viability and discriminating kernel algorithm of [39, 40]. In Section 5.2 the new viability kernel approximation is introduced. Section 5.3 discusses the necessary modification to use the path planning model in the viablility kernel algorithm. The viability and discriminating kernel based on this model are then analyzed in Section 5.4. In Sections 5.5 and 5.6 we study the performance of the resulting controller both in simulation and in experiment. Conclusions are provided in Section 5.7. Appendix 5.8 contains technical results and proofs.

5.1 Viability kernel and discriminating kernel

In this section we briefly discuss discrete-time viability theory which will later on be used to construct recursive feasible trajectories. Discrete-time viability theory addresses the question for which initial conditions does there exists a solution to a difference inclusion, which stays within a constraint set forever [31]. Consider a controlled discrete-time system $x_{k+1} = f(x_k, u_k)$, where $x \in \mathbb{R}^n$ is the state, $u \in U \subset \mathbb{R}^m$ is the control input and $f : \mathbb{R}^n \times U \to \mathbb{R}^n$ a continuous function. This system can be formulated as a difference inclusion,

$$x_{k+1} \in F(x_k), \qquad \text{with } F(x) = \{f(x,u) | u \in U\} . \tag{5.1}$$

We next briefly summarize the standard viability kernel algorithm of [39] for the above class of systems.

5.1.1 Viability kernel algorithm

Given a constraint set $K \subset \mathbb{R}^n$, solutions to the difference inclusion (5.1) which stay in K forever are known as *viable solutions.*

Definition 5.1 ([39])**.** *A set $D \subset \mathbb{R}^n$ is a* ***discrete viability domain*** *of F if $F(x) \cap D \neq \emptyset$ for all $x \in D$. The* ***discrete viability kernel*** *of a set $K \subset \mathbb{R}^n$ under F, denoted by $\mathrm{Viab}_F(K)$, is the largest closed discrete viability domain contained in K.*

Under mild assumptions on F (discussed below in Proposition 5.1) standard viability theory arguments ensure the existence of the discrete viability kernel and establish the existence of viable solutions for all the states contained in it.

Conceptually speaking, the viability kernel can be calculated through the so called viability kernel algorithm:

$$\begin{aligned} &K^0 = K \, , \\ &K^{n+1} = \{x \in K^n | F(x) \cap K^n \neq \emptyset\} . \end{aligned} \tag{5.2}$$

Proposition 5.1 ([39])**.** *Let F be an upper-semicontinuous set-valued map with closed values and let K be a closed subset of $\mathrm{Dom}(F)$. Then, $\bigcap_{n=0}^{\infty} K^n = \mathrm{Viab}_F(K)$.*

The viability kernel algorithm requires one to perform operations with arbitrary sets and as such is not implementable. In practice, viability kernels are approximated by discretizing/gridding the state space. Consider a family of countable subsets X_h of $\mathbb{R}^n$, parameterized by $h \in \mathbb{R}$, such that

$$\forall x \in \mathbb{R}^n, \quad \exists x_h \in X_h \quad \text{s.t. } \|x - x_h\|_\infty \leq \alpha(h) \, ,$$

for some continuous function $\alpha : \mathbb{R}_+ \to \mathbb{R}_+$ with $\lim_{h\to 0} \alpha(h) = 0$. Given a grid point $x_h \in X_h$, we will call the set of points $\{x \in \mathbb{R}^n \mid \|x - x_h\|_\infty \leq \alpha(h)\}$ the *cell* of x_h. Notice that the cells of different grid points may overlap. We note that, unlike [39], we resort here to the infinity norm, as this will facilitate the presentation of subsequent results.

To ensure that the discretized set-valued map $F(x_h) \cap X_h$ is non-empty it is necessary to enlarge the set-valued map before discretization, a process that is known as "expansion" [39]. The expanded set-valued map is defined as $F^r(x) = F(x) + r\mathbb{B}_\infty$, where $\mathbb{B}_\infty$ is a closed unit ball in the infinity norm centered at the origin and r is the radius of the ball. It can be shown that, for $r \geq \alpha(h)$, $F^r(x) \cap X_h$ is non empty for all x. Throughout we denote $F^r_h(x_h) = F^r(x_h) \cap X_h$ and assume for simplicity that $r = \alpha(h)$. When restricted to X_h the set-valued map F^r_h defines the following discrete time system over the countable set by

$$x_{h,k+1} \in F^r_h(x_{h,k}) \,. \tag{5.3}$$

If the set K is compact, then the set X_h can be assumed to be finite and the viability kernel algorithm (5.2) applied to (5.3) converges after a finite number of iterations, since $K_h = K \cap X_h$ is finite. Moreover, under appropriate technical assumptions [39], the resulting set inner-approximates the viability kernel, i.e.,

$$\mathrm{Viab}_{F^r_h}(K_h) \subset (\mathrm{Viab}_{F^r}(K) \cap X_h) \,. \tag{5.4}$$

Note that the guaranteed inner approximation is only with respect to the viability kernel of the extended dynamics F^r. The extension limits the granularity/resolution of features that the kernel can represent due to extension, but is necessary in the finite case. Finally, the finite viability kernel converges to the discrete viability kernel as the space discretization h goes to zero, [39].

5.1.2 Discriminating kernel algorithm

The discriminating kernel algorithm of [40] extends viability computations to systems where the state evolution $x_{k+1} = g(x_k, u_k, v_k)$ is additionally affected by a disturbance $v_k \in V \subset \mathbb{R}^d$, where $g : \mathbb{R}^n \times U \times V \to \mathbb{R}^n$ is assumed to be a continuous function. The discriminating kernel algorithm returns all states ("victory domain") where there exists a control input, which is able to prevent the disturbance from driving the system to the open set $\mathbb{R}^n \setminus K$. Notice that, in the discriminating kernel the control is able to access the disturbance's current action, which is not the case if the related leadership kernel is used, for more details we refer to [112].

The difference equation can again be written as a difference inclusion

$$x_{k+1} \in G(x, v) \quad \text{with } G(x, v) = \{g(x, u, v) | u \in U\} \,.$$

Definition 5.2 ([40]). *A set $Q \subset \mathbb{R}^n$ is a* ***discrete discriminating domain*** *of G, if for all $x \in Q$, we have that $G(x, v) \cap Q \neq \emptyset$ for all $v \in V$. The* ***discrete discriminating kernel*** *of a set $K \subset \mathbb{R}^n$ under G, denoted by $\mathrm{Disc}_G(K)$, is the largest closed discrete discriminating domain contained in K.*

One can show that the discriminating kernel $\mathrm{Disc}_G(K)$ exists if K is a closed subset of $\mathbb{R}^n$ and $G : \mathbb{R}^n \times V \rightsquigarrow \mathbb{R}^n$ is an upper-semicontinuous set-valued map with compact values [40]. The discrete discriminating kernel can be calculated using an algorithm similar to the viability kernel algorithm, by considering a sequence of nested closed sets

$$\begin{aligned} &K^0 = K\,, \\ &K^{n+1} = \{x \in K^n | \forall v \in V,\ G(x,v) \cap K^n \neq \emptyset\}\,. \end{aligned} \tag{5.5}$$

To numerically implement the discriminating kernel algorithm, the state space $\mathbb{R}^n$, the disturbance space V and the difference inclusion $G(x,v)$ have to be discretized [40]. Whenever K and V are compact the algorithm (5.5) terminates after a finite number of iterations. Similar to the viability kernel it can be shown that the finite approximation of the discriminating kernel converges to the discrete discriminating kernel as the space discretization h goes to zero [40].

5.2 Robustifying the viability kernel algorithm against space discretization errors

5.2.1 Errors due to the space discretization

We start by noting that the inner approximation property of the viability kernel algorithm (5.4) only holds for viable grid points and not for states in their cells. In practice it is desirable that the viability property of a grid point is passed on to its cell. In the following section we present a rigorous approach ensuring this property.

Assumption 5.1. *The function $f(x,u)$ is continuous in u and globally Lipschitz in x with Lipschitz constant L, that is there exists a positive number L, such that for all $u \in U$,*

$$\forall x, y \in \mathbb{R}^n : \|f(y,u) - f(x,u)\|_\infty \leq L\|y - x\|_\infty\,.$$

Consider $x_h \in X_h$, and a point $x \in x_h + r\mathbb{B}_\infty$ in the cell of x_h, then under Assumption 5.1, it holds that for all $u \in U$

$$f(x,u) \in f(x_h,u) + Lr\mathbb{B}_\infty\,. \tag{5.6}$$

Therefore, to robustify the viability kernel algorithm against the space discretization, we can model the discretization error as an additive disturbance on our nominal system, i.e.,

$$x_{k+1} = f(x_k,u_k) + v_k, \qquad \text{with } v_k \in Lr\mathbb{B}_\infty\,. \tag{5.7}$$

5.2.2 Inner approximation of viability kernel

To calculate an inner approximation of the viability kernel that guarantees viability of the entire cell of a viable point, we propose to use the discriminating kernel algorithm (5.5).

Proposition 5.2. *Under Assumption 5.1, the discriminating kernel* $\mathrm{Disc}_G(K)$ *of the difference inclusion,*

$$x_{k+1} \in G(x_k, v_k) = F(x_k) + v_k \quad v_k \in Lr\mathbb{B}_\infty\,, \tag{5.8}$$

enjoys the following properties:

(1) $\lim_{r\to 0} \mathrm{Disc}_G(K) = \mathrm{Viab}_F(K)$.

(2) $\mathrm{Disc}_G(K)$ *is a viability domain of* F.

(3) For all $x \in \mathrm{Disc}_G(K)$ *and for all* $\hat{x} \in x + r\mathbb{B}_\infty$*, there exists a* $u \in U$ *such that* $f(\hat{x}, u) \in \mathrm{Disc}_G(K)$.

Proof. Under Assumption 5.1 $F(x)$ is upper-semicontinuous, hence $G(x)$ is upper-semicontinuous, and the conditions to apply the viability and discriminating kernel algorithms are satisfied.

(1) If $r \to 0$ the disturbance set vanishes, thus $\bigcap_{r>0} Lr\mathbb{B}_\infty = \{0\}$ and using [40, Remark 4.1], the discriminating kernel is identical to the viability kernel.

(2) By definition the discriminating kernel is a discriminating domain. Thus for all $x \in \mathrm{Disc}_G(K)$ we have,

$$(F(x) + v) \cap \mathrm{Disc}_G(K) \neq \emptyset \quad \forall v \in V = Lr\mathbb{B}_\infty\,,$$

Since $v = 0 \in Lr\mathbb{B}_\infty$ is an allowed disturbance, the discriminating kernel is also a viability domain, as

$$\forall x \in \mathrm{Disc}_G(K), \quad F(x) \cap \mathrm{Disc}_G(K) \neq \emptyset\,.$$

(3) First, by (5.6) we know that for all $\hat{x} \in x + r\mathbb{B}_\infty$

$$f(\hat{x}, u) \in f(x, u) + Lr\mathbb{B}_\infty\,,$$

and therefore for a given $u \in U$, $f(\hat{x}, u) = f(x, u) + v$, for some $v \in Lr\mathbb{B}_\infty$. Second, from [113, Equation (4)] we know that the discriminating kernel $\mathrm{Disc}_G(K)$ is the largest closest subset of K such that for all $x \in \mathrm{Disc}_G(K)$ and for all $v \in V$, there exists a $u \in U$ such that $g(x, u, v) \in \mathrm{Disc}_G(K)$. Thus, together we have that for every $\hat{x} \in x + r\mathbb{B}_\infty$ there exists a $u \in U$, such that $f(\hat{x}, u) \in \mathrm{Disc}_G(K)$.

□

Note that if the leadership kernel is used instead of the discriminating kernel, a similar property can be shown.

As discussed in the previous Section 5.1, it is necessary to discretize space to implement the discriminating kernel algorithm. To state similar properties as in Proposition 5.2 if the space is discretized we need an inner approximation of the discriminating kernel $\mathrm{Disc}_{G_h^r}(K_h) \subset \mathrm{Disc}_{G^r}(K) \cap X_h$. This is generally not given, therefore we derive a modified discriminating kernel algorithm, which guarantees an inner approximation for the given system (5.8). The modified discriminating kernel algorithm is derived in Appendix 5.8 and stated in Algorithm 1. The algorithm, however needs two assumptions; (i) we only consider square grids and (ii) the set U needs to be finite. The second assumption is necessary to implement the algorithm and can be achieved by introducing a finite discretization of U, which we henceforth denote by $U_h \subset U$. Note that for our system square grids are an obvious simple choice and the path planning model has a finite input space by construction.

Proposition 5.3. *Consider the finite dynamical system corresponding to the extended and discretized system of* (5.8),

$$x_{h,k+1} \in G_h^r(x_{h,k}) = (F(x_{h,k}) + v_h + r\mathbb{B}_\infty) \cap X_h \,,$$

where $v_h \in V_h$ *is the discretization of* $Lr\mathbb{B}_\infty$*. If* $\mathrm{Disc}_{G_h^r}(K_h)$ *is computed with Algorithm 1, then the following properties hold:*

- *(1)* $\mathrm{Disc}_{G_h^r}(K_h)$ *is a viability domain of* F_h^r*.*
- *(2) For all* $x_h \in \mathrm{Disc}_{G_h^r}(K_h)$ *and for all* $\hat{x} \in x_h + r\mathbb{B}_\infty$*, there exists* $u_h \in U_h$ *such that* $(f(\hat{x}, u_h) + r\mathbb{B}_\infty) \cap X_h \in \mathrm{Disc}_{G_h^r}(K_h)$*.*

Proof. See Appendix 5.8. □

5.2.3 Reconstructing viable controls

The use of the discriminating kernel in the path planning step of the hierarchical racing controller as presented in Chapter 4 is not directly possible as the action depends on both the state and the disturbance. However, as the disturbance is related to the position of the current state within the grid cell the disturbance is indirectly known. Furthermore, we know that if the state is within a grid cell of the discriminating kernel, there exists a control which keeps the system within a grid cell of the discriminating kernel, see Proposition 5.3. Thus, we can formulate a predictive controller which enforces this property.

More precisely, we can modify the optimization problem (4.4) which defines the path planning step in Chapter 4 to constrain the state to stay within the discriminating kernel. Feasibility of the problem below is ensured by virtue of Proposition 5.3; the objective function $p(x_{N_S}, c_{N_S})$ is the progress and the system dynamics $x_{k+1} = f(x_k, u_{h,k})$ is the path planning model as described in Chapter 4,

$$\begin{aligned} \min_{\mathbf{u}_h, \mathbf{x}} \quad & p(x_{N_S}, c_{N_S})\,, \\ \text{s.t.} \quad & x_0 = x, \quad c_0 = c\,, \\ & x_{k+1} = f(x_k, u_{h,k})\,, \quad u_{h,k} \in U_h(q_k)\,, \\ & c_{k+1} = \mathcal{C}(x_k, u_k, c_k)\,, \quad k = 0, ..., N_S - 1\,, \\ & (f(x_k, u_{h,k}) + r\mathbb{B}_\infty) \cap X_h \subset \text{Disc}_{G_h^r}(K_h)\,. \end{aligned} \tag{5.9}$$

Where, $\mathbf{u}_h = [u_{h,0}, ..., u_{h,N_S-1}]$ is the control sequence, $\mathbf{x} = [x_0, ..., x_{N_S}]$ the corresponding state sequence, x the measured state, c the current lap counter, $c_{k+1} = \mathcal{C}(x_k, u_k, c_k)$ the update equation of the lap counter and $u_{h,k} \in U_h(q_k)$ the input constraints of the path planning model. We still use the subscript h to highlight that the set of admissible inputs needs to be finite, which is the case for the path planning model but maybe not the case for other applications/models. Note that only the constraints are modified and that this approach to reconstruct viable controls would also work for general objective function. Identical to Chapter 4 the problem is solved by enumeration, which limits the horizon to few steps. To reduce the computation time related to generate all feasible input and state sequences, it is possible to replace $u_{h,k} \in U_h$ with $u_{h,k} \in U_D(x_k + r\mathbb{B}_\infty \cap X_h)$, where,

$$U_D(x_h) = \begin{cases} \left\{ \begin{array}{c} u_h \in U_h \,|\, \exists v_h \in V_h \\ f(x_h, u_h) + v_h + r\mathbb{B}_\infty \\ \cap \text{Disc}_{G_h^r}(K_h) \neq \emptyset \end{array} \right\} & \begin{array}{l} \text{if } x_h \in \\ \text{Disc}_{G_h^r}(K_h) \end{array} \\ \\ \emptyset & \text{otherwise} \end{cases}\,. \tag{5.10}$$

If instead the viability kernel is used, the last constraint of (5.9) is replaced with $(f(x_k, u_{h,k}) + r\mathbb{B}_\infty) \cap X_h \subset \text{Viab}_{F_h^r}(K_h)$. And to reduce the computation time $u_{h,k} \in U_h$ can be replaced with $u_{h,k} \in U_V(x_k + r\mathbb{B}_\infty \cap X_h)$, where the set-valued feedback law $u_{k,h} \in U_V(x_h)$ can be computed similar to U_D in (5.10). However, there exists no guarantee that the control will keep the system within the viability kernel if the state does not always coincide with a grid point.

5.3 Upper-semicontinuity of the path planning model

In (5.9), we formulated the path planning step including viability constraints, however, to compute the viability and discriminating kernel the path planning is required to be upper-semicontinuous. In Chapter 3, we showed that the path planning model can be described as the following discrete-time model, $x_{k+1} = f(x_k, u_k)$, where $x_k = (X_k, Y_k, \varphi_k, q_k)$ and u_k determines the mode of the path planning model, and the discretization time of the path planning model is T_{pp}. The set-valued map needed for the viability kernel algorithm is given by $F(x) = \{f(x,u)|u \in U(x)\}$, and the output is the union of a finite number of points in the state space.

In the following, we show that by appropriately embedding the discrete state in the set of real numbers, upper-semicontinuity of the set-valued map of (3.5) can be ensured despite the hybrid structure of the model. To see this, recall that a set-valued map $F : \mathbb{R}^n \rightsquigarrow \mathbb{R}^n$ is called upper-semicontinuous if and only if for all $x \in \mathbb{R}^n$ and for all $\epsilon > 0$ there exists an $\eta > 0$ such that for all $x' \in x + \eta B$ it holds that $F(x') \subset F(x) + \epsilon B$ [114]. By embedding the discrete state q in the real numbers we have $x = (X, Y, \varphi, q) \in \mathbb{R}^4$, where the first three states (X, Y, φ) are continuous by virtue of the original model. Collecting them into the continuous state $x_c = (X, Y, \varphi) \in \mathbb{R}^3$ and defining the integer lattice $\mathbb{M} = \{1, ..., N_m\}$, where N_m is the number of modes, we can define the following upper-semicontinuous set-valued map,

$$F(x_c, q) = \begin{cases} \tilde{F}(x_c, m), & \text{if } q \in (m - \frac{1}{2}, m + \frac{1}{2}) \\ \left\{ \begin{matrix} \tilde{F}(x_c, m), \\ \tilde{F}(x_c, m+1) \end{matrix} \right\}, & \text{if } q = m + \frac{1}{2} \end{cases},$$

where $\tilde{F}(x_c, m) = \{f([x_c, m], u)|u \in U(m)\}$ is the set-valued map defined at the lattice point m. Notice that $q \in (1 - 1/2, N_m + 1/2)$ is w.l.o.g. since the automaton maps the interval to itself, forming an invariant set in $\mathbb{R}$. Finally, upper-semicontinuity is ensured because the dynamics of x_c is continuous and, using the above proposed formulation there, are no discontinuities in a set-valued sense.

To reduce the number of grid points in a practical implementation of the algorithm it is beneficial to limit the angle between 0 and 2π. Instead of using $\text{mod}(\varphi, 2\pi)$, which would lead to a discontinuous set-valued map, we propose the set-valued mod function

$$\text{Mod}(\varphi, 2\pi) = \begin{cases} \text{mod}(\varphi, 2\pi) & \text{if } \varphi \neq k2\pi \\ [0, 2\pi] & \text{otherwise} \end{cases},$$

which is upper-semicontinuous.

Finally, as already discussed in Chapter 4 with the model (3.5) it is still possible that the trajectory leaves the constraint set K and re-enters during the time interval T_{pp}. This is an issue more generally for sampled data systems and was tackled in [108] by a sampled data system viability/discriminating kernel algorithm that discards control inputs and states for which the continuous evolution of the data sampled system leaves the constraint set K.

In the following section we will use the path planning model, including the modifications and the modified sampled data system viability/discriminating kernel algorithm to compute the kernel with respect to the track constraints.

5.4 Viability kernel for the track

5.4.1 Track constraints

The goal is to find a viable solution of the path planning model within the track. To that end, we can define the constraint set K as,

$$K := \begin{cases} (X, Y) \in \mathcal{X}_{\text{Track}} \,, \\ \varphi \in [0, 2\pi] \,, \\ q \in \{1, ..., N_m\} \,. \end{cases}$$

In other words, X, Y are constrained within the track, while φ and q are unconstrained. The state space is uniformly gridded such that the distance in the respective unit (meter or radian) between two grid points is identical. This leads to the smallest r if the states are not normalized. Further, as q is already finite no gridding is necessary.

The Lipschitz constant of the path planning model (3.5) can be calculated analytically by differentiating the explicit formula for $f(x_k, u_k)$. By using different Lipschitz constants for each mode it is possible to reduce conservatism of the proposed discriminating kernel approximation.

5.4.2 Viability vs. discriminating kernel

Proposition 5.2 statement (1) in Section 5.2 suggests that the proposed discriminating kernel approximation converges to the viability kernel as the grid spacing r goes to zero. To verify this, we compare the fraction of grid points in K which fall into the respective kernels. Thus, if all grid points in K are viable, then the fraction would be one. Figure 5.2 shows the fraction for the viability kernel and the discriminating kernel approximation for different numbers of grid points. It is interesting to see that the proportion of the viability kernel stays approximately at 0.35 for all grid spacings, whereas the discriminating kernel approaches the viability kernel as the grid spacing gets smaller. We can make two observations from Figure 5.2, first, as Proposition 5.2 suggests the two approximations of the viability kernel get closer the smaller the grid spacing is. Second, while the viability kernel only changes slightly as a function of the grid spacing, the proposed discriminating kernel approximation requires a relatively fine grid to achieve an approximation that is similar to the one of the viability kernel algorithm. Intuitively speaking, this is because a coarse grid introduces a larger uncertainty than a fine grid, see also (5.7).

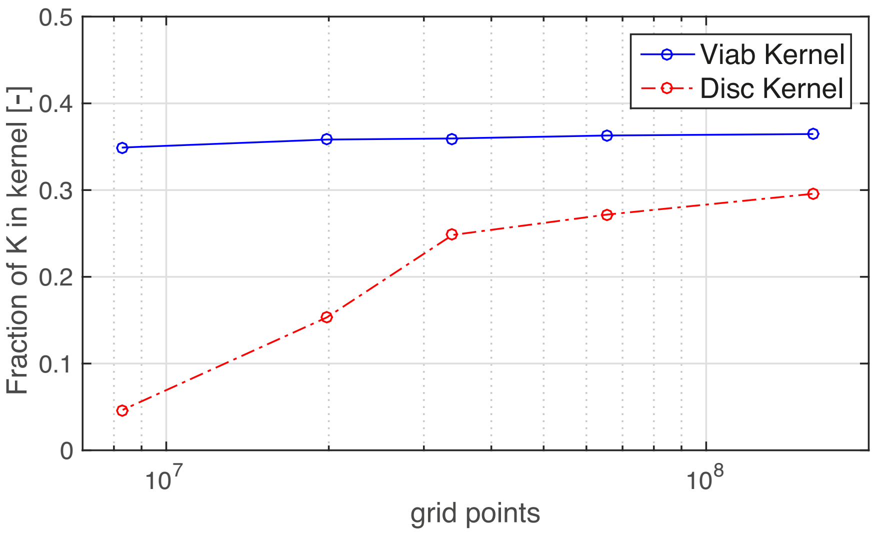

Figure 5.2: Fraction of grid points of K which fall into the viability kernel and the discriminating kernel, as a function of the total number of grid points in K.

Figure 5.3, visualizes the two kernels by comparing slices of the four dimensional kernels, this is achieved by fixing the angle φ and the mode q, resulting in just the X-Y plane which is easy to visualize. Figure 5.3, allows to see a qualitative difference between the viability and discriminating kernel for $r = 0.02$ and $r = 0.015$ (corresponding to 65'528'130 and 157'284'540 grid points respectively). Note that $r = 0.02$ implies that the grid points are 4 cm apart; as a comparison the track is 37 cm wide, and the size of the cars is about 12×5 cm. Figure 5.3 shows that the difference between the two kernels also qualitatively becomes smaller if we use a finer grid. More precisely Figure 5.3 shows all the viable points if the car is driving straight left with a forward velocity of 2 m/s. For the illustrated angle and constant velocity, mainly the top straight and the bottom S-curve are of interest (depending on the driving direction). In this region, the difference between the kernels and the effect of grid size is visible, for example, in the bottom S-curve, where the red region (which indicates that the grid point is in both kernels) gets significantly larger for the finer grid.

The difference in the kernel also influence the trajectories generated in the path planning step (5.9). See Figure 5.4 for a comparison, where we can see that if the discriminating kernel is used even fewer trajectories are generated, as a reference Figure 5.1 shows the same situation with no viability constraints.

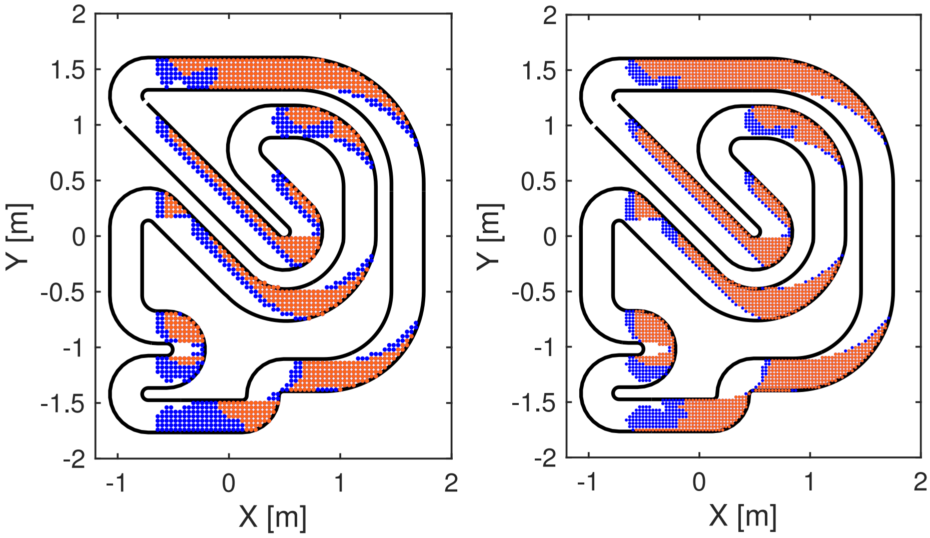

Figure 5.3: Red points indicate that the grid point is in both kernels, blue grid points are only in the viability kernel and not shown grid points are in neither kernel. Both plots show a slice through the kernels for the angle fixed to $\varphi = 180°$ and a mode where the car is driving straight with 2 m/s. In the left figure 65'528'130 grid points are used in the calculation whereas the right we use a finer grid with 157'284'540 grid points.

5.5 Simulation results and sensitivity analysis

We start with a simulation study to investigate the effect of the various design choices similar to the one shown in Chapter 4. We use the two-level hierarchical racing controller introduced in Chapter 4, however, we use the path planning step including the viability constraints (5.9). The controls are applied to the continuous time bicycle model (3.1), including control quantization motivated by the communication link in the real set-up, where both control inputs are represented using 8 bit each.

The effect of the following tuning parameters in the path planner are examined:

- use of viability kernel, discriminating kernel, and no viability constraints;
- grid size in (X, Y, φ);
- number of modes N_m;
- path planning discretization time T_{pp};
- number of constant velocity segments N_S.

To measure a controller's performance, we consider the mean lap time, the number of constraint violations (i.e., the number of time steps the car is outside the track constraints) and the computation time required by the path planner. Each controller is simulated for 10'000 time steps; with a sampling time of 20 ms, this corresponds to

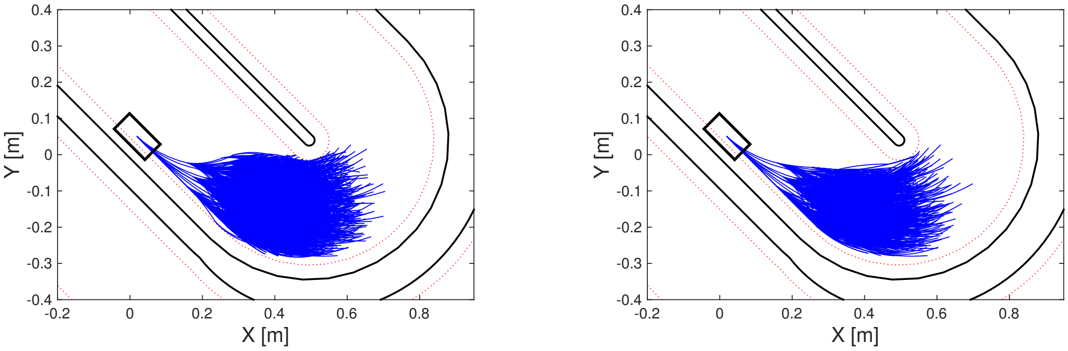

Figure 5.4: Left: Trajectories generated by the path planning step using the viabiltiy kernel. Right: Trajectories generated by the path planning step using the discriminating kernel. For $N_S = 3$, $T_{pp} = 0.16$ s and $N_m = 129$.

200 s or roughly 23 laps. The simulations are performed on a MacBook Pro with a 2.4 GHz Intel Core i7 processor, implemented in `C`, using `gcc` as a compiler and $-$`O3` code optimization. Furthermore, we investigate the offline computation time and required memory for certain combinations, where the viability and discriminating kernel are computed in *JULIA*, on a computer running Debian equipped with 16 GB of RAM and a 3.6 GHz Intel Xeno quad-core processor.

5.5.1 Viability constraints and grid size

We first compare the controller including the path planner with viability constraints ($\mathcal{C}_f^v$, $\mathcal{C}_f^d$, $\mathcal{C}_c^v$ and $\mathcal{C}_c^d$), to the naive controller without viability constraints ($\mathcal{C}^{nv}$) as studied in Chapter 4. Specially, we consider both the viability kernel and the discriminating kernel (superscript v and d), and two different grid spacing ($r = 0.02$ and $r = 0.015$) referred to as coarse and fine (subscript c and f). All the controllers use the parameters $N_m = 129$, $T_{pp} = 0.16$ s and $N_S = 3$ in the path planner.

Table 5.1 compares all five controllers in terms of driving performance (mean lap time and constraint violations) as well as online computation cost and Table 5.2 compares the offline computation times as well as the storage requirements for the kernels. We see that the number of constraint violation is negligible for all controllers as no controller triggers more than ten violations (out of 10'000 steps). When focusing on the mean lap time, we can distinguish three groups: First, we see that $\mathcal{C}^{nv}$ has the highest mean lap time. Second are $\mathcal{C}_f^v$ and $\mathcal{C}_f^d$, which have very similar mean lap times, and are clearly the two fastest algorithms. Third, we have $\mathcal{C}_c^v$ and $\mathcal{C}_c^d$ whose mean lap times lie between the first two groups.

If we compare the online computation times of the controllers, then we can mainly

Table 5.1: Influence of the viability constraints on the online performance

Controller			mean lap time [s]	# constr. violations	online comp. time [ms]	
abbr.	kernel	grid			median	max
$\mathcal{C}^{nv}$	No	N/A	8.77	10	43.71	334.23
$\boldsymbol{\mathcal{C}^{v}_{c}}$	**Viab**	**coarse**	**8.57**	**0**	**0.904**	**7.968**
$\mathcal{C}^{v}_{f}$		fine	8.39	3	0.870	7.557
$\mathcal{C}^{d}_{c}$	Disc	coarse	8.60	1	0.870	7.533
$\mathcal{C}^{d}_{f}$		fine	8.41	6	1.032	6.518

Table 5.2: Offline aspects of the viability constraints

Controller			comp. time kernel [s]	# of itera- tions	memory [MB]	
abbr.	kernel	grid			kernel	control
$\mathcal{C}^{nv}$	No	N/A	N/A	N/A	N/A	N/A
$\boldsymbol{\mathcal{C}^{v}_{c}}$	**Viab**	**coarse**	**17957**	**11**	**65**	**1769**
$\mathcal{C}^{v}_{f}$		fine	43238	11	157	4246
$\mathcal{C}^{d}_{c}$	Disc	coarse	156776	23	65	1769
$\mathcal{C}^{d}_{f}$		fine	397674	30	157	4246

differentiate between the controller without viability constraints ($\mathcal{C}^{nv}$) and controllers with viability constraints ($\mathcal{C}^{v}_{f}$, $\mathcal{C}^{d}_{f}$, $\mathcal{C}^{v}_{c}$ and $\mathcal{C}^{d}_{c}$) which are on average more than 40 times faster. As $\mathcal{C}^{nv}$ does not use any viability constraints to guide the path planning process, more branches of the tree have to be generated and checked, slowing down the computation, see Figure 5.1. Furthermore, we see from Table 5.1 that the maximal computation time of $\mathcal{C}^{nv}$ is 334 ms, which by far exceeds the sampling time of 20 ms.

When comparing $\mathcal{C}^{nv}$ with the remaining controllers using the viability constraints, it is also essential to investigate the offline computation time and the memory the viability lookup tables in (5.9) require. First, for the controllers with viability constraints, we need to compute the corresponding kernels. Their computation time mainly depends on two factors: The choice between viability or discriminating kernel, and the used grid size. The number of grid points has a direct influence on the offline computation time, as the number of operations per grid point stays the same. The observation when comparing the chosen kernel algorithm is that the discriminating kernel algorithm is significantly slower. This has two reasons: first each iteration needs more operations than the viability kernel algorithm due to the disturbance (5.5); second, the discriminating kernel algorithm needs more iterations to converge. For an implementation the required memory of the viability based path planner (5.9) is also important, as (5.9) requires to store a four-dimensional lookup table for the kernel and a five-dimensional lockup table for the viable inputs. The memory needed to store these lookup tables only depends on the grid size, and for the fine grid is 4.4 GB, and 1.8 GB for the coarse grid. Thus, we can conclude that, if the memory requirement and offline computation time are of no concern, then $\mathcal{C}_f^v$ and $\mathcal{C}_f^d$ should be preferred as they are the best performing controllers. However, if the available memory is limited, $\mathcal{C}_c^v$ and $\mathcal{C}_c^d$ with a coarse grid achieve good performance at a significantly lower memory requirement.

For the following discussions, we will compare the other tuning parameters (N_m, T_{pp} and N_S), of the path planner while using $\mathcal{C}_c^v$ as the base comparison, marked as bold text in the tables.

5.5.2 Number of modes, N_m

We examine the influence of N_m by varying the number of modes, which depends on the grid imposed on the stationary velocity manifold, see Appendix 3.5.2 for the different grids used. As expected, the lap time get slower the fewer modes are considered (mean lap time with $N_m = 89$ is $9.69\,\mathrm{s}$), but at the same time the computation time decrease by up to a factor of two. In all cases, the number of constraint violations is zero.

5.5.3 Discretization time, T_{pp}

The effect of time discretization was investigating by testing different values of $T_{pp} \in \{0.12, 0.16, 0.2, 0.24\}\,\mathrm{s}$. The results are summarized in Table 5.3, where we can see that larger T_{pp} lead to lower computation times but the lap times getting slower. This effects come from the influence of T_{pp} on the path planner model. As for the same mode (same constant velocity), a larger T_{pp} implies that the car travels further. This may render this mode non-viable as the additional traveled distance may bring the car too close to the border. In contrast slower modes more likely stay inside the track. Thus, the path planner more likely chooses slow trajectories for larger T_{pp}, which leads to slower lap times. At the same time fewer trajectories are viable, which reduces the computation time. Thus, even though, a larger T_{pp} implies a longer preview, the larger number of non-viable states mitigate this advantage.

Table 5.3: Influence of T_{pp}

T_{pp}	mean lap time [s]	# constr. violations	comp. time [ms] median	comp. time [ms] max
0.12	8.51	1	1.656	11.061
0.16	**8.57**	**0**	**0.904**	**7.968**
0.2	8.79	1	0.720	6.169
0.24	9.25	1	0.599	3.473

Table 5.4: Influence of N_S

N_S	mean lap time [s]	# constr. violations	comp. time [ms] median	comp. time [ms] max
2	8.83	0	0.179	2.435
3	**8.57**	**0**	**0.904**	**7.968**
4	8.56	1	17.015	114.278

5.5.4 Number of constant velocity segments, N_S

The influence of the number of constant velocity segments, or in other words the prediction horizon of the path planner is investigated by varying N_S from two to four. In Table 5.4 we can see that more segments improve the performance but at the same time the computation time is heavily increased. The improved lap times came from the longer preview of the controller with larger N_S. The longer preview is achieved without changing the viability of a trajectory, which stands in contrast with larger T_{pp}. Furthermore, since both T_{pp} and N_S influence the preview window, we have observed that N_S and T_{pp} should be tuned simultaneously, e.g., for $N_S = 2$, $T_{pp} = 0.2\,\mathrm{s}$ achieves a faster mean lap time than $T_{pp} = 0.16\,\mathrm{s}$.

5.5.5 Concluding remarks

The above discussion is summarized in Figure 5.5, which allows us to draw the following conclusions: First, all the tested controllers based on kernel pruning outperform $\mathcal{C}^{nv}$ which for the horizon length considered does not even meet the real time requirements. In contrast, the controllers with viability constraints and $N_S \leq 3$ are real-time feasible; put another way, the viability constraints allow one to use longer horizons in real-time than would be possible with the naive controller from Chapter 4. Second, when comparing the variations of $\mathcal{C}_c^v$ with different values of N_m, T_{pp} and N_S, we have observed that more modes N_m, shorter T_{pp} and larger N_S are beneficial for the performance. Furthermore, the numerical study indicates that, if a less powerful computer is used and the computation time has to be reduced, the most effective way to do so is to reduce N_S to 2, see Figure 5.5. Also note that $N_S = 4$ leads to no further improvement in the lap time, which indicates that $N_S = 3$ is a sufficiently long prediction horizon. This is also confirmed by using a terminal cost which captures the possible long term progress of a state, which improves the performance for $N_S = 2$, but does not help to reduce the lap time if $N_S \geq 3$.

Finally, we see from Table 5.1 that the viability-based controller is marginally faster than the discriminating-based controller. The latter, however, uses the proposed discriminating kernel approximation, which takes into account the discretization of the state space. Thus, the controller (5.9) comes with additional viability guarantees.

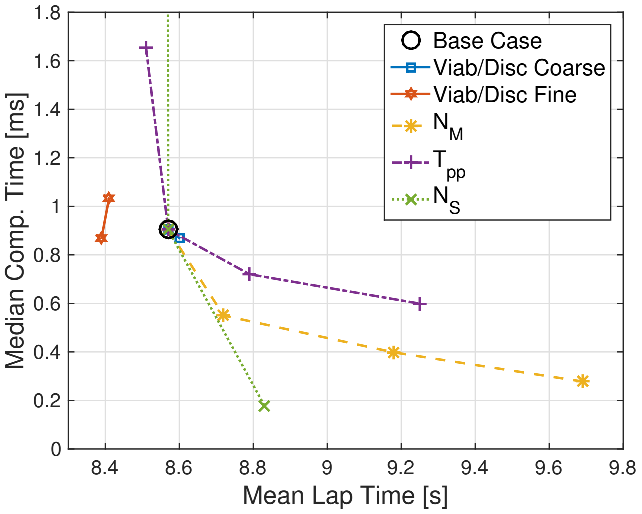

Figure 5.5: Visualization of the investigated cases, with the base case is marked with a circle. Solid lines Table 5.1, dashed line Section 5.5.2, dash-doted line Table 5.3 and doted line Table 5.4. For visualization purposes the data for $\mathcal{C}^{nv}$ and $N_S = 4$ are excluded from the plot.

5.5.6 Closed-loop behavior of $\mathcal{C}_c^v$ and $\mathcal{C}_c^d$

Since the lap time of $\mathcal{C}_c^v$ and $\mathcal{C}_c^d$ are similar, it is interesting to investigate the driving behavior of the two controllers. Figure 5.6 shows one typical lap driven with $\mathcal{C}_c^v$ and $\mathcal{C}_c^d$. The beginning of the lap is in the top left corner marked with a line perpendicular to the track, and the cars race counterclockwise. The velocity is shown relative to center line, to make it possible to compare the velocity at a given point on the track; the projection on the center line is done as discussed in Chapter 4. We see that most of the track, the two controllers drive similarly and even have the same velocity. However, $\mathcal{C}_c^d$ drives somewhat slower coming into the curves, allowing higher velocity on the curve itself and higher exit velocity, the most extreme cases are marked with green bars in Fig 5.6. It is interesting to see that, these two different driving styles lead to practically the same lap time.

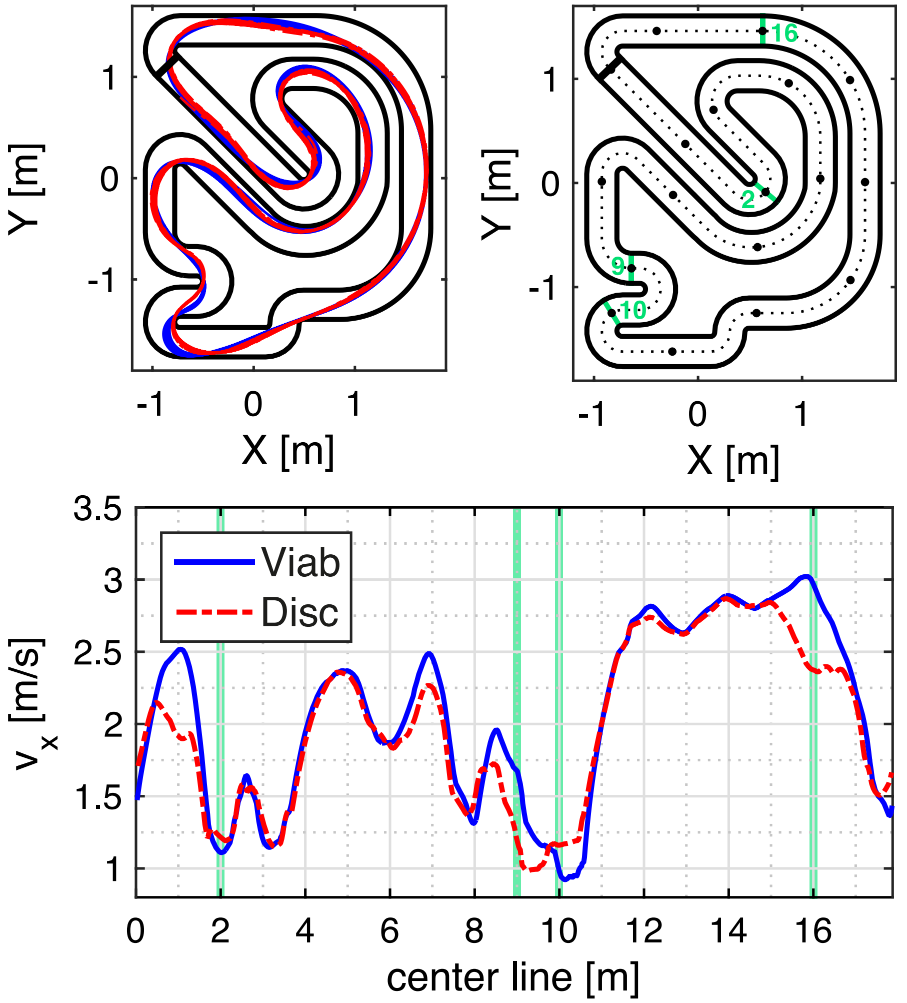

Figure 5.6: Top left: All trajectories of the controller using the viability kernel (in blue), and the discriminating kernel (in dash-dotted red). Top right: To link relative position to the center line, every meter along the center line is marked with a point, whereas points of interest are marked with the length of the center line in green. Bottom: Velocity profile relative to the center line of one lap for both controllers, points of interest from the top right figure are marked with a green bar.

5.5.7 Obstacle avoidance

To further highlight the difference between the two controllers $\mathcal{C}_c^v$ and $\mathcal{C}_c^d$ we included several obstacles at challenging position. The obstacle constraints can be included by modifying the constraint set K and recomputing the viability and discriminating kernel. We tested two obstacle configurations shown in Figure 5.7. For the first configuration (right plot in Figure 5.7) $\mathcal{C}_c^v$ successfully avoids the obstacles, but the discriminating kernel collapses to the empty set. The collapse is due to the conservatism added by considering the space discretization and the fact that the model cannot stop. The second obstacle configuration is easier to navigate, and both controllers are able to find paths around the obstacles. However, $\mathcal{C}_c^v$ is significantly faster than $\mathcal{C}_c^d$ with a mean lap time of 9.066 compared to 9.272 s.

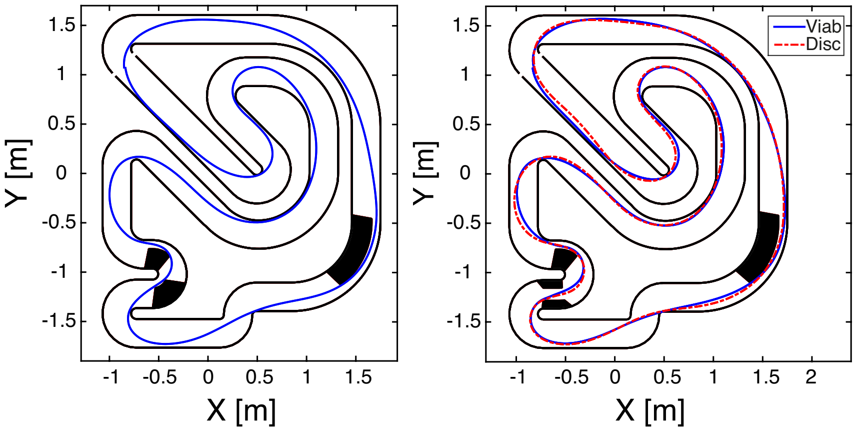

Figure 5.7: The best lap of the controllers for two obstacles= constellations marked in black using the viability kernel (in blue), and the discriminating kernel (in dash-dotted red). In the left obstacle configuration only the viability kernel based controller is able to navigate the course.

Incorporating the obstacles directly in K guarantees that the obstacles can be avoided by the path planning model if the kernel does not collapses. However, for every obstacle configuration, a new kernel needs to be computed. Alternatively, if the position of the obstacle is not a priori known, but the shape of the obstacle is known, viability computations can be used to improve upon only including the obstacle avoidance constraints in the predictive controller (5.9). The idea is to combine the viability kernel for the track constraints with a second viability kernel which only describes from which relative positions an obstacle located at the origin can be avoided. This viability kernel is illustrated in Figure 5.8, where the complement of the kernel is shown or in other words the positions and orientations where a collision cannot be avoided. It is visible that this region is the obstacle itself and a triangularly shaped region in-front which depends on the relative orientation. To include this kernel in the path planning step, the kernel needs to be translated and rotated to the position of the obstacle, and then the obstacle avoidance just becomes a second viability constraint. Note that this does not guarantee recursive feasibility of the obstacle avoidance problem, but improves compared to only use standard obstacle avoidance constraints since it marks states non-viable from which a collision with the obstacle cannot be avoided.

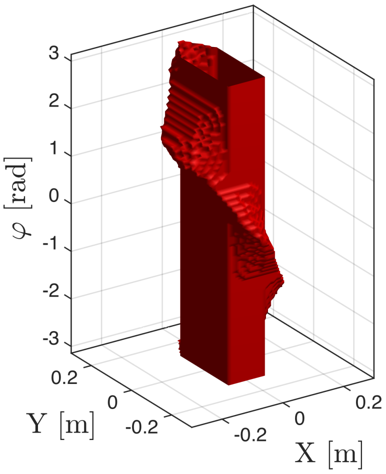

Figure 5.8: Complement of the viability kernel for the relative obstacle avoidance problem, with a car-sized obstacle in at the origin and for the ego car driving straight with 0.5 m/s.

5.6 Experimental results

To verify the simulation results, we implemented $\mathcal{C}_c^v$ and $\mathcal{C}_c^d$ in the experimental set-up described in Chapter 2. The set-up consists of 1:43 Kyosho *dnano* RC cars, which are driven autonomously around the race track shown in Figure 2.1. The control signals are sent to the cars by an external computer via Bluetooth and an infrared-based vision system captures the cars' current position, orientation, and velocity. The controllers were implemented on a desktop computer running Ubuntu 14.04 OS, equipped with 4 GB of RAM and a 3.5 GHz Intel i7 quad-core processor. The tracking MPC problem is solved using FORCES Pro [111].

For the experiments we implemented the two controllers $\mathcal{C}_c^v$ and $\mathcal{C}_c^d$ due to their good performance, and because they are feasible in terms of memory (compared to $\mathcal{C}_f^v$ and $\mathcal{C}_f^d$). We ran the experiment for 200 s and extracted all completed laps, shown in Figure 5.9, together with the velocity profile of one lap. We see from Table 5.5 that constraint violations occur more frequently than in simulation. This is mainly due to a significantly larger model mismatch when using the real car compared to the bicycle model used in simulation. The increased model mismatch can also be seen by the increased spread of the trajectories between the simulation and the experimental results, compare Figure 5.6 and 5.9.

One way to deal with the model mismatch is to tighten the track constraints of the controller to illustrate this we impose track constraints that are 1.5 cm away from the track boundary in the viability computations, and 0.5 cm in the MPC; for comparison the miniature cars are 12 × 5 cm in size. This provides the controller a certain margin of errors but of course comes at the cost of a potential increase in lap time. Constraint violation, are counted when the car is closer than 0.5 cm to the track boundary. The model mismatch can also lead to infeasibilities in the controller. This is, for example, the case for the path planner when the current state is not inside the viability kernel. In such a case we resolve infeasibilities in the path planning process using the following heuristics: First we find a viable neighboring cell and solve the path planning problem pretending to be in the closest viable neighboring grid cell. If none of the neighboring grid cells are viable, then an emergency stop is initiated, and the controller restarts if the car is back in a viable state. Infeasibilities in the lower level MPC problem are dealt with by using soft constraints, as discussed in Chapter 4.

We see from Figure 5.9 that $\mathcal{C}_c^d$ drives more conservatively, a feature we have also seen in the simulation studies before. By breaking earlier, $\mathcal{C}_c^d$ is able to achieve higher velocities on the curves itself and higher exit velocities. In contrast to the simulation, however, the difference in the mean lap times between $\mathcal{C}_c^d$ and $\mathcal{C}_c^v$ is larger, see Table 5.5, though the best and the worst laps are close with lap times in the range 8.66 to 9.54 s for $\mathcal{C}_c^v$ and 8.64 to 9.44 s for $\mathcal{C}_c^d$. Also the number of constraint violations, is nearly identical. For a video comparison of the two controllers, see `https://youtu.be/RlZdMojOniO`.

Table 5.5: Experimental implementation of $\mathcal{C}_c^v$ and $\mathcal{C}_c^d$.

Kernel	median lap time [s]	# constr. violations	comp. time [ms] median	comp. time [ms] max
Viab	8.86	42	1.125	12.936
Disc	8.94	46	1.176	11.055

5.7 Conclusion

In this chapter, we showed that the existing hierarchical racing controller of Chapter 4 can be improved by incorporating the viability kernel in the path planning phase. As a result, the path planner only generates viable trajectories that are recursively feasible, while reducing the computation time. This, in turn, allows the use of longer predic-

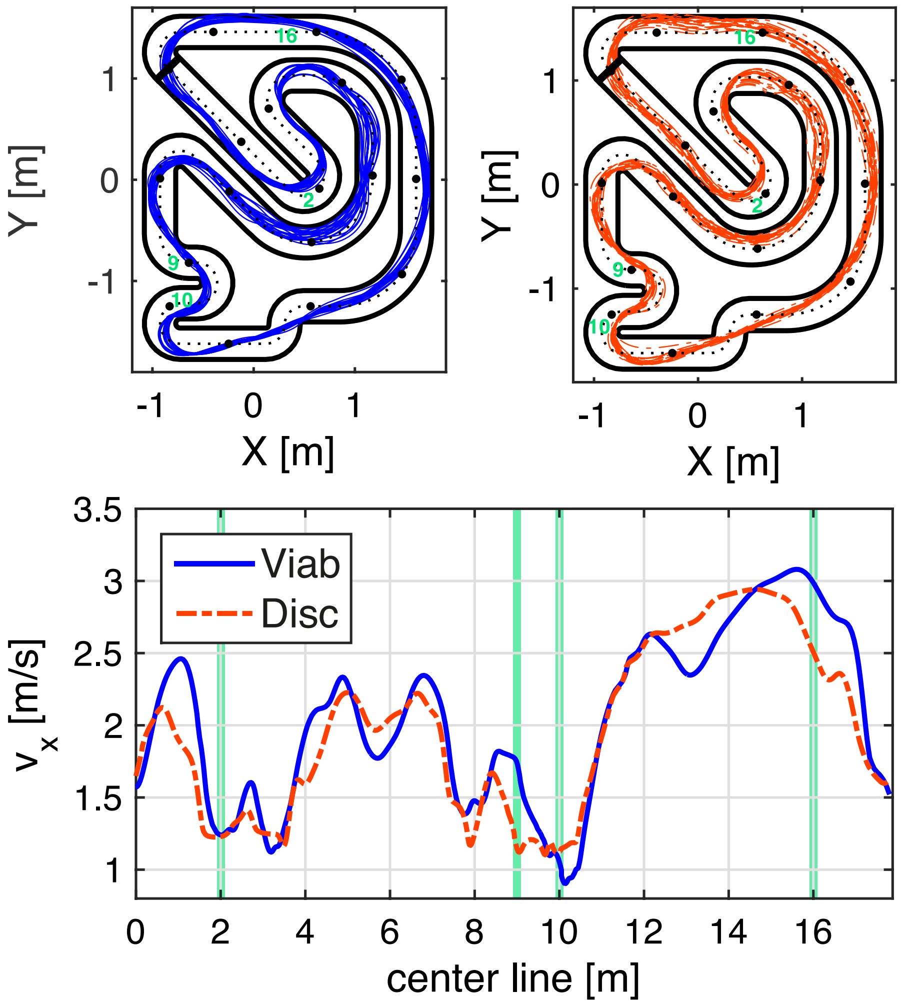

Figure 5.9: All laps of $\mathcal{C}_c^v$ (in blue) and $\mathcal{C}_c^d$ (in dash-doted red) as well as the velocity profile of one lap.

tion horizons, which leads to better performance in terms of lap times and constraint violations.

To compensate for discretization errors in the computation of the viability kernel, we formulated the viability computation problem as a game between the uncertain initial condition introduced by the state discretization and the control input. The resulting kernel was calculated using the discriminating kernel algorithm, which allows us to derive an inner approximation of the normal viability kernel. Furthermore, the new kernel guarantees that there exists a control input that keeps the system within the kernel even if the current state is not on a grid point but somewhere within the cell around a grid point.

In a numerical study, we investigated the influence of different parameters on the performance of the controller using the viability constraints in the path planning phase of our controller and compared the performance of the standard viability kernel is used compared with our inner approximation. Although the closed-loop behavior of the controller using the standard viability kernel and the proposed discriminating kernel approximation are different, the performance in terms of lap time of the two controllers is very similar. The controller based on the proposed discriminating kernel approximation seems to drive with more foresight and is less aggressive. The same behavior was also observed in our experimental implementation.

5.8 Appendix

5.8.1 Finite inner approximation

Recall from (5.4) that the finite viability kernel is an inner approximation of the discrete viability kernel with respect to an extended set-valued map. For the discriminating kernel the authors are not aware of similar results. Even though the basic algorithms (5.2) and (5.5) are similar, the discretization of the disturbance space $V_h \subset V$ required in the discriminating kernel algorithm "weakens" the uncertainty. This may lead to points in the *finite discrete* discriminating kernel $\mathrm{Disc}_{G_h^r}(K_h)$ which are outside the *discrete* discriminating kernel $\mathrm{Disc}_{G^r}(K) \cap X_h$.

In the following we present a method for addressing this problem for our dynamics (5.8). This is achieved by first showing that by only discretizing space, but not the disturbance space, the "semi-finite" discriminating kernel is an inner approximation of $\mathrm{Disc}_{G^r}(K) \cap X_h$. And second by showing that for every discrete disturbance $v_h \in V_h$ there exists a set of disturbances $\tilde{V} \subset V$ which does not change the finite dynamical system $x_{h,k+1} \in G(x_{h,k}, v_{h,k})$. This inside then allows to formulate a modified discriminating kernel algorithm, which guarantees that $\mathrm{Disc}_{G_h^r}(K_h) \subset \mathrm{Disc}_{G^r}(K) \cap X_h$.

Continuous Disturbance Space

Let us first look into the "semi-finite" discriminating kernel where the state space is discretized but the disturbance input is continuous. To this end let us introduce a new set-valued map $\tilde{G}_h^r : X_h \times V \rightsquigarrow X_h$, $\tilde{G}_h^r(x_h, v) := G^r(x_h, v) \cap X_h$, which leads to the following finite dynamical system,

$$x_{h,k+1} \in \tilde{G}_h^r(x_{h,k}, v_k)\,, \tag{5.11}$$

where $v_k \in V$. The following result holds for the new "semi-finite" discriminating kernel $\mathrm{Disc}_{\tilde{G}_h^r}(K_h)$

Proposition 5.4. *Let $G : \mathbb{R}^n \times V \rightsquigarrow \mathbb{R}^n$ be an upper-semicontinuous set-valued map with compact values, K be a closed subset of* $\mathrm{Dom}(G)$, *and V a compact set. Let r be such that, $\forall v \in V\, \forall x \in \mathrm{Dom}(G^r) \cap X_h$, $G^r(x, v) \cap X_h \neq \emptyset$. Then,* $\mathrm{Disc}_{\tilde{G}_h^r}(K_h) \subset \mathrm{Disc}_{G^r}(K) \cap X_h$.

Proof. The proof is an extension of the proof of [39, Proposition 4.1] to the case of the discriminating kernel. We have that for any $v \in V$, $\tilde{G}^r_h(x_h, v) \subset G^r(x_h, v)$ and additionally $K_h \subset K$. By the definition of the discriminating kernel, we know that for all $x_h \in \text{Disc}_{\tilde{G}^r_h}(K_h)$ there exists a solution to the finite dynamical system (5.11) starting from this grid point, which forever stays within K_h, for any sequence of disturbance inputs v_k. As $\tilde{G}^r_h(x_h, v) \subset G^r(x_h, v)$ and $K_h \subset K$, there also exists a solution to $x_{k+1} \in G^r(x_k, v_k)$, starting at the same grid point which stays in K, for any disturbance input sequence. One trivial solution to $x_{k+1} \in G^r(x_k, v_k)$, which fulfills the property is same as the one of the finite dynamical system. Therefore, if $x_h \in \text{Disc}_{\tilde{G}^r_h}(K_h)$ the grid point is also an element of $\text{Disc}_{G^r}(K)$, which concludes the proof. ☐

Proposition 5.4 would allow us to state the discrete space counterpart of Proposition 5.2. However, $\text{Disc}_{\tilde{G}^r_h}(K_h)$ cannot be computed, thus we proceed by establishing a link between $\tilde{G}^r_h(x_h, v)$ and $G^r_h(x_h, v_h)$, which allows to use Proposition 5.4 in the "fully finite" case.

Discrete Disturbance Space

By discretizing the disturbance space $V_h \subset V$, the possible actions of the disturbance input are reduced, which can result in a wrong classification of states to lay within the discriminating kernel. More precisely we can only guarantee that there exists a trajectory to the difference inclusion which stays in K_h for all sequences $v_{h,k}$. However, this does not necessarily hold for all continuous disturbance sequences v_k, as illustrated in Figure 5.10. Even under restrictive assumptions on the used grids, e.g., square grids, and fine disturbance grid, this problem is not solved.

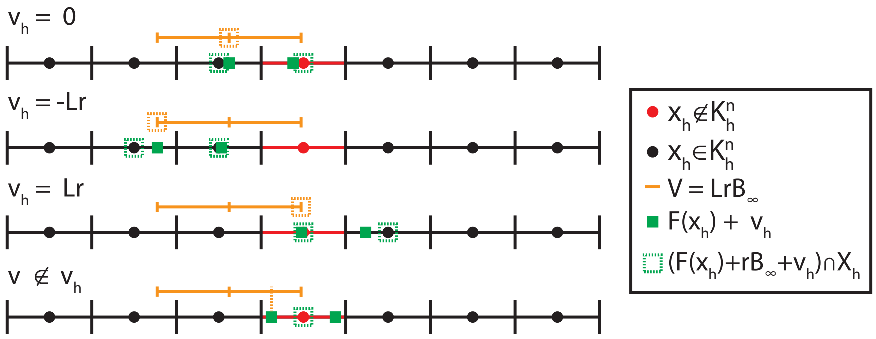

Figure 5.10: Visualization of an instance where the fully finite algorithm gives a wrong answer for a given grid point. In the example a simple 1-D system $F(x)$ with two discrete values is considered. For all finite disturbances $((F(x_h) + r\mathbb{B}_\infty + v_h) \cap X_h) \cap K^n_h$ is non empty, as the first the lines illustrate, however, there exists a $v \notin v_h$, where the condition is not fulfilled.

However, it is possible to establish a link between the continuous and discrete disturbance, for the dynamical system used in Proposition 5.2, $x_{k+1} \in F(x_k) + v_k$ with $v_k \in V = Lr\mathbb{B}_\infty$, where the uncertainty enters additive and the disturbance space is a box. We begin with stating the assumptions on the state and disturbance grid.

Assumption 5.2.

- *$X_h \subset X$ is a square grid with a spacing of $2r$ in the infinity norm.*
- *$V_h \subseteq V$ is a square grid, where any point $v \in V \subset \mathbb{R}^d$ has 2^d neighboring points in V_h which are closer than $2r$ in the infinity norm.*

The assumption on the state grid is necessary to exclude special cases that can arise with non-square grids. The assumption on the disturbance grid generates a grid where $V = Lr\mathbb{B}_\infty$ is gridded with $\lceil L \rceil + 1$ grid points in each direction, including the corner points and grid points at the boundary of the set. Additionally, the disturbance grid spacing is smaller than the state grid spacing. Therefore, it is impossible that two neighboring disturbance grid points can move $F(x_h)$ further than to the next grid cell.

In addition to Assumption 5.2 we need an assumption on the set of allowed controls. As the proposed method will check every element of $F(x)$, it is necessary that $F(x)$ is a finite set, which is the case if the set of admissible controls is finite. To achieve this, the set of control inputs U is discretized, leading to the following finite subset $U_h \subseteq U$. Notice that U_h does not need any additional structure.

Corollary 5.5. *If X_h and V_h fulfill Assumption 5.2, then for all $x_h \in X_h$, $v_h \in V_h$, and $u_h \in U_h$, it holds that*

$$\begin{aligned} x_h^* &= (f(x_h, u_h) + v_h + r\mathbb{B}_\infty) \cap X_h \,, \\ \Leftrightarrow & f(x_h, u_h) + v_h \in x_h^* + r\mathbb{B}_\infty \,. \end{aligned}$$

Proof. It is easy to see that the two statements are equivalent, as in the first statement the point is projected onto a grid point whenever the it is closer than r to this grid point in the infinity norm, and the second statement states that the point is in a cell around the grid point with radius r in the infinity norm. Both statements are only true if $\|x_h^* - (f(x_h, u_h) + v_h)\|_\infty \leq r$ □

From Corollary 5.5, we can see that there exists a neighborhood around v_h, which still leads to the same x_h^*. Thus, for a given x_h, u_h and v_h, we can compute the subset of all continuous disturbances $\tilde{V}(x_h, u_h, v_h) \subseteq V$ which maps to the same state grid point $x_h^* = (f(x_h, u_h) + v_h + r\mathbb{B}_\infty) \cap X_h$,

$$\tilde{V}(x_h, u_h, v_h) = \{v \in V | \|x_h^* - (f(x_h, u_h) + v)\|_\infty \leq r\} \,,$$

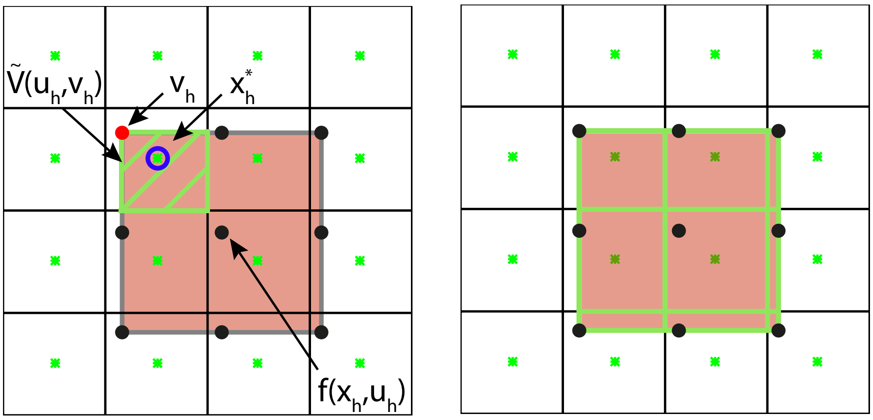

Figure 5.11: Visualization of the box $\tilde{V}(u_h, v_h)$, on the left for one disturbance grid point, and on the right all resulting $\tilde{V}(u_h, v_h)$ boxes for one control input are shown as green boxes.

see Figure 5.11 for an illustration of $\tilde{V}(x_h, u_h, v_h)$. In the interest of readability the dependency of $\tilde{V}$ on the grid point x_h will subsequently be left out, and we just refer to $\tilde{V}(u_h, v_h)$.

Thus, by only looking at finite disturbance v_h, but considering $\tilde{V}(u_h, v_h)$ it is possible to make a statement about the continuous disturbance. In the following, we will discuss how the set $\tilde{V}(u_h, v_h)$ can be used to formulate a modified discriminating kernel algorithm which is equivalent to the "semi-finite" case described in Proposition 5.4.

Given an instance of the discriminating kernel algorithm with a grid point $x_h \in K_h^n$, we can compute the set of all u_h, v_h such that the intersection of $(f(x_h, u_h) + v_h + r\mathbb{B}_\infty) \cap X_h$ and K_h^n is non-empty,

$$\begin{aligned} I_{u,v}(x_h) = \{u_h \in U_h,\ v_h \in V_h| \\ (f(x_h, u_h) + v_h + r\mathbb{B}_\infty) \cap X_h \in K_h^n\}\,. \end{aligned} \tag{5.12}$$

Proposition 5.6. *Given $x_h \in K_h^n$ and $I_{u,v}(x_h)$, it holds that for all $v \in V$ there exists a $u_h \in U_h$ such that*

$$(f(x_h, u_h) + v + r\mathbb{B}_\infty) \cap X_h \in K_h^n\,,$$

if and only if, $\bigcup_{I_{u,v}(x_h)} \tilde{V}(u_h, v_h) = V$

Proof. The *if* part directly follows from the definition, as all continuous disturbances $v \in V$ are considered. The *only if* part is satisfied by Assumption 5.2, which ensures that for a control u_h the union of all $\tilde{V}$ is equal to V. □

Similar to the proof of Proposition 5.2, by using [113, Equation (4)], we know that for the "semi-finite" discriminating kernel it holds that if $x_h \in \mathrm{Disc}_{\tilde{G}_h^r}(K_h)$, for all $v \in V$

there exists a $u_h \in U_h$ such that $(f(x_h, u_h) + v + r\mathbb{B}_\infty) \cap X_h \in \text{Disc}_{\tilde{G}_h^r}(K_h)$. Thus, $\bigcup_{I_{u,v}(x_h)} \tilde{V}(u_h, v_h) = V$ allows us to compute $\text{Disc}_{\tilde{G}_h^r}(K_h)$ by only considering a finite number of disturbances. However, we need to compute $\bigcup_{I_{u,v}(x_h)} \tilde{V}(u_h, v_h) = V$ which may be hard. Thus, we now focus on calculating and approximating of $\bigcup_{I_{u,v}(x_h)} \tilde{V}(u_h, v_h) = V$. Notice first that the set $\tilde{V}(u_h, v_h)$ is always a box. Therefore, they can be parameterized as

$$\tilde{V}(u_h, v_h) = \{v \in V | \underline{v}_j \leq v_j \leq \overline{v}_j, \forall j = 1, ..., d\} .$$

Where, the subscript j refers to each dimension of the uncertainty $V \subset \mathbb{R}^d$ and for each of these dimensions the lower and upper bound can be calculated by,

$$\underline{v}_j(u_h, v_h) = \min(x^*_{h,j} - f(x_h, u_h)_j - r, -Lr) ,$$
$$\overline{v}_j(u_h, v_h) = \max(x^*_{h,j} - f(x_h, u_h)_j + r, Lr) .$$

As the union of boxes is in general not convex, checking that $\bigcup_{I_{u,v}(x_h)} \tilde{V}(u_h, v_h) = V$ is a combinatorial problem. However, by approximating the union conservatively we are able reduce the computational burden while guaranteeing an inner approximation.

To approximate the union notice that $v_h \in \tilde{V}(u_h, v_h)$ always holds. Thus we first find a box approximation for the union of the boxes around each v_h grid point, and then calculate the union of the resulting $|V_h|$ boxes, which can be done by checking if neighboring boxes overlap. To compute a box approximation of the union at one disturbance grid point let us define $I_u(x_h, v_h) = \{u_h \in U_h | \; (f(x_h, u_h) + v_h + r\mathbb{B}_\infty) \cap X_h \in K_h^n\}$, which is a set similar to $I_{u,v}(x_h)$. Here we propose two approximations of $\bigcup_{I_u(x_h,v_h)} \tilde{V}(u_h, v_h)$, first the maximal volume box $\max_{I_u(x_h,v_h)} \text{Vol}(\tilde{V}(u_h, v_h))$ and second the intersection of all boxes $\bigcap_{I_u(x_h,v_h)} \tilde{V}(u_h, v_h)$. Both approximations result in a box and are an inner approximation of the union.

Thus, we propose the modified discriminating kernel algorithm outlined in Algorithm 1. The algorithm guarantees an inner approximation of the discriminating kernel in the following sense, $\text{Disc}_{G_h^r}(K_h) \subset \text{Disc}_{\tilde{G}_h^r}(K_h) \subset \text{Disc}_{G^r}(K) \cap X_h$, the first inner approximation follows due to the approximation of $\bigcup_{I_u(x_h,v_h)} \tilde{V}(u_h, v_h)$ and the second due to Proposition 5.4.

Note that the union approximation is not necessary if one control input is robust with respect to all discrete disturbances. Because by virtue of Assumption 5.2, we know that in this case $\bigcup_{I_{u,v}(x_h)} \tilde{V}(u_h, v_h) = V$.

As the proposed algorithm constructs an inner approximation in the case of a finite disturbance, it is possible to state the following finite version of Proposition (5.2).

Proposition 5.7. *Consider the finite dynamical system corresponding to the extended and discretized system of* (5.8)*,*

$$x_{h,k+1} \in G_h^r(x_{h,k}) = (F(x_{h,k}) + v_h + r\mathbb{B}_\infty) \cap X_h ,$$

Algorithm 1: Modified Discriminating Kernel Algorithm

```
initialization $K_h^0 = K_h$ and $n = -1$;
do
    $n = n + 1$;
    for all $x_h \in K_h^n$ do
        calculate $I_{u,v}(x_h)$, in (5.12);
        calculate the corresponding $\tilde{V}(u_h, v_h)$;
        $\forall v_h \in V_h$ compute $\max_{I_u(x_h, v_h)} \mathrm{Vol}(\tilde{V}(u_h, v_h))$;
        check if neighboring boxes overlap;
        if Yes then
            $x_h \in K_h^{n+1}$
        end
    end
while $K_h^n \neq K_h^{n+1}$;
Result: $\mathrm{Disc}_{G_h^r}(K_h) = K_h^n$
```

where $v_h \in V_h$ *is the discretization of* $Lr\mathbb{B}_\infty$ *according to Assumption 5.2. And if* $\mathrm{Disc}_{G_h^r}(K_h)$ *is computed with the Algorithm 1 and Assumption 5.2 holds, then the following properties hold:*

(1) $\mathrm{Disc}_{G_h^r}(K_h)$ *is a viability domain of* F_h^r.

(2) For all $x_h \in \mathrm{Disc}_{G_h^r}(K_h)$ *and for all* $\hat{x} \in x_h + r\mathbb{B}_\infty$*, there exists a* $u_h \in U_h$ *such that* $f(\hat{x}, u_h) \in \mathrm{Disc}_{G_h^r}(K_h) + r\mathbb{B}_\infty$.

Proof.

(1) Identical to the proof of Proposition 5.2 statement 2), as $\mathrm{Disc}_{G_h^r}(K_h)$ is a discriminating domain and $v = 0$ is an allowed disturbance.

(2) By using the finite version of [113, Equation (4)], and the guarantee of the inner approximation $\mathrm{Disc}_{G_h^r}(K_h) \subset \mathrm{Disc}_{G^r}(K) \cap X_h$ we know that for all $v \in Lr\mathbb{B}_\infty$ there exists a $u_h \in U_h$ such that $(f(x_h, u_h) + v + r\mathbb{B}_\infty) \cap X_h \in \mathrm{Disc}_{G_h^r}(K_h)$. Furthermore, by Corollary 5.5 we can reformulate the discriminating kernel condition as, for all $v \in Lr\mathbb{B}_\infty$ there exists a $u_h \in U_h$ such that $f(x_h, u_h) + v \in \mathrm{Disc}_{G_h^r}(K_h) + r\mathbb{B}_\infty$. The same Lipschitz argument as in the proof of Proposition 5.2, allows to conclude that for all $\hat{x} \in x_h + r\mathbb{B}_\infty$, there exists a $u_h \in U_h$ such that $f(\hat{x}, u_h) \in \mathrm{Disc}_{G_h^r}(K_h) + r\mathbb{B}_\infty$

□

We conclude our discussion by pointing out that if instead of the discriminating kernel the leadership kernel algorithm is used, then the inner approximation is directly guaranteed. Because by virtue of Assumption 5.2, we know that for any robust control input $\bigcup_{I_{u,v}(x_h)} \tilde{V}(u_h, v_h) = V$.

CHAPTER 6

A Non-Cooperative Game Approach to Autonomous Racing

In a car race, the goal is to finish first; mathematically this can be interpreted as a non-cooperative dynamic game where each player tries to reach the finishing line before any other player. In *F1* or similar racing series this game is tackled by separating it into two problems: the first problem concerns slowly changing strategic decisions (pit stops, power consumption, tire wear, etc.) and is solved by the pit crew [115]. These high-level decisions are then executed by a skilled race car driver, who solves the second problem by driving the car at the handling limit while interacting with other cars. In this chapter, we concentrate on the task of the race car driver and formulate it as a dynamic game.

In the previous chapter we showed that the hierarchical racing controller with viability constraints, results in an effective autonomous racing controller. Therefore, our racing game formulation builds on the controller and adds the game theoretical decision making into the high level path planner. More precisely the game is formulated in terms of trajectories, which are given by the path planning model. We propose to model the interactions between race cars as a non-cooperative non-zero-sum game, where the players only get rewarded if they do not cause a collision. By restricting our attention to finite horizon two-player games, we derive three different games that, since the number of trajectories is finite, are formulated as bimatrix games.

The three games have increasingly complex interactions between the players. In the first two games the players maximize their own progress, while avoiding collisions. However, in the first game the leading car is not concerned with collision avoidance and the task is only considered by the following car. In the second game, this sequential structure is dropped and both cars consider the collision constraints. In the third game, the players are not only concerned with maximizing their progress, but also get rewarded to stay ahead at the end of the horizon. This is done to emphasize blocking behavior often seen in racing. Given these three games we investigate the equilibrium solutions of the games and conditions when the equilibria of the games are the same and when they are different. This is especially interesting since the core structures of the games are similar but the interactions between the players are different.

Since this games are played in a receding horizon fashion, similar to the previous chapter not all trajectories are recursively feasible. Therefore we investigate how viability constraints need to be formulated to guarantee recursive feasibility of the moving horizon game. Furthermore, we introduce an exact soft constraint reformulation which guarantees feasibility of the game at all times.

Finally, we investigate the proposed games in a simulation study and investigate how the different formulations influence the racing behavior. Mainly the number of overtaking maneuvers and the collision probability and how these findings are related to the structure of the game.

This chapter is structured as follows, in Section 6.1 we introduce the ingredients to formulate the racing games. Three different racing games are formulated in Section 6.2. The optimal solution in terms of the Stackelberg and Nash equilibrium is studied in Section 6.3. In Section 6.4, we discuss feasibility of moving horizon games. The performance of the approaches is investigated in a simulation study in Section 6.5, while Section 6.6 provides some concluding remarks. Appendix 6.7 contains technical results.

6.1 Game ingredients

To formulate the racing games three ingredients are required, first an appropriate vehicle model, second an objective function representing/approximating the "finish first" goal and third state constraints characterizing collisions.

Building on the previous chapters, we use the path planning model $x_{k+1} = f(x_k, u_k)$ from Chapter 3 as the vehicle model. However, to define the progress we use the extended state progress formulation introduced in Chapter 4, the lap counter c is added to the state leading to $\bar{x} = [x, c] \in \mathbb{R}^5$ and the lap counter update equation $c_{k+1} = \mathcal{C}(x_k, u_k, c_k)$ is appended to the path planning model giving rise to the dynamics of the combined state

$$\bar{x}_{k+1} = \bar{f}(\bar{x}_k, u_k)\,.$$

Finally, given a state $\bar{x}$ we define the progress function

$$\bar{p}(\bar{x}) = \hat{p}(x) + cL\,, \tag{6.1}$$

that will form the basis for the objective function defined below, similar to the previous two chapters.

To encode the objective function of each car as well as the fact that the cars should avoid collisions with each other one needs to consider both cars; we use the superscript $\varrho = 1, 2$ to distinguish the state of the two cars. Furthermore, to formulate the racing games both cars start at their initial state denoted by $\bar{x}^1$ and $\bar{x}^2$, and even though both cars use the path planning model they may be described by different constant velocity modes, highlighted by the superscript in $\bar{f}^\varrho(\cdot,\cdot)$ and U^ϱ. Therefore the dynamics of the two cars are given by,

$$\begin{aligned} &\bar{x}_0^\varrho = \bar{x}^\varrho\,, \quad \varrho = 1,2\,, \quad k = 0,...,N_S-1\,, \\ &\bar{x}_{k+1}^\varrho = \bar{f}^\varrho(\bar{x}_k^\varrho, u_k^\varrho)\,, \quad u_k^\varrho \in U^\varrho(q_k^\varrho)\,. \end{aligned} \tag{6.2}$$

Due to the discrete nature of the admissible inputs $U^\varrho(q^\varrho)$ there exist a finite number of state-input trajectories. These different trajectories are denoted by a subscript $i = 1,...,n$ for car 1 and $j = 1,...,m$ for car 2 and the time step along the trajectory with $k = 1,...,N_S$, resulting in the notation $\bar{x}_i^1(k)$ and $\bar{x}_j^2(k)$ denoting the different trajectories of the two cars at time step k.

Each trajectory of the two cars has an associated progress payoff, which is the progress (6.1) of its final state. The progress payoff of the i-th and j-th trajectory of car 1 respectively car 2 is therefore given by $\bar{p}(\bar{x}_i^1(N_S))$ and $\bar{p}(\bar{x}_j^2(N_S))$. In addition to progress, the objective function of the game should also reflect the fact that the car would like to remain on the track and avoid collisions with other cars. The first requirement can be encoded by a set $\bar{\mathcal{X}}_\text{T} \subseteq \mathbb{R}^5$ that encodes the requirement that the first two components of the state are within the physical limits of the track, similar to (4.1) used in Chapter 4. One can then impose a constraint that for all $k = 1,...,N_S$, $\bar{x}_i^1(k) \in \bar{\mathcal{X}}_\text{T}$, and $\bar{x}_j^2(k) \in \bar{\mathcal{X}}_\text{T}$ (or more precisely, penalize the car in the objective function whenever this constraint is violated).

Finally the collision constraints couple the two cars, in the sense that they depend on the actions of both players. One can encode this constraint through the signed distance[1] [116]. If we assume that each car can be described by a rotated rectangle (indeed any fixed shape) centred at the (X, Y, φ) component of the state, then the signed distance of the corresponding sets induces a function $d : \mathbb{R}^5 \times \mathbb{R}^5 \to \mathbb{R}$ such that $d(\bar{x}^1, \bar{x}^2) \geq 0$ if and only if the two cars do not physically overlap. Thus, a trajectory pair (i, j) does not have a collision if

$$d(\bar{x}_i^1(k), \bar{x}_j^2(k)) \geq 0\,, \quad k = 1,...,N_S\,. \tag{6.3}$$

6.2 Game formulation

In this section the idea is to combine the game ingredients discussed in Section 6.1 to formulate bimatrix games representing the racing objective. Therefore, let us assume

---

[1] Where the singed distance between two sets $R^1 \in \mathbb{R}^2$ and $R^2 \in \mathbb{R}^2$ is defined as $sd(R^1, R^2) = \text{dist}(R^1, R^2) - \text{pert}(R^1, R^2)$, with $\text{dist}(R^1, R^2) = \inf\{\|T\| | (T + R^1) \cup R^2 \neq \emptyset\}$ and $\text{pert}(R^1, R^2) = \inf\{\|T\| | (T + R^1) \cup R^2 = \emptyset\}$.

that the two cars are the players; player 1 (P1) has n possible trajectories, and player 2 (P2) has m possible trajectories to choose from, constructed by combinatorially selecting inputs from the corresponding sets $U^1(q^1)$ and $U^2(q^2)$. The bimatrix game has two $m \times n$ payoff matrices A and B with elements $a_{i,j}$ and $b_{i,j}$ representing, respectively, the payoffs of P1 and P2, if P1 adopts trajectory i and P2 trajectory j. The payoff associated with a trajectory pair (i, j) encodes information about both the progress and the constraints.

Before formally introducing our different racing games, let us give two definitions and an assumption used in the rest of the chapter.

Definition 6.1. *A trajectory pair* (i, j) *is* feasible *if* $\bar{x}^1_i(k) \in \bar{\mathcal{X}}_T$, $\bar{x}^2_j(k) \in \bar{\mathcal{X}}_T$, *and* $d(\bar{x}^1_i(k), \bar{x}^2_j(k)) \geq 0$ *for all* $k = 1, ..., N_S$.

Definition 6.2. *The leader of the game is the car which is ahead at the beginning of the game, in other words P1 if* $\bar{p}(\bar{x}^1(0)) \geq \bar{p}(\bar{x}^2(0))$ *and P2 otherwise.*

Without loss of generality we assume that P1 is the leader of the game.

Assumption 6.1. *At the beginning of the game P1 is ahead of P2 or in other words* $\bar{p}(\bar{x}^1(0)) \geq \bar{p}(\bar{x}^2(0))$.

In following we discuss three different approaches to combine the progress payoff, collision, and track constraints to formulate three different bimatrix games, with increasingly complex interactions between the players. In the first two games, the winning objective is encoded as “drive as fast as possible while avoiding collision with other cars”. In the third game, a more complex payoff structure is used to promote blocking behavior, or in other words prevent an overtaking maneuver by driving sub-optimally with respect to the progress along the track. For simplicity, we consider racing of only two cars. Racing with multiple opponents can be treated similarly but is considerably more challenging from a computational point of view.

6.2.1 Sequential game

In the first game, the payoff matrices are designed such that the players stay on track, drive as fast as possible and avoid collisions. Roughly speaking, feasible trajectory pairs receive a payoff equal to their progress if a trajectory leaves the track, this trajectory receives a constant payoff of κ strictly smaller than the payoff of any trajectory which remains within the track. Likewise, trajectory pairs that collide also receive a low payoff. However, due to the leader-follower structure present in racing, we assume that only the follower (P2) is concerned with collision avoidance. Therefore, if a trajectory pair has a collision, P2 receives a low payoff of λ, whereas P1 receives the payoff as though the collision was not present. The resulting payoff matrices can be computed as follows,

$$
\begin{aligned}
a_{i,j} &= \begin{cases} \kappa & \text{if } \exists k \in \{1, ..., N_S\} : \bar{x}_i^1(k) \notin \bar{\mathcal{X}}_{\mathrm{T}} \\ \bar{p}(\bar{x}_i^1(N_S)) & \text{else} \end{cases}, \\
b_{i,j} &= \begin{cases} \kappa & \text{if } \exists k \in \{1, ..., N_S\} : \bar{x}_j^2(k) \notin \bar{\mathcal{X}}_{\mathrm{T}} \\ \lambda & \begin{aligned} &\text{else if} \quad \exists k \in \{1, ..., N_S\} : \\ &\qquad d(\bar{x}_i^1(k), \bar{x}_j^2(k)) < 0 \end{aligned} \\ \bar{p}(\bar{x}_j^2(N_S)) & \text{else} \end{cases}.
\end{aligned}
\tag{6.4}
$$

Intuitively, if κ and λ are smaller than $\min(\min_i(\bar{p}(\bar{x}_i^1(N_S)), \min_j \bar{p}(\bar{x}_j^2(N_S)))$ both player, if possible, choose a trajectory which remains within the track and P2 will avoid a collision, see Section 6.3 for a formal discussion. This condition can be fulfilled by choosing $\kappa, \lambda < 0$, as the progress is always positive. For the rest of the chapter, we assume that $0 > \lambda \geq \kappa$, which implicitly penalizes collisions less than leaving the track.

Note that in this game the payoff of the leader does not depend on the actions of the follower; in other words, all elements of a row of the payoff matrix A will have the same value. We denote by a_i the leader payoff for action $i \in \Gamma_1$ (in other words, the value of all elements of row i in matrix A) to highlight the fact that it does not depend on the actions of the follower. Due to this purely sequential structure the game is called the *sequential game*.

To visualize the sequential game (6.4) let us look at one racing situation with only three possible trajectories for each player, see Figure 6.1. We determine the progress by assuming that the zero point is at the beginning of the track interval shown and that the interval is 0.95 m long, with 12 cm long cars, which corresponds to the test track introduced in Chapter 2 and the scale used in the subsequent simulation study. We use $\kappa = -10$ and $\lambda = -1$ leading to the payoff matrices A (for the leader P1) and B (for P2) of the racing game (6.4)

$$
A = \begin{bmatrix} 0.83 & 0.83 & 0.83 \\ 0.88 & 0.88 & 0.88 \\ -10 & -10 & -10 \end{bmatrix} \; B = \begin{bmatrix} 0.81 & 0.86 & -10 \\ 0.81 & -1 & -10 \\ 0.81 & 0.86 & -10 \end{bmatrix}. \tag{6.5}
$$

For each player, the bottom trajectory leaves the track, so the payoff is $\kappa = -10$ for the corresponding trajectory, irrespective of the actions of the other player. This leads to the third row of A and third column of B identically equal to -10. If both players select their middle trajectory, there is a collision, giving rise to the payoff $\lambda = -1$ in the $(2, 2)$ entry of B, note that $a_{2,2}$ is equal to the progress. The remaining combinations of actions neither leave the track nor cause a collision. Hence the payoff is the progress, giving rise to the entries in the order of 0.8 in the two matrices.

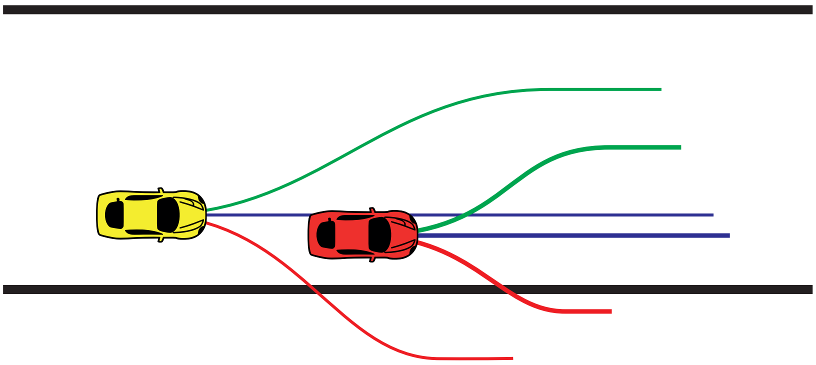

Figure 6.1: The leader P1 (red car) is in front of P2 (yellow car). Each player has three trajectories green, blue, red from top to bottom, and the trajectories are numbered from top (1) to bottom (3).

6.2.2 Cooperative game

Compared to the sequential game in the cooperative game, the leader does also consider collisions. This is incorporated into the game by modifying the payoff such that also the leader receives a payoff of λ if there is a collision. In other words, both players only receive the progress payoff if a trajectory pair is feasible. The resulting payoff matrices can be computed as follows,

$$
\begin{aligned}
a_{i,j} &= \begin{cases} \kappa & \text{if } \exists k \in \{1, \ldots, N_S\} : \bar{x}_i^1(k) \notin \bar{\mathcal{X}}_\mathrm{T} \\ \lambda & \begin{aligned}\text{else if} \quad & \exists k \in \{1, \ldots, N_S\} : \\ & d(\bar{x}_i^1(k), \bar{x}_j^2(k)) < 0 \end{aligned} \\ \bar{p}(\bar{x}_i^1(N_S)) & \text{else} \end{cases}, \\
b_{i,j} &= \begin{cases} \kappa & \text{if } \exists k \in \{1, \ldots, N_S\} : \bar{x}_j^2(k) \notin \bar{\mathcal{X}}_\mathrm{T} \\ \lambda & \begin{aligned}\text{else if} \quad & \exists k \in \{1, \ldots, N_S\} : \\ & d(\bar{x}_i^1(k), \bar{x}_j^2(k)) < 0 \end{aligned} \\ \bar{p}(\bar{x}_j^2(N_S)) & \text{else} \end{cases},
\end{aligned}
\tag{6.6}
$$

Note that if we choose $\kappa, \lambda < 0$ as suggested in the sequential game, in case of the cooperative game the optimal trajectory pair will be feasible whenever possible since none of the players would benefit from violating the constraints, see Theorem 6.1 for a formal discussion. Due to this cooperative obstacle avoidance approach, we call it the *cooperative game.*

When investigating the racing situation in Figure 6.1, the payoff matrices A and B

for the cooperative game are

$$A = \begin{bmatrix} 0.83 & 0.83 & 0.83 \\ 0.88 & -1 & 0.88 \\ -10 & -10 & -10 \end{bmatrix} \; B = \begin{bmatrix} 0.81 & 0.86 & -10 \\ 0.81 & -1 & -10 \\ 0.81 & 0.86 & -10 \end{bmatrix}. \tag{6.7}$$

Compared to the payoff matrices of the sequential game (6.5) the only difference is $a_{2,2}$ which is now -1, since P1 also considers collisions.

6.2.3 Blocking game

When racing, maximizing progress is only a means to an end, the real objective is to finish first. In some cases, it may, therefore, be beneficial to prevent the opponent from overtaking, even if that means less progress. We refer to this behavior as "blocking". In our third game, we modify the payoff matrices of the cooperative game (6.6) to reward staying ahead at the end of the horizon, giving rise to the so-called *blocking game*. More precisely we add a parameter $w \geq 0$, which allows to trade-off staying ahead and maximizing progress. The resulting payoff matrices can be computed as follows,

$$\begin{aligned} a_{i,j} &= \begin{cases} \kappa & \text{if } \exists k \in \{1, ..., N_S\} : \bar{x}_i^1(k) \notin \bar{\mathcal{X}}_\mathrm{T} \\ \lambda & \begin{aligned} &\text{else if } \;\; \exists k \in \{1, ..., N_S\} : \\ &\qquad d(\bar{x}_i^1(k), \bar{x}_j^2(k)) < 0 \end{aligned} \\ \bar{p}(\bar{x}_i^1(N_S)) & \text{else if } \;\; \bar{p}(\bar{x}_i^1(N_S)) < \bar{p}(\bar{x}_j^2(N_S)) \\ \bar{p}(\bar{x}_i^1(N_S)) + w & \text{else} \end{cases}, \\ b_{i,j} &= \begin{cases} \kappa & \text{if } \exists k \in \{1, ..., N_S\} : \bar{x}_j^2(k) \notin \bar{\mathcal{X}}_\mathrm{T} \\ \lambda & \begin{aligned} &\text{else if } \;\; \exists k \in \{1, ..., N_S\} : \\ &\qquad d(\bar{x}_i^1(k), \bar{x}_j^2(k)) < 0 \end{aligned} \\ \bar{p}(\bar{x}_j^2(N_S)) & \text{else if } \;\; \bar{p}(\bar{x}_i^1(N_S)) \geq \bar{p}(\bar{x}_j^2(N_S)) \\ \bar{p}(\bar{x}_j^2(N_S)) + w & \text{else} \end{cases}. \end{aligned} \tag{6.8}$$

Note that in this case there are two terms linking the payoff of the two players, the collision constraint in the second line in (6.8) and the blocking reward in the fourth line.

To visualize the effect of the additional payoff term, we look at a slightly more complex racing situation illustrated in Figure 6.2, where compared to the first racing situation in Figure 6.1 both players have one additional trajectory and P2 has the chance to overtake P1.

The payoff matrices of the bimatrix game corresponding to the racing situation in

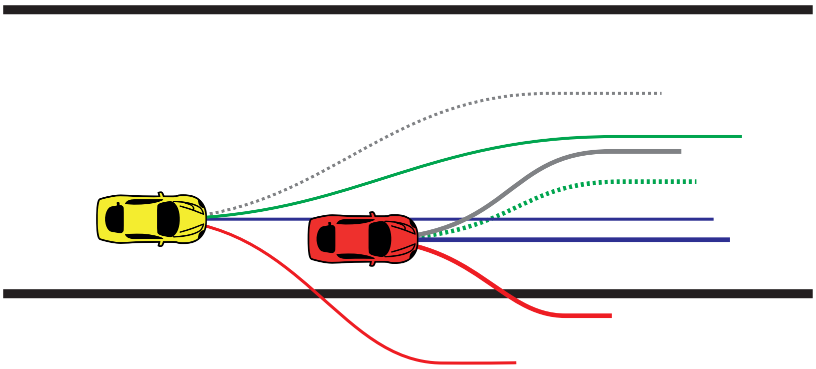

Figure 6.2: Illustration of a blocking situation, where P2 (yellow car) has a chance to overtake P1 (red car) at the end of the horizon by choosing the green trajectory (2). However, by choosing the dashed green blocking trajectory P1 can avoid an overtaking maneuver, forcing P2 to choose his gray dashed trajectory (1) to avoid a collision. The trajectories are again numbered from top to bottom.

Figure 6.2, with $w = 0.5$ are,

$$A = \begin{bmatrix} 0.83+0.5 & -1 & 0.83 & 0.83+0.5 \\ 0.85+0.5 & -1 & -1 & 0.85+0.5 \\ 0.88+0.5 & 0.88 & -1 & 0.88+0.5 \\ -10 & -10 & -10 & -10 \end{bmatrix},$$

$$B = \begin{bmatrix} 0.81 & -1 & 0.86+0.5 & -10 \\ 0.81 & -1 & -1 & -10 \\ 0.81 & 0.9+0.5 & -1 & -10 \\ 0.81+0.5 & 0.9+0.5 & 0.86+0.5 & -10 \end{bmatrix}. \tag{6.9}$$

6.3 Characterization of equilibria

6.3.1 Equilibrium concepts and feasibility

Given the bimatrix games (6.4), (6.6), and (6.8), the question is how to find the optimal trajectory pair. Note that we only consider pure strategies, where the action space of the two players is given by $\Gamma^1 := \{1, 2, .., n\}$ and $\Gamma^2 := \{1, 2, .., m\}$. Further, we consider two types of equilibria, the classical Stackelberg and Nash.

Stackelberg Equilibrium

In the Stackelberg equilibrium, there exists a leader, and a follower and the leader can enforce his strategy on the follower. It is assumed that the follower plays rationally, in the sense that he plays the best response with respect to the strategy of the leader. Thus, the leader chooses the strategy which maximizes his payoff given the best response of the follower.

Definition 6.3 ([117])**.** *A strategy pair* $(i^*, j^*) \in \Gamma^1 \times \Gamma^2$ *is a Stackelberg equilibrium of a bimatrix game A, B, with P1 as leader, if:*

$$i^* = \arg\max_{i \in \Gamma^1} \min_{j \in R(i)} a_{i,j}\,, \quad j^* = \arg\max_{j \in \Gamma^2} b_{i^*,j}\,,$$

where $R(i) := \arg\max_{j \in \Gamma^2} b_{i,j}$ *is the best response of P2 to the strategy* $i \in \Gamma^1$ *of P1.*

Note that the best trajectories of both the leader and the follower may be non-unique. For the leader all optimal trajectories lead to the same pay-off; the leader can, therefore, pick anyone among them and announce it to the follower [117]. The follower can then select any one of the (possibly many) best responses to this announced trajectory of the leader.

Nash Equilibrium

The Nash equilibrium is a trajectory pair, such that there is no incentive for either of the players to deviate unless the other player does.

Definition 6.4 ([117])**.** *A strategy pair* $(i^*, j^*) \in \Gamma^1 \times \Gamma^2$ *is a Nash equilibrium of a bimatrix game A, B if the following inequalities are fulfilled:*

$$\begin{aligned} a_{i^*,j^*} &\geq a_{i,j^*} \quad \forall i \in \Gamma^1\,, \\ b_{i^*,j^*} &\geq b_{i^*,j} \quad \forall j \in \Gamma^2\,. \end{aligned}$$

We note that, unlike zero sum games, existence of Nash pure strategy equilibria (as Definition 6.4) is not guaranteed for our bimatrix games, even though the payoff matrices are bounded; we address the issue of existence of pure strategy Nash equilibria below. Moreover, if Nash equilibria do exists, there may be multiple of them with potentially different payoffs for the two players [117]. To distinguish between them we use the notion of "betterness" following terminology of [117].

Definition 6.5 ([117])**.** *A strategy pair* (i^1, j^1) *is said to be better than another strategy pair* (i^2, j^2)*, if* $a_{i^1,j^1} \geq a_{i^2,j^2}$*,* $b_{i^1,j^1} \geq b_{i^2,j^2}$*, and at least one of the inequalities is strict.*

If neither equilibrium is better than the other we then say that the two equilibria are incomparable. The existence of better equilibria forms the basis for a non-cooperative equilibrium consensus [117].

Feasibility Assumptions

Since feasibility of a trajectory pair is fundamental for the game, but the constraints are only considered in the payoff, we investigate assumptions under which a feasible trajectory pair is obtained.

Assumption 6.2.a. *There exists a feasible trajectory pair* (i, j) *according to Definition 6.1.*

To establish a connection between the sequential and the cooperative game we need a more restrictive assumption involving the payoff of the leader (P1) in the sequential game (6.4).

Assumption 6.2.b. *There exists a trajectory pair* (i, j)*, such that* $\max_{i \in \Gamma^1} a_i > \kappa$ *and* $\max_{j \in \Gamma^2} \min_{i' \in \arg\max_{i \in \Gamma_1}} b_{i',j} > \lambda$.

Note that, the payoff of the follower is identical in both the cooperative game (6.6) and the sequential game (6.4), thus the assumption is well defined for both games. One can easily verify that Assumption 6.2.b is more restrictive, since any cooperative game which fulfills Assumption 6.2.b also fulfills Assumption 6.2.a, however, the opposite does not necessarily hold. We note that Assumption 6.2.b is not always fulfilled in practice, but the assumption is necessary for our technical results.

6.3.2 Equilibrium of the sequential game

When investigating the sequential game (6.4) in more detail one can see that it is not really a game since the decisions of the follower do not influence the decisions of the leader. Thus the optimal trajectory pair can simply be computed by a sequential maximization approach, where the leader selects an optimal trajectory without considering the follower. Similar to the Stackelberg equilibrium this trajectory is then announced to the follower, who selects a best response. This can be summarized in the following two-step sequential maximization, which determines the sequential optimal trajectory pair (i^s, j^s),

$$i^s = \arg\max_{i \in \Gamma^1} a_i \,, \quad j^s = \arg\max_{j \in \Gamma^2} b_{i^s,j} \,. \tag{6.10}$$

One can show that trajectory pairs (i^s, j^s) that satisfy (6.10) are both Stackelberg and Nash equilibria of the sequential game (6.4); the proof is similar to that of Theorem 6.1 and is omitted in the interest of space. Moreover, if Assumption 6.2.b holds, these trajectory pairs are also feasible; this follows directly from Assumption 6.2.b, which guarantees that the follower has a feasible response to the announced optimal trajectory of the leader. Once again multiple optimal trajectories for the leader do not pose a problem since their payoffs are the same and the chosen trajectory is announced to the follower.

In the racing situation in Figure 6.1 Assumption 6.2.b is fulfilled and the sequential optimal trajectory pair is $(2, 1)$, where the car in front goes straight and the car in the back avoids the car in front by going left. It is easy to verify that this trajectory pair is indeed a Nash and a Stackelberg equilibrium for (6.5).

6.3.3 Equilibrium of the cooperative game

Let us now focus on the more complex cooperative game (6.6) which does not exhibit the sequential structure of (6.4). When investigating the racing situation in Figure 6.1 with the corresponding payoff matrices (6.7), we see that the Stackelberg equilibrium of the game is the trajectory pair $(2, 1)$, which is identical to the sequential optimal trajectory pair. Furthermore, we can also verify that the sequential optimal trajectory pair is also a Nash equilibrium. Note that this is not the only Nash equilibrium, however: the trajectory pair $(1, 2)$ is also a Nash equilibrium[2].

A natural question that arises is how can the two cars choose one of the two Nash equilibria. This is generally a difficult question in game theory. In our example, if each player assumes that the equilibrium that maximizes his own progress will be chosen, this would result in the trajectory pair (2,2) which is not a Nash equilibrium and results in a collision. This is because none of the Nash equilibria is better than the other. To resolve this issue ahead of the game the players can agree on rules about how to pick a Nash equilibrium in the case of multiple equilibria; we call those "rules of the road". Here we use a rule that says that if there are multiple Nash equilibria, the equilibrium with the largest payoff for the leader (P1) is chosen. In the racing situation in Figure 6.1, the Nash equilibrium which fulfills the rules of the road is the trajectory pair $(2, 1)$.

Note that in this case all the different equilibrium concepts, as well as the sequential game, lead to the same *feasible* trajectory pair. Indeed one can show that under Assumption 6.2.b this property holds generally.

Theorem 6.1.

(a) *If Assumption 6.2.a holds, all Stackelberg equilibria are feasible, there exists at least one feasible Nash equilibrium and all feasible Nash equilibria are better than all infeasible Nash equilibria.*

(b) *Let Π_s denote the set of all Nash/Stackelberg equilibria of the sequential game, Π_{st} the set of all Stackelberg equilibria of the cooperative game, $\Pi_{n,RoR}$ the set of all*

---

[2] Note that in this example the best-response dynamics as proposed in [58] would cycle between trajectory pair $(2, 2)$ and $(1, 1)$.

Nash equilibria of the cooperative game that fulfill the rules of the road, and Π_n the set of all Nash equilibria of the cooperative game. If Assumption 6.2.b holds, then

$$\emptyset \neq \Pi_s = \Pi_{st} = \Pi_{n,RoR} \subseteq \Pi_n\,.$$

Proof. We start by noting that due to the payoff structure in the cooperative game (6.6), specifically the symmetry of the collision constraints, it holds that

$$a_{i,j} = \lambda \Leftrightarrow b_{i,j} = \lambda\,. \tag{6.11}$$

Part (a): Under Assumption 6.2.a there exists $i \in \Gamma^1$ such that $b_{i,j} > \lambda$ for all $j \in R(i)$, hence by (6.11) $\max_{i\in\Gamma^1} \min_{j\in R(i)} a_{i,j} > \lambda$ and therefore all Stackelberg equilibria are feasible.

We now show that the Stackelberg equilibrium is a Nash equilibrium, which by virtue of the previous point shows that a feasible Nash equilibrium exists. Note that $\max_{i\in\Gamma^1} \min_{j\in R(i)} a_{i,j} = \max_{i\in\Gamma^1} \max_{j\in\Gamma^2} a_{i,j}$ due to the payoff structure. This observation together with P2 playing best response shows that the Nash inequalities are fulfilled.

Finally, note that for any infeasible Nash equilibrium at least one of the two players is not using their feasible trajectory, as otherwise, the Nash would be feasible. Moreover, none of the two players violates the track constraint; if they did, they could improve their payoff from κ to (at least) λ by switching to another trajectory, contradicting Nash. Hence neither is using their trajectory from the feasible pair, as if one was the other could unilaterally improve their payoff by also switching to their feasible trajectory, contradicting Nash again. Therefore an infeasible Nash equilibrium only exists if both players have a payoff of λ but for both players, any unilateral change would result in a payoff of λ or κ. However, any feasible Nash equilibrium is better than such an infeasible Nash equilibrium, as in the feasible case both players receive the progress payoff, which is by definition greater than λ.

Part (b): If Assumption 6.2.b holds then $b_{i^s,j} > \lambda$ for all $j \in R(i^s)$, which again implies by (6.11) that $\min_{j\in R(i^s)} a_{i^s,j} = \bar{p}(\bar{x}^1_{i^s}(N_S))$. This also shows that the sequential optimal trajectory pair is feasible and the set of trajectory pairs is non-empty.

Furthermore, the sequential optimal trajectory pair (i^s, j^s) is identical to the Stackelberg equilibrium since by the above observation $\max_{i\in\Gamma^1} \min_{j\in R(i)} a_{i,j} = \max_{i\in\Gamma^1} a_i$. Thus, we can reformulate the Stackelberg equilibrium as,

$$\begin{aligned} i^* &= \arg\max_{i\in\Gamma^1} \min_{j\in R(i)} a_{i,j} = \arg\max_{i\in\Gamma^1} a_i\,, \\ j^* &= \arg\max_{j\in\Gamma^2} b_{i^*,j} \end{aligned}$$

which is identical to the sequential maximization approach (6.10) and proves the second equality.

Finally, we show that the sequential optimal trajectory pair (i^s, j^s) is a Nash equilibrium that fulfills the rules of the road. We know that $a_{i^s,j^s} = \bar{p}(\bar{x}^1_{i^s}(N_S))$, leading to

$$\begin{aligned} a_{i^s,j^s} &= \bar{p}(\bar{x}^1_{i^s}(N_S)) \geq a_{i,j^s} \quad \forall i \in \Gamma^1 \,, \\ b_{i^s,j^s} &= b_{i^s,R(i^s)} \geq b_{i^s,j} \quad \forall j \in \Gamma^2 \,. \end{aligned}$$

The first inequality holds since $\bar{p}(\bar{x}^1_{i^s}(N_S))$ is the best possible payoff for P1, the second because P2 plays best response. Therefore, the trajectory pair (i^s, j^s) is a Nash equilibrium. Furthermore, the trajectory pair (i^s, j^s) is the Nash equilibrium which fulfills the rules of the road, as it yields to the best possible payoff for P1, which shows the last two relationships. □

To illustrate the proof of Theorem 6.1 (a), consider the racing situation depicted in Figure 6.3 with the payoff matrices in (6.12). In this case, two Nash equilibria exist, (2, 2) which is feasible with a payoff of (0.87, 0.89), and (1, 3) which is infeasible with a payoff of (−1, −1). The feasible Nash is better than the infeasible one and is thus preferable for both players.

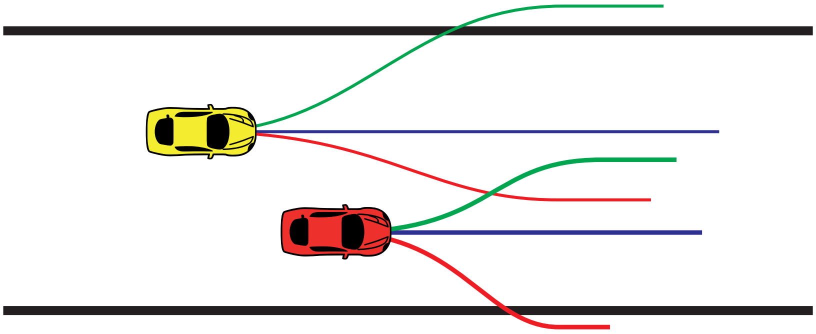

Figure 6.3: A racing game with an infeasible Nash equilibrium. Each player has three trajectories green, blue, red from top to bottom, and the trajectories are numbered from top (1) to bottom (3). The green trajectory (1) of P1 (red car) collides both with the blue (2) and the red (3) trajectory of P2 (yellow car) and the blue trajectory (2) of P1 also collides with the red trajectory (3) of P2.

$$A = \begin{bmatrix} 0.84 & -1 & -1 \\ 0.87 & 0.87 & -1 \\ -10 & -10 & -10 \end{bmatrix} \quad B = \begin{bmatrix} -10 & -1 & -1 \\ -10 & 0.89 & -1 \\ -10 & 0.81 & 0.81 \end{bmatrix} . \tag{6.12}$$

Thanks to Theorem 6.1, if Assumption 6.2.b holds we can use the sequential maximization approach (6.10) to compute Nash and Stackelberg equilibria. This can offer significant computational advantages since it is not necessary to generate all entries of A and B. Moreover, collision checks are only necessary between the optimal trajectory of the leader and the possible trajectories of the follower; this dramatically reduces the computational burden as we will discuss in Section 6.5. We note that the only structure necessary for Theorem 6.1 is the fact that the decisions of the players are only linked through the constraints as the progress payoff solely depends on the players own action. Thus, Theorem 6.1 would still hold if one uses objective functions other than progress, e.g., fuel efficiency, or even in other applications that enjoy a similar structure. This suggests that the theorem may have wider applicability, as many robotic multi-agent problems with collision constraints can be formulated such that they have the same structure.

Note that if Assumption 6.2.b is not fulfilled, in the cooperative game the leader will choose a trajectory which allows the follower to make a choice such that the resulting trajectory pair is feasible (as long as Assumption 6.2.a holds). On the other hand, in the sequential game the leader does not consider the follower and a trajectory pair that satisfies (6.10) may be not feasible even if Assumption 6.2.a is satisfied.

6.3.4 Equilibrium of the blocking game

Finally let us investigate the blocking game. To formalize the discussion, we start by defining a blocking trajectory pair. Let (i^{cg}, j^{cg}) denote a Stackelberg equilibrium of the cooperative game (6.6). Note that if there are multiple Stackelberg equilibria, all these trajectory pairs have the same progress payoff. Therefore, all the progress based arguments still hold in the case of multiple equilibria.

Definition 6.6. *A trajectory pair* (i, j) *is blocking if the following properties hold:* (i) *the Stackelberg equilibrium of the cooperative game* (i^{cg}, j^{cg}) *is such that P1 gets overtaken at the end of the horizon* $\bar{p}(\bar{x}^1_{i^{cg}}(N_S)) < \bar{p}(\bar{x}^2_{j^{cg}}(N_S))$, (ii) *the pair* (i, j) *is such that P1 does not get overtaken* $\bar{p}(\bar{x}^1_i(N_S)) > \bar{p}(\bar{x}^2_j(N_S))$, (iii) *the pair* (i, j) *is feasible* $a_{i,j} > \lambda$ *and* $b_{i,j} > \lambda$, (iv) *for all* $j^c \in \Gamma^2$ *such that* $\bar{p}(\bar{x}^2_{j^c}(N_S)) > \bar{p}(\bar{x}^2_j(N_S))$, *it holds that* $b_{i,j^c} \leq \lambda$.

A blocking trajectory pair corresponds to a collision-free trajectory pair where P1 is ahead at the end of the horizon. P1 achieves this by choosing a trajectory, such that any trajectory of P2 that would achieve a larger progress collides with this trajectory of P1. The last condition also implies that P2 plays a best response, as any other trajectory would lead to a smaller payoff. Among all blocking trajectory pairs fulfilling properties (i)-(iv), let (i^b, j^b) denote the one which achieves the largest progress for the leader.

When investigating the racing situation in Figure 6.2 with the payoff matrices (6.9), we can see that the additional payoff term promotes blocking behavior since the Stackelberg equilibrium $(2, 1)$ of the game is a blocking trajectory where P1 drives a curve to prevent P2 from overtaking at the end of the horizon. In contrast, the (unique) Nash equilibrium is $(3, 2)$, where P1 drives straight to maximize progress, but P2 can overtake P1 at the end of the horizon. Interestingly, this is the same trajectory pair which would be optimal if the cooperative game payoff would be used. Similar to the cooperative game, the observations from the simple racing situation can again be generalized.

Theorem 6.2. *Under Assumption 6.2.a:*

(a) If $\bar{p}(\bar{x}^1_{i^{cg}}(N_S)) > \bar{p}(\bar{x}^2_{j^{cg}}(N_S))$, and if $\bar{p}(\bar{x}^1_{i^{cg}}(N_S)) \leq \bar{p}(\bar{x}^2_{j^{cg}}(N_S))$ but there does not exists a blocking trajectory pair, then the Stackelberg equilibrium (i^{cg}, j^{cg}) of the cooperative game (6.6) *is identical to the Stackelberg equilibrium of the blocking game* (6.8)*. If there exists a blocking trajectory pair and if $\bar{p}(\bar{x}^1_{i^{cg}}(N_S)) \leq \bar{p}(\bar{x}^1_{i^b}(N_S)) + w$, then (i^b, j^b) is the Stackelberg equilibrium of the blocking game* (6.8)*.*

(b) The Stackelberg equilibrium (i^{cg}, j^{cg}) of the cooperative game (6.6) *is a Nash equilibrium of the blocking game* (6.8)*. And only if a blocking trajectory (i^b, j^b) is a Nash equilibrium of the cooperative game* (6.6) *it is a Nash equilibrium of the blocking game* (6.8)*.*

Proof. We start by noting that by Assumption 6.2.a and Theorem 6.1 we know that (i^{cg}, j^{cg}) is feasible, and since the constraints are the same in the cooperative game and the blocking game, we know that this trajectory pair is also feasible for the blocking game.

Part (a): Given this observation we know that if $\bar{p}(\bar{x}^1_{i^{cg}}(N_S)) > \bar{p}(\bar{x}^2_{j^{cg}}(N_S))$, it follows that the payoffs for the blocking game of this trajectory pair are, $a_{i^{cg},j^{cg}} = \bar{p}(\bar{x}^1_{i^{cg}}(N_S)) + w$ and $b_{i^{cg},j^{cg}} = \bar{p}(\bar{x}^2_{j^{cg}}(N_S))$. We now show that j^{cg} is the best response of P2 to i^{cg} given the payoff of the blocking game (6.8). We know that j^{cg} maximizes the progress given i^{cg} due to the formulation of the cooperative game (6.6), and since the additional payoff only depends on the progress, we know that no other trajectory would get the additional reward, thus j^{cg} is the best response. Second, i^{cg} maximizes the payoff for P1, therefore, (i^{cg}, j^{cg}) is also the Stackelberg equilibrium of the blocking game.

Second, if $\bar{p}(\bar{x}^1_{i^{cg}}(N_S)) \leq \bar{p}(\bar{x}^2_{j^{cg}}(N_S))$ and there exists no blocking trajectory pair (i^b, j^b), or in other words P1 cannot avoid the overtaking maneuver, the payoff of P1 does not depend on the blocking payoff, and is therefore maximized by i^{cg}. As j^{cg} leads to the largest possible progress for P2 and by definition gets the additional reward w, j^{cg} is also the best response for the blocking game (6.8).

Third, let us assume there exists a blocking trajectory pair (i^b, j^b). The pair will be chosen if it leads to the largest payoff for the two players. For P1 this is the case if $\bar{p}(\bar{x}^1_{i^b}(N_S)) + w \geq a_{i,j}$ for all i and j, since i^b is the best possible blocking trajectory other blocking strategies have a lower payoff. Thus, the inequality holds if the payoff

a_{i^b,j^b} is larger than the largest payoff of a feasible non-blocking trajectory pair, which is $\bar{p}(\bar{x}^1_{i^{cg}}(N_S))$. Thus P1 chooses i^b if $\bar{p}(\bar{x}^1_{i^{cg}}(N_S)) \leq \bar{p}(\bar{x}^1_{i^b}(N_S)) + w$ and by definition j^b is the optimal response for P2. Thus (i^b, j^b) is a Stackelberg equilibrium.

Part (b): To show that (i^{cg}, j^{cg}) is a Nash equilibrium of the blocking game (6.8), two cases are possible. First, $\bar{p}(\bar{x}^1_{i^{cg}}(N_S)) > \bar{p}(\bar{x}^2_{j^{cg}}(N_S))$, where $a_{i^{cg},j^{cg}} = \bar{p}(\bar{x}^1_{i^{cg}}(N_S)) + w$ and $b_{i^{cg},j^{cg}} = \bar{p}(\bar{x}^2_{j^{cg}}(N_S))$, and second, $\bar{p}(\bar{x}^1_{i^{cg}}(N_S)) \leq \bar{p}(\bar{x}^2_{j^{cg}}(N_S))$, where $a_{i^{cg},j^{cg}} = \bar{p}(\bar{x}^1_{i^{cg}}(N_S))$ and $b_{i^{cg},j^{cg}} = \bar{p}(\bar{x}^2_{j^{cg}}(N_S)) + w$. In the first case, because j^{cg} is the best response in terms of progress we know that if j^{cg} does not get the additional reward, no other trajectory gets the reward. Therefore, it holds that for all $j \in \Gamma^2$, $b_{i^{cg},j^{cg}} \geq b_{i^{cg},j}$. As a consequence for P2 the trajectory pair (i^{cg}, j^{cg}) fulfills the Nash inequality. For P1 the Nash inequality $a_{i^{cg},j^{cg}} \geq a_{i,j^{cg}}$ is fulfilled since it is the largest feasible payoff for P1. In the second case, a similar argument holds, which is omitted in the interest of space. Thus, (i^{cg}, j^{cg}) is a Nash equilibrium of the blocking game (6.8) even though the additional payoff is not considered in the computation of the trajectory pair.

For the second part, we now assume that the Nash equilibrium is a blocking trajectory pair (i^b, j^b). The Nash inequality for P2 $b_{i^b,j^b} \geq b_{i^b,j}$ is always fulfilled by Definition 6.6, since no action of P2 can get the additional payoff w and j^b is the best response of P2. However, the Nash inequality for P1,

$$\bar{p}(\bar{x}^1_{i^b}(N_S)) + w \geq a_{i,j^b} \quad \forall i \in \Gamma^1 \,,$$

is only fulfilled, if for all $i \in \Gamma^1$ such that $\bar{p}(\bar{x}^1_i(N_S)) > \bar{p}(\bar{x}^1_{i^b}(N_S))$, it holds that $a_{i,j^b} \leq \lambda$. This condition, however, is identical to the one required by the Nash equilibrium of the cooperative game (6.6). Thus, for (i^b, j^b) to be a Nash equilibrium of the blocking game (6.8), it also needs to be a Nash equilibrium of the cooperative game (6.6).

□

This illustrates, that if the goal is to encourage blocking the Stackelberg equilibrium is the appropriate equilibrium concept. Furthermore, this also shows that blocking needs an asymmetric information pattern, which is logical as the leader needs to be sure that he can enforce the blocking trajectory on the follower.

One can recognize that the blocking game subsumes the cooperative game, more precisely for $w = 0$ the two games are identical. This poses the question for which values of w the equilibrium changes. This can be answered for the Stackelberg equilibrium using Theorem 6.2 (a). For a game instance where a blocking trajectory pair (i^b, j^b) exists, there is a discrete change in the optimal trajectory pair when increasing w. This switch occurs the moment $\bar{p}(\bar{x}^1_{i^b}(N_S)) + w$ becomes larger that $\bar{p}(\bar{x}^1_{i^{cg}}(N_S))$; in (6.9) the switch occurs if $w \geq 0.03$. From a racing point of view, it is best to choose w large enough such that whenever possible the blocking trajectory is selected.

6.4 Moving horizon games

The racing games presented in Section 6.2 are finite horizon games. To introduce feedback we propose to play the games in a moving horizon fashion, solving the game with the full horizon but only applying the first input, then reformulating the game based on the current state measurement and solving again. Similar to [61] and [62], this allows to approximately solve an infinite horizon game, which would not be tractable using dynamic programming, by solving a series of finite horizon games.

An inherent problem of moving horizon approaches is that they, in general, do not guarantee feasibility of closed-loop operation. This problem can be tackled by using terminal set constraints which ensure recursive feasibility by guaranteeing that the terminal state is within an invariant set [10]. Alternatively one can use soft constraints, rendering the problem feasible at all times [110].

6.4.1 Terminal viability constraints

If the goal is to guarantee feasibility of the racing games at all times when played in a receding horizon fashion, one can use tools from viability theory similar to the approach proposed in Chapter 5. To apply this approach to ensure recursive feasibility in our recursive gaming setting we need to distinguish the sequential game from the cooperative and blocking games since the former requires the more restrictive Assumption 6.2.b to hold whereas for the latter two Assumption 6.2.a suffices to guarantee feasibility.

For the sequential game the aim is to fulfill Assumption 6.2.b recursively this causes technical difficulties due to discontinuities that arise in the dynamics which need to be tackled appropriately. One discontinuity arises because the leader and follower may switch roles from one solution of the game to the next. This problem can be resolved using a hybrid state in combination with a spatial regularization to prevent unwanted switching behavior, see [118]. Note that the system is now hybrid requiring a hybrid viability kernel algorithm [105]. Second, the optimal viable policy of the leader leads to discontinuities which can be tackled by appropriately smoothing the policy of the leader. Given this continuous dynamical model incorporating the sequential maximization approach, the viability kernel with respect to the track and collision constraints can be computed. If the game starts with the car states in this viability kernel then the follower always has a viable trajectory if the leader plays his optimal viable trajectory. A more detailed discussion can be found in Appendix 6.7.1.

The cooperative and blocking game, require a conceptually simpler viability kernel since guaranteeing Assumption 6.2.a recursively can be assured if there always exists a *cooperative* collision avoiding trajectory pair. Therefore, both cars need a viable trajectory with respect to track and collision constraints. For more details see Appendix

6.7.1.

For all three games, we can theoretically compute an appropriate viability kernel, which can be imposed as a terminal constraint to guarantee recursive feasibility. This can be achieved by penalizing trajectory pairs for which the terminal state is not in the viability kernel by a payoff λ.

Unfortunately, practically it is not possible to compute the described viability kernels as they require one to consider the state of both players simultaneously. As this joint state is ten-dimensional the computation is not tractable with the standard, grid based viability kernel algorithm. From a practical point of view, it is possible to use the viability kernel with respect to the track constraints, as proposed in Chapter 5, as a first approximation. This computation is tractable and it is easy to see that the product of the track viability kernels for the two cars is a superset of all racing viability kernels discussed above; ensuring that the system remains in this product set therefore provides a necessary condition for the recursive feasibility of the games. We adopt this approach in our simulation study below.

6.4.2 Soft constraints

Alternatively one can use soft constraints that allow formulating a game which always finds an optimal trajectory. The idea is to relax the constraints and consider them by reducing the payoff depending on how much the constraints are violated. The advantage compared to the above discussed terminal constraints is that no viability kernels need to be computed. However, the resulting trajectory is not always feasible with respect to the constraints.

The idea is similar to the soft constraints in MPC [110], however, the implementation in the game set-up is quite different. In our racing games, we use soft constraints for the collision constraints only. In a first step, given a trajectory pair we can compute the smallest relaxation of the constraint necessary, which we call slack multipliers of a trajectory pair $S_{i,j} = \sum_{k=1}^{N_S} \max(-d(\bar{x}_i^1(k), \bar{x}_j^2(k)), 0)$. Given our collision constraints, this corresponds to how much the two cars penetrate each other and therefore $S_{i,j}$ is always positive. Second, instead of penalizing a colliding trajectory pair with a payoff of λ, the soft constrained game computes the payoff neglecting the collision constraints and then reduces this payoff by $\sigma S_{i,j}$, where σ is a positive weight. Thus, if σ is chosen large enough, the players in the game have still no benefit from violating the collision constraints, and therefore the optimal trajectory pair does not change. This is often called an exact soft constraint reformulation, and it can be shown that there always exist such a σ for all racing games. The main reason this holds is that there is only a finite number of trajectories and all payoffs as well as the slack multipliers $S_{i,j}$ are bounded. On the other hand, if there is no feasible trajectory the soft constrained game does still find a reasonable trajectory pair by trading off violation of the collision constraint and payoff depending on the value of σ. See Appendix 6.7.2 for the reformulation of the cooperative game with soft collision constraints as well as the conditions on σ to guarantee an exact reformulation.

6.5 Simulation results

6.5.1 Simulation set-up

Our simulation study replicates the miniature race car set-up, introduced in Chapter 2 and to simulate the cars we use the dynamical bicycle model with nonlinear tire forces as described in Chapter 3.

To achieve high-performance driving we use a two-level hierarchical controller following the structure introduced in Chapter 4. The upper level of the controller generates a trajectory pair based on one of the three racing games and the path planning model studied in the experimental set-up of Chapter 5. More precisely, for the path planning model we use number prediction steps $N_S = 3$, discretization time $T_{pp} = 0.16\,\mathrm{s}$ and number of constant velocities $N_m = 129$. Figure 6.4 shows an example of trajectories resulting from the path planning model. At the lower level, we employ an MPC controller to track the trajectories corresponding to the optimal trajectory pair. The lower level MPC uses the linearized bicycle model, and also enforces track constraints; identical to the MPC (4.6) described in Chapter 4.

Note that brute force implementation of the racing bimatrix game would, in this case, require forming two $129^3 \times 129^3$ payoff matrices, clearly a formidable computational task. To reduce the computation we rely on viability-based pruning of the trajectories introduced in Chapter 5, which excludes trajectories that are doomed to leave the track either immediately or at a later stage. Following Chapter 5 we use either the viability or the discriminating kernel to prune the trajectories. The two kernels lead to comparable lap times but differences in driving style that, in the context of this chapter, results in interesting racing behaviors. More precisely, pruning based on the discriminating kernel results in a more conservative driving style, breaking earlier into curves but achieving higher exit velocities out of curves. Pruning based on the viability kernel, on the other hand, results in a more aggressive driving style. In the following, we will use PV to denote the player using the viability kernel and PD the player using the discriminating kernel to prune the trajectories.

To generate the bimatrix game, we assume that each car shares their possible trajectories with the opponent. Alternatively, it would also be possible to share the path planning model and the pruning strategy at the beginning of the race, and online only share the current state. For real racing, this can be extended by estimating the opponents constant velocity points as well as estimating the state based on sensors. The

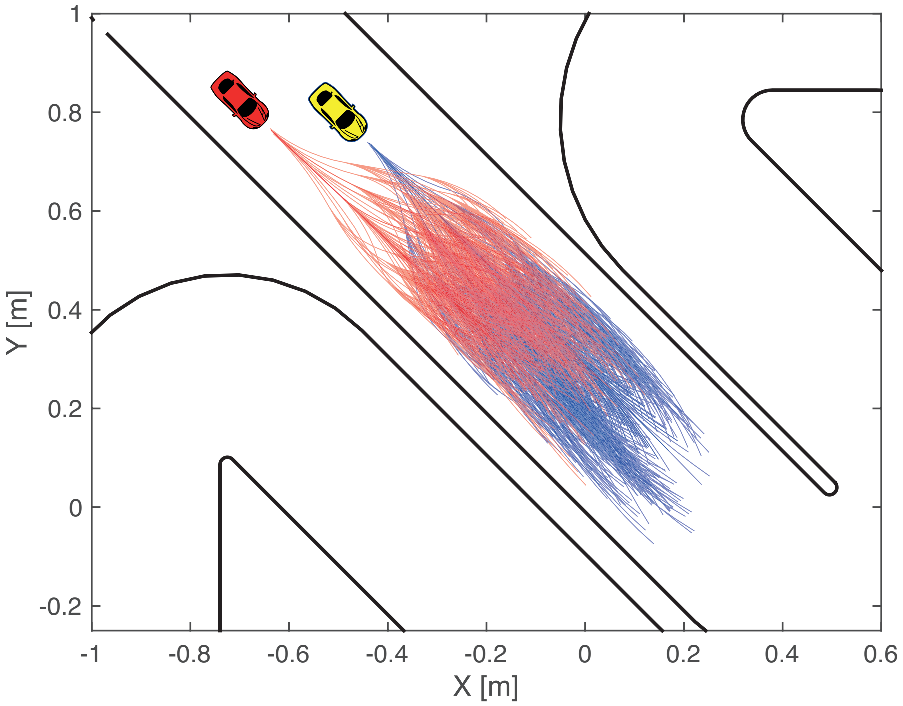

Figure 6.4: Possilbe trajectories of the path planning model for two cars using the viability-based pruning presented in Chapter 5

estimation of the constant velocity points is simplified by the fact that similar cars have similar constant velocities. Thus, each car only needs to adapt its own model to get an estimate of the opponent's model. Alternatively, for the cooperative game the modified best response approach described in Appendix 6.7.3 can be used, where only a sequence of best response trajectories needs to be shared to solve each game.

To further reduce the computation time, we compute collisions using a two-step collision detection: first, the distance between the center of the cars is checked, only if the distance is small enough for a collision to be possible in a second step a separating axis collision detection algorithm is used [119]. Finally, we compute progress of all remaining trajectories as described in Chapter 4. All the tasks except the path planning model run at a sampling time of 20 ms allowing for accurate collision checks and fast feedback. To be able to react fast to the changes of the opponent player and to deal with model mismatch the game is also repeated every 20 ms.

The last implementation detail is concerned with the soft constraints, any constraint violation of more than 1 cm is treated as a collision between the cars. Note that thereby the problem is not always feasible and that in a case where no feasible solution can be found the car behind initiates an emergency braking maneuver, until the problem is feasible again.

6.5.2 Results

The racing games were evaluated in a simulation study, where the cars are initiated at 500 different random initial positions along the center line of the track. To guarantee feasible initial conditions the cars are initiated at a low forward velocity of 0.5 m/s, tangentially to the center line. To promote interesting racing situations we choose the initial position of the two cars in close proximity (0-20 cm apart) and the physical parameters of both cars are identical, including engine power, braking power, and tire models. For each initial position, the simulation is run for 40 s, which corresponds to approximately four laps.

We compare three cases of the presented racing games. In the first case the optimal strategy pair is computed using the Sackelberg equilibrium of the sequential game (6.4), in the second case that of the cooperative game (6.6) and in the third case that of the blocking game (6.8) with $w = 100$. Recall that the solutions of the first two approaches are identical if Assumption 6.2.b is fulfilled, but may differ otherwise. We also assume that the players play truthfully. This is reasonable since deviating from this trajectory leads to a lower payoff, either due to a collision or to worse progress. We note, however, that in the blocking game the leader could benefit from being untruthful. Simulation studies also showed that other methods such as random play do lead to a worse payoff, in some cases even for both players.

Table 6.1: Simulation study of the racing games

	seq. game (PV/PD)	coop. game (PV/PD)	blocking game (PV/PD)
# of overtaking maneuvers	113 (4/109)	857 (502/355)	414 (231/183)
# of runs with overtaking maneuvers	106	309	220
collision probability	$5.42 \cdot 10^{-3}$	$4.59 \cdot 10^{-3}$	$5.28 \cdot 10^{-3}$

Let us start by noting that in all cases we observe a substantial number of overtaking maneuvers (see Table 6.1), even though both cars have the same power, and only differ in the pruning strategies they employ. However, it is visible that the sequential game has the fewest overtaking maneuvers. The cooperative game has by far the most overtaking maneuvers, whereas the blocking game has approximately half as many overtaking maneuvers. It is also interesting to see that the cooperative and blocking game lead to a more even distribution of overtaking maneuvers between the different pruning strategies. This stands in contrast to the sequential game, where nearly all overtaking maneuvers are performed by the player using the discriminating kernel as a their pruning strategy (PD). When looking at the probability of a collision, we can see that all three implementations have a low empirical collision probability of around $5 \cdot 10^{-3}$, especially given the close initial proximity of the cars. The cooperative game has the lowest collision probability and the sequential game the highest. Note that some of the collisions can also be caused by the fact that the low-level MPC does not consider the collision constraints. Even though experimental and simulation results suggest that the low-level MPC can follow the trajectory very precisely, the games are sometimes so tight that even sub-millimeter tracking errors can result in a collision.

The main reason for the observed trends is first and foremost the fact that in the sequential game the leader does not consider the follower. This explains the fewer overtaking maneuvers, the higher collision probability, and the observation that PV has nearly no overtaking maneuvers. Almost all overtaking maneuver need a certain cooperation between the cars, and this holds especially when PV overtakes PD. In the sequential game the leader does not consider the follower and ignores the risk of a collision. Due to the layout of the track, this is an advantage for PD, which can prevent nearly all overtaking maneuvers, whereas PV is not able to do so. In the cooperative and blocking games the leader adapts his strategy to accommodate the follower. This helps to prevent collisions, but at the same time allows for overtakes by the opposing car. In the blocking game the leader does consider the follower, and helps to prevent a collision, but at the same time actively tries to avoid overtaking maneuvers. This, in the end, results in a middle ground both in terms of overtaking and collision probability.

Table 6.2: Payoff of the racing games

	seq. game (PV/PD)	coop. game (PV/PD)	blocking game (PV/PD)
mean progress [m]	82.89	82.49	82.53
# of stay	384	292	329
ahead runs	(148/236)	(230/62)	(196/133)
# of wins	(156/344)	(412/88)	(307/193)

Similar trends can also be seen when analyzing the overall payoff of the game. Table 6.2, shows the mean progress of the different games, in how many runs the car starting ahead was able to also be ahead at the end and how many times which player won. The first two metrics are directly the ones we try to maximize in our games, and the third is, in a sense, the long term objective of the players. We can see that the sequential game achieves the largest progress of the three games, whereas the cooperative and the blocking game achieve very similar values. The sequential game achieves the largest progress because the leader does not adapt his strategy thus only optimizing the progress. In the other games, there are also more overtaking maneuvers which slow down the cars. The number of stay ahead runs, and winning runs show a similar trend to overtaking in Table 6.1. The sequential game is the best to stay ahead and win but does this by causing more and also deeper collisions. From the stay ahead and winning runs, it is even better to see how the additional term in the blocking game helps to be more competitive compared to the cooperative game.

To further highlight the difference between the three different game approaches, one of the 500 simulation runs is shown in a video which can be found at: `https://youtu.be/3Skl5qeFum8` for the sequential game, `https://youtu.be/Cp9OchB2S_M` for the cooperative game, and `https://youtu.be/Xxa8W9D3Z_A` for the blocking game. The videos emphasize the points discussed above, where the sequential game has zero overtaking maneuvers but there are collisions, in the blocking game there is only one overtaking maneuver, but in the cooperative game there are three overtaking maneuvers.

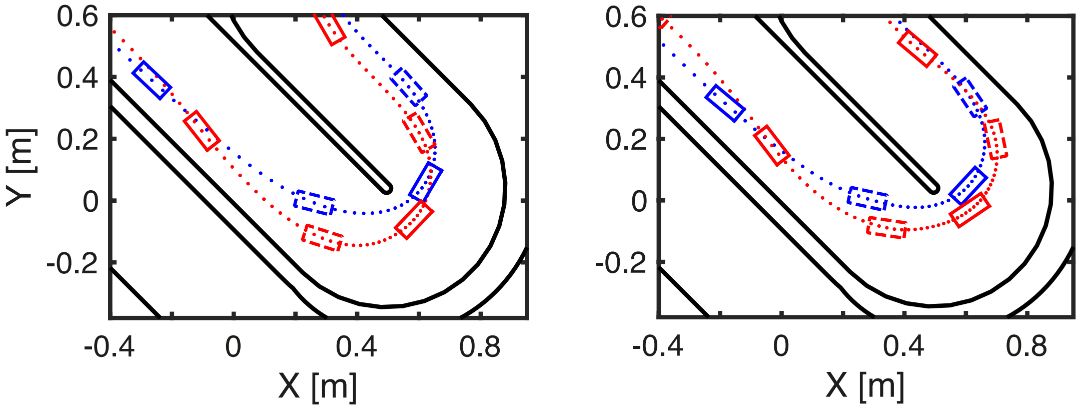

Figure 6.5: Time elapse of two overtaking maneuvers. Left side cooperative game, right blocking game. The blue car is PV and the red car PD, the cars are shown every 0.3 s and every second time the car is dashed to highlight which cars are at the same time step. The driven trajectory also is also shown as a dotted line, where each dot is 0.02 s apart. In both cases, the blue car can overtake the red car, but in the blocking game, the red car drives a more aggressive trajectory which tries to prevent the overtaking maneuver.

Remarks on difference in driving style and car dynamics

The above simulation study investigated the case where one player PV uses the viability and the other player PD the discriminating kernel to prune their trajectories. Thus, the driving style of the two players is quite different, with PV being “aggressive” and PD “cautious.” However, both players have the same model and identical cars. To further investigate the influence of different driving styles and differences in the cars, we also investigated the effect of a reduced motor power of one of the car as well as using the same approach to prune the trajectories for both players; the latter results in a race between identical cars and drivers. The simulation results show as expected that reducing the motor power of one car leads to more overtaking maneuvers. However, the influence is smaller than expected as the power constraint mainly influenced the maximal velocity, which is not too important in the used track layout. Using the same pruning strategy for both players drastically reduced the amount of overtaking maneuvers. However, even though the drivers and the cars are identical overtaking maneuvers still occur. In conclusion, we noticed that the change in the driving style induced by different pruning strategies seems to be more important than power differences between the cars.

6.5.3 Computation time

Simulations were carried out on a computer running Debian equipped with 16 GB of RAM and a 3.6 GHz Intel Xeon quad-core processor. When investigating the computation times reported in Table 6.3, one can see that the computation times for the trajectory generation step is very similar for all three games, which is expected as the different cases only have a minor influence on the trajectory planning phase. However, the collision check for the sequential game is significantly faster than the collision checks for the other two games.

Table 6.3: Computation time of the racing games

	seq. game	coop. game	blocking game
mean traj. generation [s]	0.0027	0.0027	0.0027
max traj. generation [s]	0.0358	0.0314	0.0317
mean collision check [s]	0.0012	0.3365	0.2545
max collision check [s]	0.0738	109.81	112.31

The reason is that the cooperative and blocking game require checking collisions of all trajectories of both players with each other. Roughly speaking this computation grows quadratically in the number of trajectories available to the players and can result in up to 10^9 collision checks. In the sequential game, when using the sequential maximization approach, collisions are only checked between the best trajectory of the leader and all trajectories of the follower. In this case, the number of collision checks grows only linearly in the number of trajectories available to the players. We can say that the sequential game is close to real-time feasible, whereas the cooperative and blocking game are significantly slower and would need radical changes to implement in real-time.

6.6 Conclusion

In this chapter we presented a non-cooperative game approach for one-to-one racing of two autonomous cars. Three approaches to model the interaction between the two players were considered and formulated as bimatrix games. In the first two approaches the interaction is limited to collision constraints, and each player optimizes his progress. The third approach augments the cost function by rewarding staying ahead at the end of the horizon to emphasize blocking behavior. For online implementation the game is played in a moving horizon fashion and two methods were proposed to deal with the loss of feasibility in closed-loop. A simulation study shows that the proposed games can be used for competitive autonomous racing of two cars. The main observation is that the sequential game where the follower is completely neglected seems to be the most efficient blocking technique in closed-loop, but also comes with the highest risk of a collision.

6.7 Appendix

6.7.1 Terminal viability constraints

In the main text, the challenges and solutions how to compute the appropriate viability kernels are only roughly outlined, here we investigate them in more detail.

Sequential Game

In the case of the sequential game we use a one-step game to simplify the derivation of the necessary constraints and models for the viability computations. Similar ideas work for longer horizons, however, the derivation and conditions get more complex.

First, let us define the viable one-step optimal policy of the leader P1, as the following parametric optimization problem,

$$\begin{aligned} \psi^1(\bar{x}^1) = \arg \max_{u^1 \in U^p(q^1)} \quad & \bar{p}(\bar{f}^1(\bar{x}^1, u^1)) \,, \\ \text{s.t.} \quad & \bar{f}^1(\bar{x}^1, u^1) \in \text{Viab}_{\bar{F}^1}(K^1) \,, \end{aligned} \tag{6.13}$$

where, $K^1 = \{\bar{x}^1 \in \mathbb{R}^5 | \, \bar{x}^1 \in \bar{\mathcal{X}}_{\text{T}}\}$, $\bar{F}^1(\bar{x}^1) = \{\bar{f}^1(\bar{x}^1, u^1) | \, u^1 \in U^1(q^1)\}$ and $\text{Viab}_{\bar{F}^p}(K^1)$ is the corresponding viability kernel. Since the feedback policy could be non-unique, it is necessary to add a deterministic tie-breaking rule, which is part of the rules of the road.

If the leader plays this feedback control law this mimics the sequential maximization approach. However, the resulting set-valued map $\{\bar{f}^1(\bar{x}^1, u^1) | \, u^1 \in \psi^1(\bar{x}^1)\}$ is not upper-semicontinuous, due to the discontinuities in the feedback control law. Therefore, it is necessary to smooth the feedback policy to achieve an upper-semicontinuous approximation of the set-valued map, which then can be used to compute the viability kernel. We propose to use the set-valued map whose graph is the closure of $\psi^1(\bar{x}^1)$,

$$\bar{\psi}^1(\bar{x}^1) = \{y|(x,y) \in \overline{\mathrm{Graph}(\psi^1)}\}\,. \tag{6.14}$$

This smooths the discontinuities in a set-valued sense, and by [31, Proposition 2.2.2] and the fact that the graph is closed we know that $\bar{\psi}^1(\bar{x}^1)$ is upper-semicontinuous, and further by [31, Proposition 2.1.4] we know that the resulting set-valued map $\{\bar{f}^1(\bar{x}^1, u^1)|\, u^1 \in \bar{\psi}^1(\bar{x}^1)\}$ is upper-semicontinuous. However, the policy $\bar{\psi}^1(\bar{x}^1)$ is now set-valued, therefore, at these smoothed discontinuities the leader has several possible trajectories, which allows to cooperate with the follower since the players only need one avoiding trajectory pair. Note that this only influences the outcome of the algorithm if the discontinuities of $\psi^1(\bar{x}^1)$ do not have measure zero. If this would turn out to be an issue, a discriminating kernel algorithm could be used, where the action of the leader would be adversarial. This, however, would lead to a state dependent adversary player, which is not covered in the standard discriminating kernel theory [40].

As described in the main text, the second discontinuity results from the leader-follower switch. This discontinuity can be avoided by introducing a hysteresis based switch, which uses a discrete state for each leader and spatial regularization to prevent unwanted switching behavior, see [118]. The discrete state h is 1 if P1 is the leader and 2 if P2 is the leader. The disadvantage is that the resulting system is a discrete time hybrid system which requires a hybrid viability kernel algorithm [105]. However, the advantage is that then for each hybrid state the dynamics are continuous. Notice that the changed leader-follower switch needs to be included in the rules of the road.

To describe the dynamics of the two cars, including the above discussed modifications, we need to consider the stacked state $\xi = [\bar{x}^1, \bar{x}^2]$, as well as the discrete states h, resulting in the following difference inclusion $\xi_{k+1} \in \mathcal{F}(h_k, \xi_k)$, with the set-valued map $\mathcal{F}(h, \xi)$ defined as follows,

$$\mathcal{F}(h,\xi) = \begin{cases} \left\{ \begin{bmatrix} \bar{f}^1(\bar{x}^1, u^1) \\ \bar{f}^2(\bar{x}^2, u^2) \end{bmatrix} \middle| \begin{matrix} u^1 \in \bar{\psi}^1(\bar{x}^1) \\ u^2 \in U^2(q^2) \end{matrix} \right\} & \text{if } h = 1 \\ \left\{ \begin{bmatrix} \bar{f}^1(\bar{x}^1, u^1) \\ \bar{f}^2(\bar{x}^2, u^2) \end{bmatrix} \middle| \begin{matrix} u^1 \in U^1(q^1) \\ u^2 \in \bar{\psi}^2(\bar{x}^2) \end{matrix} \right\} & \text{if } h = 2 \end{cases} \tag{6.15}$$

Note that the reset function and guard are omitted in interest of space, however, since they result from a spatial regularization of the leader-follower switch they can be obtained similar to the examples in [118]. Furthermore, due to the above discussed modifications, in addition to the modifications introduced in Chapter 5 Section 5.3 the set-valued map (6.15) is upper-semicontinuous and describes the sequential maximization approach.

Finally, the closed constraint set K, includes the track and collision constraints,

$$K = \{\xi \in \mathbb{R}^{10}|\, d(\bar{x}^1, \bar{x}^2) \geq 0,\, \bar{x}^1 \in \bar{\mathcal{X}}_{\mathrm{T}},\, \bar{x}^2 \in \bar{\mathcal{X}}_{\mathrm{T}}\}\,. \tag{6.16}$$

The following fact is immediate from the definition of the viability kernel.

Proposition 6.3. *The sequential game* (6.4) *is recursively feasible if played in a moving horizon fashion, when the leader plays* $\bar{\psi}^{\varrho}(\bar{x}^{\varrho})$ *the viable optimal policy* (6.14)*, and the terminal state of the follower remains within* $Viab_{\mathcal{F}}(K)$*, where* $\mathcal{F}(\xi)$ *as defined in* (6.15) *and* K *as defined in* (6.16)*.*

Progress and Blocking Game

For these two games it suffices that the state of both player remains within the viability kernel with respect to the track and collision constraints. This viability kernel describes all states for which there exist cooperative collision free trajectories of the two players that remain within the track. To compute the viability kernel we again consider the stacked state $\xi = [\bar{x}^1, \bar{x}^2]$ with the corresponding difference inclusion,

$$\xi_{k+1} \in \mathcal{F}(\xi_k) = \left\{ \begin{bmatrix} \bar{f}^1(\bar{x}_k^1, u_k^1) \\ \bar{f}^2(\bar{x}_k^2, u_k^2) \end{bmatrix} \middle| \begin{array}{l} u_k^1 \in U^1(q_k^1) \\ u_k^2 \in U^2(q_k^2) \end{array} \right\}, \tag{6.17}$$

Following Chapter 5 Section 5.3 we know that the set-valued map $F(\xi)$ is upper-semicontinuous and K is closed. Similar to the sequential game the following fact is immediate from the definition of the viability kernel.

Proposition 6.4. *The cooperative or blocking game is recursively feasible if played in a moving horizon fashion, if terminal state of both players remains within* $Viab_{\mathcal{F}}(K)$*, where* $\mathcal{F}(\xi)$ *as defined in* (6.17) *and* K *as defined in* (6.16)*.*

6.7.2 Soft constraints

In the following we show how the soft constrained version of the of the cooperative game is formulated and what condition σ needs to fulfill to be an exact reformulation. First the soft constrained cooperative game can be computed as follows,

$$\begin{aligned} a_{i,j} &= \begin{cases} \kappa & \text{if } \exists k \in \{1, ..., N_S\} : \bar{x}_i^1(k) \notin \bar{\mathcal{X}}_{\mathrm{T}} \\ \bar{p}(\bar{x}_i^1(N_S)) - \sigma S_{i,j} & \text{else} \end{cases}, \\ b_{i,j} &= \begin{cases} \kappa & \text{if } \exists k \in \{1, ..., N_S\} : \bar{x}_j^2(k) \notin \bar{\mathcal{X}}_{\mathrm{T}} \\ \bar{p}(\bar{x}_j^2(N_S)) - \sigma S_{i,j} & \text{else} \end{cases}, \end{aligned} \tag{6.18}$$

with $S_{i,j} = \sum_{k=1}^{N_S} \max(-d(\bar{x}_i^1(k), \bar{x}_j^2(k)), 0)$.

Now lets assume that Π_{st} is the set of Stackelberg equilibria of the cooperative racing game with (i^{cg}, j^{cg}) denoting on of these equilibria. Further let Γ_{T}^1 and Γ_{T}^2 be all trajectories which fulfill the track constraints. We first notice that even for the soft constrained game, as long as for both players there exists a trajectory which stays inside the track and κ is chosen small enough the Stackelberg equilibrium does not leave the track and there exists a Nash equilibrium which does not leave the track. Then if for all $(i,j) \in \Gamma^1_T \times \Gamma^2_{\mathrm{T}} \setminus \Pi_{\mathrm{st}}$, σ is chosen such that the following two inequalities hold,

$$\begin{aligned} \bar{p}(\bar{x}^1_i(N_S)) - \sigma S_{i,j} &< \bar{p}(\bar{x}^1_{i^{cg}}(N_S))\,, \\ \bar{p}(\bar{x}^2_j(N_S)) - \sigma S_{i,j} &< \bar{p}(\bar{x}^2_{j^{cg}}(N_S))\,, \end{aligned}$$

the reformulation is exact. Or in other words, for all trajectory pairs which stay within the track and are not a Stackelberg equilibria the payoff received is smaller than the one received by all the Stackelberg equilibria. Note that similar inequalities can be formulated for all three games, however, in the interest of space they are omitted.

6.7.3 Sequential iterative best response

Recently several research groups proposed to use iterative best response (best response dynamics) in the context of autonomous racing of cars [58], as well as quadcopters [59]. The idea is to find a Nash equilibrium by iterating the best response of the players. This approach is well known to find Nash equilibria of populations of players. However, it has one fundamental problem, the approach is not guaranteed to find a Nash equilibrium and is prune to cycling behavior.

In the context of this thesis the iterative best response can be formulated as follows,

$$i^{k+1} = \arg\max_{i \in \Gamma^1} a_{i,j^k}\,, \quad j^{k+1} = \arg\max_{j \in \Gamma^2} b_{i^k,j}\,. \tag{6.19}$$

Where (i^k, j^k) is the trajectory pair at iteration k and (i^0, j^0) is the initial trajectory pair, which is determined randomly or using a smart guess. It is easy to show that if the sequence of trajectory pairs converges $(i^k, j^k) = (i^{k+1}, j^{k+1})$ the trajectory pair is a Nash equilibrium.

However, if the approach is applied to the cooperative game (6.7) with the corresponding racing situation in Figure 6.1, and the initial guess is not one of the two Nash equilibria, the iterative best response cycles between the trajectory pair $(2,2)$ and $(1,1)$. Where the trajectory pair $(2,2)$ leads to a collision and $(1,1)$ is worst feasible outcome for both players. The problem is related to the fact that there exist two Nash equilibria which are not distinguishable for the given game. However, this structure including multiple non-distinguishable Nash equilibria is common for the cooperative game (as well as the blocking game). Often one Nash equilibrium corresponds to the player ahead doing what is best for him and the following player avoiding a collision, and the second is the opposite where the player ahead moves out of the way for the following player. Thus, this problem is likely to occur also in real racing situations, such as shown in Figure 6.4.

In the following, we will propose a sequential iterative best response approach. This approach has the advantage that we can show that the sequence of trajectory pairs converges to a Nash equilibrium of the cooperative game if started with a feasible trajectory pair. The idea is inspired by the sequential maximization approach (6.10), and instead of both players determining the best response given the trajectory pair of the last iteration, the idea is to sequentially update the trajectory, with the leader starting first. This can be summarized by the following sequential iterative best response scheme,

$$i^{k+1} = \arg\max_{i \in \Gamma^1} a_{i,j^k}\,, \quad j^{k+1} = \arg\max_{j \in \Gamma^2} b_{i^{k+1},j}\,. \tag{6.20}$$

It is still straightforward to show that if the sequence converges the resulting strategy pair is a Nash equilibrium. Further, we can even weaken this statement and only require the value of the payoff to converge for the trajectory pair (i^k, j^k) to be a Nash equilibrium. Actually, if the payoff converges but (i^k, j^k) and (i^{k+1}, j^{k+1}) are not equal both trajectory pairs are Nash equilibria.

Proposition 6.5. *If the sequential iterative best response* (6.20) *is used to find a Nash equilibrium of the cooperative game* (6.6) *and Assumption 6.2.a holds then the following properties hold,*

1. *If (i^k, j^k) is feasible all subsequent iterates are feasible.*
2. *Every feasible sequence of trajectory pairs has a monotonically increasing payoff for the leader (P1).*
3. *Every feasible sequence of trajectory pairs converges to a Nash equilibrium.*

Proof. Since we look at the cooperative game and Assumption 6.2.a holds, the symmetry in the collision constraints used in Theorem 6.1 still holds.

1. If (i^k, j^k) is feasible, there exists at least one i such that $a_{i,j^k} > \lambda$, thus $a_{i,j^{k+1}} > \lambda$ and due to the symmetry in the collision constraints there also exist a j such that $b_{i^{k+1},j} > \lambda$.
2. First, it is straight forward to see that $a_{i^{k+1},j^k} \geq a_{i^k,j^k}$, given the first step in (6.20). Second, since we know that P2 will chose a feasible trajectory, due to point (1), and due to the symmetry in the cooperative game we know that $a_{i^{k+1},j^{k+1}} = a_{i^{k+1},j^k}$. Thus we have $a_{i^{k+1},j^{k+1}} \geq a_{i^k,j^k}$ which shows that the payoff of the leader (P1) increases monotonically.
3. Since the cost of the leader monotonically increases, and there are a finite number of trajectory pairs, the sequential iterative best response scheme will converge to a point where the cost does not change. Every trajectory pair with this cost is a Nash equilibrium.

□

CHAPTER 7

Conclusion and Outlook

Conclusion

In this thesis, we presented a path planning and control approach for autonomous racing which can race around a given race track and interact with other race cars. Therefore, we proposed a hierarchical racing controller, which is able to drive a car at the handling limit. Further, we used viability theory to improve the upper-level path planning step in the hierarchical controller. As a result, the path planner only generates viable trajectories that are recursively feasible. Furthermore, by only generating viable trajectories the computation time is reduced. These savings in computation times allow using longer prediction horizons, which together with the guaranteed recursive feasibility, improves the performance in terms of lap times and constraint violations. Due to the reduced computation the controller including the viability constraints is real-time feasible. This allowed for a successful implementation of the controller on the experimental miniature race car set-up.

Finally, to allow the controller to interact and race against other cars, we presented a non-cooperative game approach for one-to-one racing of two autonomous cars, which builds on the path planning step of the hierarchical racing controller. Three approaches to model the interaction between the two players were considered and formulated as bimatrix games. Given these games the equilibrium solutions in terms of Stackelberg and Nash equilibria were studied. For online implementation, the game is played in a moving horizon fashion, and two methods were proposed to deal with the loss of feasibility in closed-loop. A simulation study shows that the proposed games can be used for competitive autonomous racing of two cars, with the different games resulting in different racing styles.

Outlook

Theory and Computation

There are several theoretical and computational research question which are interesting to investigate in future work. First, how can uncertainty be incorporated into the decision process? Mainly the uncertainty in the path planning model. There are two places where these uncertainties should be considered; first, in the computation of the viability constraints and second the online path planning step. However, compared to the space discretization errors in Chapter 5 these uncertainties are not known. Thus, different techniques are required, e.g., the leadership kernel seems the more natural concept. The second, and arguably more interesting place to consider these uncertainties is in the racing games. Where the collision constraints could be formulated as robust or stochastic constraints, hopefully leading to fewer collisions, but with the potential drawback of fewer overtaking maneuvers.

In Chapter 5 we noticed that a terminal cost can improve the performance of controllers with short horizons. Currently, this terminal cost is computed heuristically, and future work could investigate how to compute a terminal cost properly and if it is possible to include viability constraints in these computations. A terminal cost could be especially interesting in the racing game where strategic positioning on the race track is important to prevent overtaking maneuvers. However, similar to the terminal viability constraints computing a terminal cost is not tractable due to the high state dimension. Therefore, it would be interesting to investigate machine learning tools, to learn from the cars racing against each other, similar to methods proposed in [120, 121]. The main challenge is to come up with a method to trade-off exploration and exploitation while learning this terminal cost.

Currently, the main drawback of the presented controller is the high computational cost, especially for the racing games. Preliminary work showed that using a GPU instead of a CPU can significantly speed up the computation times, since the path planning step of the controller is parallelizable [122]. For example, the worst case computation time for the path planning step for a horizon of $N_S = 4$ can be reduced from 85 ms to below 3 ms. However, this only covers the path planning step. It would also be necessary to use a GPU in the computation of the game payoffs, mainly when checking if trajectories collided. Therefore, future work should investigate how to parallelize the whole racing game controller, especially how to efficiently split the problem into (small) parallel executable work packages.

Finally, it is also possible to improve the lower level MPC controller, for example by considering the uncertainty in the bicycle model such as done in [67] and [68], or by using an NLP solver which allows using the nonlinear dynamics directly and considering combined slip tire force constraints, similar to [89]. Furthermore, in the racing games currently no collision constraints are considered in the lower level MPC. One possibility to consider collision constraints is to use an NLP solver and the collision constraint formulation proposed in [69].

Experimental

In future work, it would be interesting to test the hierarchical racing controller including the racing games and the above discussed ideas on the experimental miniature race car set-up. However, it would also be interesting to test the controller on real cars. One challenge with real sized cars is that real circuits are larger in relative size. Therefore, significantly longer horizons are needed to determine a good racing trajectory. In theory, this problem can be tackled using the proposed viability constraints as well as a meaningful terminal cost. Thus, this would be a real challenge for several of the previously described future directions.

Bibliography

[1] R. Horowitz and P. Varaiya, “Control design of an automated highway system,” *Proceedings of the IEEE*, vol. 88, no. 7, pp. 913–925, 2000.

[2] M. Buehler, K. Iagnemma, and S. Singh, *The 2005 DARPA grand challenge: the great robot race.* Springer, 2007.

[3] ——, *The DARPA urban challenge: Autonomous vehicles in city traffic.* Springer, 2009.

[4] B. Paden, M. Čáp, S. Z. Yong, D. Yershov, and E. Frazzoli, “A survey of motion planning and control techniques for self-driving urban vehicles,” *IEEE Transactions on Intelligent Vehicles*, vol. 1, no. 1, pp. 33–55, 2016.

[5] D. González, J. Pérez, V. Milanés, and F. Nashashibi, “A review of motion planning techniques for automated vehicles,” *IEEE Transactions on Intelligent Transportation Systems*, vol. 17, no. 4, pp. 1135–1145, 2016.

[6] W. Schwarting, J. Alonso-Mora, and D. Rus, “Planning and decision-making for autonomous vehicles,” *Annual Review of Control, Robotics, and Autonomous Systems*, 2018.

[7] K. Kritayakirana and C. Gerdes, “Using the centre of percussion to design a steering controller for an autonomous race car,” *Vehicle System Dynamics*, vol. 15, pp. 33–51, 2012.

[8] R. Verschueren, S. De Bruyne, M. Zanon, J. V. Frasch, and M. Diehl, “Towards time-optimal race car driving using nonlinear MPC in real-time,” in *Conference on Decision and Control (CDC)*, 2014, pp. 2505–2510.

[9] U. Rosolia, A. Carvalho, and F. Borrelli, “Autonomous racing using learning model predictive control,” in *American Control Conference (ACC)*, 2017, pp. 5115–5120.

[10] D. Q. Mayne, J. B. Rawlings, C. V. Rao, and P. O. M. Scokaert, “Constrained model predictive control: Stability and optimality,” *Automatica*, vol. 36, no. 6, pp. 789 – 814, 2000.

[11] M. Althoff, O. Stursberg, and M. Buss, “Model-based probabilistic collision detection in autonomous driving,” *IEEE Transactions on Intelligent Transportation Systems*, vol. 10, no. 2, pp. 299–310, 2009.

[12] A. Carvalhoa, Y. Gaoa, S. Lefevrea, and F. Borrellia, “Stochastic predictive control of autonomous vehicles in uncertain environments,” in *International Symposium on Advanced Vehicle Control (AVEC)*, 2014.

[13] A. Carvalho, S. Lefevre, G. Schildbach, J. Kong, and F. Borrelli, “Automated driving: The role of forecasts and uncertainty - A control perspective,” *European Journal of Control*, vol. 24, pp. 14 – 32, 2015.

[14] A. Liniger, A. Domahidi, and M. Morari, “Optimization-based autonomous racing of 1:43 scale RC cars,” *Optimal Control Applications and Methods*, vol. 36, no. 5, pp. 628–647, 2015.

[15] J. Funke, M. Brown, S. M. Erlien, and J. C. Gerdes, “Collision avoidance and stabilization for autonomous vehicles in emergency scenarios,” *IEEE Transactions on Control Systems Technology*, vol. 25, no. 4, pp. 1204–1216, 2017.

[16] F. Borrelli, P. Falcone, T. Keviczky, and J. Asgari, “MPC-based approach to active steering for autonomous vehicle systems,” *International Journal of Vehicle Autonomous Systems*, vol. 3, no. 2, pp. 265–291, 2005.

[17] P. Falcone, F. Borrelli, J. Asgari, H. E. Tseng, and D. Hrovat, “Predictive active steering control for autonomous vehicle systems,” *IEEE Transactions on Control Systems Technology*, vol. 15, no. 3, pp. 566–580, 2007.

[18] T. Besselmann and M. Morari, “Hybrid parameter-varying model predictive control for autonomous vehicle steering,” *European Journal of Control*, vol. 14, no. 5, pp. 418–431, 2008.

[19] J. Ackermann, J. Guldner, W. Sienel, R. Steinhauser, and V. I. Utkin, “Linear and nonlinear controller design for robust automatic steering,” *IEEE Transactions on Control Systems Technology*, vol. 3, no. 1, pp. 132–143, 1995.

[20] S. Fuchshumer, K. Schlacher, and T. Rittenschober, “Nonlinear vehicle dynamics control - a flatness based approach,” in *Conference on Decision and Control (CDC)*, 2005, pp. 6492–6497.

[21] Y. Gao, T. Lin, F. Borrelli, H. Tseng, and D. Hrovat, “Predictive control of autonomous ground vehicles with obstacle avoidance on slippery roads,” in *Dynamic Systems and Control Conference*, 2010, pp. 265–272.

[22] A. Gray, Y. Gao, T. Lin, J. K. Hedrick, H. E. Tseng, and F. Borrelli, "Predictive control for agile semi-autonomous ground vehicles using motion primitives," in *American Control Conference (ACC)*, 2012, pp. 4239–4244.

[23] E. Frazzoli, M. Dahleh, and E. Feron, "Maneuver-based motion planning for nonlinear systems with symmetries," *IEEE Transactions on Robotics*, vol. 21, no. 6, pp. 1077–1091, 2005.

[24] Y. Kuwata, J. Teo, G. Fiore, S. Karaman, E. Frazzoli, and J. P. How, "Real-time motion planning with applications to autonomous urban driving," *IEEE Transactions on Control Systems Technology*, vol. 17, no. 5, pp. 1105–1118, 2009.

[25] E. Velenis, P. Tsiotras, and J. Lu, "Modeling aggressive maneuvers on loose surfaces: The cases of trail-braking and pendulum-turn," in *European Control Conference (ECC)*, 2007, pp. 1233–1240.

[26] J. Frasch, A. Gray, M. Zanon, H. Ferreau, S. Sager, F. Borrelli, and M. Diehl, "An auto-generated nonlinear MPC algorithm for real-time obstacle avoidance of ground vehicles," in *European Control Conference (ECC)*, 2013, pp. 4136–4141.

[27] J. Jeon, S. Karaman, and E. Frazzoli, "Anytime computation of time-optimal off-road vehicle maneuvers using the RRT*," in *Conference on Decision and Control (CDC)*, 2011, pp. 3276–3282.

[28] J. Jeon, R. Cowlagi, S. Peters, S. Karaman, E. Frazzoli, P. Tsiotras, and K. Iagnemma, "Optimal motion planning with the half-car dynamical model for autonomous high-speed driving," in *American Control Conference (ACC)*, 2013, pp. 188–193.

[29] M. Diehl, H. Bock, J. Schlöder, R. Findeisen, Z. Nagy, and F. Allgöwer, "Real-time optimization and nonlinear model predictive control of processes governed by differential-algebraic equations," *Journal of Process Control*, vol. 12, no. 4, pp. 577–585, 2002.

[30] E. Kerrigan and J. M. Maciejowski, "Invariant sets for constrained nonlinear discrete-time systems with application to feasibility in model predictive control," in *Conference on Decision and Control (CDC)*, 2000, pp. 4951 – 4956.

[31] J.-P. Aubin, *Viability Theory.* Springer, 1991.

[32] L. Grüne and J. Pannek, *Nonlinear Model Predictive Control.* Springer, 2011.

[33] I. Kitsios and J. Lygeros, "Aerodynamic envelope computation for safe landing of the hl-20 personnel launch vehicle using hybrid control," in *International Symposium on Intelligent Control*, 2005, pp. 231–236.

[34] N. Seube, R. Moitie, and G. Leitmann, “Aircraft take-off in windshear: a viability approach,” *Set-Valued Analysis*, vol. 8, pp. 163–180, 2000.

[35] R. Moitie and N. Seube, “Guidance algorithms for UUVs obstacle avoidance systems,” in *OCEANS*, 2000, pp. 1853–1860.

[36] P. Dimitra, M. Kostas, S. Sean, L. John, and K. Kostas J., “A viability approach for the stabilization of an underactuated underwater vehicle in the presence of current disturbances,” in *Conference on Decision and Control (CDC)*, 2005, pp. 8612–8617.

[37] J. Nilsson, J. Fredriksson, and A. C. Odblom, “Verification of collision avoidance systems using reachability analysis?” in *IFAC World Congress*, vol. 19, no. 1, 2014, pp. 10 676–10 681.

[38] A. Tinka, S. Diemer, L. Madureira, E. B. Marques, J. B. De Sousa, R. Martins, J. Pinto, J. E. Da Silva, A. Sousa, P. Saint-Pierre, and A. M. Bayen, “Viability-based computation of spatially constrained minimum time trajectories for an autonomous underwater vehicle: Implementation and experiments,” in *Conference on Decision and Control (CDC)*, 2009, pp. 3603–3610.

[39] P. Saint-Pierre, “Approximation of the viability kernel,” *Applied Mathematics and Optimization*, vol. 29, no. 2, pp. 187–209, 1994.

[40] P. Cardaliaguet, M. Quincampoix, and P. Saint-Pierre, “Set-valued numerical analysis for optimal control and differential games,” *Stochastic and Differential Games*, vol. 4, pp. 177–247, 1999.

[41] J. Lygeros, “On reachabiltiy and minimum cost optimal control,” *Automatica*, vol. 40, no. 6, pp. 917–927, 2004.

[42] I. M. Mitchell, A. M. Bayen, and C. Tomlin, “A time-dependent hamilton-jacobi formulation of reachable sets for continuous dynamic games,” *IEEE Transactions on Automatic Control*, vol. 50, no. 7, pp. 947–957, 2005.

[43] J. N. Maidens, S. Kaynama, I. M. Mitchell, M. M. Oishi, and G. A. Dumont, “Lagrangian methods for approximating the viability kernel in high-dimensional systems,” *Automatica*, vol. 49, no. 7, pp. 2017–2029, 2013.

[44] A. Chutinan and B. H. Krogh, “Verification of polyhedral-invariant hybrid automata using polygonal flow pipe approximations,” in *Hybrid Systems: Computation and Control (HSCC)*, 1999, pp. 76–90.

[45] A. Girard, C. Le Guernic, and O. Maler, “Efficient computation of reachable sets of linear time-invariant systems with inputs,” in *Hybrid Systems: Computation and Control (HSCC)*, 2006, pp. 257–271.

[46] A. A. Kurzhanskiy and P. Varaiya, “Ellipsoidal techniques for reachability analysis of discrete-time linear systems,” *IEEE Transactions on Automatic Control*, vol. 52, no. 1, pp. 26–38, 2007.

[47] A. Girard, C. Le Guernic *et al.*, “Efficient reachability analysis for linear systems using support functions,” in *IFAC World Congress*, 2008, pp. 8966–8971.

[48] M. Korda, D. Henrion, and C. N. Jones, “Convex computation of the maximum controlled invariant set for polynomial control systems,” *SIAM Journal on Control and Optimization*, vol. 52, no. 5, pp. 2944–2969, 2014.

[49] M. W. Mueller, M. Hehn, and R. D’Andrea, “A computationally efficient algorithm for state-to-state quadrocopter trajectory generation and feasibility verification,” in *International Conference on Intelligent Robots and Systems*, 2013, pp. 3480–3486.

[50] M. Kalisiak and M. van de Panne, “Faster motion planning using learned local viability models,” in *International Conference on Robotics and Automation (ICRA)*, 2007, pp. 2700 – 2705.

[51] T. A. Wood, P. Mohajerin, Esfahani, and J. Lygeros, “Hybrid modelling and reachability on autonomous RC-cars,” in *Analysis and Design of Hybrid Systems*, 2012, pp. 430–435.

[52] J. Rieger, “Shadowing and the viability kernel algorithm,” *Applied Mathematics and Optimization*, vol. 60, no. 3, pp. 429–441, 2009.

[53] M. Lhommeau, L. Jaulin, and L. Hardouin, “Capture basin approximation using interval analysis,” *International Journal of Adaptive Control and Signal Processing*, vol. 25, no. 3, pp. 264–272, 2011.

[54] C. Tomlin, G. J. Pappas, and S. Sastry, “Conflict resolution for air traffic management: A study in multiagent hybrid systems,” *IEEE Transactions on Automatic Control*, vol. 43, no. 4, pp. 509–521, 1998.

[55] J. Lygeros, D. N. Godbole, and S. Sastry, “Verified hybrid controllers for automated vehicles,” *IEEE Transactions on Automatic Control*, vol. 43, no. 4, pp. 522–539, 1998.

[56] N. Li, D. W. Oyler, M. Zhang, Y. Yildiz, I. Kolmanovsky, and A. R. Girard, “Game theoretic modeling of driver and vehicle interactions for verification and validation of autonomous vehicle control systems,” *IEEE Transactions on Control Systems Technology*, 2017.

[57] J. H. Yoo and R. Langari, “Stackelberg game based model of highway driving,” in *Dynamic Systems and Control Conference*, 2012, pp. 499–508.

[58] G. Williams, B. Goldfain, P. Drews, J. M. Rehg, and E. A. Theodorou, “Autonomous racing with autorally vehicles and differential games,” *arXiv preprint arXiv:1707.04540*, 2017.

[59] R. Spica, D. Falanga, E. Cristofalo, E. Montijano, D. Scaramuzza, and M. Schwager, “A game theoretic approach to autonomous two-player drone racing,” *arXiv preprint arXiv:1801.02302*, 2018.

[60] J. P. Hespanha, M. Prandini, and S. Sastry, “Probabilistic pursuit-evasion games: A one-step nash approach,” in *Conference on Decision and Control (CDC)*, 2000, pp. 2272–2277.

[61] J. Cruz, M. A. Simaan, A. Gacic, and Y. Liu, “Moving horizon nash strategies for a military air operation,” *IEEE Transactions on Aerospace and Electronic Systems*, vol. 38, no. 3, pp. 989–999, 2002.

[62] K. Virtanen, J. Karelahti, and T. Raivio, “Modeling air combat by a moving horizon influence diagram game,” *Journal of Guidance, Control, and Dynamics*, vol. 29, no. 5, pp. 1080–1091, 2006.

[63] A. Liniger and J. Lygeros, “A Viability Approach for Fast Recursive Feasible Finite Horizon Path Planning of Autonomous RC Cars,” in *Hybrid Systems: Computation and Control (HSCC)*, 2015, pp. 1 – 10.

[64] ——, “Real-time control for autonomous racing based on viability theory,” *IEEE Transactions on Control Systems Technology*, 2017.

[65] ——, “A non-cooperative game aproach to autonomous racing,” *IEEE Transactions on Control Systems Technology (submitted)*, 2017.

[66] M. Tranzatto, A. Liniger, S. Grammatico, and A. Landi, “The debut of aeolus, the autonomous model sailboat of eth zurich,” in *OCEANS 2015-Genova*. IEEE, 2015, pp. 1–6.

[67] J. V. Carrau, A. Liniger, X. Zhang, and J. Lygeros, “Efficient implementation of randomized MPC for miniature race cars,” in *European Control Conference (ECC)*, 2016, pp. 957–962.

[68] A. Liniger, X. Zhang, P. Aeschbach, A. Georghiou, and J. Lygeros, “Racing miniature cars: Enhancing performance using stochastic MPC and disturbance feedback,” in *American Control Conference (ACC)*, 2017, pp. 5642–5647.

[69] X. Zhang, A. Liniger, and F. Borrelli, "Optimization-based collision avoidance," *IEEE Transactions on Control Systems Technology (submitted)*, 2018.

[70] T. Novi, A. Liniger, R. Capitani, M. Fainello, G. Danisi, and C. Annicchiarico, "The influence of autonomous driving on passive vehicle dynamics," in *SAE World Congress Experience (WCX)*, 2018.

[71] L. Hewing, A. Liniger, and M. N. Zeilinger, "Cautious NMPC with gaussian process dynamics for autonomous miniature race cars," in *European Control Conference (ECC)*, 2018.

[72] T. Novi, A. Liniger, R. Capitani, M. Fainello, G. Danisi, and C. Annicchiarico, "The influence of autonomous driving on passive vehicle dynamics," *SAE International Journal of Vehicle Dynamics, Stability, and NVH*, 2018.

[73] X. Zhange, A. Liniger, and F. Borrelli, "Cautious NMPC with gaussian process dynamics for autonomous miniature race cars," in *Conference on Decision and Control (CDC)*, 2018.

[74] T. Novi, A. Liniger, R. Capitani, and C. Annicchiarico, "Real-time control for at-limit handling driving on a predefined path," *Vehicle System Dynamics (submitted)*, 2018.

[75] C. Duenner and S. Merkli, "DC motor controller with estimator and new track with optimal trajectory for ORCA," 2012, Group Project, IfA ETH Zürich.

[76] M. Dahinden and C. Stocker, "Design and implementation of an embedded sensing and control platform for 1:43 scale R/C race cars," 2012, Semester Thesis, IfA ETH Zürich.

[77] J. Schaefer, "Improved embedded sensing and communication for 1:43 RC cars," 2015, Semester Thesis, IfA ETH Zürich.

[78] "Vicon motion system ltd," www.vicon.com, 2017.

[79] B. Kaiser, "Redesign of infrared camera system for ORCA," 2013, Semester Thesis, IfA ETH Zürich.

[80] Itseez, "Open source computer vision library," https://github.com/itseez/opencv, 2015.

[81] S. Tanner, "Improved computer vision for tracking of 1:43 RC cars," 2014, Semester Thesis, IfA ETH Zürich.

[82] M. Rutschmann, "Infrared based vision system for tracking 1:43 scale race cars," 2010, Semester Thesis, IfA ETH Zürich.

[83] D. Stettler, “State and friction estimation for 1:43 RC cars,” 2015, Semester Thesis, IfA ETH Zürich.

[84] D. Lam, C. Manzie, and M. Good, “Model predictive contouring control,” in *Conference on Decision and Control (CDC)*, 2010, pp. 6137–6142.

[85] T. Faulwasser, B. Kern, and R. Findeisen, “Model predictive path-following for constrained nonlinear systems,” in *Conference on Decision and Control (CDC)*, 2009, pp. 8642–8647.

[86] T. Nägeli, L. Meier, A. Domahidi, J. Alonso-Mora, and O. Hilliges, “Real-time planning for automated multi-view drone cinematography,” *ACM Transactions on Graphics*, vol. 36, no. 4, pp. 132–142, 2017.

[87] W. Schwarting, J. Alonso-Mora, L. Paull, S. Karaman, and D. Rus, “Safe nonlinear trajectory generation for parallel autonomy with a dynamic vehicle model,” *IEEE Transactions on Intelligent Transportation Systems*, 2017.

[88] A. Domahidi, A. Zgraggen, M. Zeilinger, M. Morari, and C. Jones, “Efficient interior point methods for multistage problems arising in receding horizon control,” in *Conference on Decision and Control (CDC)*, Maui, HI, USA, Dec. 2012, pp. 668–674.

[89] F. Leutwiler, “Nonlinear MPC for miniature RC race cars,” 2016, Semester Thesis, IfA ETH Zürich.

[90] A. Zanelli, A. Domahidi, J. Jerez, and M. Morari, “Forces nlp: an efficient implementation of interior-point methods for multistage nonlinear nonconvex programs,” *International Journal of Control*, pp. 1–17, 2017.

[91] G. C. Calafiore and M. C. Campi, “The scenario approach to robust control design,” *IEEE Transactions on Automatic Control*, vol. 51, no. 5, pp. 742–753, 2006.

[92] Y. Darouach, “Racing backwards with 1:43 RC cars,” 2014, Semester Thesis, IfA ETH Zürich.

[93] N. Kariotoglou, M. Kamgarpour, T. H. Summers, and J. Lygeros, “The linear programming approach to reach-avoid problems for markov decision processes,” *Journal of Artificial Intelligence Research*, vol. 60, pp. 263–285, 2017.

[94] F. Böwing, A. Schaper, and J. Schuurmans Stekhoven, “How to jump with RC-cars,” 2016, Group Project, IfA ETH Zürich.

[95] H. Banzhaf, “Trajectory-tracking using MPC for an autonomous solar vehicle,” 2014, Master Thesis, IfA ETH Zürich/TUM.

[96] R. N. Jazar, *Vehicle dynamics: Theory and Applications.* Springer, 2008.

[97] H. Pacejka, *Tire and vehicle dynamics.* Elsevier, 2005.

[98] M. Zanon, J. V. Frasch, and M. Diehl, "Nonlinear moving horizon estimation for combined state and friction coefficient estimation in autonomous driving," in *European Control Conference (ECC)*, 2013, pp. 4130–4135.

[99] E. Velenis, E. Frazzoli, and P. Tsiotras, "On steady-state cornering equilibria for wheeled vehicles with drift," in *Conference on Decision and Control (CDC)*, 2009, pp. 3545–3550.

[100] C. Voser, R. Y. Hindiyeh, and J. C. Gerdes, "Analysis and control of high sideslip manoeuvres," *Vehicle System Dynamics*, vol. 48, pp. 317–336, 2010.

[101] E. Bakker, L. Nyborg, and H. Pacejka, "Tyre modelling for use in vehicle dynamics studies," *SAE Technical Paper*, 1987.

[102] C. Tomlin, G. J. Pappas, and S. Sastry, "Conflict resolution for air traffic management: A study in multiagent hybrid systems," *IEEE Transactions on Automatic Control*, vol. 43, no. 4, pp. 509–521, 1998.

[103] M. Herceg, M. Kvasnica, C. Jones, and M. Morari, "Multi-Parametric Toolbox 3.0," in *European Control Conference (ECC)*, 2013, pp. 502–510.

[104] J. Lygeros, K. H. Johansson, S. N. Simic, J. Zhang, and S. Shankar, "Dynamical properties of hybrid automata," *IEEE Transaction on Automatic Control*, vol. 48, no. 1, pp. 2 – 17, 2003.

[105] K. Margellos and J. Lygeros, "Viable set computation for hybrid systems," *Nonlinear Analysis: Hybrid Systems*, vol. 10, pp. 45–62, 2013.

[106] J. Kong, M. Pfeiffer, G. Schildbach, and F. Borrelli, "Kinematic and dynamic vehicle models for autonomous driving control design," in *Intelligent Vehicles Symposium*, 2015, pp. 1094–1099.

[107] A. Liniger, "Autonomous drift control," 2012, Master Thesis, IfA ETH Zürich.

[108] I. M. Mitchell and S. Kaynama, "An improved algorithm for robust safety analysis of sampled data systems," in *Hybrid Systems: Computation and Control (HSCC)*, 2015, pp. 21–30.

[109] D. G. Luenberger, *Linear and nonlinear programming.* Springer, 2003.

[110] E. C. Kerrigan and J. M. Maciejowski, "Soft constraints and exact penalty functions in model predictive control," in *Proc. UKACC International Conference (Control 2000)*, Cambridge, UK, September 2000.

[111] A. Domahidi and J. Jerez, "FORCES Professional," embotech GmbH (http://embotech.com/FORCES-Pro), Jul. 2014.

[112] P. Cardaliaguet, "A differential game with two players and one target," *SIAM Journal on Control and Optimization*, vol. 34, no. 4, pp. 1441–1460, 1996.

[113] P. Cardaliaguet, M. Quincampoix, and P. Saint-Pierre, "Some algorithms for differential games with two players and one target," *Mathematical Modelling and Numerical Analysis*, vol. 28, no. 4, pp. 441–461, 1994.

[114] J.-P. Aubin and H. Frankowska, *Set-valued analysis.* Springer, 1990.

[115] J. Bekker and W. Lotz, "Planning formula one race strategies using discrete-event simulation," *Journal of the Operational Research Society*, vol. 60, no. 7, pp. 952–961, 2009.

[116] J. Schulman, J. Ho, A. X. Lee, I. Awwal, H. Bradlow, and P. Abbeel, "Finding locally optimal, collision-free trajectories with sequential convex optimization," in *Robotics: Science and Systems*, 2013, pp. 1–10.

[117] T. Basar and G. J. Olsder, *Dynamic noncooperative game theory.* Academic Press, 1982.

[118] K. H. Johansson, M. Egerstedt, J. Lygeros, and S. Sastry, "On the regularization of zeno hybrid automata," *Systems and Control Letters*, vol. 38, no. 3, pp. 141–150, 1999.

[119] S. Gottschalk, M. C. Lin, and D. Manocha, "Obbtree: A hierarchical structure for rapid interference detection," in *Computer Graphics and Interactive Techniques*, 1996, pp. 171–180.

[120] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[121] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton *et al.*, "Mastering the game of go without human knowledge," *Nature*, vol. 550, no. 7676, p. 354, 2017.

[122] S. Norwin, "High performance path planning using GPU for autonomous racing car," 2017, Semester Thesis, IfA ETH Zürich.