Real-time and efficient control for autonomous racing

Nan Li

INSTITUT POLYTECHNIQUE DE PARIS

NNT : 2023IPPAT008

Thèse de doctorat

Real-Time and Efficient Control for Autonomous Racing

Thèse de doctorat de l'Institut Polytechnique de Paris
préparée à Télécom Paris

École doctorale n°626 École doctorale de l'Institut Polytechnique de Paris
(ED IP Paris)
Spécialité de doctorat : Informatique

Thèse présentée et soutenue à Palaiseau, le 24 mars 2023, par

NAN LI

Composition du Jury :

Liliana Cucu-Grosjean Directrice de recherche, INRIA (KOPERNIC)	Présidente, Examinatrice
Michel Kieffer Professeur, CentraleSupélec (L2S)	Rapporteur
Frank Singhoff Professeur des Universités, Université de Bretagne Occidentale (STICC)	Rapporteur
Pierre-Loïc Garoche Professeur, ENAC (LII)	Examinateur
Laurent Pautet Professeur, Télécom Paris (LTCI)	Directeur de thèse
Eric Goubault Professeur, École Polytechnique (LIX)	Co-directeur de thèse
Sylvie Putot Professeur, École Polytechnique (LIX)	Invitée

Acknowledgements

I am deeply grateful to my thesis advisors, Sylvie, Eric, and Laurent, for their invaluable guidance and support throughout my doctoral journey. Having met them during my master's studies, I was immediately drawn to their rigorous academic attitude and energetic teaching. Throughout my Ph.D. study, the regular meetings and discussions with them helped me understand the paradigm of scientific research and enhance my presentation skills, both orally and in writing. While the research paradigm of "identifying and defining the problem, targeting the problem and developing the solution, presenting the result in a scientifically rigorous way" may seem simple, it was not easy for me to internalize and incorporate it into my own thinking. Fortunately, with the help of my mentors, I gradually gained insight into this process, which not only enhanced my thesis work but will also certainly benefit my future career. By the way, I believe that this research paradigm will serve as a valuable foundation for understanding, designing, and evaluating prompts in the era of large language models (LLM). In addition to their academic guidance, they provided an excellent working environment and offered compassionate care, encouraging me to maintain a healthy work-life balance, especially during the challenging times of the pandemic. During the writing of the dissertation and the preparation for the defense, they were always aware of my emotional state and provided me with genuine encouragement and support as both mentors and friends.

I extend my heartfelt gratitude to the reviewers, Michel and Frank, for their precious feedback. Their comments provided me with profound insights and helped me delve deeper into my research, refine my dissertation, and prepare for my defense. Michel, together with Xavier, also participated in my mid-term defense, offering crucial suggestions that enabled me to identify issues in my project and explore potential solutions. I am immensely grateful for their help, which was important for the completion of subsequent work. I would also like to thank the examiners, Liliana and Pierre-Loïc, for agreeing to be members of the thesis committee and for sharing their expertise in a very friendly and supportive manner!

Thanks to the enjoyable atmosphere in the lab, I was able to happily carry out my Ph.D. work for over three years. Engaging in discussions with my colleagues has

provided me with a lot of new ideas and spiritual support. Amidst the pandemic, François, Maria, and I worked together to participate in an online competition. The experience was enriching as we collaborated and learned from one another. François and Elena provided me with valuable advice on the writing of my Ph.D. thesis. With Aloysio and Mathile, I had a lot of enlightening discussions regarding robot projects, and I appreciate very much their help in rehearsing my defense. I also had many fascinating conversations with Sergio and Riccardo regarding life and Italian cuisine. Emmanuel's genuine smile always exudes warmth, and I appreciate his introduction to French and Swiss culture as well as our discussions on BigLiu's science-fiction books. I am grateful to Nathan and Aly-Bora for their help in reviewing and improving the French version of the abstract and introduction in this thesis. I really appreciate Bernardo's kind invitation to several parties and his efforts in creating a convivial and cheerful office ambiance. Martin generously provided me with PowerPoint templates and insights for the defense. Conversions with Jiayi, Jiong, Yanzhu, and Mathieu have been inspiring and have made my life less monotonous. I am grateful to Jakie for her enthusiastic help and encouragement. Other colleagues in the LIX lab were incredibly supportive as well and they shared a lot of interesting ideas. Moreover, I want to express my gratitude to Samuel and Etienne from the LTCI lab for offering me the opportunity to participate in the courses and the monitoring of student projects, which allowed me to gain a fresh perspective on teaching in Télécom Paris.

I am genuinely grateful for the support of my friends outside the lab throughout my doctoral studies. I would like to express my sincere gratitude to my friend Bernard, whom I first met during my time at Télécom ParisTech in the Rue Barrault era, more specifically, at the train station of Dax for the school's "week-end d'intégration". Bernard has always been a great listener, empathetic to my feelings, and blessed with intelligence and a nice sense of humor. I have pleasant memories of my trips to Clément-Ferrand, where I received a very warm reception from Bernard's parents, Paula and Michel. Their generosity and thoughtfulness are deeply appreciated. I would like to extend my sincere thanks to Adrien and Fabien, whom I first met during summer school in Poitier. Since then, we have continued to meet regularly and offer each other constant support and encouragement. In addition, I must express my profound gratitude to my Chinese friends, especially my book club companions, who have had a significant impact on my personal and intellectual growth beyond the scope of my Ph.D. research. Our shared experiences, such as reading and discussing together, traveling together, and collaborating together on our podcast "Books and Faraway Life", have been tremendously enjoyable.

Ever since I set foot in France in July 2016, I have been benefiting from its openness and tolerance that encourages self-development. A rich recreational life (exhibitions, skiing, etc.) has broadened my horizons and provided me with great inspiration for self-knowledge and personal growth. I am grateful to the French higher education system for its characteristic engineering program and doctoral system that prioritizes personal growth. A quote from Ernest Hemingway, though maybe clichéd, perfectly captures how I feel and what I hope at this moment: "If you are lucky enough to have lived in Paris as a young man, then wherever you go for the

rest of your life, it stays with you, for Paris is a moveable feast."

Among everything else, my parents have always been a pillar of great support, despite being thousands of miles away. Ever since I was a child, I have looked up to them as role models for their diligent work ethic and kind demeanor. Their commitment to understanding and accepting my new perspectives and their empathy toward my various situations have been invaluable to me. Their unconditional support has enabled me to pursue my dreams and make important decisions with confidence, and for that, I am forever grateful.

Contents

List of Figures

List of Tables

Nomenclature

Abbreviations

ACADO	Toolkit for Automatic Control and Dynamic Optimization
ADAS	Advanced Driver Assistance Systems
CoG	Center of Gravity
CPS	Cyber-Physical Systems
DAG	Directed Acyclic Graph
EK	Extended Kinematic Model
HiL	Hardware-in-the-Loop
IP	Interior Point method
KKT	Karush-Kuhn-Tucke condition
KS	Kinematic Single-track Model
LiDAR	Light Detection and Ranging
LOROFO	Left-side Overtaking, Right-side Overtaking, and FOllowing
MB	Multi-Body Model
MINLP	Mixed Integer Nonlinear Programming
MIP	Mixed Integer Programming
MIQP	Mixed Integer Quadratic Programming
MPC	Model predictive control

NMPC	Nonlinear Model predictive control
OD	Opponent Detection
ODE	Ordinary Differential Equations
PF	Particle Filter
QP	Quadratic Programming
ROS	Robot Operating System
SLAM	Simultaneous Localization and Mapping
SQP	Sequential Quadratic Programming
ST	Single-Track Model
WCET	Worst-Case Execution Time

Notations

α_F	slip angle at front wheels
α_R	slip angle at rear wheels
$\Delta\delta$	change rate of *delta*
Δd	change rate of d
Δt_k^c	calculation time used for step k
Δt_k^p	progress time for step k
δ	steering angle
$\kappa(s)$	curvature of the center line at progress s
$\mathcal{L}$	Lagrangian
ω	angular velocity
degree	the actual number of DAG components that are executed in parallel
$degree_{DAG}$	the maximum number of tasks in the DAG that can be executed in parallel on an unbounded number of processors
ξ	state vector
d	motor engine parameter
e_y	relative distance between the CoG of the vehicle and the projection point
e_ψ	relative angle between the orientation of the vehicle and the tangent angle at the projection point
L, R	parameters in task execution models: end-to-end latency, control refresh rate
s	progress distance along the track's center line
t	progress time
T^{ctrl}	control update rate
u	control vector
v_x	lateral velocity
v_y	longitudinal velocity

CHAPTER 1

Introduction

1.1 Motivation and background

In recent years, the research of sophisticated control algorithms has gained significant attention. Lots of efforts have been made to enhance the quality of service in autonomous systems. Despite the notable progress, there is still a need to further investigate their application in real-time settings. This dissertation focuses on real-time control for autonomous driving and more specifically autonomous race car racing.

Autonomous driving is classified by the Society of Automotive Engineers [1] from Level 0 (no driving automation), Level 1 (driver assistance), up to Level 5 (full driving automation under all conditions). Advanced Driver-Assistance Systems [2] (ADAS, including electronic stability control, roll stability control, lane departure warning, adaptive cruise control, parking assistance systems, etc.) provide Level 1 functions, which assist human drivers to drive safely, efficiently, and with a better user experience. The study of ADAS has been ongoing for decades. DARPA Urban Challenge [3] is one milestone that has greatly promoted the advancement of the field of autonomous driving. Since then, related research has been massively carried out in both academia and industry. Although several companies have released autonomous driving systems with high levels of automation, e.g. Tesla's Full Self-Driving (FSD) [4], Level 5 driving is still a decade or more away from the grand public because of ethical and regulatory issues, along with several technical difficulties: the inability to handle unexpected extreme situations, the absence of sophisticated understanding of implicit behaviors of traffic agents, etc.

Autonomous vehicle racing [5] is a special form of autonomous driving, in which the race car should achieve a lap time as short as possible while avoiding potential collisions with track boundaries or rival vehicles. Since the race car usually runs at a high speed and requires extreme handling capacities (active drift, sharp steering, etc.), we have the opportunity to study the system's performance at its physical

limits. These investigations on edge cases can also help advance the study of autonomous passenger car driving [6]. Other relevant domains may derive benefits from the study of autonomous vehicle racing as well. For instance, the time-optimal control problem for a race car, which is under engine and traction saturation constraints and over a confined trajectory, is analogous to the time-optimal control problem for a robot arm that moves through a constrained path with actuator saturation constraints. In both cases, in order to achieve desired behaviors, the control system takes into account constraints and objectives in a similar problem structure. From the perspective of the academic world, the setting up of autonomous racing is an ideal research platform for the study of Cyber-Physical Systems (CPS) [7], where the physical plants (i.e. race cars) have rapidly changing system dynamics, and the computation is typically carried out on embedded systems as widely used in CPS.

Model predictive control (MPC) is a state-of-the-art control technique for controlling a physical plant in the receding horizon framework to achieve a specified objective while satisfying a set of constraints. Among different control techniques for the autonomous vehicle racing problem, we are especially interested in MPC for the following reasons:

1. MPC is one of the most promising model-based methods for taking into account accurate physical dynamic models and pertinent constraints. It makes sense to apply MPC to racing problems since MPC can effectively incorporate our prior knowledge of race car dynamics and make use of pre-defined state/control constraints of race cars.

2. MPC is one typical representative of optimization-based controllers that in general provide optimal results at the cost of high complexity. By studying MPC on a race car with embedded computing units, we gain a better understanding of how to effectively design and deploy optimization-based algorithms on devices with limited calculation resources.

3. MPC is extensible for working in conjunction with other control techniques. For instance, it can cooperate with Mixed Integer Programming (MIP) for decision-making problems. In the literature, it is coupled with several other tools to enhance its capability and improve its performance, e.g. game theory [8] for enabling MPC to react to the opponents' intended actions, viability theory [9] for ensuring the recursive feasibility of MPC, Koopman operator theory [10] for assisting the system identification of a vehicle with unknown dynamics in MPC, etc.

In this study, to precisely capture the system dynamics of the autonomous race car, we represent them using ordinary differential equations (ODE). They will appear in MPC as nonlinear equality constraints. Under some circumstances, inequality constraints in MPC, such as collision avoidance constraints, might also be nonlinear. The focus of this work is hence Nonlinear MPC (NMPC).

NMPC needs to work *correctly* in 2 senses. On one hand, its prediction horizon should be long enough for providing reliable control. On the other hand, NMPC with

a longer prediction horizon requires more computation resources. It might result in an obsolete result, i.e. result arriving after the deadline, that is regarded as useless and incorrect. Since deterministic behaviors are desirable in real-time systems, the worst-case execution time of NMPC should be bounded within a reasonable value considering the limited resource of embedded systems.

Controllers, including NMPC, collaborate with other system components, such as perception tasks for self-localization and opponent detection. They work as a task chain and share computation resources. Different computational components may be parallelizable, and modern processors, such as multi-core CPUs and GPUs, have the ability to parallelize tasks. It is therefore crucial building a suitable task execution model to maximize the use of the processor's parallelization capabilities for decreasing latency and increasing control update rate, which improves overall controller performance and prevents non-deterministic behaviors.

1.2 Research scope

In this section, we define the research scope of the dissertation. The research problem is limited to autonomous vehicle racing in 2 modes: single-vehicle mode (similar to the time trial mode in Formula 1 qualifying sessions, in which a single race car aims to achieve the fastest lap time) and head-to-head mode (in which two race cars compete on the same racetrack for being first to cross the finish line).

In Chapters 3 and 4, which cover algorithm-level controller design, we assume that the controller of the ego vehicle can access its own exact state as the control input, as well as the pose and intended trajectory of the opponent vehicle. Both vehicles are assumed to have prior knowledge of the curvature information anywhere on the racetrack.

In Chapter 5, which is related to system-level task execution model design, we assume that the LiDAR data serves as the ego vehicle's control input. Perception algorithms are needed for self-localization and opponent detection. Additionally, the opponent vehicle's intended trajectory is not provided. The experiment is performed in a Hardware-in-the-Loop (HiL) setup, i.e. software components of the ego vehicle are executed on an embedded device while other simulation components run on a standard laptop.

In this dissertation, we address the following problems.

- NMPC typically requires a relatively long solving time, especially when the prediction horizon is long. Meanwhile, long-horizon NMPC generally yields a better outcome. We are interested in knowing how to make long-horizon NMPC feasible within a given time budget while both reducing computation costs and obtaining satisfactory performance.

- Decision-making, e.g. overtaking, is a crucial element in the racing problem. Combining decision-making methods with existing control frameworks is still an open question. We will discuss how to formulate the overtaking problem

in a MIP form and how to create a simpler yet online feasible overtaking controller.

- A task execution model needs to be properly built to reduce latency, increase control update rate and ensure that tasks are completed before their deadlines. We consider the task's parallelism and the processor's parallelization capabilities when constructing this model. As a result, it helps to improve control performance and reliability.

There are still some interesting topics not included in the research scope. For example, taking into account the uncertainty (which might come from either the environment or the model mismatch of system dynamics) is meaningful for deploying algorithms on vehicles operating in imperfect circumstances.

1.3 Thesis outline

The dissertation is organized in the manner described below. We first introduce the context and state-of-the-art of the autonomous racing problem. The aforementioned problems in Section 1.2 are then addressed respectively in different chapters.

- Chapter 2, Autonomous race car control: state of the art, covers topics on control techniques for the autonomous racing problem. A racetrack model and vehicle dynamics are discussed. The MPC principles and associated solving techniques are presented. We introduce the time-optimal NMPC that serves as a basis for the following chapters. A discussion of the head-to-head racing problem gives insight into decision-making problems in the racing scenario.

- Chapter 3, Autonomous racing in single-vehicle mode, focuses on the complexity of NMPC and presents the timing issue raised by the classical NMPC framework in single-vehicle racing mode. We come up with a triggering-based solution for meeting the time budget constraint and reducing computational workload while maintaining adequate lap time performance.

  Contribution: The proposed method uses two triggering conditions. One is associated with the time budget constraint and enables real-time NMPC execution with a long prediction horizon. The other is inspired by the racing behavior of human drivers. It seeks to allocate computational loads in accordance with how the surrounding environment is changing, which can be regarded as an on-demand computing model.

- Chapter 4, Autonomous racing in head-to-head mode, concentrate on the autonomous racing problem in head-to-head racing mode. We suggest encoding collision-avoidance constraints into the form of MIP for making overtaking decisions within an NMPC framework in the curvilinear coordinate system. Based on the observations in the experiment of the MIP-based approach, an alternative overtaking strategy is proposed to enable online execution.

Contribution: We provide a technique for representing the geometry of the vehicle, which can be used to set up collision-avoidance constraints in the curvilinear coordinate. The combination of NMPC and MIP provides one effective way for resolving decision-making challenges in the racing problem. As an alternative method to MIP, the algorithm LOROFO (Left-side Overtaking, Right-side Overtaking, and FOllowing) efficiently attempts grouped overtaking decisions. This algorithm is online feasible and serves as an example of showing how to reduce the complexity of decision-making problems.

- Chapter 5, Task execution model for autonomous racing systems, first introduces a generic system architecture for the autonomous race car, including software components, task chain, and hardware platforms. A task execution model is then proposed for assigning software components to available processors with different parallelism degrees. This task execution model aims to decrease the latency and enhance the control update rate. A Hardware-in-the-Loop simulation is performed to demonstrate how the task execution model is implemented in the Robot Operating System (ROS) and how it performs.

  Contribution: The task execution model is validated to efficiently improve system performance in the race scenario by making use of the parallelization capabilities of processors available. It provides insight for building task execution models in other similar autonomous systems.

CHAPTER 2

Autonomous race car control: state of the art

In this chapter, we review the development of control techniques from autonomous passenger car driving to autonomous race car racing. Essential ingredients such as the racetrack model and vehicle dynamics are investigated. As one state-of-the-art control method that is used throughout this dissertation as a basis, the Model Predictive Control (MPC) based time-optimal controller is comprehensively discussed. For head-to-head racing, various approaches such as Mixed Integer Programming (MIP) are introduced.

2.1 Control design for autonomous vehicles

Techniques in the autonomous passenger car driving domain have been developed for decades, which typically hierarchically structure the control system into 4 levels [11]: global planning, behavioral decision-making, local motion planning, and feedback control. We begin by introducing various techniques used for autonomous passenger car driving at each control level, particularly those that have strong connections to the autonomous race car racing problem. With the reason stated at the end of this section, in this dissertation, we will primarily investigate a control framework that integrates decision-making, local planning, and feedback control in a one-level structure. Despite this, we present the state-of-the-art control techniques for the autonomous race car racing problem still in a 4-level structure to facilitate the comparison with the development in the autonomous passenger car driving field.

The **global planning**, also known as route planning in the passenger car driving field, is on the highest level where a vehicle selects a minimum-cost path on a road network graph using algorithms such as Dijkstra [12] and A* [13]. After a route plan has been found, the autonomous vehicle uses the **behavioral decision-making** component to interact with other traffic participants in accordance with traffic regulations and conventions.

After determining a local driving intention, the **local motion planning** component generates a collision-free path/trajectory (time-parametrized path) that satisfies kinematic or dynamic constraints on the vehicle's motion. It usually comes with the problem of optimum seeking for a given objective function with different purposes: minimizing travel time, penalizing hazardous motions, maximizing passenger comfort degree, etc. Three categories of planning methodology are available: graph-searching methods, incremental tree-based methods (e.g. RRT [14] and RRT* [15]), and optimization-based methods. Since optimization-based methods can explicitly take into consideration the satisfaction of constraints and the minimization of objectives, they are more desirable than other planning methods. However, they often involve a high level of complexity. The **feedback control** on the low level is used to stabilize the reference path/trajectory tracking in the presence of modeling error and other forms of uncertainty. Various methods are available: geometric-based method (e.g. Pure Pursuit [16]), output feedback linearization [17], MPC, etc.

Table 2.1: Comparison between autonomous passenger car driving and autonomous race car racing.

	Autonomous Passenger Car Driving	**Autonomous Race Car Racing**
Road type	high-way / urban road with lane segment	closed racetrack
Objective	safety, travel time, smoothness, passenger comfort degree	safety, lap time
Physical state	(usually) far away from limit	(often) close to limit
Behavior constraint	traffic conventions	competition rules

Autonomous passenger car driving and autonomous race car racing share many techniques. For instance, most works on autonomous racing also use a similar 4-level control structure as autonomous driving [5]. There are some distinctions between the two research fields as well, which are listed in Table 2.1 and discussed in more detail below.

On the **global planning** level of the autonomous racing system, the primary task for the race car is to solve the minimum lap time problem, i.e. find an optimal trajectory which is called the race line. Existing methods can be classified into two main groups [18]: quasi-steady state methods and transient optimal methods. Quasi-steady state methods explore the feature that the racing is performed on a closed racetrack and can be split into a sequence of segments. The vehicle is assumed in stationary conditions for each segment, i.e. with constant speed and lateral acceleration. Geometrical constraints (e.g. the track boundaries) and mechanical constraints (e.g. engine power, braking capacity, and tire limits) are imposed on each segment for calculating the race line profile. Quasi-steady state methods include several variants: using evolutionary algorithms to find a set of control point positions [19][20], mimicking human drivers' behavior to define straight paths and

curve paths along the racetrack [21][22], finding the best compromise between the shortest and the least curvature track based on geometric properties of the track [23]. Transient optimal methods optimize an objective function, e.g. maximizing the traveled distance along a racetrack in a given time [24], maximizing the vehicle speed at the final point of the maneuver [25], or directly minimizing the lap time [26][27] by transferring the time-dependent system into a space-dependent and space-variant system.

On the **behavioral decision-making** level, approaches such as game theory can be used for finding the best action to win the racing game by taking into consideration adversary vehicles' intentions and responses [28][29][30]. The appearance of the opponent vehicles can also be handled on the **local motion planning** level by modifying the global plan, i.e. to find a collision-free local trajectory by changing the constraints or cost functions [31][32], assembling motion primitives [33][34][35][36], or dynamically sampling feasible local paths in free space [37][38][39].

The low-level **feedback control** usually serves to track the race line at the potential physical limit. Authors in [40] use the G-G diagram (a graphical representation of the performance envelope of a car, with x/y axis standing for lateral/longitudinal acceleration) to design a feedforward controller and use the safe $\beta - r$ phase portrait (slide slip angle - yaw velocity, calculated using the Pacejka Magic Formula [41]) envelopes to generate active yaw moment for tracking the race line at its limits. Authors in [42] set up feedforward and feedback controllers to make use of all the available friction force. The MPC-based method [43] has the ability to forecast the future trajectory and act proactively to the potential tracking error. It is widely used with several variations: Tube-MPC [44] for dealing with nonlinear effects and external disturbances, Stochastic MPC [45] for handling model uncertainty at high speeds/accelerations, etc.

The hierarchical control structure that was described above has several disadvantages. For instance, the smoothness level of the high-level path planning is a factor that the low-level controller depends on. The system performance could be unsatisfactory if the intended racing line is not sufficiently smooth. Additionally, because racing lines are often computed offline by high-level planners, it is difficult to instantly adapt the race line to the changing environment from a global perspective. Actually, one can combine high-level planning and low-level action into one single control framework, which is the approach used in this dissertation.

Authors in [34] formulate the racing problem in a predictive control framework to incorporate system constraints (track boundaries, state/control constraints) and the objective function (maximization of progress distance). This framework directly optimizes the low-level control variable, allowing the vehicle to follow a locally optimal trajectory. Authors in [28] developed a one-level sampling-based MPC algorithm that is based on the path integral control [46] which relies on the stochastic sampling of trajectories. Authors in [47] directly optimize the progress time of the race car by using an MPC framework under the curvilinear coordinate system. As demonstrated in the aforementioned articles, the MPC framework can actually transfer the hierarchical control structure into one-level optimal control paradigms.

2.2 Racetrack model

A racetrack consists of a single lane that is defined as a drivable space with an inner and an outer bound. We focus on tracks in the form of 2 dimensions, where the inner and outer bounds are represented by sampling points (x, y), as used in the open-source racetrack library [48]. A common way to build a racetrack model is to construct an occupancy grid mapping from bound lines, which can assess whether a given position is on or off the track [28]. Another widely used method is to establish a reference line, such as the racetrack's center line that is determined by the inner and outer bounds [34][47]. The reference line can be represented as spline polynomials and parameterized by its arc length $s \in [0, L]$, where L is the total length of the center line.

In this dissertation, we use the latter method to model the racetrack. We first sample M control points $(x, y)_i, i = 1, \ldots, M$ on the center line of the racetrack as a reference. Then, we fit each segment between the control point i and $i + 1$ into a cubic spline interpolation in 2 dimensions: $(f_x(s), f_y(s))$, where s is the progress along the reference line and $f_{x/y}$ is the spline function for the axis x/y. Since the cubic spline is smooth and 2-order differentiable, it is convenient to calculate one important character of the racetrack: the curvature of the reference line at the progress s, i.e. $\kappa(s) = (f'_x(s)f''_y(s) - f'_y(s)f''_x(s))/(f'_x(s)^2 + f'_y(s)^2)^{3/2}$.

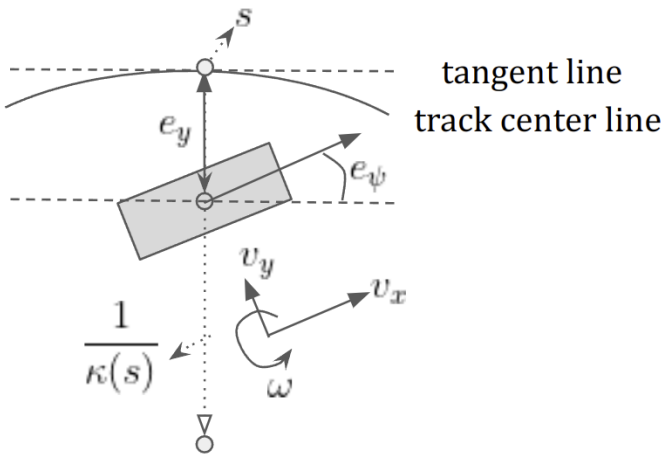

Figure 2.1: Vehicle state in the curvilinear coordinate.

As shown in Fig. 2.1, the vehicle's center of gravity (CoG) has a projection point on the reference line with the arc length s (i.e. the progress along the center line since the starting point). The deviation distance between the vehicle's CoG and the projection point on the reference line is denoted as e_y. The difference between the vehicle's orientation and the tangent angle at the projection point is denoted as e_ψ. We can thus set up the curvilinear coordinate along the reference line using the cubic spline interpolation, in which the vehicle's position is represented as (s, e_y, e_ψ).

In addition to the convenient calculation of curvature, there are other advantages to employing this racetrack model. For instance, the track boundary can be presented as a simple interval form, i.e. $e_y \in [\underline{e_y}, \overline{e_y}]$. In the curvilinear coordinate, we

can also set the progress s as the independent variable and use the progress time t as the dependent variable, which will be used later in Section 2.5 for directly setting the progress time as the optimization objective.

However, using this racetrack model in the curvilinear coordinate brings inconvenience to the representation of the vehicle's occupied area. In the Cartesian coordinate, it is usually represented as a simple rectangle. The existence of an overlap between two rectangles indicates the collision. The representation can be further simplified as a circle with a diameter equal to the length of the vehicle's diagonal. In this case, we compare the distance between the centers of the two circles with the diameter to verify the collision situation. The vehicle's occupied area can also be decomposed as several circles for getting a more precise representation than the single-circle one. The collision-free constraint requires that the distance between any pair of centers should be greater than the diameter. In the curvilinear coordinate, either a rectangle or circle from the Cartesian coordinate will be transformed into an irregular shape, making it difficult to directly port them for collision checking. In Section 4.2, we discuss how to design a new representation method under the curvilinear coordinate and set up the corresponding collision-avoidance constraints.

2.3 Vehicle dynamics

Different vehicle dynamics are reviewed in [49] and [50]. The point-mass model is the simplest model which contains the position, velocity, and acceleration at the vehicle's CoG. The Kinematic Single-Track Model (KS), which represents the front and rear wheel-pairs by two single wheels, is a more precise model. This kinematic model does not consider any tire slip and assumes that the vehicle follows exactly the desired kinematic path even if it is indeed dynamically infeasible. Single-Track Model (ST), which is also called the bicycle model, considers the slip angle of tires and thus can simulate important effects such as drifting, understeer driving, and oversteer driving. A more precise model, Multi-Body Model (MB), can model the load transferring between 4 wheels and other effects. Extended Kinematic Model (EK) [51] [52] is one model combining features of KS and ST, which takes into account the existence of tire slip angle but does not explicitly model the lateral force.

The high-fidelity vehicle models enable the controller to produce a more reliable solution, which is crucial for the racing problem. However, complex models also bring high computational costs. Since the ST model represents most of the dynamic effects that we need and its complexity is reasonably acceptable, we use the ST model in the control algorithm design in Chapter 3 and in Section 4.3 of Chapter 4. The analysis in [51] shows that, when vehicles run below a threshold velocity, the EK model results in a precise enough solution. Given that the vehicle model is a replaceable component in the controller, in Section 4.4 of Chapter 4 and in Chapter 5, we use the EK model within the controller to reduce calculation complexity and, as a result, to facilitate online calculation. The EK model is also attractive due to another fact that it needs fewer parameters for system identification, making it

easier to deploy in real-world settings in further research.

The ST model in the curvilinear coordinate is stated as in Eq. 2.1, in which the system state is defined as $\xi = [e_y, e_\varphi, v_x, v_y, \omega, t, s, d, \delta]$, where: v_x, v_y are the longitudinal and lateral velocities; ω is the angular velocities; d is the parameter for the motor engine; δ is the vehicle steering angle; t is the progress time of the vehicle. The control vector is $u = [\Delta d, \Delta\delta]$, i.e. change rates of d and δ.

$$
\begin{aligned}
\frac{d}{dt}\begin{bmatrix} e_y \\ e_\psi \\ v_x \\ v_y \\ \omega \\ t \\ s \\ d \\ \delta \end{bmatrix} &= \begin{bmatrix} v_x \sin(e_\psi) + v_y \cos(e_\psi) \\ \omega - \kappa(s) \cdot \dot{s} \\ \frac{1}{m}(F_{R,x} - F_{F,y} \sin\delta + m v_y \omega) \\ \frac{1}{m}(F_{R,y} + F_{F,y} \cos\delta - m v_x \omega) \\ \frac{1}{I_z}(l_f F_{F,y} \cos\delta - l_r F_{R,y}) \\ 1 \\ \dot{s} \\ \Delta d \\ \Delta\delta \end{bmatrix}, \\
\dot{s} &= \frac{v_x \cos(e_\psi) - v_y \sin(e_\psi)}{1 - e_y \cdot \kappa(s)} \\
F_{R,x} &= (C_{m_1} - C_{m_2} v_x) d - C_r - C_d v_x^2 \\
F_{F,y} &= D_f \sin(C_f \arctan(B_f \alpha_F)) \\
\alpha_F &= -\arctan((\omega l_f + v_y)/v_x) + \delta \\
F_{R,y} &= D_r \sin(C_r \arctan(B_r \alpha_R)) \\
\alpha_R &= \arctan((\omega l_r - v_y)/v_x)
\end{aligned}
\tag{2.1}
$$

where m is the vehicle weight; l_f and l_r are the distance between the vehicle center of mass and front/rear wheels respectively; $F_{R,x}$ denotes the longitudinal force of the rear wheel, which includes motor parameters C_{m_1} and C_{m_2} as well as friction parameters C_r and C_d; $F_{F/R,y}$ is a function of $(v_x, v_y, \omega, d, \delta)$ for modeling the lateral forces of the front and rear tires based on the Pacejka tire model, which depends on empirical parameters $(B_{f/r}, C_{f/r}, D_{f/r})$; slip angles at front/rear wheels, $\alpha_{F/R}$, capture intricate behaviors including drifting and some other phenomena.

The EK model is presented as in Eq. 2.2, in which the system state is defined as $\xi = [e_y, e_\varphi, v_x, v_y, \omega, t, s, \delta]$ and the control vector is $u = [a, \Delta\delta]$, i.e. the longitudinal acceleration a and the change rate of steering angle δ.

$$\frac{d}{dt}\begin{bmatrix} e_y \\ e_\psi \\ v_x \\ v_y \\ \omega \\ t \\ s \\ \delta \end{bmatrix} = \begin{bmatrix} v_x \sin(e_\psi) + v_y \cos(e_\psi) \\ \omega - \kappa(s) \cdot \dot{s} \\ a \\ \frac{l_r}{l_f + l_r}(\dot{\delta} v_x + \delta \dot{v_x}) \\ \frac{1}{l_f + l_r}(\dot{\delta} v_x + \delta \dot{v_x}) \\ 1 \\ \dot{s} \\ \Delta\delta \end{bmatrix}, \tag{2.2}$$

$$\dot{s} = \frac{v_x cos(e_\psi) - v_y sin(e_\psi)}{1 - e_y \cdot \kappa(s)}.$$

2.4 Experiment platform

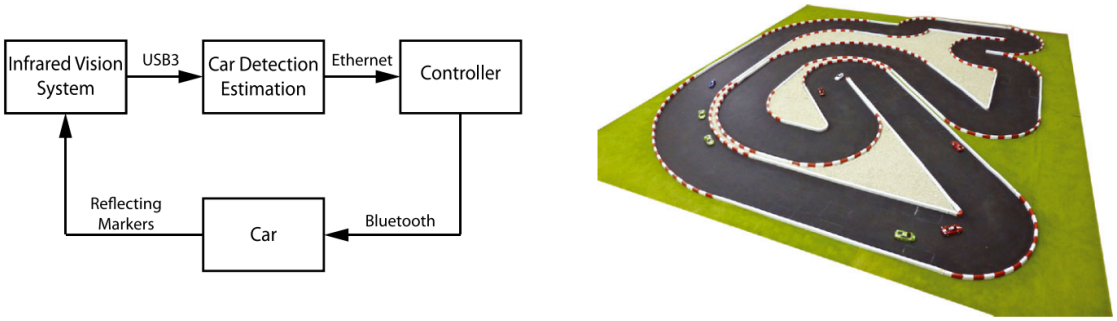

Figure 2.2: Control loop, 1:43 race car, and racetrack. Image taken from [34].

A 1:43 scale race car named Kyosho dnano is widely studied, e.g. in [34], [47] and [53]. Since we can make use of existing knowledge (such as system identification parameters) and use the previous research as a baseline, we use this miniature race car model in the simulation of Chapter 3 and in Section 4.3 of Chapter 4 for validating the controller design. The race car is too small (with a length of about $6cm$) to accommodate a powerful onboard computation card. As shown in Fig. 2.2, in [34], the calculation is performed in a controller implemented on the host PC and the race car simply executes the control command that it receives.

A 1:10 scale race car named F1tenth [35] is another experiment target that captures our attention. We use this model in the simulation of Chapter 5 and in Section 4.4 of Chapter 4. As shown in Fig. 2.3, it is a modularized experiment platform with an NVIDIA Jetson series card and a high-precision LiDAR which is similar to a full-scale autonomous vehicle. The chassis of F1tenth delivers realistic dynamics, allowing researchers to safely and cost-effectively test their algorithms on the 1/10 scale. F1tenth supports a popular software platform: the Robot Operating System (ROS), making it simple to leverage existing work in the robotics community. A realistic F1tenth simulator exists and it facilitates the development and deployment of algorithms.

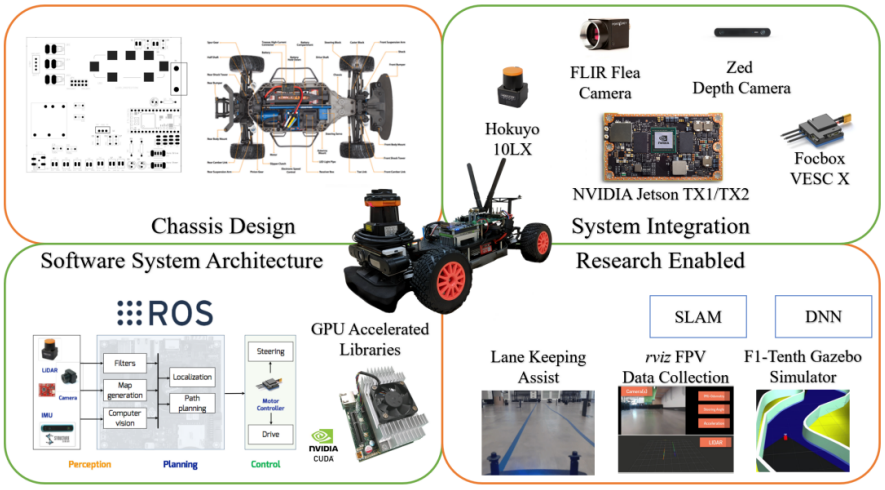

Figure 2.3: F1tenth race car and its related resources. Image taken from [35].

2.5 Model Predictive Control (MPC)

Model Predictive Control (MPC) is suited for controlling the autonomous race car since it explicitly takes into account system dynamics and physical constraints. In order to give a smooth and optimal control strategy, it can combine high-level planning and low-level action into a unified control framework. In this dissertation, we use a variant of MPC, Nonlinear MPC (NMPC), as the base controller for the autonomous race car to capture the non-linearity in dynamics and constraints. In this section, we first describe the principles of MPC. Then, we discuss time-optimal NMPC formulations. Finally, we present the main ingredients for solving an NMPC problem.

2.5.1 MPC principles

MPC, a.k.a receding horizon control, is originally used as an alternative for traditional feedback controllers (e.g. PID) in the application of oil and chemical industries [54][55]. With the development in the last decades, MPC can run reliably at millisecond sampling times even on embedded devices [56]. It aims to synthesize a feedback law μ while obeying constraints on state ξ and control u in a finite prediction horizon with the length N.

The first type of MPC, **tracking/stabilizing MPC**, synthesizes a feedback law μ for stabilizing the system state ξ to stay around the equilibrium state ξ^*. Its stage cost $l(\xi, u)$ penalizes the control effort u and the difference between ξ and ξ^*, e.g., $l(\xi, u) = ||\xi - \xi^*||^2 + \lambda||u||^2, \lambda \geq 0$.

An important property of the tracking/stabilizing MPC is asymptotic stability which implies both the attraction ($\xi_k \to \xi^*$ when step $k \to \infty$) and the stability (for an initialization close to ξ^*, the final solution remains close to ξ^*). With the help of the Lyapunov function, the asymptotic stability has been analyzed and proven under different conditions: with "equilibrium terminal constraint" (i.e. a constraint $\xi_N =$

ξ^* at the N-th/terminal step) [57], with "regional terminal constraint and terminal cost" (i.e. a regional terminal constraint $\xi^* \in \Xi$, in which Ξ is a region around ξ^*, and a Lyapunov function terminal cost F) [58], and without any stabilizing terminal constraints (i.e. without the aforementioned constraints) [59][60].

While MPC is effective at solving tracking/stabilizing problems, such as vehicle guidance, pressure maintenance, and temperature regulation, it can also be applied to problems that involve objectives other than target tracking. **Economic MPC**, in contrast to the tracking/stabilizing MPC, does not penalize the term for the difference between the predicted state and the equilibrium state in the stage cost. Instead, it directly set the economic criterion as the objective function. For example, if we let $l(\xi, u) = ||\xi||^2$, it directly optimizes the squared sum of the state.

In the economic MPC, if the optimal equilibrium is known a priori and set as a terminal constraint, the asymptotic stability and the optimality can be proven in an averaged sense [61], which means that the average long-run performance is no worse than the best admissible steady state. Without such terminal constraints, the asymptotic stability is only ensured around a small neighbor zone of ξ^* [62][63], and the averaged performance is only approximately achieved in a practical manner: a larger N, a better control result [64].

Once we select the type of MPC and determine the kind of stage cost function, we have a formal definition of MPC with a prediction horizon of length N and an initial state $\tilde{\xi}_k$ at control step k:

$$\min_{\text{u admissible}} J_N(\xi, u) = \Sigma_{i=0}^{N-1} l(\xi_{i|k}, u_{i|k}) + l(\xi_{i|N}), \xi_{0|k} = \tilde{\xi}_k \tag{2.3}$$

where $J_N(\cdot)$ is the objective function over N steps; states $\xi_{i|k}, i = 0, \ldots, N$ follow the requirement of dynamics evolution $\frac{d}{dt}\xi_{i+1|k} = f_{\text{dyn}}(\xi_{i|k}, u_{i|k})$; "u admissible" means that control and state stay in the feasible set, i.e. $u \in \mathcal{U}^N$ and $\xi \in \Xi^{N+1}$. The resulting optimal state series is denoted as $\xi^*_{0|k}, \ldots, \xi^*_{N|k}$, and the resulting optimal control sequence is denoted as $u^*_{0|k}, \ldots, u^*_{N-1|k}$. The MPC feedback law μ for a system with the initial state $\tilde{\xi}_k$ is defined as $\mu(\tilde{\xi}_k) := u^*_{0|k}$.

We should mention the similarity between the economic MPC and the continuous Optimal Control Problem (OCP). We first formulate the OCP as follows:

$$\min_{\xi,u} \int_t^{t+T} l(\xi(\tau), u(\tau))d\tau, \tag{2.4a}$$

$$s.t.\ \dot{\xi}(\tau) = f(\xi(\tau), u(\tau)), \tag{2.4b}$$

$$h(\xi(\tau), u(\tau)) \leq 0. \tag{2.4c}$$

where $f(\cdot)$ is the dynamic function and $h(\cdot)$ represents the state and control constraints. The OCP can be discretized into a form with finite parameterization, using either single-shooting (the intermediate states $\xi_0, \ldots, \xi_N$ disappear and there exist only piece-wise constant control variables $u_0, \ldots, u_{N-1}$) or multiple-shooting method (both state and control variables are treated as variables to be optimized, control variables are piece-wise constant while control variables are continuous) [65]. The resulting discretized OCP is equivalent to the economic MPC problem.

In this dissertation, we regard the racing problem as an OCP which optimizes an objective related to the lap time and solve it in the form of discretized OCP/economic MPC. More specifically, the racing problem will be solved in the form of time-optimal Nonlinear MPC that integrates the system dynamics, physical constraints, and objective function into a first-principle model-based optimization framework.

2.5.2 Time-optimal Nonlinear MPC (NMPC) formulation

In this section, we discuss several existing works on how to set up an objective function that leads to the fastest lap time. In the case that we discretize the prediction horizon in terms of progress time t, the time t is a dependent variable. It hence cannot be optimized directly, for which case there exist methods for approximating the time-optimal objective. Otherwise, the use of a curvilinear coordinate system allows the representation of progress s as the dependent variable and time t as the independent variable, allowing for the direct incorporation of time t into the objective function. In this case, there are several varieties of time-optimal objective functions to choose from. The following is a detailed explanation of the methods in both cases.

The model predictive contouring control (MPCC) based approach in [34] is the very first attempt in the literature to combine both path planning and path following into a one-level control framework as an NMPC problem. The authors approximate the time-optimal objective by maximizing the progress of the race car, which is measured as the distance traveled by a projection of the vehicle's center along the racetrack's center line. To handle the nonlinear constraints, they locally linearize the continuous nonlinear system dynamics to obtain a linear time-varying (LTV) model. The resulting convex optimization problem is then solved at each sampling time by an interior point solver.

Instead of indirectly maximizing progress to achieve the time-optimal objective, authors in [53] employ a transformation from the Cartesian coordinate to the curvilinear coordinate, making time t as a dependent variable to be optimized. Similar ideas for transforming time-dependent vehicle dynamics to track-dependent dynamics have been proposed in [66] and [67] for making it convenient to represent obstacles and road boundaries. As shown in Fig. 2.4, in a time-dependent system dynamics, the prediction horizon is discretized in terms of t, while in a track-dependent system dynamics, steps are discretized in terms of s. The transformation on the state vector ξ in the curvilinear coordinate is:

$$\frac{d}{ds}\xi = \frac{d\xi}{dt}\frac{dt}{ds} = \frac{d\xi}{dt}\frac{1}{\dot{s}} \tag{2.5}$$

where $\dot{s}$ is defined by the kinematic relation, i.e. as the velocity of the projection of the vehicle's center on the center line of the racetrack (see Fig. 2.4):

$$\dot{s} = (v_x cos(e_\psi) - v_y sin(e_\psi)) \cdot \frac{||\frac{1}{\kappa(s)}||}{||\frac{1}{\kappa(s)}|| + ||e_y||} = \frac{v_x cos(e_\psi) - v_y sin(e_\psi)}{1 - e_y \cdot \kappa(s)}.$$

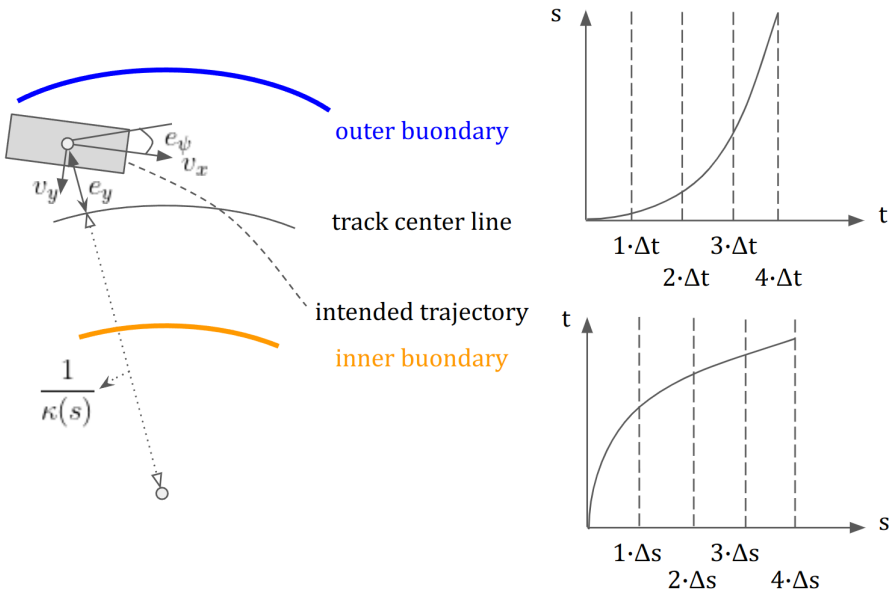

Figure 2.4: An example scenario demonstrating the vehicle's state and intended trajectory, including the relationship between time t and progress s along the trajectory in both a time-dependent and track-dependent system.

Authors in [53] describe the racing problem as a continuous optimal control problem (OCP). By using a curvilinear coordinate and applying the dynamic transformation (2.5), the system dynamics are transformed from time-dependent to track-dependent (i.e. depending on the vehicle's progress distance s). The progress time at the end of the prediction horizon is explicitly included in the objective function: $T = \int_{t_0}^{t_f} 1dt = \int_{s_0}^{s_f} \frac{1}{\dot{s}} ds$. To take the advantage of the existing efficient algorithms that rely on least-squares formulations and the generalized Gauss-Newton method, the authors propose the formulation as in (2.6).

$$\min_{\xi(\cdot),u(\cdot)} \quad ||T - T_{\text{ref}}||^2 \tag{2.6a}$$

$$s.t. \quad \xi(s_0) = \tilde{\xi}_0, \tag{2.6b}$$

$$\frac{d}{ds}\xi(s) = f_{\text{dyn}}(\xi(s), u(s)), \ \forall s \in [s_0, s_f] \tag{2.6c}$$

$$\xi(s) \in [\underline{\xi}, \overline{\xi}], \ \forall s \in [s_0, s_f] \tag{2.6d}$$

$$u(s) \in [\underline{u}, \overline{u}], \ \forall s \in [s_0, s_f] \tag{2.6e}$$

where T_{ref} in the objective function (2.6a) is a sufficiently small "target time". (2.6b) is the initial value condition that depends on state measurement or state estimation. (2.6c) requires that the evolution of system dynamics conform to system dynamics and it is formulated as ordinary differential equations (ODE). (2.6d) and (2.6e) impose physical restrictions on state and control in the form of intervals.

Although the least-squares objective function in formulation (2.6) is a good approximation for the time-optimal objective, the choice of parameter T_{ref} has a signif-

icant impact on the outcome and it is difficult to determine. Authors in [47] propose to directly set the progress time T as the objective function as shown in (2.7).

$$\min_{\xi(\cdot),u(\cdot)} \quad T \tag{2.7a}$$
$$s.t. \quad \xi(s_0) = \tilde{\xi}_0, \tag{2.7b}$$
$$\frac{d}{ds}\xi(s) = f_{\text{dyn}}(\xi(s), u(s)), \ \forall s \in [s_0, s_f] \tag{2.7c}$$
$$\xi(s) \in [\underline{\xi}, \overline{\xi}], \ \forall s \in [s_0, s_f] \tag{2.7d}$$
$$u(s) \in [\underline{u}, \overline{u}], \ \forall s \in [s_0, s_f]. \tag{2.7e}$$

The authors use a direct multiple-shooting method [68] for discretizing the prediction horizon of the continuous OCP (2.7) into N intervals. Control in each interval is piece-wise constant. The multiple-shooting approach differs from the single-shooting method in how they handle system dynamics; the former discretizes both control and state variables, whilst the latter discretizes only control variables. The multiple-shooting method is better suited for high-dimension state spaces and long prediction horizons, which is exactly the need for racing problems. The resulting formulation as in (2.8) is thus in a standard form of time-optimal Nonlinear MPC. The subscript k means that we are currently at the k-th control step.

$$\min_{\xi_i,u_i} \quad J_N(\xi, u) = t_{N|k} \tag{2.8a}$$
$$s.t. \quad \xi_{0|k} = \tilde{\xi}_k, \tag{2.8b}$$
$$\frac{d}{ds}\xi_{i+1|k} = f_{\text{dyn}}(\xi_{i|k}, u_{i|k}), \ i = 0, \dots, N-1 \tag{2.8c}$$
$$\xi_{i|k} \in [\underline{\xi}, \overline{\xi}], \ i = 0, \dots, N \tag{2.8d}$$
$$u_{i|k} \in [\underline{u}, \overline{u}], \ i = 0, ..., N-1. \tag{2.8e}$$

The challenge with formulation (2.8) is that the computationally efficient Gauss-Newton method, which is based on first-order sensitivities, is no longer appropriate. It must be solved using an exact Hessian-based technique that is based on second-order sensitivities and is usually expensive. Thanks to the work in [69], where a time and memory efficient algorithm is proposed using a novel symmetric algorithmic differentiation scheme, the propagation of second-order derivatives is online feasible. Combining this efficient method with the Hessian matrix regulation and condensing techniques, authors in [47] finally succeed to solve (2.8) for real-time racing problem in single-vehicle racing mode.

We note that a formulation similar to continuous OCP (2.7) is previously solved in [70] for a Formula 1 vehicle racing problem. However, the selected solving technique is time-consuming and the calculation must be performed offline, making it impractical for online use.

In this dissertation, we employ NMPC formulation (2.8) as a baseline controller for the autonomous race car in the single-vehicle racing mode. We are interested in how to enable its online execution and extend it to the head-to-head racing mode.

2.5.3 Ingredients for solving NMPC

In this subsection, we introduce ingredients for solving NMPC. We write NMPC formulation in a generic and compact form as in (2.9).

$$\min_{y} \quad f(y) \tag{2.9a}$$
$$s.t. \quad g(y) = 0 \tag{2.9b}$$
$$h(y) \leq 0 \tag{2.9c}$$

where we have vector $y = (\xi_i, u_i)$. (2.9a) represents a generic objective function, while (2.9b) and (2.9c) represent equality and inequality constraints.

The local optimal solution y^* of (2.9) satisfies the Karush-Kuhn-Tucker (KKT) conditions, i.e. there exists multiplier vectors λ^* and μ^* such that the following equations hold:

$$\nabla_y \mathcal{L}(y^*, \lambda^*, \mu^*) = 0 \tag{2.10a}$$
$$g(y^*) = 0 \tag{2.10b}$$
$$h(y^*) \leq 0 \tag{2.10c}$$
$$\mu^* \geq 0 \tag{2.10d}$$
$$h_i(y^*)\mu_i^* = 0, i = 1, \ldots, N_h \tag{2.10e}$$

where $\mathcal{L}(y, \lambda, \mu) = f(y) + g(y)^T\lambda + h(y)^T\mu$.

There are two families of Newton-type optimization methods [65] for solving NMPC problems in form (2.9): Sequential Quadratic Programming (SQP) [71] and Interior Point (IP) method [72].

With the SQP method, the KKT system (2.10) is sequentially solved. The KKT condition (2.10a) can be linearized around initial guessed values $(\overline{y}, \overline{\lambda}, \overline{\mu})$ as (2.11). (2.10b) and (2.10c) can be linearized as (2.12). Actually, (2.11) and (2.12) are exactly the KKT conditions for the Quadratic Programming (QP) problem (2.13). Solving (2.11) and (2.12) is equivalent to solving the sub-problem (2.13).

$$\begin{aligned}
&\nabla_y \mathcal{L}(\overline{y}, \overline{\lambda}, \overline{\mu}) + \\
&\nabla_y(\nabla_y \mathcal{L}(\overline{y}, \overline{\lambda}, \overline{\mu})) \cdot \Delta y + \\
&\nabla_{\overline{\lambda}}(\nabla_y \mathcal{L}(\overline{y}, \overline{\lambda}, \overline{\mu})) \cdot \Delta\lambda + \\
&\nabla_{\overline{\mu}}(\nabla_y \mathcal{L}(\overline{y}, \overline{\lambda}, \overline{\mu})) \cdot \Delta\mu = 0 \\
\rightarrow \nabla_y \mathcal{L}(\overline{y}, \overline{\lambda}, \overline{\mu}) &+ \nabla_y^2 \mathcal{L}(\overline{y}, \overline{\lambda}, \overline{\mu}) \cdot \Delta y + \nabla_y g(\overline{y}) \cdot \Delta\lambda + \nabla_y h(\overline{y}) \cdot \Delta\mu = 0 \\
\rightarrow \nabla_y f(\overline{y}) &+ \nabla_y^2 \mathcal{L}(\overline{y}, \overline{\lambda}, \overline{\mu}) \cdot \Delta y + \nabla_y g(\overline{y}) \cdot (\overline{\lambda} + \Delta\overline{\lambda}) + \nabla_y h(\overline{y}) \cdot (\overline{\mu} + \Delta\overline{\mu}) = 0
\end{aligned} \tag{2.11}$$

$$\begin{aligned}
g(\overline{y}) + \nabla_y g(\overline{y}) \cdot \Delta y = 0 \\
h(\overline{y}) + \nabla_y h(\overline{y}) \cdot \Delta y \leq 0
\end{aligned} \tag{2.12}$$

$$\begin{aligned} \min_{\Delta y} \quad & \frac{1}{2}\Delta y^T \nabla_y^2 \mathcal{L}(\cdot)\Delta y + \nabla_y f(\bar{y})\Delta y \\ s.t. \quad & g(\bar{y}) + \nabla_y g(\bar{y})\Delta y = 0 \quad | \ \bar{\lambda} + \Delta\lambda \\ & h(\bar{y}) + \nabla_y h(\bar{y})\Delta y \leq 0 \quad | \ \bar{\mu} + \Delta\mu \\ \text{where } & \mathcal{L}(\cdot) = f(\bar{y}) + g(\bar{y})^T\bar{\lambda} + h(\bar{y})^T\bar{\mu} \end{aligned} \tag{2.13}$$

After obtaining the QP solution $(\Delta y, \Delta\lambda, \Delta\mu)$, we update $(\bar{y}^+, \bar{\lambda}^+, \bar{\mu}^+)$ as in (2.14), where α is the step size that can be varied. We repeat the initialization and calculation of QP sub-problems until the original KKT condition (2.10) is fully satisfied.

$$\begin{aligned} \bar{y}^+ &= \bar{y} + \alpha\Delta y \\ \bar{\lambda}^+ &= \bar{\lambda} + \alpha\Delta\lambda \\ \bar{\mu}^+ &= \bar{\mu} + \alpha\Delta\mu \end{aligned} \tag{2.14}$$

We should mention that the QP sub-problems are convex only when the Hessian matrix $\nabla_y^2\mathcal{L}(\cdot)$ is positive and semi-definite. Convexification procedures are available in [73] and [69] to regularize the Hessian matrix. One example approach is to use the mirroring technique to ensure the convexity of the QP sub-problems: by using the eigenvalue decomposition result, we can approximate the exact Hessian matrix as $\tilde{H} := V\text{abs}(\Gamma)V^T$ with $H = V\Gamma V^T$, where the operator $\text{abs}(\gamma) = \gamma($ if $\gamma > \epsilon)$ or $\epsilon($ if $\gamma \leq \epsilon)$, ϵ is a small positive constant value.

A major difference between the IP method and SQP is that the IP method solves a modified KKT system (2.15) by introducing a positive parameter τ. This system contains only equality constraints and can be solved with Newton's method. We start the calculation from a fixed value of τ. Once one approximated solution is obtained, we decrease the value of τ and redo the calculation until the original KKT conditions (2.10) are fully satisfied. In this way, we actually solve a sequence of barrier problems (2.16).

$$\nabla_y \mathcal{L}(y^*, \lambda^*, \mu^*) = 0 \tag{2.15a}$$
$$g(y^*) = 0 \tag{2.15b}$$
$$h_i(y^*)\mu_i^* = \tau, i = 1, \ldots, N_h \tag{2.15c}$$

$$\min_y \quad f(y) - \tau\sum_{i=1}^{N_h} \ln(-h_i(y)) \tag{2.16a}$$
$$s.t. \quad g(y) = 0 \tag{2.16b}$$

When the KKT system (2.10) is satisfied with the given precision, both IP and SQP methods converge to the same local optimum, i.e. resulting in the same solution. The execution time of both methods depends on concrete problem structures and available solvers.

ACADO Code Generation Toolkit [74] is one state-of-the-art open-source software for employing the SQP method. It provides options on discretization algorithms (single or multiple-shooting), Hessian representation methods, condensing methods (for exploiting the sparse Hessian matrix structure), convexification procedures, integration routines (for handling system dynamics in the form of equality constraints), automatic differentiation methods, etc. The resulting SQP calculation framework is self-contained C++ code that facilitates code implementation on embedded systems. For further solving QP sub-problems, the ACADO toolkit offers an interface for various QP solvers, such as qpOASES [75] (a parametric active-set method) and HPMPC [76] (an interior-point QP solver). For employing the IP method, the open-source software IPOPT [77] is a very popular option.

Let us review the solving techniques chosen in the works that are discussed in Section 2.5.2. Authors in [53] and [47] use the ACADO Toolkit to generate an SQP framework with an interface to QP solver qpOASES for solving the time-optimal NMPC. The main purpose of choosing SQP in both works is to cooperate with the real-time iteration (RTI) scheme [78] for online feasible implementation. We should mention that the RTI scheme addresses the issue of online feasibility, but does not ensure system constraints are not violated. In contrast, one goal of this dissertation is to handle the same problem while ensuring constraint satisfaction. The NMPC problem in [34] is also solved under the SQP framework, while its extension work [51] uses the IP method under a commercial framework ForcesPro NLP [79]. In this dissertation, we choose the SQP method and use the ACADO Toolkit for two reasons. First, it is the same choice as in baseline works [53] and [47]. Second, the ACADO Toolkit offers interfaces for integrating additional tools, such as Mixed Integer Programming solvers that we will discuss in Section 2.6, and it is suited for implementation on embedded devices.

2.6 Head-to-head racing

In this section, we first introduce general control techniques for the head-to-head racing problem, more specifically for enabling safe and efficient overtaking maneuvers. Then, we present the principles of Mixed-integer programming (MIP), which will be used in Chapter 4.

2.6.1 General techniques

In head-to-head racing mode, the ego vehicle runs together with the opponent vehicle on the same racetrack. Both vehicles compete by attempting to overtake each other or maintain the leading position in order to complete the lap faster.

We focus on the head-to-head racing scenario instead of directly studying the multiple-vehicle racing problem for several reasons. First, the head-to-head scenario is clear in structure and easy to analyze, making it an ideal setting to understand the underlying principles of the racing problem. Second, by studying the head-to-head scenario in detail, we can gain insights and develop strategies that can be extended to

the multiple-vehicle racing scenario. An intriguing challenge that will be taken into account in future work is how to build task execution models in multi-vehicle cases to effectively allocate computation resources and prevent the violation of real-time constraints as the number of race cars increases. The key action in the head-to-head racing scenario, overtaking maneuver, is one main subject in this dissertation.

Overtaking behavior has been largely studied in the autonomous passenger car driving domain for lane-changing/merging problems. Classical graph/tree-based motion planning methods, such as A* and RRT, can take into account the appearance of vehicles on different lanes. However, these methods are usually designed for multiple-lane traffic systems, while in the racing scene, the vehicle can freely maneuver on the track without adhering to specific lanes. More importantly, they usually make use of imprecise kinematic models for a smooth and safe driving style while maintaining a conservative minimum distance between vehicles for safety purposes. It is not suitable for aggressive racing maneuvers in a highly dynamic racing environment that requires the vehicle to operate close to the limits of handling. In contrast, the optimization-based method, such as MPC, can catch the complex nonlinear system dynamics, which is suitable for racing scenarii. In [80], the authors build a local risk map and use a robust tube-based MPC to generate a feasible and safe trajectory for realizing one-side overtaking maneuvers in high-speed structured environments. In [67], the vehicle's objective is to follow the center line of a straight track while avoiding collisions with static obstacles.

For the autonomous racing problem, few works take into consideration the dynamic opponents. In [81], trajectory planning is considered as an optimization problem by maximizing the progress of the race car while penalizing deviation from a reference trajectory using a receding horizon approach, on top of which a dynamic-programming-based high-level corridor planner is proposed to generate convex constraints for moving obstacles. In [82], a similar idea of maximizing the progress of the race car is used. The authors take into consideration the relative position and velocity between the ego vehicle and the opponent vehicle for setting up constraints to avoid a potential collision. In [83], an off-line viability kernel computation is used to ensure continuous feasibility, which allows for improving the performance of the online controller. The authors also take into account the interaction between two race cars and formulate it as a bimatrix game, a type of game theory problem.

Recently, the learning-based method shows its potential for solving complex decision-making problems by leveraging a large volume of data generated in either simulation or the real world. For instance, authors in [84] succeed to tackle the high-speed autonomous race car overtaking problem in the video game Gran Turismo Sport by using curriculum reinforcement learning. One challenge for such kind learning-based methods is that overtaking is a sparse signal and can be difficult to collect. In this work, we focus on the NMPC-based method for overtaking problems. As a byproduct, it could generate well-structured data for use in learning-based approaches and improve data efficiency.

Mixed-integer programming (MIP) catches our interest since it can be used for encoding discrete decisions in different tasks [85]: assignment problems, control of hybrid systems, control of piece-wise-affine systems, and problems with non-convex

constraints. Several MIP-based works exist for collision avoidance in structured environments. For example, it is used in [86] and [87] to encode and solve the two-lane expressway overtaking problem. The authors represent the moving obstacles (vehicles other than the ego vehicle) as rectangles enclosed by hyperplanes. They further define the feasible region as the intersection of the complement of the obstacles. The collision-free decision is to make the ego vehicle lay in at least one feasible halfspace split by hyperplanes. The feasibility of each halfspace is set as a binary variable, resulting in a MIP problem. In [88], authors aim to realize safe lane-changing maneuvers in an MPC framework. A binary variable at each prediction step encodes 3 stages of the lane maneuver: Pre (following the initial lane), Peri (lane change), and Post (following the goal lane). The objective is to prevent the total number of Peri steps from falling below a threshold, i.e. to guarantee a sufficient time budget for lane switching.

MIP can also be used in unstructured environments. We notice that in [89], a MIP encoding method combined with the NMPC framework is used for solving the subclass problem of autonomous racing, which requires the race car to stay as close as possible to the optimal race line while crossing as many as possible pre-defined bonus zones for collecting rewards. Their NMPC formulation's objective function is different from the one used in this dissertation: instead of directly optimizing the time variable t, they set the minimization of the deviation from the optimal race line, the maximization of the collected rewards from pre-defined zones, and other terms as the objective function. In the Frenet-Serret frame, they define "gates" (free space between obstacles, or areas where exist rewards) for cars to pass through. The MIP problem arises from the combinatorial selection of "gates" at different prediction steps.

2.6.2 MIP related method

We select MIP as a base method for handling the overtaking problem in this dissertation because it can naturally model the decision-making series for overtaking maneuvers and ultimately outcome efficient and optimal results by working together with NMPC.

In this section, we first introduce how MIP can be used for modeling overtaking maneuvers. Then, we discuss several sub-problem classes of MIP and the solving technique. Finally, we talk about how we can reduce the complexity of MIP.

Basic concept

The overtaking maneuver can be readily encoded in a MIP form. As an example shown in Fig. 2.5, the ego vehicle has the option to position its center of gravity (CoG) in 4 different configurations: fully behind, fully in front of, to the left of, or to the right of the opponent vehicle's CoG. These configurations are represented by the letters A, B, C, and D, respectively. For 2-step planning, the decision series can be represented as (d_1, d_2) with $d_1, d_2 \in (A, B, C, D)$. There are a total of 16 decision combinations that can be represented as integers in a MIP form and solved

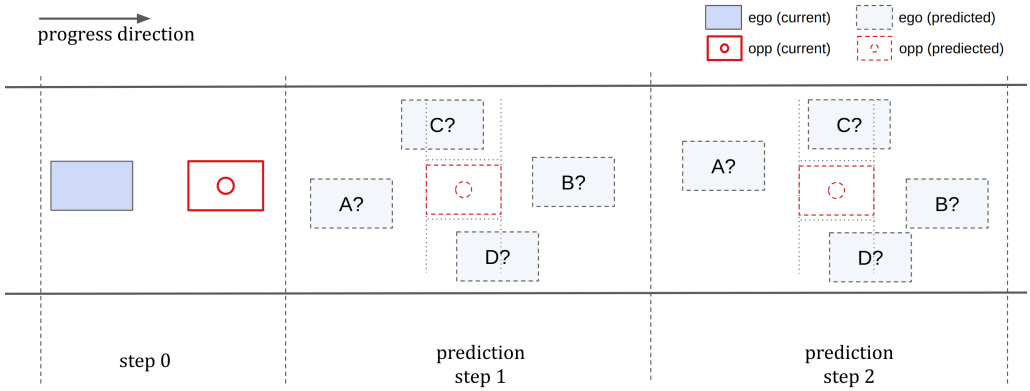

Figure 2.5: An example of overtaking maneuver encoding.

for a given objective. It is important to note that, in Fig. 2.5, the ego vehicle's step size is not restricted to a particular value in order to more clearly show the relative position between the two vehicles. In Chapter 4, the step size is limited to the length of the vehicle. This means that the decision series $d_1 = B$ or $d_1 = A \;\;\&\;\; d_2 = B$ are infeasible since they require a step size larger than the vehicle's length. The MIP solver will automatically identify these decisions as violating the constraints and eliminate them. A formal and detailed presentation on this MIP-based method is discussed in Chapter 4.

Taxonomy of MIP and the solving technique

MIP is a general optimization problem containing decision variables in the form of integers as in equation (2.17).

$$\min_{x\in\mathbb{R},z\in\mathbb{Z}} \quad f(x,z) \tag{2.17a}$$

$$s.t. \quad g(x,z) \leq 0 \tag{2.17b}$$

In the case that the objective function (2.17a) and the constraints (2.17b) are all linear (i.e. $f(x,z) = f_1^T x + f_2^T z$ and $g(x,z) = A_1 x + A_2 z \leq 0$ with matrices A_1, A_2 and vectors f_1, f_2), it refers to Mixed Integer Linear Programming (MILP). In the case that the objective function is quadratic (i.e. $f(x,z) = \frac{1}{2}x^T F_1 x + \frac{1}{2}z^T F_2 z + c_1^T x + c_2^T z$ with matrices F_1, F_2 and vectors c_1, c_2) while the constraints are linear, it refers to Mixed Integer Quadratic Programming (MIQP). Mixed Integer Nonlinear Programming (MINLP) is a special form of MIP where the objective function $f(x,z)$ and/or the constraints $g(x,z)$ are nonlinear.

In Section 4.1 of Chapter 4, we will introduce that the overtaking problem can be written in a generic form (2.18), where $f(\cdot)$ denotes the objective function, $g(\cdot)$ denote equality constraints, $h_*(\cdot)$ denote inequality constraints, along with vectors C and d for representing constants and integer variables, respectively. It is an extension of the generic formulation of time-optimal NMPC (2.9), while we separate inequality constraints (2.9c) as two parts, $h_1(y)$ and $h_2(y)$, for distinguishing the constraints with/without integer variables. Since both the equality constraints (2.18c) that

represent the system dynamics and the inequality constraints (2.18d) that represent the collision avoidance conditions are nonlinear, formulation (2.18) is actually an MINLP problem.

$$\min_{y\in\mathbb{R},d\in\mathbb{Z}} \quad f(y) \tag{2.18a}$$
$$s.t. \quad g(y) = 0 \tag{2.18b}$$
$$h_1(y) \leq 0 \tag{2.18c}$$
$$h_2(y) \leq Cd \tag{2.18d}$$

MINLP solvers are rarely built entirely from scratch [90]. Solvers are usually classified either as extending NLP solvers to handle integer variables or as extending MIP solvers to handle nonlinear objectives and constraints. In this dissertation, the SQP technique, as one type of NLP solver for formulation (2.9), is still available for solving (2.18) except that the intermediate sub-problem is no more a simple QP but an MIQP. The previous QP sub-problem (2.13) is modified in an MIQP form:

$$\begin{aligned}
\min_{\Delta y} \quad & \frac{1}{2}\Delta y^T \nabla_y^2 \mathcal{L}(\cdot)\Delta y + \nabla_y f(\bar{y})\Delta y \\
s.t. \quad & g(\bar{y}) + \nabla_y g(\bar{y})\Delta y = 0 \quad | \ \bar{\lambda} + \Delta\lambda \\
& h_1(\bar{y}) + \nabla_y h_{1,y}(\bar{y})\Delta y \leq 0 \quad | \ \bar{\mu}_1 + \Delta\mu_1 \\
& h_2(\bar{y}) + \nabla_y h_{2,y}(\bar{y})\Delta y \leq Cd \quad | \ \bar{\mu}_2 + \Delta\mu_2 \\
\text{where } & \mathcal{L}(\cdot) = f(\bar{y}) + g(\bar{y})^T\bar{\lambda} + h_1(\bar{y})^T\bar{\mu}_1 + h_2(\bar{y})^T\bar{\mu}_2
\end{aligned} \tag{2.19}$$

There exist standard algorithms to solve MIQP (2.19), which are implemented in solvers such as GUROBI [91], CPLEX [92] and COIN-Cbc [93]. The branch-and-bound algorithm [94] is a popular technique choice for these solvers. It searches in a "tree" with different integer settings to find a globally optimal solution. The heuristic methods belong to another approach family that includes simulated annealing, genetic algorithms, Tabu search, etc. Its randomness reduces the possibility of getting "stuck" in local optimality. In this work, we use the branch-and-bound algorithm in the GUROBI solver. Once the sub-problem (2.19) is solved, we repeat the SQP procedure (2.14) until the KKT value reaches the given precision.

Methods for reducing MIP complexity

Combining NMPC and MIP will result in a long execution time which is usually 2 to 4 times more than simple NMPC [95], depending on the length of the prediction horizon. The additional complexity of MIP makes it more difficult to complete the calculation in time. There are various approaches [85] for reducing the complexity of MIP.

- *Removal of constraints.* In some cases, we can exploit the prior knowledge to initially ignore some constraints. At each iteration, we check the feasibility of the solution to the original problem, and constraints are re-applied if necessary.

One example of using this technique to reduce the complexity of MIP is shown in [96], where MIP is used to prevent vehicles from collisions with circular obstacles.

- *Feasible initialization.* If we can find a feasible solution as initialization (even potentially sub-optimal), the branch-and-bound procedure can be terminated earlier and finished with faster calculation. Using the dynamic programming method as a high-level planner [34] is one effective solution for finding such feasible initialization.

- *Approximated Cost-to-go.* We can calculate an exact solution for a horizon in the near future and approximate the cost-to-go beyond this horizon, which will largely simplify the optimization problem. For instance, in the case of [97] for handling maneuvers of autonomous fixed-wing aerial vehicles, MIP is used to plan a precise trajectory in a short prediction horizon, combining with a cost function accounts for decisions beyond the planning horizon.

- *Time-step grouping.* The computation can be simplified by "sharing" the variables across adjacent time steps. Take the vehicle overtaking problem as an example, the action usually lasts for several consecutive time-steps. Using the time-step grouping will significantly reduce the problem size [98], but at the cost of conservatism compared to the original problem.

In Section 4.4 of Chapter 4, we will propose a method similar to "time-step grouping" for regrouping and simplifying decision sequences for overtaking maneuvers. Using Fig. 2.5 as an example, the combinational decision-making problem can be reduced to 3 options: left-side overtaking (C, C), right-side overtaking (D, D), or following (A, A). The resulting trajectory might be sub-optimal but the complexity is decreased.

2.7 Conclusion

In this chapter, we reviewed general control techniques for the autonomous race car racing problem and compared them with the methods developed for the autonomous passenger car driving problem. We introduced the racetrack model, vehicle dynamics, and their integration into a one-level time-optimal NMPC framework. NMPC is selected as the base controller in this dissertation since it explicitly takes into account system dynamics and physical constraints for generating optimal feasible trajectories. It complies with the requirements of safety, high model accuracy, and optimality in the autonomous racing problem. With decades of development, NMPC has become more and more reliable and efficient. However, there remain a few challenges when applying it in real-time systems, which is the research subject in Chapters 3 and 5. We also described techniques for solving NMPC, which will be used in subsequent chapters.

The head-to-head racing problem was also discussed in this chapter. We reviewed various state-of-the-art methods for handling the overtaking problem, among which

the MIP-based works were presented in detail. MIP is a natural form to encode decision series for overtaking maneuvers, producing efficient and optimal results. We will use this technique in Chapter 4 for modeling and solving the decision-making problem.

CHAPTER 3

Autonomous racing in single-vehicle mode

In this chapter, we first present the classical NMPC framework along with the real-time related challenge that it raises. To address this problem, we introduce a triggering-based approach that enables the online execution of long-horizon NMPC, which is validated in an experiment.

3.1 NMPC framework and real-time related issue

NMPC provides locally optimal results at the cost of high complexity. We discuss this issue using formulation (2.8) within the control framework. We recall that: instead of discretization in terms of time t as in other NMPC formulations, the prediction horizon of formulation (2.8) is discretized in terms of progress distance s along the racetrack's center line; at the progress step k, the initial state is noted as $\tilde{\xi}_k$, the state series in the prediction horizon is noted as $\xi_{i|k}, i = 0, \ldots, N$, while the piece-wise constant control series is noted as $u_{i|k}, i = 0, \ldots, N-1$. $\xi^*_{i|k}$ and $u^*_{i|k}$ represent the resulting optimal results.

In the classical NMPC framework, only the first-step control command in the optimal control series, i.e. the feedback law $\mu(\tilde{\xi}_k) = u^*_{0|k}$, is deployed at the beginning of the progress step k. On the left of Fig. 3.1, the orange dash line represents the resulting optimal control series $u^*_{i|k}$, while on the right of the figure, the purple solid line depicts the actually deployed 1st-step control command. The green solid line illustrates how the system could evolve toward a state that differs from the prediction. The rest of the optimal control series is dropped and an NMPC recalculation is supposed to be done during the progress step k. A new optimal control series $u^*_{i|k+1}$ is available at the beginning of the progress step $k+1$. On the right of Fig. 3.1, the orange dash line represents the dropped optimal control series and the purple dash line represents the newly generated optimal control series. The new control command, i.e. the feedback law $\mu(\tilde{\xi}_{k+1}) = u^*_{0|k+1}$, will be deployed. This

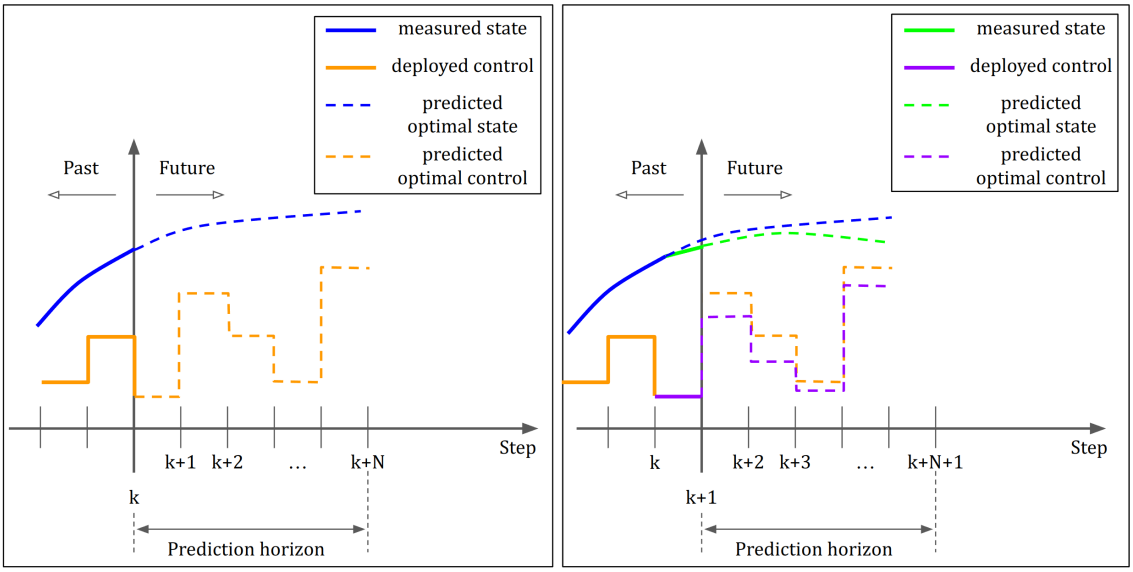

Figure 3.1: Deployment procedure for NMPC in the classical framework.

procedure repeats.

As shown in Fig. 3.2, the system evolution and the NMPC calculation occur simultaneously. On one hand (the system evolution part), the vehicle moves forward using the optimal control input $u^*_{0|k}$. The system ultimately evolves to a new state ξ_{k+1} during the progress time $\Delta t^p_k = t_{k+1} - t_k$. On the other hand (the calculation part), we prepare the optimal control for the next evolution step: first, we measure the current state ξ_k and predict the initial state at the beginning of the next step, $\tilde{\xi}_{k+1}$, using the integration of system dynamics or a simple interpolation; then, taking this initial state as input, NMPC is calculated using time Δt^c_k. The necessary condition for enabling online execution is that: $\Delta t^p_k \geq \Delta t^c_k$. Using the classical NMPC framework, this condition is not always respected, especially in long-horizon NMPC. The computed trajectory may be obsolete taking into account the progress of the vehicle during the calculation, i.e. the result might arrive after the task deadline.

3.1.1 A motivation example

We demonstrate this problem by using an experiment in the classical NMPC framework. Formulation (2.8) and model parameters from the system identification in [34] are used to generate a controller for the race car. We consider two representative prediction horizon lengths, $N = 15$ and 30, to investigate the online feasibility under different computational complexity. The SQP method is used for solving formulation (2.8) with a pre-defined maximum iteration number 20, and a KKT precision 10^{-4} requiring that the violation of KKT condition (2.10) does not exceed this value.

Two well-studied racetracks, [47] and [81], are used for the vehicle to start with the pose $(s, e_y, e_\psi) = (0, 0, 0)$ and completes a lap with the maximum velocity $1.6[m/s]$. The first racetrack has straight sections and corners which have the same

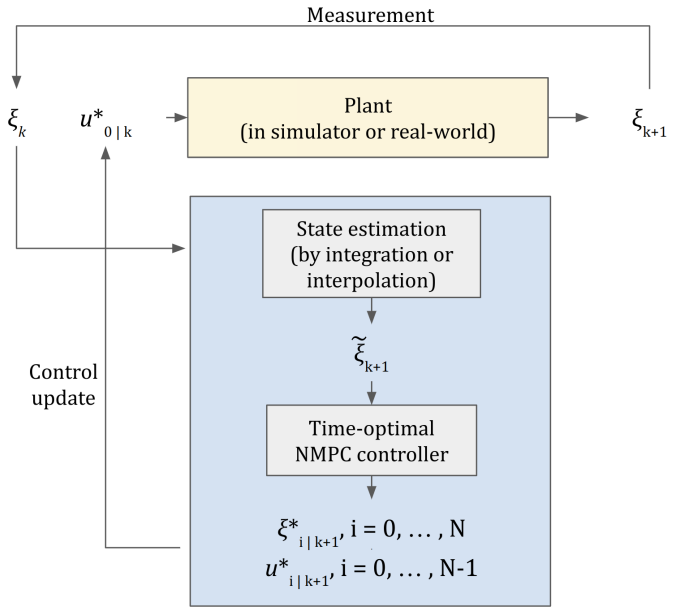

Figure 3.2: NMPC framework.

Table 3.1: Statistics for classical NMPC framework: progress time, calculation time, and SQP iteration numbers.

		Track 1		Track 2	
horizon N		15	30	15	30
Per-step progress time [ms]	max	57.8	54.5	73.7	66.5
	min	17.2	17.3	12.8	12.2
	mean	33.2	32.7	33.9	33.4
Per-step calculation time [ms]	max	19.7	81.6	19.2	101.3
	min	2.4	8.1	1.2	3.8
	mean	6.8	27.0	6.4	25.2
SQP iteration number	max	15	11	14	18
	min	2	2	1	1
	mean	4.4	4.3	3.9	4.0
Lap time [s]		4.852	4.773	10.189	10.064

curvature, whereas the second is more complex with a combination of segments with varying and large curvatures. Layouts of both tracks are shown in Fig. 3.7 and they take respectively 145 and 300 progress steps to run through.

The experiment runs on a standard laptop with an Intel i7 processor with the code optimization level set to be '-O3'. In order to observe the online non-feasible cases, the calculation and the simulation are done alternately (instead of simultaneous execution): we get the vehicle state at step k, then we calculate the optimal control series of NMPC; once the calculation finishes, we simulate the evolution of system dynamics during step $k+1$ by an integration method using the control command from NMPC; the NMPC calculation and the simulation repeat.

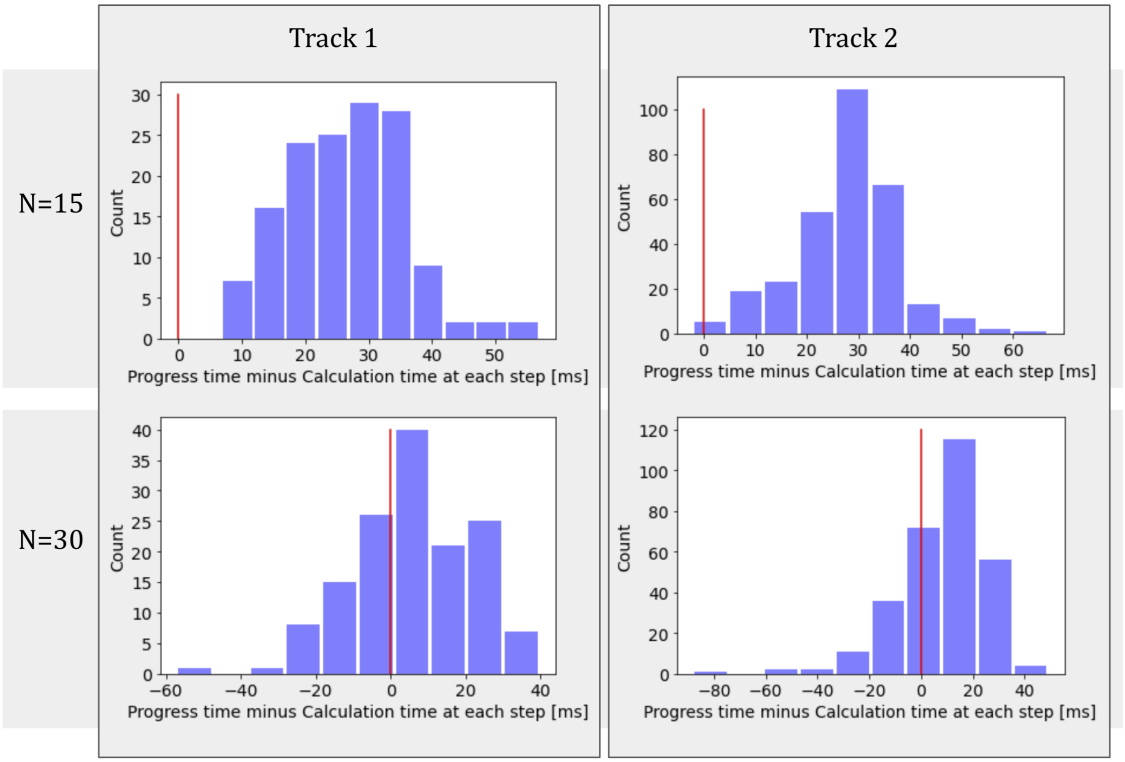

Figure 3.3: Histograms for the value of $\Delta t = \Delta t_k^p - \Delta t_k^c$.

In Table 3.1, the first group of rows shows the statistic result for per-step progress time. The vehicle's actual traveled distance and velocity differ at each step, resulting in a varied progress time. The second/third group of rows demonstrate the calculation time/SQP iteration numbers for solving each NMPC problem. They fluctuate at each step due to the change in NMPC input. Since the maximum iteration number does not exceed the allowed maximum number 20, constraints in (2.8) are fully satisfied within a precision 10^{-4}. We can observe that the average calculation time for $N = 30$ is about 4 times more than that for $N = 15$. To examine the constraint $\Delta t_k^p \geq \Delta t_k^c$ for each step, we display the value of $\Delta t = \Delta t_k^p - \Delta t_k^c$ in Fig. 3.3. The value of Δt should always be positive to allow online execution. We see that on track 1, the constraint is totally satisfied with $N = 15$ while this is not the case for certain steps with $N = 30$. On track 2, there are 2 steps out of 300 where the calculation time exceeds the progress time by $2.2[ms]$ and $0.7[ms]$ respectively. With $N = 30$, the constraint is violated at more steps.

From this motivation example, we can find that: with a long prediction horizon (e.g. $N = 30$ in this experiment), the NMPC calculation time Δt^c might be higher than Δt^p; with large curvature segments (e.g. corners on track 2), and when the race car travels along the inside of the track, the actual traveled distance will be short and results in a short progress time Δt^p. Both situations can prevent the constraint $\Delta t_k^p \geq \Delta t_k^c$ from being satisfied in the classical NMPC framework.

Therefore, we need to find a method to guarantee that the online feasibility constraint $\Delta t_k^p \geq \Delta t_k^c$ is always satisfied, especially in the case of long-horizon NMPC

which is appealing because of its superior performance in terms of lap time (as shown in the last row in Table 3.1).

3.2 Triggering-based recalculation method

In this section, we propose a triggering-based recalculation method for NMPC in single-vehicle racing mode, which ensures the online feasibility of NMPC with limited computation resources, especially for one with a long horizon, and allocates the computation load efficiently. To be precise, we use two triggering conditions in this method: one for ensuring the constraint $\Delta t_k^p \geq \Delta t_k^c$ will be respected, and another one for maintaining the performance in terms of lap time while using as less as possible the computation resources. The latter is related to the type change of racetrack segments, which is indicated by the curvature change on the racetrack's center line. To the best of our knowledge, no triggering methods using either of the above two conditions for NMPC recalculating have been previously proposed in the literature.

3.2.1 Basic concept

The triggering method is well-studied in path-tracking problems. For example, authors in [99] propose two triggering modes: self-triggering and event-triggering. The difference is that the event-triggered mode relies on the environment and requires continuous measurement of system states, while in the self-triggered mode, the next triggering point is pre-computed based on the current state and a predicted control sequence. In [100], authors determine the threshold in an event-based triggering method by applying the statistical method to the historical data. Through an experiment, they demonstrate that their approach reduces the computational and communication burdens without compromising performance. Both works attempt to alleviate the computation burden in the task of path tracking (tracking/stabilizing MPC problem, as defined in Section 2.5.1 of Chapter 2), which is distinct from ours. The triggering-based method used in this work aims to ensure that the time budget is respected while maintaining sufficient system performance and leveraging computation resources efficiently in a time-optimal control problem (economic MPC problem, as defined in Section 2.5.1 of Chapter 2). Moreover, the triggering conditions used in these 2 works differ from ours.

In our method, the first triggering condition for an NMPC recalculation is straightforward: waiting for a sufficient amount of calculation time to be available. It allows us to run long-horizon NMPC without exceeding the allotted time budget. The second triggering condition is inspired by human drivers' behavior in single-vehicle racing mode. They have been observed to adjust the intended trajectory only when they are aware that the vehicle is about to enter a track segment that is different from the one they had originally planned for (e.g. from a straight line to a corner, or in reverse). It inspires us that the planned trajectory of NMPC needs to be significantly changed only when the vehicle is about to run into different

racetrack types. In other words, it is only necessary to recalculate NMPC under this specific circumstance. Based on this fact, we propose the second triggering condition to reduce total calling times to the NMPC solver while maintaining a sufficient level of system performance.

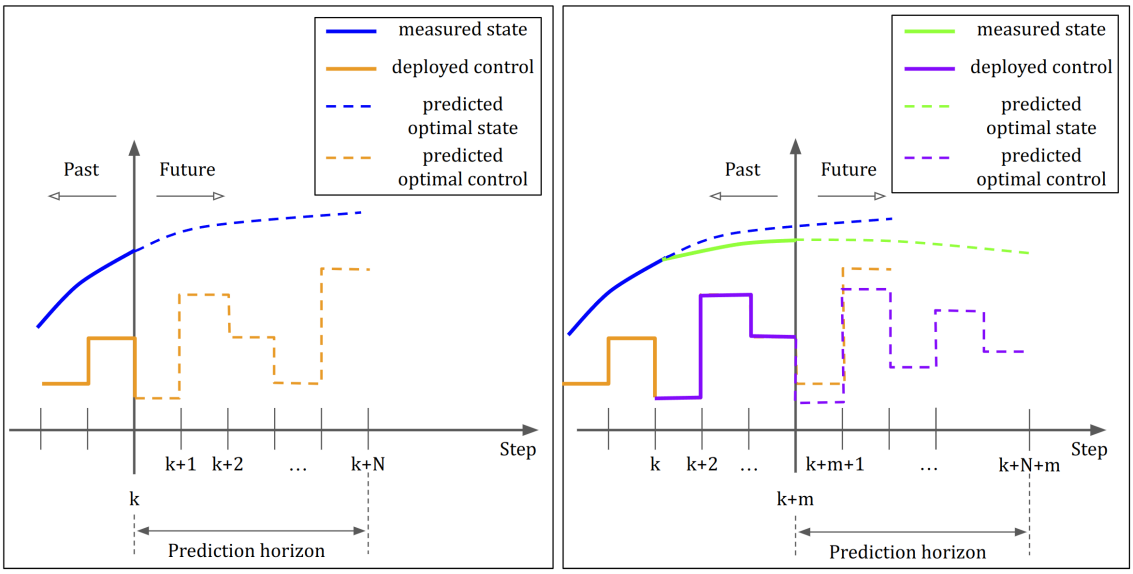

Figure 3.4: Deployment procedure for NMPC using the triggering-based method.

Our triggering method actually belongs to the self-triggering mode. As shown in Fig. 3.4, m-steps in the optimal control series, instead of one single step in a classical framework, will be deployed. The main problem is to define a triggering condition for determining the value of m. In the experiments in Section 3.3, we will demonstrate that our triggering method significantly reduces the number of recalculations while still maintaining comparable lap times to the classical NMPC framework, in which recalculations occur at every progress step. It is worth mentioning that we solve NMPC under this triggering method by the SQP procedure (2.14) until its convergence. In this way, the convergence criteria of NMPC can naturally guarantee that the system constraints are not violated within given precision.

3.2.2 Triggering conditions

As the first triggering condition, we require that the progress time between the current step k and recalculation step $k+m$ to be long enough for finishing the recalculation in time budget C: $\Delta t_k^p = \Delta t_{k\to k+m}^p = t_{k+m} - t_k \geq C \geq \Delta t_k^c, 1 \leq m \leq N-1$. We should find a minimum step number m_1 for satisfying this constraint.

To save computation resources, we reuse $m_2, m_2 \geq m_1$ steps in the NMPC prediction horizon until other factors trigger the recalculation. In the following paragraphs, we present the second triggering condition based on the curvature of the racetrack's center line $\kappa(s)$, which depends on the progress s along the center line and indicates the type of segments of the racetrack. As an example, we demonstrate in Fig 3.5 that racetracks used in Section 3.1 are characterized by the curvature distribution.

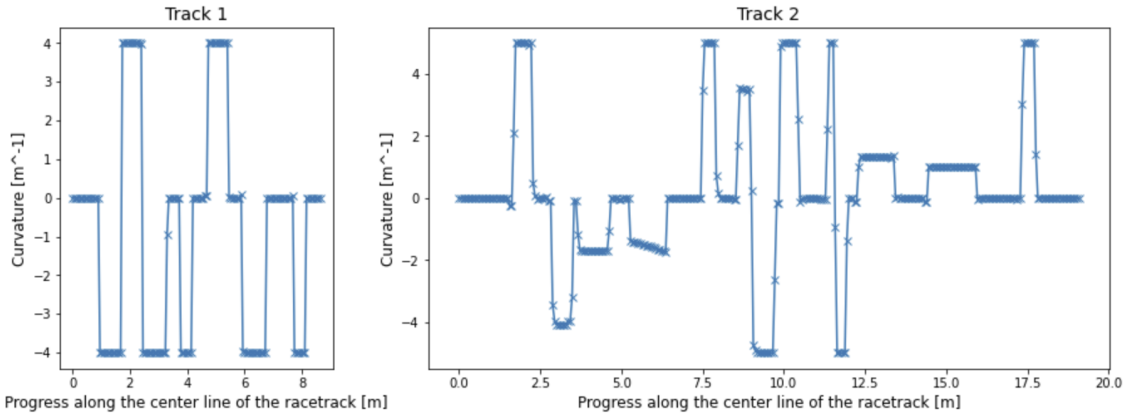

Figure 3.5: Curvature distribution for the center line of racetracks used in Section 3.1. The layout of both tracks can be found in Fig. 3.7.

Intuitively, we can scan the curvature change value and when it is greater than a threshold ϵ_κ, a recalculation will be triggered. At step k, we check for a candidate recalculation step $k+i, i = 1, \ldots, N-1$. The end of this potential prediction horizon is at step $k+i+N$. If there is a significant difference between the racetrack curvature at step $k+i+N$ and step $k+N$ (i.e. the end of the current prediction horizon), the recalculation of NMPC could provide us with a different optimal trajectory; otherwise, the resulting trajectory should not be significantly different from the previous optimization result. The condition is formally expressed as finding the first $i = m, i \in [1, \ldots, N-1]$ such that:

$$\begin{aligned} \Delta t^p_{k\to k+i} &= t_{k+i} - t_k \geq C \\ \text{and } & |\kappa(s_{k+i+N}) - \kappa(s_{k+N})| \geq \epsilon_\kappa \end{aligned} \tag{3.1}$$

The first term is a necessary condition for meeting the requirement of the time budget. The second term indicates the potential change of the optimal trajectory. It is meant to trigger the recalculation as late as possible to save computation resources while maintaining the system's performance in terms of lap time.

It is difficult to calculate the first term in (3.1) since the progress time at step $k+i$, t_{k+i}, is an unknown future state variable. However, we can make a conservative estimation of the vehicle's traveling path: assuming that it travels along the inner boundary of the track (the side of the track that is closer to the center of the curve, as shown in Fig. 3.6), the vehicle will take the shortest route which results in the shortest progress time. We use a simple geometric relation to under-approximate the progress time between steps $k+j$ and $k+j+1$, $0 \leq j < i$ as:

$$\Delta t_{j,\min} = d^{\text{progress}}_{\min} / v^{\text{ego}}_{\max} = \Delta s \cdot (1 - \left| e_{y_{\max}} \cdot \kappa(s_j) \right|) / v^{\text{ego}}_{\max} \leq t_{k+j+1} - t_{k+j}.$$

Since we have $\Delta t^p_{k\to k+i} = t_{k+i} - t_k = \sum_{j=0}^{i-1}(t_{k+j+1} - t_{k+j}) \geq \sum_{j=0}^{i-1} \Delta t_{j,\min}$, the triggering condition is reformulated as finding the first $i = m, i \in [1, \ldots, N-1]$ such that:

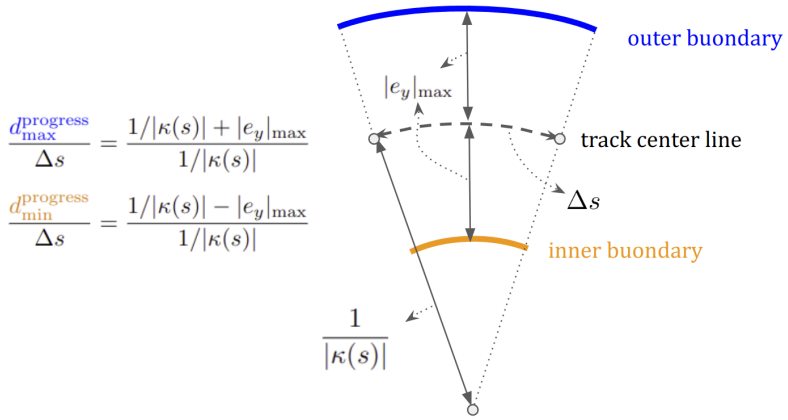

Figure 3.6: Estimation for vehicle's traveling path.

$$\begin{aligned} &\textstyle\sum_{j=0}^{i-1} \Delta t_{\text{j, min}} \geq C \\ &\text{and } \ |\kappa(s_{k+i+N}) - \kappa(s_{k+N})| \geq \epsilon_\kappa \end{aligned} \tag{3.2}$$

It is noteworthy that the second term in triggering conditions (3.2) can be replaced by other factors. For example, the difference between the predicted state and the actually measured state can also be used as a triggering condition, which can be studied in future work. In this study, we suppose that there is no external disturbance or model mismatch that could cause this kind of state difference.

3.2.3 Initialization related issues

Skipping recalculation points makes it harder to design an initialization strategy. In the classical NMPC framework, we have the optimal solution at progress step $k-1$, i.e. $\xi^*_{i|k-1}, u^*_{i|k-1}$. Only the first step in the optimal control series, $u^*_{0|k-1}$, will be applied. When we recalculate NMPC at progress step k, the state and control at the first $N-1$ prediction steps can be initialized by the previous optimal state/control series (it is called "state/control shifting", which is a key component of the RTI method [101]):

$$\begin{aligned} \xi_{i|k} &= \xi^*_{i+1|k-1}, i = 0, \ldots, N-1 \\ u_{i|k} &= u^*_{i+1|k-1}, i = 0, \ldots, N-2 \end{aligned} \tag{3.3}$$

The state/control values at the last prediction step can be a direct copy from the second last prediction step: $\xi_{N|k} = \xi_{N-1|k}$ and $u_{N-1|k} = u_{N-2|k}$. Another option is to only copy the control at the last prediction step $u_{N-1|k} = u_{N-2|k}$ and estimate the last step state by extrapolation: $\xi_{N|k} = f^{\text{integration}}_{\text{Runge Kutta}}(\xi_{N-1|k}, u_{N-1|k})$.

Using the triggering-based recalculation, the state/control shifting is only feasible for the first $N-m$ steps. For the remaining m steps, we currently use an all-zero strategy with several exceptions, e.g. the velocity is set to be a constant non-zero

value inside its value range. Although no initialization-related errors appeared in the following experiments, this simple initialization method might need more SQP iterations than other approaches to converge.

3.3 Experiment: baseline method v.s. triggering-based method

In this section, we show the experiment results for single-vehicle racing, including the performance at baseline and the performance of the suggested triggering-based method.

3.3.1 Implementation and experimental setup

We first implement problem formulation (2.8) in the syntax of ACADO Code Generation Toolkit [102]. In this implementation, we use the vehicle model identification parameters described in [81], which corresponds to a 1:43 miniature race car, and set various physical constraints as listed in Table 3.2. We select code generation options in ACADO Toolkit: Hessian approximation method [103] is set to *EXACT_HESSIAN*; discretization type is set to *MULTIPLE_SHOOTING*; QP solver is set to *QP_QPOASES* [75]. We set the KKT value to 10^{-4} for the SQP procedure, which means that the optimization process ends when the KKT condition reaches this precision level, indicating that it is sufficiently optimal and all constraints are satisfied as well. In terms of vehicle position, it means that the maximum precision error is no more than $0.01[cm]$, which is relatively small and acceptable given that the vehicle size is about $6[cm]$. The maximum QP iteration number in an SQP procedure is set to 20. After the configuration, an SQP framework is generated by ACADO Toolkit.

Table 3.2: Physical constraints on state variables.

Variable	Range
e_y	$[-0.17, +0.17]\ m$
e_ψ	$[-1.5, +1.5]\ rad$
v_x	$[0.05, +1.6]\ m/s$
v_y	$[-1.0, +1.0]\ m/s$
r	$[-8.0, +8.0]\ rad$
d	$[-1.0, +1.0]$
δ	$[-0.6, +0.6]\ rad$
Δd	$[-10.0, +10.0]\ s^{-1}$
$\Delta\delta$	$[-10.0, +10.0]\ rad/s$

The racetrack is characterized by sample points on the center line using Cartesian coordinates. We create a cubic spline using these sample points and this spline allows

us to readily obtain information on the curvature of the racetrack at any location. Two classic racetracks are employed in the experiment: track 1, used in [47] with the full length of 8.7$[m]$ and track 2, used in [81] with the full length of 18.0$[m]$. Both tracks have a constant width of 0.34$[m]$. The vehicle is initially located at the $(s, e_y, e_\psi) = (0, 0, 0)$ and runs along the racetrack for finishing a full lap.

We write a script to simulate the vehicle's progress along the racetrack. We recall that, as same as the motivation experiment in Section 3.1.1, the experiment is based on alternate execution, a method that allows testing the controller even though some steps were online infeasible due to constraint violations of $\Delta t_k^p \geq \Delta t_k^c$. Both the baseline method and the triggering-based method operate in this alternate manner: we get the vehicle state and calculate the optimal control series of NMPC; once the calculation finishes, we simulate the evolution of system states for advancing one step; the NMPC calculation and the simulation repeat.

In this racing scenario, the progress step length is selected to be 0.06$[m]$ which is close to the length of vehicles. The choice of horizon length N has an important impact on the lap time performance and calculation time, which is discussed in Section 3.3.2. In this simulation script, the functionality of visualization is also provided. The raw data of the experiment is logged and will be used for data analysis after the experiment.

The code implementation discussed above is stored in a code repository [1]. The following experiments are performed on a standard laptop featuring an Intel i7 CPU and 32 GB of RAM under Ubuntu 18.04.

3.3.2 Racing with the baseline method

Table 3.3: Simulation results for single-vehicle racing using the baseline method.

	Horizon length N	Lap time [s]	Mean calculation time per step [s]
Track 1	15	4.852	0.137
	20	4.802	0.171
	30	4.773	0.245
	40	4.768	0.355
	50	4.767	0.483
Track 2	15	10.189	0.118
	20	10.107	0.140
	30	10.064	0.205
	40	10.059	0.307
	50	10.059	0.460

We test the single-vehicle racing scenario using the classical NMPC framework as a baseline to compare the influence of different lengths N of prediction horizon on

[1] https://github.com/nanli42/NMPC_baseline_vs_triggering_based_method

the lap time. The result is shown in Table 3.3. For track 1, the lap times achieved for $N = 40$ and 50 indicate that the minimum lap time is around 4.767 s. This shows that $N = 30$ achieves a good enough lap time (0.13% slower than $N = 50$) while maintaining a relatively low calculation time. $N = 15$ is also an appealing choice because of its low computational cost. Experiments on track 2 show very similar results. We use the prediction horizon $N = 30$ to conduct the following experiment for testing the online feasibility of long-horizon NMPC with the triggering-based method. We will compare its performance with this baseline method.

3.3.3 Racing with the triggering-based method

Parameter configuration

To ensure the robustness and efficacy of the triggering condition (3.2), careful consideration should be given to the selection of the parameter ϵ_κ. A small value of ϵ_κ makes the triggering condition more sensitive, thus, more calculations will be performed. A large value of ϵ_κ makes it less sensitive and we might miss the recalculation points that lead to the global optimal trajectory. We select $\epsilon_\kappa = 10\% \cdot (\kappa^{\max} - \kappa^{\min})$ as a compromise according to the general curvature condition on both tracks. Another important parameter to be determined is the time budget C. According to statistics in Table 3.1, we find that the average calculation time of one QP is about $6.3[ms]$. The total QP number is usually less than 20. We set the maximum allowed QP number as 20 and give a 20% margin to estimate the WCET as time budget: $C = (1 + 20\%) \cdot \Delta t^{\text{mean observed time}} = (1 + 20\%) \cdot (20 \cdot 6.3[ms]) \sim 150[ms]$.

Resulting trajectory and check on triggering conditions

The resulting trajectory is shown in Fig. 3.7. We find that: on track 1 (resp. track 2), NMPC is triggered 12 (resp. 26) times for recalculation, while it is recalculated 145 (resp. 300) times in the baseline NMPC.

Starting from the triggering-based NMPC recalculation points (shown as big green dots in Fig. 3.7), we find that the track type (indicated by the curvature value of the racetrack's center line) at the end of the prediction horizon (i.e. $N = 30$ steps away or $N \cdot \Delta s = 30 \cdot 0.06 = 1.8[m]$ away) is different from the one that is at the end of the previous prediction horizon. It shows the effectiveness of the second term in the triggering condition (3.2) for reacting to a potential trajectory modification.

We then verify whether the first term in (3.2) is satisfied for all recalculation steps, i.e. whether the progress time is always long enough to allow the NMPC recalculation to be completed. Fig. 3.9 gives an intuitive comparison of progress time and calculation time at each step. Table 3.4 shows detailed statistical information. The minimum progress time on track 1 (resp. track 2) is $196.5[ms]$ (resp. $161.8[ms]$), which is longer than $C = 150[ms]$. The maximum calculation time is $118.8[ms]$ (resp. $144.6[ms]$), which does not exceed C. As a result, the constraint on the time budget is respected.

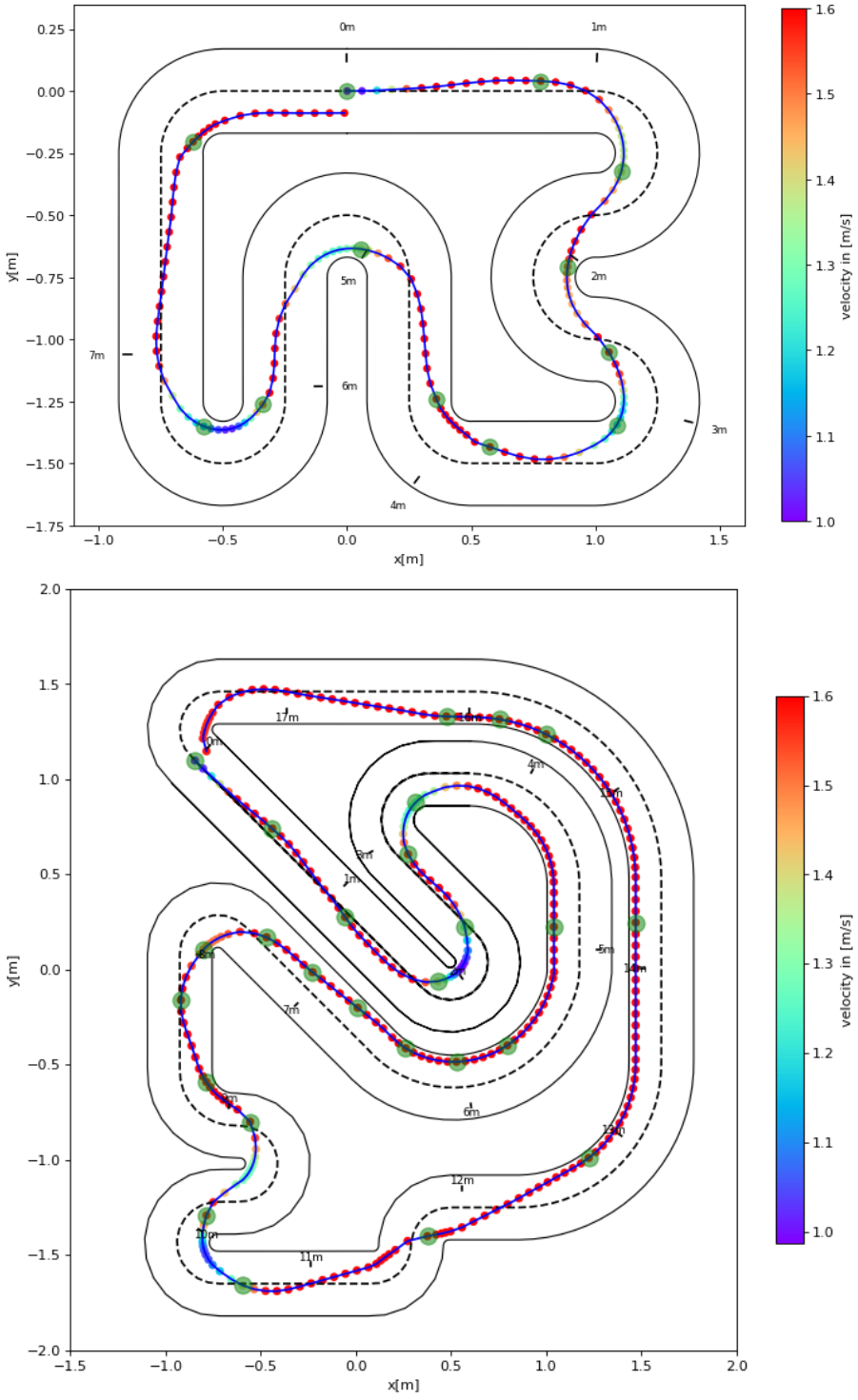

Figure 3.7: Trajectories using the triggering-based method ($N = 30$) (Top/Down: track 1/2). Green dots represent progress points triggering-based NMPC recalculation events; red points represent the baseline NMPC recalculation events. Colorbars represent speed.

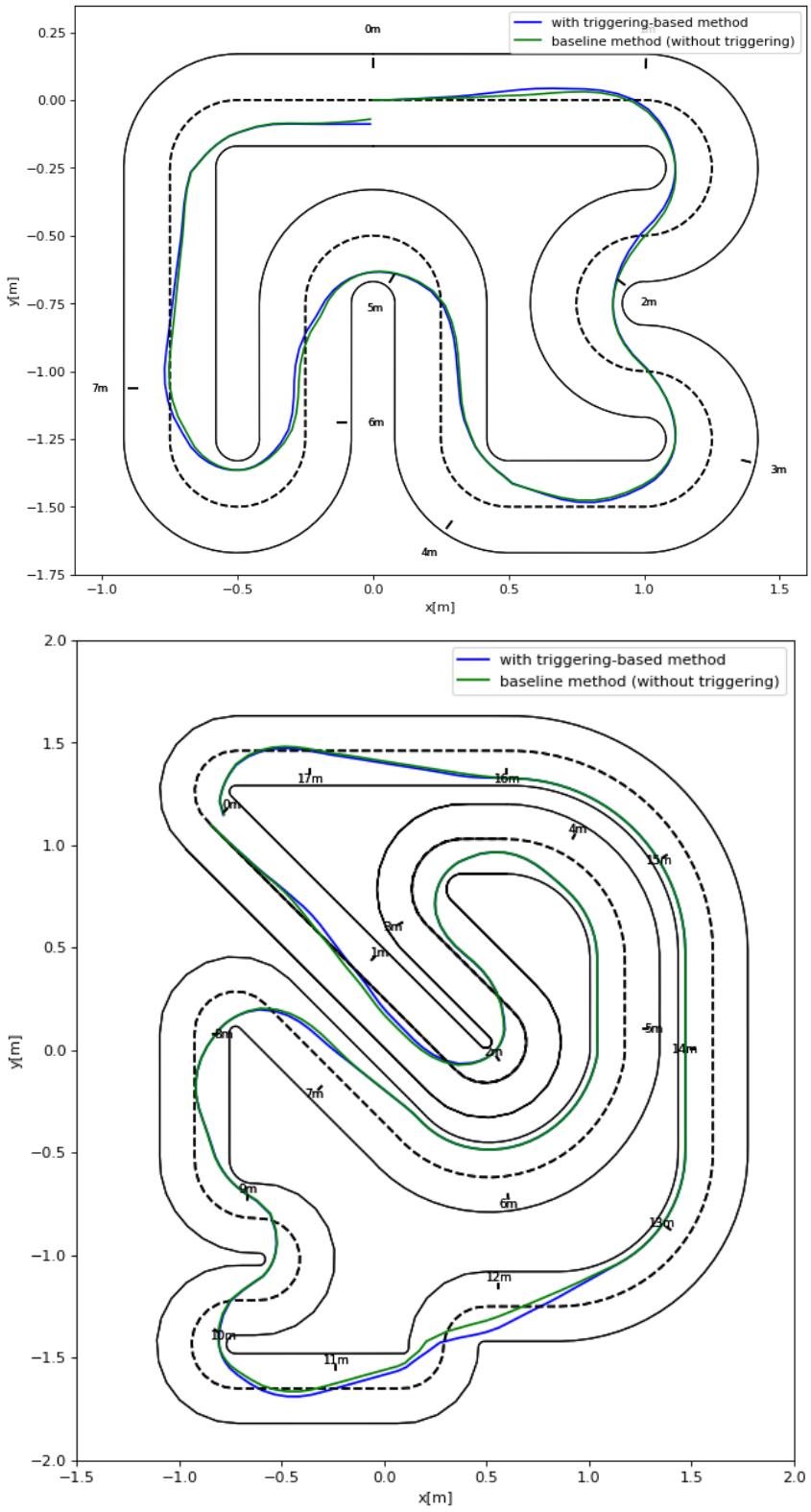

Figure 3.8: Trajectories with/without triggering-based method ($N = 30$) (Top/Down: track 1/2).

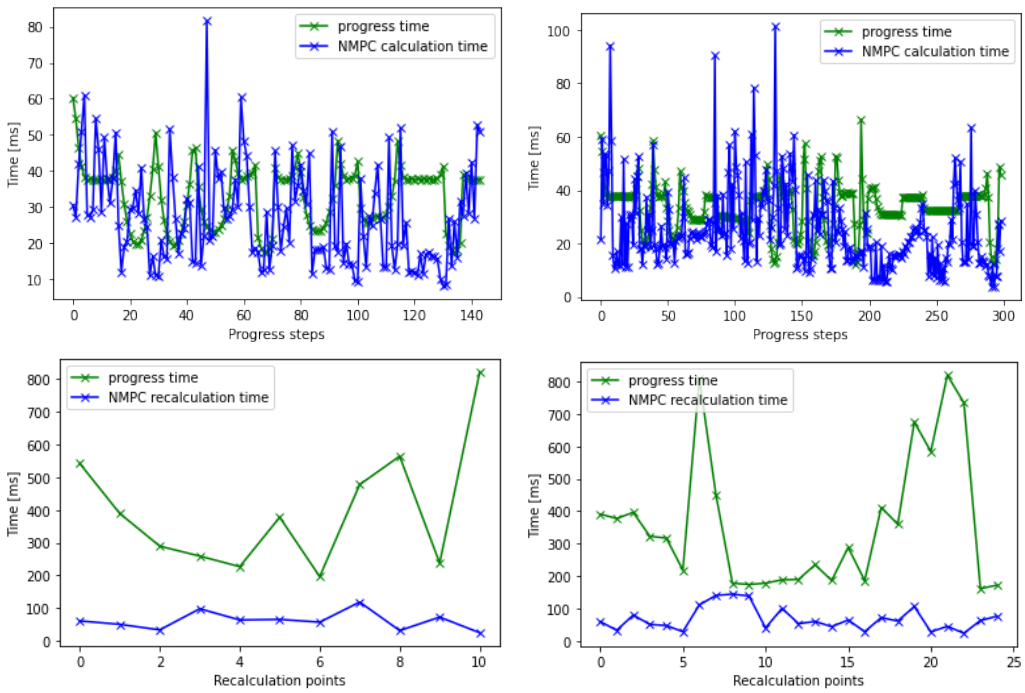

Figure 3.9: Progress time and NMPC recalculation time (Left/Right: track 1/2; Top/Down: without/with triggering method).

Table 3.4: Statistics for NMPC with triggering method ($N = 30$): progress time, calculation time, and SQP iteration numbers.

		Track 1	Track 2
Per-step progress time [ms]	max	820.0	819.2
	min	196.5	161.8
	mean	398.2	360.4
Per-step calculation time [ms]	max	118.8	144.6
	min	25.4	24.3
	mean	62.0	68.5
SQP iteration number	max	20	20
	min	6	4
	mean	10.2	10.4

Table 3.5: Lap time [s] for baseline and triggering-based methods on 2 tracks.

		Track 1	Track 2
baseline method	N = 15	4.852	10.189
	N = 30	4.773	10.064
triggering-based method	N = 30	4.775	10.075

Table 3.6: Statistics of activation times of triggering conditions.

	Track 1	Track 2
1st term in conditions (3.2) is firstly activated	8	14
2nd term in conditions (3.2) is firstly activated	4	10
two terms are activated at the same time	0	2

Lap time and analysis on triggering conditions

We also check the performance in terms of lap time. By comparing the lap times between different methods in Table 3.5, we find that the proposed method has a slightly worse lap time than the baseline method with $N = 30$ but is still better than the one with $N = 15$. It is also confirmed in Fig. 3.8 that the resulting trajectories using the triggering-based method are quite close to those using the baseline method.

The recalculation is only triggered when both terms of the triggering conditions (3.2) are activated. The recalculation must be delayed as long as the first term is not activated (i.e. the time budget is insufficient). Once the first term is active, the recalculation can occur until the end of the horizon at the cost of a potential loss of optimality. The second term aims at forcing the recalculation before the end of the horizon by estimating that a change of curvature causes a loss of optimality. As shown in Table 3.6, the recalculation is delayed by the time budget constraints 4 (resp. 10) times on track 1 (resp. track 2); in the rest of 8 (resp. 16) times recalculation on track 1 (resp. track 2), the recalculation is triggered only when there is the necessity to perform a re-planning, which is indicated by the change of the curvature. We can conclude that the slight loss of optimality is caused by the delayed recalculations for fulfilling the requirement of the first term in the triggering conditions (3.2). It is still close to the optimal result because the second term in the triggering conditions (3.2) relaunches NMPC efficiently, i.e. the recalculation is launched in a sporadic manner without deviating too much from the global optimal trajectory.

Using the following supplementary experiment, we analyze in detail the functionality of both terms of the triggering conditions (3.2). Suppose that in case 1, once the first term of the triggering conditions (i.e. $\sum_{j=0}^{i-1} \Delta t_{\text{j, min}} \geq C$) is activated, we launch the recalculation without waiting for the activation of the second term. In case 2, after the activation of the second term (i.e. $|\kappa(s_{k+i+N}) - \kappa(s_{k+N})| \geq \epsilon_\kappa$),

we do not immediately launch the recalculation but wait until the end of the horizon (the first term is automatically activated since the progress time is long enough to cover the time budget). In case 3, we use the triggering conditions as normal, i.e. launch the recalculation when both terms are activated.

Table 3.7: Analysis of two terms in the triggering conditions.

	Track 1			Track 2		
	lap time [s]	cumulative calculation time [ms]	recalculation number	lap time [s]	cumulative calculation time [ms]	recalculation number
case 1	4.777	1153.2	22	10.071	2484.6	51
case 2	5.331	340.8	4	10.533	692.6	10
case 3	4.775	795.4	12	10.075	1743.0	26

From the testing result in Table 3.7, we can find that recalculations in case 1 are launched more than in the other 2 cases and it consumes more computation resources. In case 2, the number of recalculations is reduced to a minimum but at the cost of poor performance in terms of lap time. As shown in Fig. 3.10, its resulting trajectory is much worse than the other 2 cases. In case 3, the number of recalculations is between cases 1 and 2. On track 2, it has a slightly worse lap time than case 1. On track 1, it has even a marginally faster lap time than case 1, which may be due to the fact that NMPC only provides a locally optimal solution, and thus, more recalculations do not necessarily result in a globally optimal solution. As shown in Fig. 3.10, the resulting trajectories of case 1 and 3 are close to each other, which demonstrate the effectiveness of the second term of the triggering conditions (3.2).

Initialization related issue

From the statistics of NMPC recalculation in Table 3.4, we notice that several NMPC recalculations reach the maximum iteration number, i.e. 20. To be precise, on track 1 (resp. track 2), 1 out of 12 (resp. 3 out of 26) recalculation points reaches the maximum iteration number, finishing with the KKT value 1.2×10^{-4} (resp. 2.3×10^{-4}). As discussed in Section 3.3.3, the maximum precision error is limited to $0.012[cm]$ (resp. $0.023[cm]$), which is acceptable because it is small given the size of the vehicle.

By comparing Table 3.4 with Table 3.1, we find that the average number of SQP iterations using the proposed triggering method is about twice that in the classical NMPC framework, resulting in a per-step calculation time that is also about twice as long. This can be explained by the fact that the classical NMPC framework benefits from a better initialization method, i.e. state/control shifting, as discussed in Section 3.2.3. With the current simple initialization, the proposed triggering-based method needs more SQP iterations to make the NMPC algorithm converge. To further reduce the iteration number and thus reduce the total calculation time, a better initialization method should be studied in future work. One interesting study

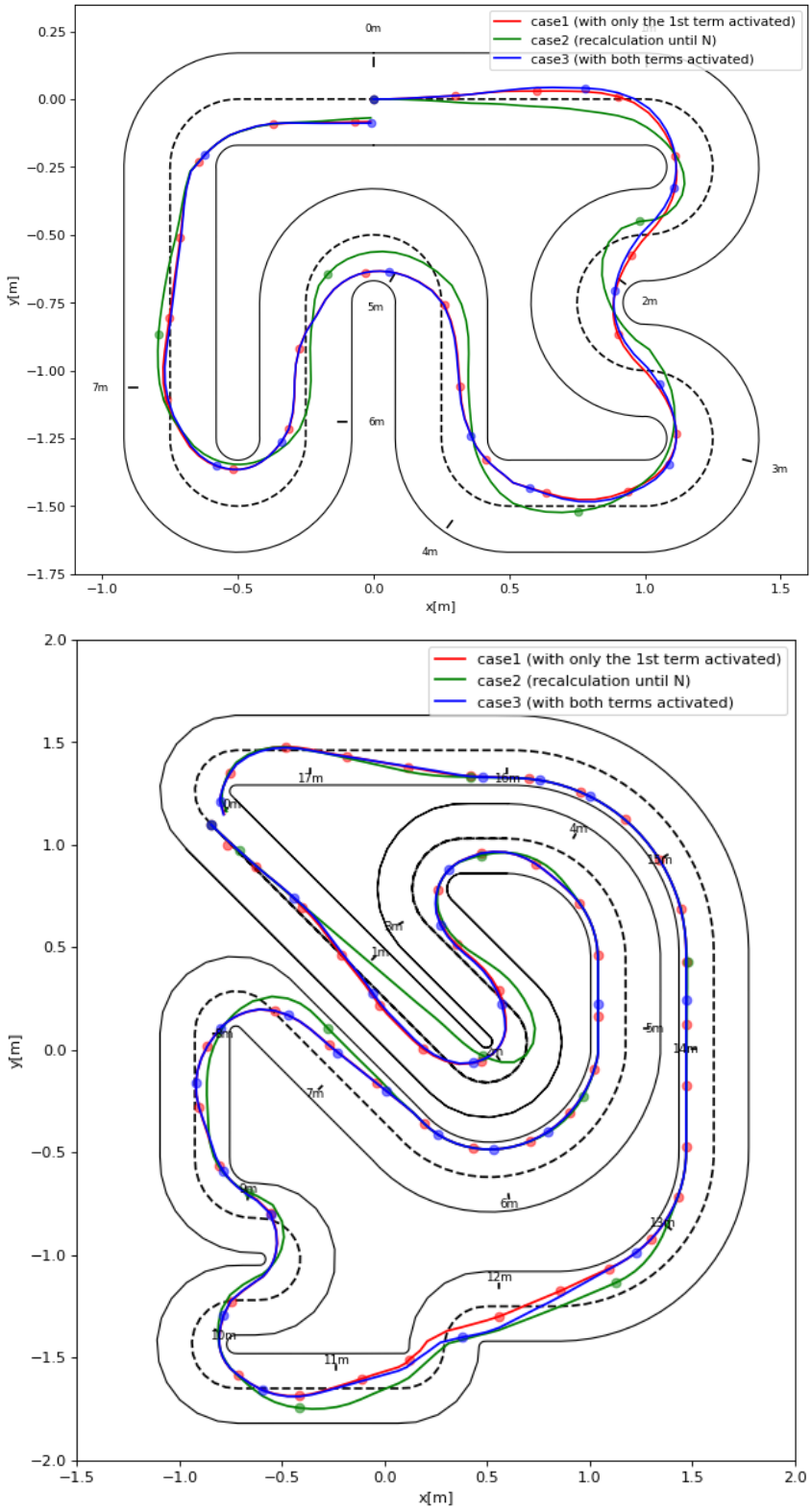

Figure 3.10: Trajectories using different configurations of the triggering conditions (Top/Down: track 1/2). Dots represent recalculation events.

direction is to use imperfect but low-cost algorithms, such as A* [13] and RRT [14], to make the initial states close to optimal.

3.4 Conclusion

This chapter presented NMPC for single-vehicle racing mode. We identified an online feasibility issue with the classical NMPC framework: one-step recalculation might become infeasible due to the violation of the necessary condition for online execution: $\Delta t_k^p \geq \Delta t_k^c$. To enable the real-time execution of long-horizon NMPC, we suggested a triggering-based method to perform m-step recalculation.

Experiments were performed to demonstrate the effectiveness of the proposed method: long-horizon NMPC has a sufficient time budget to complete the computation, which is ensured by the first term in the triggering conditions; the computation resource is conserved by skipping $m-1$ recalculation steps; the incorporation of the second term in the triggering conditions, which detects potential changes in optimal solutions, ensures that the results remain close to those of step-by-step recalculation. In the next chapter, we will extend NMPC to head-to-head racing mode, which involves a decision-making problem for collision avoidance between vehicles.

CHAPTER 4

Autonomous racing in head-to-head mode

In this chapter, we extend the autonomous racing problem from single-vehicle racing mode to a more general mode: head-to-head racing. Studying this racing mode serves as the foundation for research on the multi-vehicle (more than 2 race cars) racing problem. Additionally, it provides inspiration for the development of autonomous passenger cars for scenarii such as lane switching/merging.

We concentrate on the scene in which the ego vehicle is behind the opponent vehicle. In the case that the ego vehicle is in the leading position, it behaves the same as in the single-vehicle mode. The blocking behavior is not considered in this research. We suppose that when the ego vehicle falls behind, it is the responsibility of the ego vehicle to avoid the potential collision. Moreover, we assume that the opponent vehicle's trajectory is calculated by the same controller as the ego vehicle, and the trajectory is provided to the ego vehicle. In future work, algorithms for estimating the trajectory of the opponent vehicle can take the place of this assumption. We impose a lower maximum allowable speed for the opponent vehicle than the ego vehicle in order to observe the ego vehicle's overtaking behavior in the experiment.

4.1 Introduction: decision-making for safe and efficient overtaking maneuvers

In this dissertation, we focus on one of the most critical problems in head-to-head racing mode: how to realize safe and efficient overtaking maneuvers. The term "safe" means that no collisions should occur throughout the overtaking maneuver. The term "efficient" means that the control strategy should not be overly conservative.

In order to take use of the local optimality offered by NMPC, we consider addressing this problem within the NMPC framework. The objective is to model the overtaking maneuvers as a decision-making problem, that is, to find a series of decisions for each step in the prediction horizon, enabling the overtaking behavior while

preventing potential collisions. We formally define the problem in (4.1) which is an extension of formulation (2.8).

$$\min_{\xi_i \in \mathbb{R}, u_i \in \mathbb{R}, d_i \in \mathbb{Z}} \quad t_N \tag{4.1a}$$

$$s.t. \quad \xi_0 = \tilde{\xi}_0, \tag{4.1b}$$

$$\xi'_{i+1} = f_{dyn}(\xi_i, u_i), \ i = 0, ..., N \tag{4.1c}$$

$$\xi_i \in [\underline{\xi}, \overline{\xi}], \ i = 0, ..., N \tag{4.1d}$$

$$u_i \in [\underline{u}, \overline{u}], \ i = 0, ..., N-1 \tag{4.1e}$$

$$h_{coll}(\xi_i, \xi_i^{\text{opp}}, d_i) \leq 0, \ i = 0, ..., N. \tag{4.1f}$$

where the function $h_{coll}(\cdot)$ represents the collision avoidance constraints between the ego vehicle (with state ξ_i) and the opponent vehicle (with state ξ_i^{opp}). The decision d_i corresponds to different relative positions between two vehicles at each prediction step.

The NMPC formulation (4.1a-4.1e) is in the form of Nonlinear Programming (NLP) since it is actually an optimization problem that involves minimizing an objective function while being subject to a set of constraints, where the equality constraints that represent the system dynamics and the inequality constraints that represent the collision avoidance conditions are nonlinear. We will introduce later in Section 4.3.1 that constraint (4.1f) can be rewritten in a form of $h_{coll}(\xi_i, \xi_i^{\text{opp}}) \leq C_i \cdot d_i$, where h_{coll} is a function for representing the relative positions between two vehicles, C_i are constants, and d_i are binary variables. The introduction of binary variables makes formulation (4.1) result in an Mixed Integer NLP (MINLP) problem (2.18).

There are 2 main difficulties related to formulation (4.1). The first one is the need to represent the vehicle's projected 2-dimensional shape on the ground, a.k.a. the footprint in the context of the robotics community [104], prior to representing collision avoidance constraints (4.1f). Since the NMPC framework is developed in a curvilinear coordinate, the footprint should adapt to the same coordinate. The second difficulty is how to encode decisions at each step and how to solve the resulting decision-making problem.

We deal with the first problem in Section 4.2. Two different approaches are proposed for addressing the second problem. The approach based on Mixed Integer Programming (MIP) [85] catches our interest since it can naturally encode collision avoidance constraints as a combinatorial problem. We present in Section 4.3 how to model and solve the problem using the MIP-based method. To enable the online execution of the controller, in Section 4.4, we propose a method to group and simplify overtaking decisions. In Section 4.5, we introduce the potential refinement of the vehicle's footprint and discuss its integration into the two suggested approaches.

4.2 Representation of the vehicle's footprint

In this section, we investigate how to approximate the vehicle's footprint. First, we go over the state of the art of how the vehicle's footprint is represented in

the Cartesian coordinate. Then, we suggest a method for approximating it in a curvilinear coordinate.

4.2.1 State of the art: the footprint in the Cartesian coordinate

Authors in [105] propose to decompose the vehicle's footprint into several circles for building collision avoidance constraints between the ego vehicle and the oncoming vehicle on a structured road. Authors in [106] extend this method for enabling overtaking behavior on large-curvature roads using the artificial potential field (APF) method that incorporates the distance between representation circles into a cost function. To avoid an aggressive maneuver on large curvature roads, they propose to set "touchable constraint circles" in front of and behind the ego vehicle (in contrast, the circles representing the vehicle itself are not "touchable") for planning the overtaking trajectory as early as possible. A similar concept is developed by authors in [107] in a more generic and theoretical way. They represent the vehicle's footprint by using the control barrier function (CBF) in order to encode faraway environment information into constraints.

For unstructured environments, authors in [108] represent the moving obstacle as an ellipse and approximate the collision region using the Minkowsky sum. For dynamic environments with uncertainties, authors in [109] build probabilistic collision regions and linearize them to a minimal polytope as the constraint, which will provide probabilistic guarantees on the safety of motion planning.

4.2.2 Approximation of the footprint in the curvilinear coordinate

We get inspiration from the state of the art of footprint representation in the Cartesian coordinate and will concentrate on how to appropriately approximate the vehicle's footprint in the curvilinear coordinate, in order to further make use of it in the time-optimal NMPC framework (4.1).

In single-vehicle racing mode, dimension-related requirements only apply to the distance between the vehicle's center of gravity and the boundary of the track, which is usually set as a value more than half the length of the vehicle's diagonal. In head-to-head racing mode, we must take into account the vehicle dimension also for potential collisions between vehicles. To facilitate the handling of collision avoidance constraints, we introduce the technique for representing the geometry of the vehicle as intervals in the curvilinear coordinate system.

As shown in Fig. 4.1, the vehicle's length and width are L and D. For simplicity, we define an approximation of the vehicle's footprint using a single circle in the Cartesian coordinate system. Its center locates at (s_0, e_{y0}) in the curvilinear coordinate system. This approximation simplifies the problem by disregarding the orientation of the vehicle but limits the operational space as a consequence. To avoid the circle being transformed into a shape that is difficult to handle in the curvilinear coordinate system, we over-approximate the circle into a sector in the Cartesian

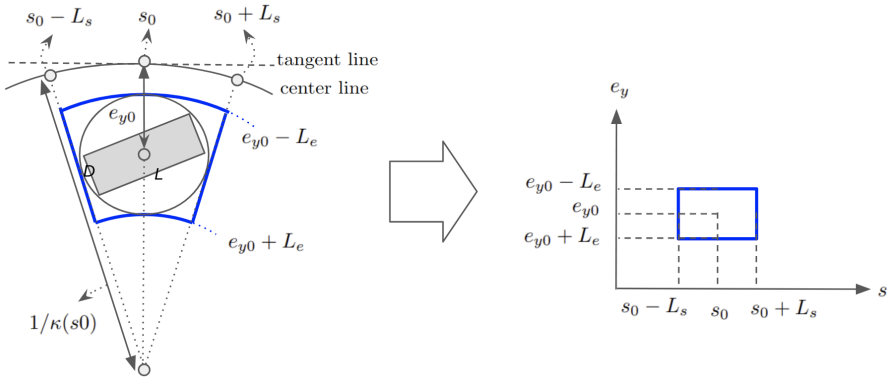

Figure 4.1: Approximation of the vehicle's footprint. The vehicle's shape is approximated as a sector (in blue) in the Cartesian coordinate and then converted into a rectangle in the curvilinear coordinate.

coordinate system and then project it into the curvilinear coordinate system as a rectangle. The vehicle's footprint is represented as:

$$\begin{aligned} &(s, e_y) \in [s_0 - L_s, s_0 + L_s] \times [e_{y0} - L_e, e_{y0} + L_e] \\ &\text{with } L_s = \frac{1}{\kappa(s)} \arcsin \frac{\sqrt{L^2 + D^2}/2}{1/\kappa(s) - e_{y0}}, L_e = \sqrt{L^2 + D^2}/2 \end{aligned} \tag{4.2}$$

In Section 4.5, we will discuss how to refine the approximation by first decomposing the vehicle's shape into multiple identical parts in the Cartesian coordinate, and then representing them in the curvilinear coordinate using rectangles. In sections 4.3 and 4.4, we will use formulation (4.2) for ease of use.

4.3 MIP-based approach for overtaking problem

In this section, we first describe how Mixed Integer Programming (MIP) is used to encode collision avoidance constraints at each step of the prediction horizon and how to compute the optimal overtaking strategy, taking into account the relative positions of both vehicles (the ego vehicle situated at left, right, ahead, or behind of the opponent vehicle). Then, we discuss the experiment result using this MIP-based method.

4.3.1 Encode collision avoidance constraints as a MIP

In this section, we first describe how to interpolate the trajectory of the opponent vehicle for further integration into the collision avoidance constraints. Next, we set up collision avoidance constraints using MIP for encoding overtaking maneuvers.

A requirement for setting up collision avoidance constraints is that: at each progress step in the prediction horizon of the ego vehicle, we should identify paired poses of both vehicles (i.e., the position and orientation of both vehicles existing at

the same time instant) to represent their relative position for building constraints (4.1f). The difficulty is that the ego vehicle's prediction steps are discretized in terms of progress s, while the trajectory of the opponent vehicle is usually estimated or communicated in terms of time t.

Suppose that the trajectory of the ego vehicle is discretized at progress step k. Based on the previous prediction of the ego vehicle at progress step $k-1$, we initialize a first guess for the ego vehicle's N-steps progress time series in the new prediction horizon: $\tilde{t}^{ego}_{i|k} = t^{ego}_{i-1|k-1}, i = 1, \dots, N$. Using this time series, we estimate the paired position of the opponent vehicle, say between trajectory sampling points $[s_1^{opp}, {e_y}_1^{opp}, t_1^{opp}]$ and $[s_2^{opp}, {e_y}_2^{opp}, t_2^{opp}]$. Finally, we can use a linear interpolation method for representing the opponent vehicle's position at each prediction step $i, i = 1, \dots, N$:

$$\begin{cases} s_i^{opp}(t^{ego}_{i|k}) = \frac{s_2^{opp} - s_1^{opp}}{t_2^{opp} - t_1^{opp}} \cdot (t^{ego}_{i|k} - t_1^{opp}) + s_1^{opp} \\ {e_y}_i^{opp}(t^{ego}_{i|k}) = \frac{{e_y}_2^{opp} - {e_y}_1^{opp}}{t_2^{opp} - t_1^{opp}} \cdot (t^{ego}_{i|k} - t_1^{opp}) + {e_y}_1^{opp} \end{cases} \tag{4.3}$$

We now study how to represent the collision-avoidance constraints (4.1f) by using the approximation of the vehicle's footprint for both vehicles (4.2) and the interpolation of position for the opponent vehicle (4.3). For the ego vehicle at progress step k, the collision-avoidance constraint for each prediction step $i = 1, .., N$ in the NMPC horizon is:

$$\begin{aligned} (A)\ & s_i^{opp} + L_s(s_i^{opp}, {e_y}_i^{opp}) \leq s_i^{ego} - L_s(s_i^{ego}, {e_y}_i^{ego}) \\ OR\ (B)\ & s_i^{ego} + L_s(s_i^{ego}, {e_y}_i^{ego}) \leq s_i^{opp} - L_s(s_i^{opp}, {e_y}_i^{opp}) \\ OR\ (C)\ & {e_y}_i^{opp} + L_e \leq {e_y}_i^{ego} - L_e \\ OR\ (D)\ & {e_y}_i^{ego} + L_e \leq {e_y}_i^{opp} - L_e \end{aligned} \tag{4.4}$$

We name each of these constraints using functions f_A, f_B, f_C and f_D so that constraints (4.4) become: $(f_A(i) \leq 0) \vee (f_B(i) \leq 0) \vee (f_C(i) \leq 0) \vee (f_D(i) \leq 0)$.

The above constraints correspond to the following configurations: (A) the ego vehicle is ahead of the opponent vehicle; (B) the ego vehicle is behind the opponent vehicle; (C) the ego vehicle is at the left of the opponent vehicle; (D) ego vehicle is at the right of the opponent vehicle. However, there is an overlap among these 4 configurations. We refine them into 4 new non-overlapping configurations, by adding to cases (C) and (D) the constraint that the ego vehicle is neither totally ahead of the opponent vehicle nor totally behind the opponent vehicle. We formally write them as:

$$\begin{aligned} & (f_A(i) \leq 0) \\ \vee & (f_B(i) \leq 0) \\ \vee & (f_C(i) \leq 0 \wedge (f_A(i) > 0 \wedge f_B(i) > 0)) \\ \vee & (f_D(i) \leq 0 \wedge (f_A(i) > 0 \wedge f_B(i) > 0)) \end{aligned}$$

We then use the big-M method [110] to encode this disjunction into a more standard set of constraints:

$$\begin{cases} f_A(i) \leq c_1 \cdot M \\ f_B(i) \leq c_2 \cdot M \\ f_C(i) \leq c_3 \cdot M \\ -f_A(i) \leq c_3 \cdot M \\ -f_B(i) \leq c_3 \cdot M \\ f_D(i) \leq c_4 \cdot M \\ -f_A(i) \leq c_4 \cdot M \\ -f_B(i) \leq c_4 \cdot M \end{cases} \tag{4.5}$$

where $c_k \in \{0, 1\}, k = 1, 2, 3, 4$ are binary variables satisfying $c_1 + c_2 + c_3 + c_4 \leq 3$, and M is a sufficiently large positive number.

We finally use the method from [111] to reduce the number of binary variables from 4 to 2:

$$\begin{cases} f_A(i) \leq (1 + a_1 - a_2) \cdot M \\ f_B(i) \leq (1 - a_1 + a_2) \cdot M \\ f_C(i) \leq (a_1 + a_2) \cdot M \\ -f_A(i) \leq (a_1 + a_2) \cdot M \\ -f_B(i) \leq (a_1 + a_2) \cdot M \\ f_D(i) \leq (2 - a_1 - a_2) \cdot M \\ -f_A(i) \leq (2 - a_1 - a_2) \cdot M \\ -f_B(i) \leq (2 - a_1 - a_2) \cdot M \end{cases} \tag{4.6}$$

where a_1 and a_2 are binary variables. If $a_1 = 0, a_2 = 1$, the first constraint is active and other constraints are relaxed. If $a_1 = 1, a_2 = 0$, the second constraint is active. If $a_1 = a_2 = 0$, the third group of constraints (3rd - 5th constraints) is active. If $a_1 = a_2 = 1$, the last group of constraints (6th - 8th constraints) is active.

For each prediction step in the NMPC horizon, we introduce 2 binary variables as shown in (4.6). In total, $2 \cdot N$ binary variables are introduced in the collision-avoidance constraints (4.1f). Following the sequential resolution of MIQP sub-problems, the NMPC problem (4.1) will be solved. Finally, a N-steps collision-free time-optimal control strategy for the ego vehicle is generated.

4.3.2 Experiment

We use the same experiment configuration as in Section 3.3. In the code implementation [1], the NMPC framework is generated by ACADO Toolkit [74] and the MIQP problem is solved by the GUROBI Optimizer [91]. The ego vehicle is located at 3 possible initial positions, $(0, 0)$, $(0, -0.1)$, and $(0, 0.1)$, to express a certain diversity of competition scenarii. In different test scenarii, the ego vehicle starts running when the opponent vehicle starts from the origin point $(0, 0)$ and reaches a progress length of $0.1\ m, 0.2\ m, \ldots, 0.8\ m$ for track 1 and $0.1\ m, 0.2\ m, \ldots, 1.5\ m$ for track 2. The

[1] https://github.com/nanli42/MIP_in_NMPC

Table 4.1: Simulation result for head-to-head racing using MIP.

	Horizon length N	# of cases where collision happens	Average lap time [s]	Average calculation time per step before overtaking [s]
Track 1	15	3/24	4.942	0.247
	30	0/24	4.899	0.905
Track 2	15	0/45	10.277	0.243
	30	0/45	10.148	0.832

vehicles' initial longitudinal speed is set to 1.0 m/s. The maximum allowed speed for the ego vehicle is 1.6 m/s, whereas the opponent vehicle's maximum permissible speed is 1.2 m/s. This constitutes 24 and 45 scenarii respectively for track 1 and track 2. These scenarii cover overtaking maneuvers at different positions on both tracks.

Table 4.1 summarizes the lap time and computation time in all scenarii. As can be expected, a longer horizon yields better lap time and higher computation cost. The trade-off between trajectory optimality and computation time naturally depends on the available computing resources. On the other hand, a sufficient horizon is necessary for finding feasible trajectories. Three failed cases with $N = 15$ on track 1 are due to a short prediction horizon: the ego vehicle is traveling at a high speed and is too close to the opponent vehicle to safely brake, which it has not anticipated this situation in advance.

A typical overtaking behavior resulting from the proposed method is presented in Fig. 4.2. It corresponds to one of the 24 scenarii that we test on track 1: the ego vehicle is initially located at $(0, 0)$ and starts running when the opponent vehicle reaches 0.8 m. The figure shows the moment that the ego vehicle arrives at $s = 5.04$ m. From this example, we can observe the following behavior in the ego vehicle's prediction horizon: the ego vehicle plans to follow the opponent vehicle from step 1 to 10 (condition B in (4.4) is active), to overtake the opponent vehicle at the right from step 11 to 19 (condition D in (4.4) is active), to completely be ahead of the opponent vehicle at step 20 (condition A in (4.4) is active) and to keep this advantage until step 26, to keep at the left of the opponent vehicle at the last 4 steps (condition C in (4.4) is active). It demonstrates that the collision-free time-optimal trajectory generated by formulation (4.1) is capable of enabling overtaking maneuvers.

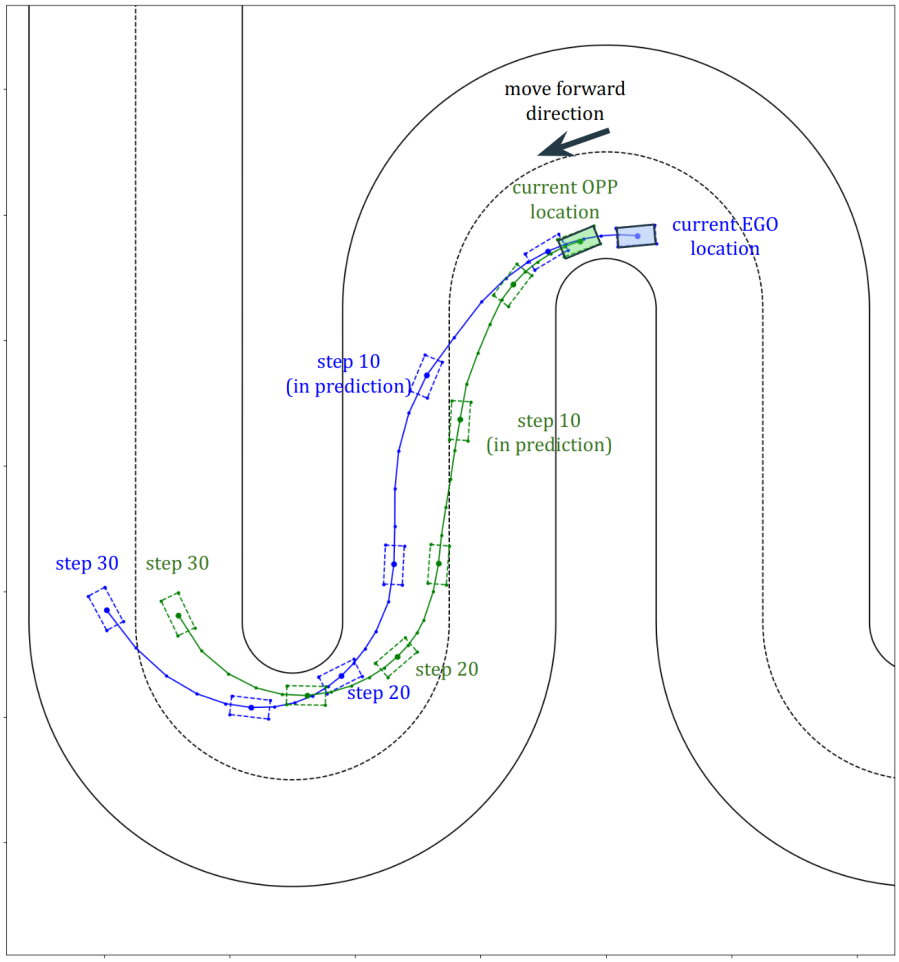

Figure 4.2: A typical example of predicted trajectories of the ego/opponent vehicle on track 1. Blue/green rectangles: the ego/opponent vehicle at the actual location, followed by predicted positions at steps 5 / 10 / 15 / 20 / 25 / 30. Dot lines: predicted trajectories of the ego/opponent vehicle.

4.4 Simplified decision-making approach for overtaking problem

In Section 4.3.2, the MIP-based method is validated in the simulation for generating collision-free overtaking-enable trajectories. However, the calculation cost of NMPC combined with MIP is high, making it difficult to employ in real-time scenarii.

In this section, we propose a simplified control strategy to set all overtaking decisions (4.1f) on either the left or right side to achieve overtaking maneuvers while also maintaining fast computation. Authors in [112] proposed a similar method for switching the overtaking side between left and right, which is based on Model Predictive Contouring Control (MPCC) [34]. They determine which side to overtake by using a learning-based method and only carry out the computation for the chosen side. Our work uses a different base controller: time-optimal NMPC (2.8). In our work, trajectories on both sides are computed, potentially in parallel, and the one that results in the faster progress time is selected as the driving strategy.

4.4.1 Deterministic control period and short-horizon NMPC

Before introducing the simplified control strategy, we describe a slightly modified control framework for facilitating the online execution of the controller.

In the classical control framework, monitoring of the progress s is needed to trigger NMPC recalculations with a non-constant control frequency, which costs additional computation resources for continuously converting the pose information in the Cartesian coordinate into the representation in the curvilinear coordinate which contains s. We propose to launch the recalculation at a fixed frequency. In this way, we monitor the time t instead of s for triggering the recalculation, which is computationally cheap and more accurate. The classical control framework discussed in Section 3.1 is therefore modified as follows.

The NMPC is calculated in the same manner as in the classical framework. As shown in the left of Fig. 4.3, the prediction horizon is discretized in terms of progress distance s into N steps. The optimal control series calculated for time instant t is $u^*_{i|t}, i = 0, \dots, N-1$. Instead of deploying control commands with integer multiples of steps (i.e. steps at distance $s = s_0 + \{\Delta s, 2\Delta s, \dots, (N-1)\Delta s\}$), we release the control periodically using the parameter T^{ctrl}. As shown in the right of Fig. 4.3, the deployed control is denoted as $u^*(t \to t + T^{\text{ctrl}}) = [u^*_{i|t}], 0 \leq i < N_T$, where N_T satisfies that $1 \leq N_T \leq N \;\;\&\;\; t^*_{N_T-1} - t \leq T^{\text{ctrl}} \;\;\&\;\; t^*_{N_T} - t > T^{\text{ctrl}}$. During the period $t \to t + T^{\text{ctrl}}$, we recalculate the optimal control series $u^*_{i|t+T^{\text{ctrl}}}, i = 0, \dots, N-1$ for deploying at the time instant $t + T^{\text{ctrl}}$.

With the appearance of the opponent vehicle (here we assume that the exact pose of the opponent vehicle is available, while in Chapter 5, a specific algorithm for detecting the opponent vehicle will be introduced), we should take into account the evolution of the system dynamics of both vehicles. In other words, the constraints are based on the opponent vehicle's intended trajectory, which might become invalid as time passes and lead to functional incorrectness. For safety reasons, we enforce a

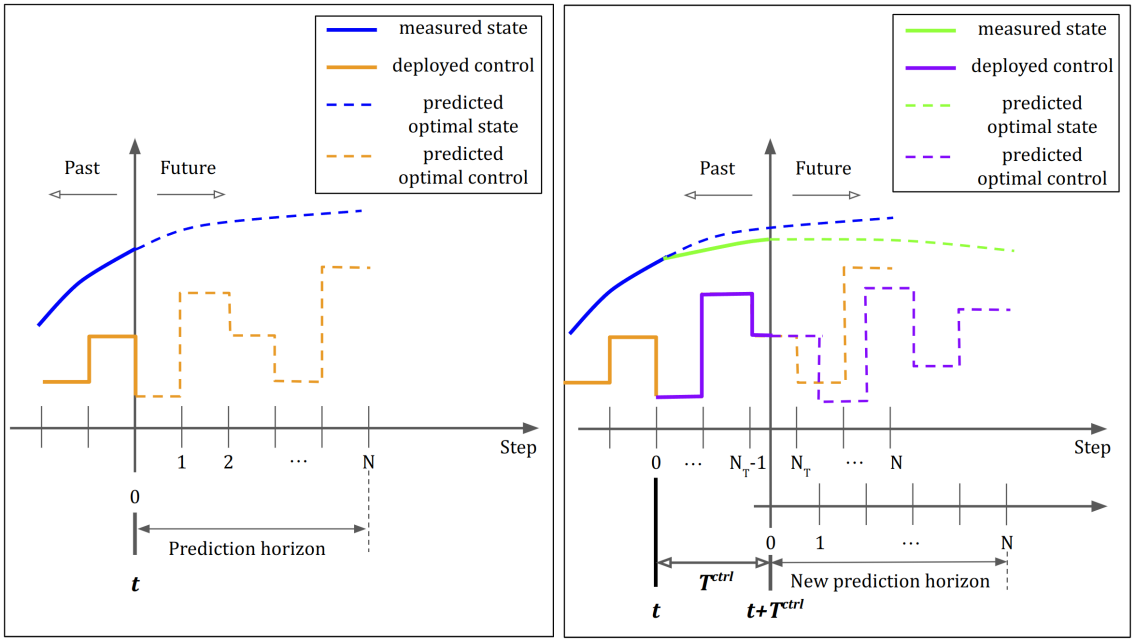

Figure 4.3: Deployment procedure for NMPC using periodic control.

safety distance $d_{\min}$ that allows the ego vehicle to have sufficient braking space and time to react. In an extreme case, the ego vehicle follows closely to the opponent vehicle with the maximum speed, thus the required braking distance is ${v_{\max}^{\text{ego}}}^2/2a_{\max}$. We also take into account the factor that: before receiving the braking command from the control update, the ego vehicle continues to move forward. The safety distance between the front of the ego vehicle and the rear of the opponent is thus defined as $d_{\min} = {v_{\max}^{\text{ego}}}^2/2a_{\max} + v_{\max}^{\text{ego}} \cdot T^{\text{ddl}}$, where T^{ddl} refers to the control deadline, i.e. the time between the start and completion of the task. To ensure that the task is completed before its deadline, we should update the control signal in a time shorter than T^{ddl}. In other words, we specify an upper bound for the control period, $T_{\max}^{\text{ctrl}} = T^{\text{ddl}} = (d_{\min} - {v_{\max}^{\text{ego}}}^2/2a_{\max})/v_{\max}^{\text{ego}}$, to prevent potential collisions caused by control delays.

In addition to meeting the above requirement of functional correctness, we should also guarantee that the controller has sufficient time to finish the calculation even in the worst case. The minimum control period should be no shorter than the time budget: $T_{\min}^{\text{ctrl}} = C$, where time budget $C = \text{WCET}_{\text{per iter}}(N) \cdot N_{\text{iter}}^{\max}$ and $\text{WCET}_{\text{per iter}}(N)$ represents the worst-case execution time (WCET) of one iteration in the SQP procedure for an NMPC with the prediction length N. Finally, we have the constraint on the control period: $C = T_{\min}^{\text{ctrl}} \leq T^{\text{ctrl}} \leq T_{\max}^{\text{ctrl}} = T^{\text{ddl}}$, i.e. $\text{WCET}_{\text{per iter}}(N) \cdot N_{\text{iter}}^{\max} \leq T^{\text{ctrl}} \leq (d_{\min} - {v_{\max}^{\text{ego}}}^2/2a_{\max})/v_{\max}^{\text{ego}}$.

In the head-to-head racing mode, a quick reaction time, i.e. a short enough control period, is needed for avoiding a potential collision with the opponent vehicle. Also, the procedure of SQP should always be completed to ensure the satisfaction of system constraints (especially collision-free constraints). To cut down the computation time below a given time budget (or a pre-determined control period), we

employ a short-horizon NMPC, i.e. reduce $\text{WCET}_{\text{per iter}}(N)$ by shortening N. In this way, we can recalculate the NMPC with high frequency. In head-to-head competition, the correctness of the result (i.e. it must be delivered within the given time budget) is of paramount importance. We may slightly degrade the lap time performance by using short-horizon NMPC, but it is worth making this trade-off to guarantee functional correctness. As an example shown in Table 3.1, long-horizon ($N = 30$) has only a relatively small advantage compared to short-horizon ($N = 15$) in terms of lap time: 1.6% and 1.2% better on the two tracks. This lap time difference is significantly smaller compared to the amount of time needed to react to unexpected events in the environment (e.g. emergency braking to avoid a collision with the opponent vehicle caused by the delayed control), which shows that using short-horizon NMPC is a suitable choice for obtaining high reacting frequency in head-to-head racing mode.

4.4.2 LOROFO (Left-side Overtaking, Right-side Overtaking, and FOllowing) algorithm

While the MIP-based method provides an optimal choice among all possible overtaking strategies (which relies on the relative position of both vehicles at each prediction step), the introduction of integer variables makes the computation costly. In some circumstances (e.g. when the onboard calculation resource is limited), we might prefer to sacrifice some optimal performance in order to decrease the cost of calculation. In both human racing scenarii and the numerical simulation presented in Section 4.3, we can observe that overtaking maneuvers usually occur in successive steps and only happens on one single side (either left or right). We can simplify the control strategy as one-side overtaking, which is one kind of "Time-step grouping" as discussed in Section 2.6.2. The resulting overtaking strategy might be sub-optimal, but the computation time is reduced while safety is still guaranteed by the satisfaction of collision avoidance constraints. The algorithm is named LOROFO (Left-side Overtaking, Right-side Overtaking, and FOllowing).

A formal description of LOROFO is defined in Algo. 1, in which D_1 represents the distance threshold to trigger the overtaking strategy, a larger D_1 value indicates an earlier involvement in the planning for overtaking planning; D_2 is the safe distance in terms of e_y, a larger D_2 value corresponds to a larger lateral safety distance when single-side overtaking is active; $d_{\min}$ is the emergency braking distance that allows the ego vehicle to have sufficient braking space and therefore enough time to react. Line 1-3 in Algo. 1 means that, if two vehicles are far away from each other, we use NMPC formulation (2.8) to build the controller for its forward (FW) progress, which is the same controller used in the single-vehicle racing mode. Otherwise, in the prediction horizon of the ego vehicle, we find the first prediction step i where collisions with the opponent vehicle might occur (line 5). Then, starting from step i, we set up the constraints for the next M steps ($i + M \leq N$) for left and right side overtaking (LO and RO, line 6-8) as shown in Fig. 4.4. They can be solved in parallel and the one resulting in a better lap time performance will be selected as the optimal overtaking strategy (line 10). To ensure that this simplified overtaking

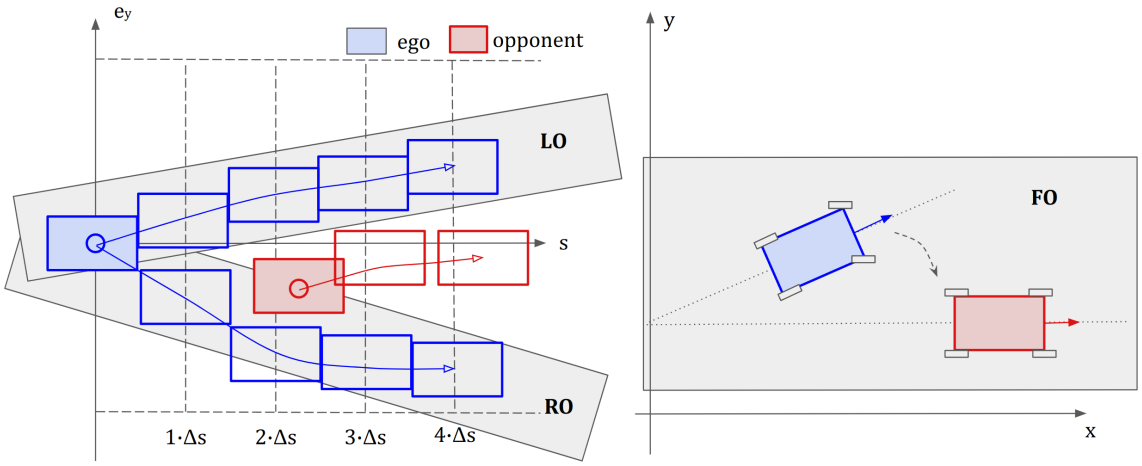

Figure 4.4: An illustrative example for the LOROFO algorithm.

attempt is safe, we propose the following technique for the situation where both sides overtaking is impossible: if two vehicles still keep a safe distance, we let the ego vehicle run at the same speed as the opponent vehicle and continuously steer towards it, i.e. follow (FO) the opponent vehicle (line 15-18) as shown in Fig. 4.4; otherwise, the ego vehicle brakes (line 13).

4.4.3 Experiment

In this part, we demonstrate how to choose parameters in the LOROFO algorithm and how it performs in an overtaking scenario.

Experimental setup

In this experiment, a Robot Operating System (ROS) [113] node [2] runs on the NVIDIA Jetson TX2 (featuring two CPUs and one GPU) as the ego vehicle's controller. The CPU frequency is set to the maximum value: 2.0 GHz. We set the code optimization option to '-O3' for compilation.

We perform the experiment with a Hardware-in-the-Loop configuration: a simulator named f1tenth_gym_ros [114] runs on the laptop and interacts with the controller running on the Jetson TX2 via a USB cable. The simulator sends the odometry information to the controller. And the controller sends back the vehicle speed/steering angle as the command input. The ping test indicates that the highest latency is less than $1[ms]$, which meets our experimental requirements. We use the vehicle dynamics (2.2) and identification parameters described in [35], which corresponds to the F1tenth vehicle model (a 1:10 miniature race car). The racetrack used in the simulator has the same shape as in [47], but is 10 times larger to accommodate the size of the experiment target model.

[2] https://github.com/nanli42/ROS_impl_F1tenth/tree/master/src/f110_nmpc

Algorithm 1 Control strategy LOROFO in head-to-head racing

Input: the position and orientation of both vehicles.

Output: the control command for the ego vehicle.

prepare an NMPC problem formulation FW using formulation (2.8) by ignoring the opponent vehicle
if ($s_0^{\text{ego}} > s_0^{\text{opp}}$ or $s_0^{\text{opp}} - s_0^{\text{ego}} > D_1$) **then**
  solve NMPC FW problem
else
  find the first prediction step $i, 0 \leq i \leq N$ which satisfies: $|s_i^{\text{ego}} - s_i^{\text{opp}}| \leq d_{\min}$
  set up LO: ${e_y}_i^{\text{opp}} + D_2 \leq {e_y}_j^{\text{ego}} \leq {e_y}^{\max}, j = i, \ldots, i + M$
  set up RO: ${e_y}^{\min} \leq {e_y}_j^{\text{ego}} \leq {e_y}_i^{\text{opp}} - D_2, j = i, \ldots, i + M$
  solve LO and RO problems in parallel
  if one of these converges to a feasible solution **then**
    select the one with faster progress time
  else
    if $s_0^{\text{ego}} < s_0^{\text{opp}}$ and $s_0^{\text{opp}} - s_0^{\text{ego}} \leq d_{\min}$ **then**
      brake by letting: $v^{\text{ego}} = 0, \delta^{\text{ego}} = 0$
    else
      calculate $\Delta\psi$ difference between the ego's yaw and the angle towards the opponent vehicle
      suppose that the vehicle's path follows the kinematic model: $\delta^{\text{ego}} = \arctan(\Delta\psi / T^{\text{ctrl}} \cdot l^{\text{vehicle}} / v^{\text{ego}})$
      $v^{\text{ego}} = v^{\text{opp}}$
      apply $v^{\text{ego}}, \delta^{\text{ego}}$ to perform FO strategy
    end if
  end if
end if

WCET Measurements

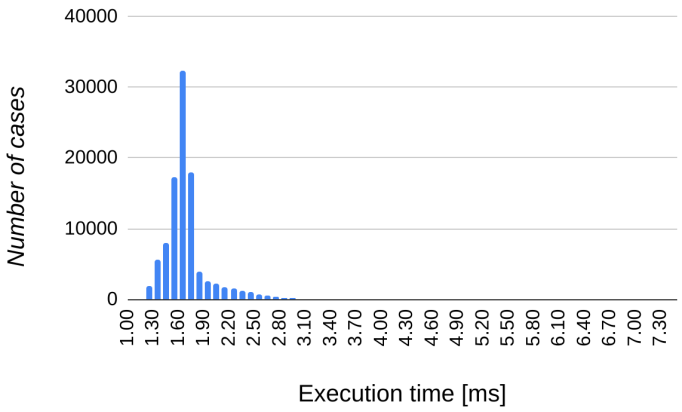

Figure 4.5: Histogram of "NMPC FW" per-iteration execution time.

According to results from several preliminary experiments on the testing track, we find appropriate values for the horizon length and the maximum iteration number: $N = 10$, $N_{\text{iter}}^{\max} = 10$. We use these values as parameters to test the NMPC controllers on 100000 random sampling scenarii. An example of WCET distribution is shown in Fig. 4.5. Detailed statistics on the execution time of the different controller components are provided in Table 4.2. Since LO/RO/FW share the same NMPC problem structure with only the difference in one constraint (see lines 6 and 7 in Algo. 1), their performance is close to each other. We use an optimistic estimation of the WCET for NMPC: $\text{WCET}_{\text{per iter}}(10) \approx 3.0[ms]$. The FO is a simple reactive controller that requires quite a little execution time.

Table 4.2: Controller execution time on Jetson TX2 (in [ms]).

	NMPC			FO
	LO per iter	RO per iter	FW per iter	
max	7.333	7.380	7.322	0.027
99%*	2.928	2.929	2.822	0.007
mean	1.710	1.694	1.688	0.002

*99% means the value at the 99th percentile in the distribution

NMPC Parameters

We recall the criteria discussed in Section 4.4.1 for choosing control period:

$$C = T_{\min}^{\text{ctrl}} \leq T^{\text{ctrl}} \leq T_{\max}^{\text{ctrl}} = T^{\text{ddl}},$$

where $C = \text{WCET}_{\text{per iter}}(N) \cdot N_{\text{iter}}^{\max}$ and $T^{\text{ddl}} = (d_{\min} - {v_{\max}^{\text{ego}}}^2/2a_{\max})/v_{\max}^{\text{ego}}$. The safety distance between the two vehicles' edges is set to be the same as the vehicle's length:

$d_{\min} = 0.58[m]$. The maximum velocity is $v_{\max}^{\text{ego}} = 2[m/s]$ and the maximum braking acceleration is $a_{\max} = 8[m/s^2]$. The upper bound for the control period is calculated as $T^{\text{ddl}} = 165[ms]$. We have the lower bound for the control period, i.e. the time budget: $C = \text{WCET}_{\text{per iter}}(10) \cdot 10 = 30[ms]$. To react to the environment as quickly as possible, we set the control period equal to its lower bound: $T^{\text{ctrl}} = C = 30[ms]$.

Overtaking Behavior

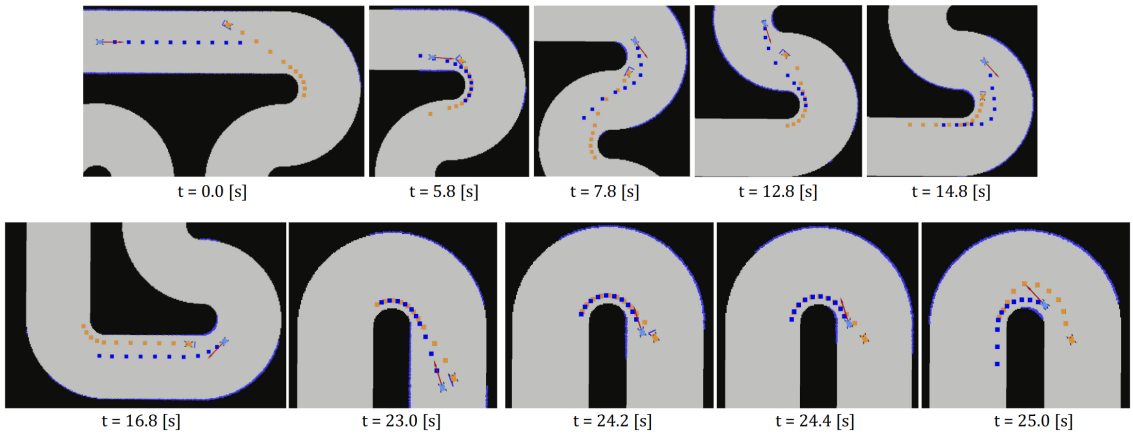

Figure 4.6: A typical scenario in head-to-head competition using the LOROFO method. The ego/opponent vehicle is shown in blue/orange. The dotted lines are planned optimal trajectories.

Using the predetermined value of T^{ctrl}, we test the proposed control method LOROFO (Algo. 1) in the simulator. A typical scenario is shown in Fig. 4.6. In the beginning, the ego vehicle is behind the opponent: $s_0^{\text{ego}} = 0.0[m], e_{y0}^{\text{ego}} = 0.0[m], e_{\psi 0}^{\text{ego}} = 0.0[rad]$ and $s_0^{\text{opp}} = 7.0[m], e_{y0}^{\text{opp}} = 1.0[m], e_{\psi 0}^{\text{opp}} = -0.5[rad]$. At $t = 5.8[s]$, neither LO nor RO strategies give a feasible solution, but the distance between the two vehicles is still safe. The ego vehicle then follows the opponent vehicle: as seen in the second frame of Fig. 4.6, the previously planned optimal trajectory (blue dotted line) is no longer employed because it is invalid at this time and the ego vehicle uses FO strategy instead. At the time $t = 7.8[s]$, only the LO strategy is feasible (no space for RO). At $t = 12.8[s]$, both LO and RO strategies are feasible, the ego vehicle decides to attempt RO at this step since the predicted progress time is shorter than the one on the left side. At $t = 14.8, 16.8, 23.0[s]$, the ego vehicle alters to LO strategy and finally succeeds at $t = 24.2[s]$. From this time instant, the ego vehicle becomes the leader, it is thus the opponent vehicle's responsibility to prevent collisions. Since the two vehicles are close enough to each other, the emergency braking of the opponent vehicle is triggered. The opponent vehicle's position remains nearly unchanged between $t = 24.2[s]$ and $24.4[s]$, proving that it succeeds to brake.

In summary, as seen in the simulation: the safety (collision-free) of the system is ensured by the emergency braking mechanism (see lines 12-13 in Algo. 1); the

execution time is guaranteed to be under the given time budget based on the WCET estimation; the overtaking is enabled, and we get a near-optimal control strategy by taking into consideration the possibility of both left and right side overtaking.

4.5 Refinement for vehicle shape representation

In Section 4.2, we approximate the vehicle's footprint by first fitting it into a circle with a radius equal to the diagonal of the vehicle. We then fit this circle into a sector in the Cartesian coordinate and convert it to a rectangle in the curvilinear coordinate. In this way, the vehicle's footprint is represented in the form of intervals as equation (4.2). It is actually a simple but relatively rough approximation since it represents the vehicle's footprint conservatively.

In the following, we refine this approximation by investigating the potential to decompose the vehicle shape into two identical parts and then generalizing it to the case of m-decomposition. Finally, we discuss how to integrate the refined approximation into the collision avoidance constraints (4.1f).

4.5.1 Decomposition into multiple identical parts

As shown in Fig. 4.7, we first decompose the vehicle's occupied area into two identical rectangles and then cover them using two circles with radii equal to the diagonal length of each decomposed rectangle. Two circles are approximated as two sectors in the Cartesian coordinate, which are finally transferred into rectangles in the curvilinear coordinate.

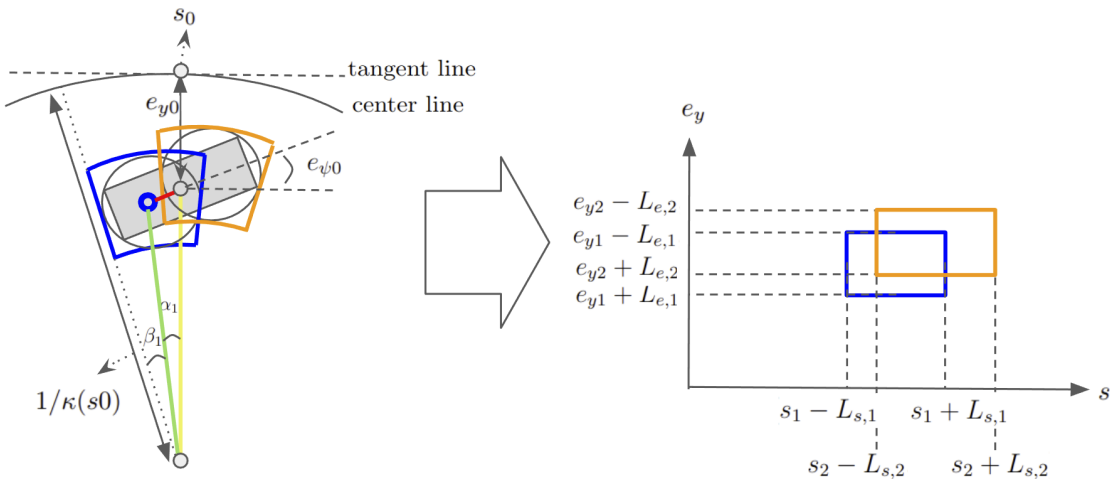

Figure 4.7: Refined approximation of the vehicle's footprint. The vehicle's shape is decomposed into two rectangles which are covered by two circles. They are further approximated as sectors (in blue and orange) in the Cartesian coordinate and converted into rectangles in the curvilinear coordinate.

We suppose the vehicle has length L and width D, with a pose $(s_0, e_{y0}, e_{\psi 0})$ in the curvilinear coordinate. Centers of 2 circles are (s_1, e_{y1}) and (s_2, e_{y2}). We define

the distance between the center of the left circle and the center of the vehicle as a_1 (red solid line in Fig. 4.7), the distance between the center of the vehicle and the local center of the racetrack curve as b_1 (yellow solid line in the figure), and the distance between the center of the left circle and the local center of the racetrack curve as c_1 (green solid line in the figure). Two angles, α_1 and β_1, are defined as in Fig. 4.7. For the right circle, we accordingly define a_2, b_2, α_2, and β_2.

The radius of each circle is $R = \sqrt{(\frac{L}{4})^2 + (\frac{D}{2})^2}$. We have the following intermediate calculation results:

$$
\begin{aligned}
&a_1 = a_2 = a = \frac{L}{4} \\
&b_1 = b_2 = b = \frac{1}{\kappa(s)} - e_{y0} \\
&c_1 = \operatorname{sgn}(b)\sqrt{a^2 + b^2 - 2ab\sin e_{\psi 0}}, c_2 = \operatorname{sgn}(b)\sqrt{a^2 + b^2 + 2ab\sin e_{\psi 0}} \\
&\alpha_1 = \arccos\frac{b^2 + c_1^2 - a^2}{2bc_1}, \alpha_2 = \arccos\frac{b^2 + c_2^2 - a^2}{2bc_2} \\
&\beta_1 = \arcsin\frac{R}{|c_1|}, \beta_2 = \arcsin\frac{R}{|c_2|}.
\end{aligned}
\tag{4.7}
$$

According to geometric relations, we calculate the center position of both circles and their size indicators (L_s and L_e) in the curvilinear coordinate system:

$$
\begin{aligned}
&s_1 = s_0 - \frac{\alpha_1}{|\kappa(s)|}, e_{y1} = \frac{1}{\kappa(s)} - c_1 \\
&s_2 = s_0 + \frac{\alpha_2}{|\kappa(s)|}, e_{y2} = \frac{1}{\kappa(s)} - c_2 \\
&L_{s,1} = \frac{\beta_1}{|\kappa(s)|}, L_{s,2} = \frac{\beta_2}{|\kappa(s)|}, L_{e,1} = L_{e,2} = R.
\end{aligned}
\tag{4.8}
$$

The vehicle's footprint is finally represented by two rectangles in the curvilinear coordinate system: $[s_1 - L_{s,1}, s_1 + L_{s,1}] \times [e_{y1} - L_{e,1}, e_{y1} + L_{e,1}]$ and $[s_2 - L_{s,2}, s_2 + L_{s,2}] \times [e_{y2} - L_{e,2}, e_{y2} + L_{e,2}]$.

If we generalize it to a decomposition with m circles, for each circle $i, i =$

$1, \ldots, m$, the radius is $R = \sqrt{(\frac{L}{2m})^2 + (\frac{D}{2})^2}$ and we have:

$$
\begin{aligned}
a_i &= -\frac{L}{2} + (\frac{L}{m} \cdot i - \frac{L}{2m}) \\
b_i &= \frac{1}{\kappa(s)} - e_{y0} \\
c_i &= \operatorname{sgn}(b) \sqrt{a_i^2 + b_i^2 - 2a_i b_i \sin e_{\psi 0}} \\
\alpha_i &= \operatorname{sgn}(a) \arccos \frac{b_i^2 + c_i^2 - a_i^2}{2bc_1} \\
\beta_i &= \arcsin \frac{R}{|c_i|} \\
s_i &= s_0 + \frac{\alpha_i}{|\kappa(s)|}, e_{yi} = \frac{1}{\kappa(s)} - c_i \\
L_{s,i} &= \frac{\beta_i}{|\kappa(s)|}, L_{e,i} = R.
\end{aligned}
\tag{4.9}
$$

The circle i is approximated as a sector and finally mapped into the curvilinear coordinate system as the rectangle i: $[s_i - L_{s,i}, s_i + L_{s,i}] \times [e_{yi} - L_{e,i}, e_{yi} + L_{e,i}]$.

4.5.2 Integration of refined footprint into constraints

We suggest an approach to further incorporate the refined approximation of the vehicle's footprint into collision constraints (4.1f) in order to improve the performance of the MIP and LOROFO method. We can calculate a minimum bounding rectangle, i.e. an envelope, for covering m resulting rectangles in the curvilinear coordinate. It is actually the leftmost and rightmost components that determine this minimum bounding rectangle. We let $m \to \infty$ and have $R = \frac{D}{2}, a_1 = -\frac{L}{2}, a_m = \frac{L}{2}$ according to (4.9). The minimum bounding rectangle is formally written as:

$$
(s, e_y) \in [s_1 - L_{s,1}, s_m + L_{s,m}] \times [\min(e_{y1}, e_{ym}) - R, \max(e_{y1}, e_{ym}) + R] \tag{4.10}
$$

We take the MIP-based method as an example for showing how the refined footprint (4.10) can be integrated into the constraints. We can replace the previous footprint approximation (4.2) with the refined one (4.10). The collision avoidance constraints for each step $i = 1, .., N$ in the prediction horizon of NMPC are rewritten as (4.11), instead of (4.4), by partitioning the inequality constraints that contain max and min functions.

$$
\begin{aligned}
(A)\ & s_{m,i}^{opp} + (L_{m,s})_i^{opp} \leq s_{1,i}^{ego} - (L_{1,s})_i^{ego} \\
OR\ (B)\ & s_{m,i}^{ego} + (L_{m,s})_i^{ego} \leq s_{1,i}^{opp} - (L_{1,s})_i^{opp} \\
OR\ (C)\ & ({e_{y1}}_i^{opp} + R \leq {e_{y1}}_i^{ego} - R \\
& \ \& \ {e_{ym}}_i^{opp} + R \leq {e_{ym}}_i^{ego} - R \\
& \ \& \ {e_{y1}}_i^{opp} + R \leq {e_{ym}}_i^{ego} - R \\
& \ \& \ {e_{ym}}_i^{opp} + R \leq {e_{y1}}_i^{ego} - R) \\
OR\ (D)\ & ({e_{y1}}_i^{ego} + R \leq {e_{y1}}_i^{opp} - R \\
& \ \& \ {e_{ym}}_i^{ego} + R \leq {e_{ym}}_i^{opp} - R \\
& \ \& \ {e_{y1}}_i^{ego} + R \leq {e_{ym}}_i^{opp} - R \\
& \ \& \ {e_{ym}}_i^{ego} + R \leq {e_{y1}}_i^{opp} - R)
\end{aligned}
\tag{4.11}
$$

Although (4.11) has $6 \cdot N$ more constraints than (4.4), it can still be classified into 4 groups: $(f_A(i) \leq 0) \vee (f_B(i) \leq 0) \vee (f_C(i) \leq 0) \vee (f_D(i) \leq 0)$. The required number of binary variables remains the same as before: $2 \cdot N$ (each binary variable has 2-bit information and represents 4 conditions in total). The rest of the problem-solving steps is the same as in Section 4.3.1.

Table 4.3: Simulation result for head-to-head racing using MIP and the refined approximation of the vehicle's footprint.

	Horizon length N	# of cases where collision happens	Average lap time [s]	Average calculation time per step before overtaking [s]
Track 1	15	3/24	4.803	0.469
	30	0/24	4.736	1.843
Track 2	15	2/45	10.085	0.411
	30	0/45	9.913	1.416

We use exactly the same experiment settings as in Section 4.3.2. The result is shown in Table 4.3. By comparing it to Table 4.1, we can see that the lap time performance is improved when using the refined approximation of the vehicle's footprint. As shown in Fig. 4.8, starting from the same position and with the same condition, the method with the refined footprint approximation allows for a closer distance between two vehicles, while the method with the rough approximation is more conservative and keeps a larger distance. However, using a less conservative approximation also leads to more aggressive racing behavior, as indicated by the higher number of collisions in Table 4.3. Additionally, the use of complicated representation (4.9) and the inclusion of more constraints in (4.11) contribute to a longer calculation time.

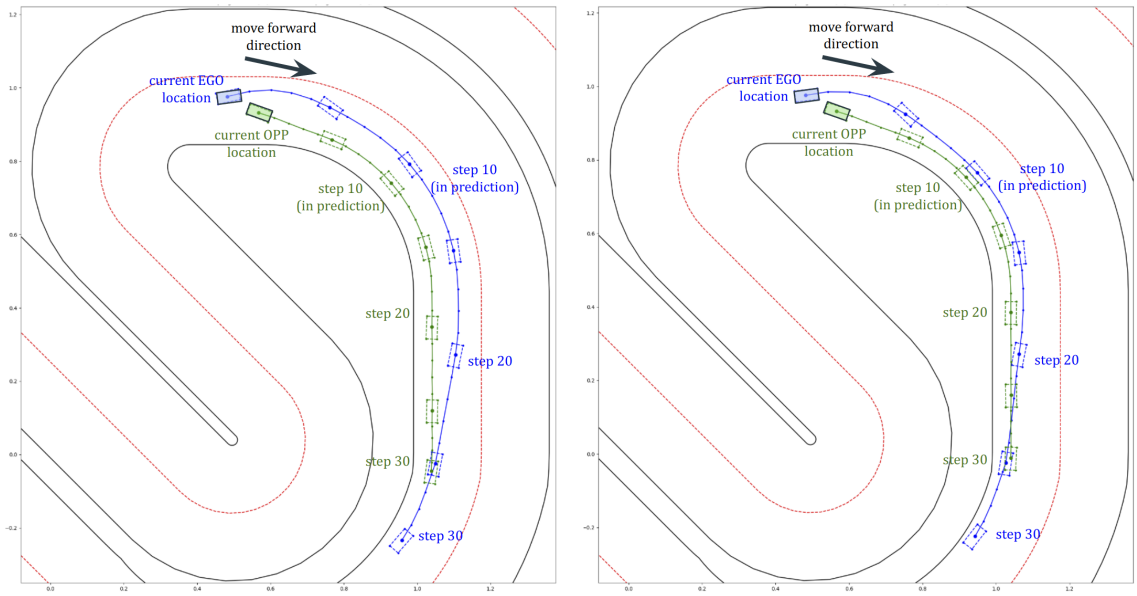

Figure 4.8: A typical example of predicted trajectories of the ego/opponent vehicle on track 2 (Left/Right: using a rough/refined approximation for the vehicle's footprint). Blue/green rectangles: the ego/opponent vehicle at the actual location, followed by predicted positions at steps 5 / 10 / 15 / 20 / 25 / 30. Dot lines: predicted trajectories of the ego/opponent vehicle.

Therefore, the decision of whether or not to use a more accurate representation of the vehicle's footprint involves balancing the benefits of improved lap time against the downsides of increased computational cost and potentially more aggressive driving style that it leads to.

Incorporating the refined approximation of the vehicle's footprint into collision constraints can also be achieved through other methods. One approach is to set a safe relative position between each decomposed component of both vehicles as collision avoidance constraints, i.e. the absence of footprint overlap. Taking the MIP-based method as an example, we can replace the "4 configurations of relative positions" (4.4) with:

$$\begin{aligned} &(A)\ s_i^{opp,j} + (L_s)_i^{opp,j} \leq s_i^{ego,k} - (L_s)_i^{ego,k} \\ OR\ &(B)\ s_i^{ego,k} + (L_s)_i^{ego,k} \leq s_i^{opp,j} - (L_s)_i^{opp,j} \\ OR\ &(C)\ e_{y_i}^{opp,j} + (L_e)_i^{opp,j} \leq e_{y_i}^{ego,k} - (L_e)_j^{ego,k} \\ OR\ &(D)\ e_{y_i}^{ego,k} + (L_e)_i^{ego,k} \leq e_{y_i}^{opp,j} - (L_e)_i^{opp,j} \\ &\text{where } j = 1, \ldots, m, k = 1, \ldots, m. \end{aligned} \tag{4.12}$$

The rest of the procedure is the same as in Section 4.3.1. Using this approach, we can get a more refined approximation of the vehicle's footprint. However, it increases the complexity of the calculation due to the increased number of constraints: instead of using $4 \cdot N$ constraints in (4.4), it needs $4 \cdot m^2 \cdot N$ constraints in total. For the LOROFO algorithm, we should have a similar replacement for lines 6-7 in Algo. 1,

which likewise increases the number of constraints by m^2 times.

Another approach is to use a convex hull to enclose all m resulting rectangles in the curvilinear coordinate. This requires further investigation to define the relative positions between the convex hulls of two vehicles, which will be used to replace "4 configurations of relative position" (4.4) in the MIP-based method. Similarly, we need to define these relative positions in the LOROFO method by replacing lines 6-7 in Algo. 1.

4.6 Conclusion

In this chapter, we presented two approaches for enabling safe and efficient overtaking maneuvers in head-to-head racing mode. The main challenge is to efficiently model the overtaking decisions within the time-optimal NMPC framework to obtain optimal results.

In order to represent collision avoidance constraints in an easy-to-handle form, we introduced a method and its refined version to appropriately define the vehicle's footprint in the curvilinear coordinate. Using the MIP-based method, we encode the overtaking behavior into a series of integers within the NMPC framework. The big-M method is used to convert the disjunction condition into a conjunction form to facilitate the integration of decision-making into NMPC. This approach can model intricate combinational overtaking actions and provide a reliable and effective control strategy. To overcome the issue of the high complexity of MIP, we introduced the LOROFO algorithm to simplify this decision-making problem. It is based on the NMPC framework and works with a deterministic control period and a short prediction horizon. In the experiment, the LOROFO algorithm is validated to be online feasible and to offer a satisfactory control strategy.

CHAPTER 5

Task execution model for autonomous racing systems

In this chapter, we first introduce a generic system architecture for the autonomous racing problem. Then, we study a task execution model for mapping software components in the system, which is a Directed Acyclic Graph (DAG), onto an embedded system. The feasibility and performance of the proposed model are finally examined using a Hardware-in-the-Loop simulation.

5.1 A generic system architecture

In this section, a generic system architecture is proposed to assist us in understanding how different algorithms work together to form a coherent and time-sensitive system. We first present software components of an autonomous racing system. A DAG is then built up for representing dependencies between different components. Finally, we compare different computation platforms for enabling the online execution of software components in the DAG.

5.1.1 Software components

We assume that the vehicle is equipped with the Light Detection and Ranging (LiDAR) sensor to sense the surrounding environment, as well as odometry information to roughly estimate its movement. They serve as input for opponent detection and self-localization algorithms. The Algo. 1 LOROFO described in section 4.4 is used to generate a time-optimal and collision-free control strategy. The following subsections introduce different system components in detail.

Since racetracks are usually static, the majority of localization-related research is based on a fixed 2D map, in the form of occupancy grids or landmarks. The map can be built either from satellite images for outdoor racetracks or by a popular method, Simultaneous Localization and Mapping (SLAM) [115] [116] and its advanced version

GraphSLAM [117], for indoors racetracks. The following work uses a pre-defined 2D occupancy grid map without losing generality.

Opponent detection

In Chapter 4, we focus on the controller design and assume that the exact pose of the opponent vehicle is provided as the controller's input. However, such information is not directly available in a realistic racing scenario. In this case, we rely on sensor information such as LiDAR data to detect the presence of the opponent vehicle and estimate its pose. This perception information will be further provided to the controller. LiDAR-based opponent detection is largely studied in the community of robotics and autonomous passenger car driving, which can be separated into two groups: first, geometric clustering, e.g. Euclidean Clustering [118], it takes into account basic geometric features of objects to be recognized; second, learning-based method [119], it classifies the LiDAR reflection points into different categories, such as track boundaries and outlines of vehicles respectively in racing cases.

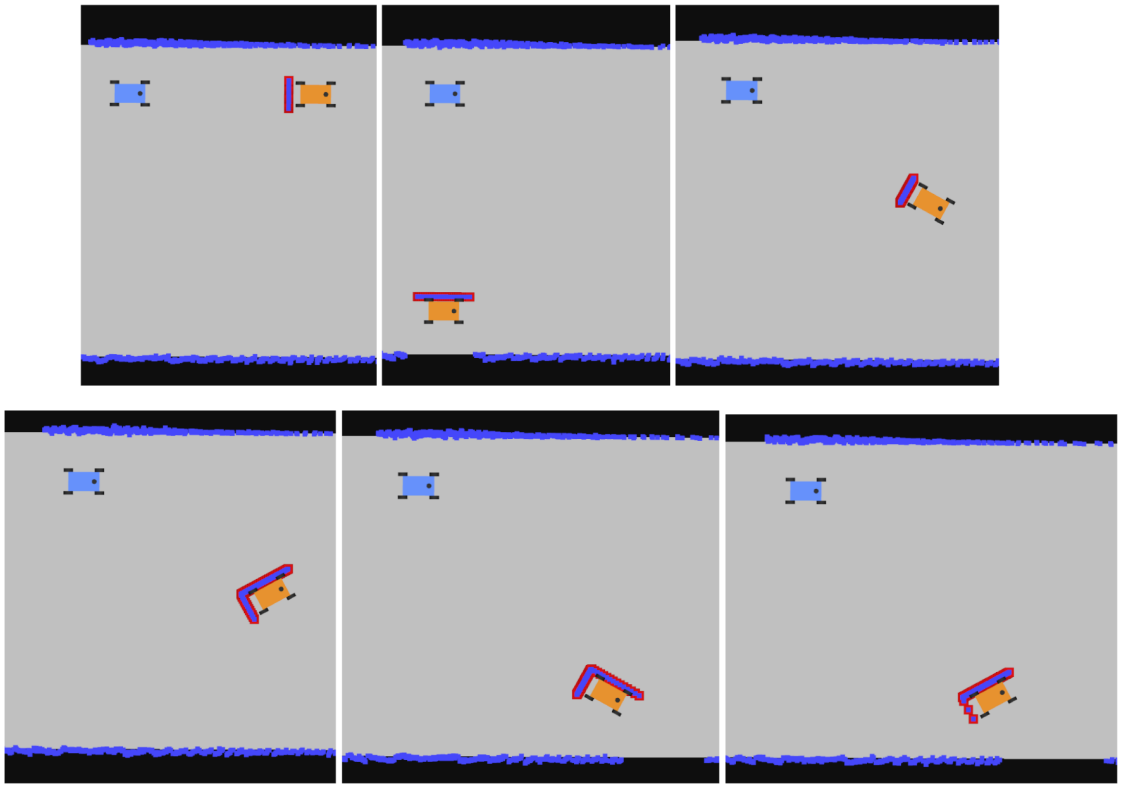

Figure 5.1: Examples for showing relative poses between two vehicles: ego/opponent vehicle in blue/orange. Blue points: LiDAR points captured by the ego vehicle. Red points: laser rays reflected from the opponent vehicle.

Opponent detection in the head-to-head racing scenario is simpler than in a general racing case. On a given 2D map, LiDAR points can be classified into 2 types: ones corresponding to the track boundary (blue points in Fig. 5.1) will lie on

the borderline between the drivable and non-drivable zones (grey and black areas), while others corresponding to the opponent vehicle (red points) will lie inside the drivable zones (grey areas). The opponent vehicle's shape is a rectangle with a known size. From the view of the ego vehicle, LiDAR points corresponding to the opponent vehicle are grouped into a form of either I-shape or L-shape (first-row v.s. second-row images in Fig. 5.1). It is thus straightforward to determine the opponent vehicle's pose using its simple geometric characteristics.

Algorithm 2 Opponent Detection (OD)

Input: LiDAR points captured by ego vehicle; last pose estimation of the ego vehicle $(x^{ego}, y^{ego}, \psi^{ego})'$; last driving command u.

Output: If the opponent vehicle exists, return: True, its relative pose to the ego vehicle $(\Delta x, \Delta y, \Delta\psi)$, and LiDAR points corresponding to the opponent vehicle; otherwise, return False.

1: Load the static 2D occupancy grid map.
2: Estimate the current pose of the ego vehicle $(x^{ego}, y^{ego}, \psi^{ego})_{\text{est}}$, using the command u and time duration Δt (which is between the current time instant and the time instant when the pose is lastly estimated):
$(x^{ego}, y^{ego}, \psi^{ego})_{\text{est}} = (x^{ego}, y^{ego}, \psi^{ego})' + u \cdot \Delta t$.
3: **for** $i \leftarrow 0, N$ **do**
4: Calculate the position of LiDAR point i using $(x^{ego}, y^{ego}, \psi^{ego})_{\text{est}}$ and information of the laser ray i.
5: **if** LiDAR point i lies inside a drivable area of the grid map **then**
6: Collect it into set S_{opp}.
7: **end if**
8: **end for**
9: **if** $S_{opp} = \emptyset$ **then**
10: **return** False (i.e. No opponent exists.)
11: **else**
12: Use points in S_{opp} to find the vertices of the rectangle that represents the opponent's vehicle.
13: **if** 3 vertexes found **then**
14: Fit the rectangle shape into L-shape.
15: **else**
16: Fit the rectangle shape into I-shape.
17: **end if**
18: Infer the opponent vehicle's relative position and orientation $(\Delta x, \Delta y, \Delta\psi)$.
19: **return** True, $(\Delta x, \Delta y, \Delta\psi)$, S_{opp}
20: **end if**

In the Algo. 2, we show how the Opponent Detection (OD) algorithm is designed in our architecture. In lines 1-2, we roughly estimate the current pose of the ego vehicle. In lines 3-8, we analyze each laser ray and collect LiDAR points potentially corresponding to the opponent vehicle. If there is no such point, we return "False" for reporting the absence of the opponent vehicle. Otherwise, in lines 12-19, we fit

LiDAR points into a rectangle shape and infer the relative pose of the opponent vehicle.

Self-localization

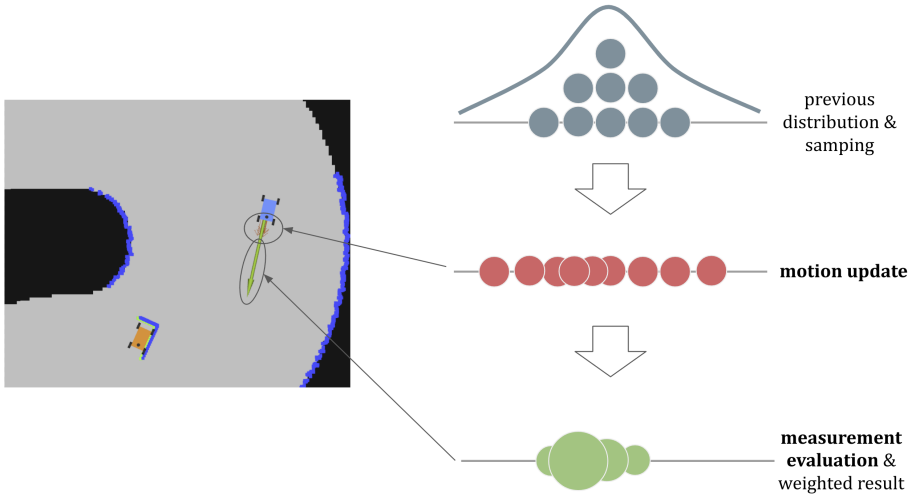

Figure 5.2: Example for showing Particle Filter algorithm. Ego/opponent vehicle in blue/orange. Small orange arrows: pose of different particles. Big green arrow: averaged pose estimation.

In a realistic racing scenario, the ego vehicle needs to infer its pose information using onboard sensors. Particle Filter (PF) localization [120][121] is a widely used approach. As shown in Fig. 5.2, it consists of two important phases: motion update, the estimation of the vehicle's pose is updated using a dynamic model; measurement evaluation, the captured LiDAR points are evaluated to generate a matching score for each particle's pose. Finally, it calculates the most probable pose by averaging the pose of all particles in a given grid map. In comparison to other methods of localization, PF is fairly simple to implement. Furthermore, the calculation for each particle can be parallelized, which will be favorable for systems with multi-core CPUs and GPUs.

In this work, we employ the PF method to localize the ego vehicle. The standard approach usually deals with a static environment, while in head-to-head competition there is a moving opponent vehicle. One possible solution for handling the presence of the opponent vehicle is to sample the pose-pairs of both vehicles instead of just sampling the ego vehicle [122]. However, this combination of both vehicles' poses adds to the algorithm's complexity. We propose to use a simple workaround suitable for the head-to-head racing scenario: first remove laser rays obscured by the opponent vehicle from the LiDAR data using the information provided by Algo. 2 OD; then perform a standard PF technique for localizing the ego vehicle. This simple method enhances localization accuracy, particularly when two vehicles are close to one another, while not increasing the algorithm's complexity. Note that

the removed laser rays represent only a relatively small range of angles. For example, by supposing that the ratio of the vehicle's length and width is 2:1, in an extreme case when two vehicles are closely side-by-side, the obscured angles will be $\arctan(2)\cdot 2 \approx 127°$. Modern LiDAR usually covers a large field of view (FOV) [123]. For instance, in the subsequent experiment, the used LiDAR has a FOV of 270°. A trustworthy result from PF can still be obtained by using the remaining laser rays. In the case of LiDAR with a small FOV, we should consider using cameras or other sensors to obtain extra information to make up for this deficiency.

Algorithm 3 Particle Filter (PF)

Input: LiDAR points captured by ego vehicle; last pose estimation of the ego vehicle $(x^{ego}, y^{ego}, \psi^{ego})'$; last driving command u; set S_{opp} from Algo. 2 OD; relative position of the opponent vehicle $(\Delta x, \Delta y, \Delta\psi)$.

Output: Pose of both vehicles.

1: Filter the captured LiDAR points by removing points that belong to set S_{opp}.
2: Perform the "motion update" phase using $(x^{ego}, y^{ego}, \psi^{ego})'$ and u.
3: Perform the "measurement update" phase using the filtered LiDAR points.
4: Calculate the average pose of the ego vehicle $(x^{ego}, y^{ego}, \psi^{ego})$ of all particles.
5: Calculate the pose of the opponent vehicle: $(x^{opp}, y^{opp}, \psi^{opp}) = (x^{ego}, y^{ego}, \psi^{ego}) + (\Delta x, \Delta y, \Delta\psi)$.
6: **return** $(x^{ego}, y^{ego}, \psi^{ego})$ and $(x^{opp}, y^{opp}, \psi^{opp})$.

Algo. 3 formally describes our proposal for the self-localization algorithm. The first line means to preprocess the LiDAR data. Lines 2 and 3 represent a standard PF procedure. In the PF procedure, a key component called "ray marching" can be performed on GPU for speeding up the evaluation of the particles' pose. Lines 4-5 calculate the pose of both vehicles as the final result.

Control

To achieve the objective of time-optimal control, we use the online algorithm, Algo. 1 LOROFO, as the race car's controller. We should be aware that this algorithm does not explicitly take into account the latency between receiving the LiDAR data and releasing the control command: as a controller, it receives as input the estimated pose of both vehicles from OD and PF, which is calculated using the received LiDAR data that date $\Delta t_1 = t_{\text{OD}} + t_{\text{PF}}$; its output will be released after the NMPC calculation $\Delta t_2 = t_{\text{NMPC calculation}}$.

To compensate for the latency $\Delta t_{\text{latency}} = \Delta t_1 + \Delta t_2$, there are two different approaches. The first one is to adapt the input of Algo. 1 from the measured state ξ_{measured} to a predicted state $\xi_{\text{predicted}}$, i.e. integrating an ordinary differential equation (ODE) to simulate the evolution of system dynamics under ongoing control command u_{current}:

$$\xi_{\text{compensation}} = \xi_{\text{predicted}} = f^{\text{integration}}_{\text{Runge Kutta}}(\xi_{\text{measured}}, u_{\text{current}}, \Delta t_{\text{latency}}).$$

Using this compensated state as input, the controller generates a control command that is adjusted for the control deploying time instant.

The second approach is that: we still use the measured state as the input for Algo. 1, however, instead of employing the control command $u^*(t \to t + T^{\text{ctrl}})$ for time instant t, we use $u^*(t + \Delta t_{\text{latency}} \to t + \Delta t_{\text{latency}} + T^{\text{ctrl}})$ as the appropriate control command during this control period T^{ctrl}.

The approach to take depends on which task model we use. In brief, when the latency $\Delta t_{\text{latency}}$ can be determined in advance, we use the first approach because it is more accurate; otherwise, we use the second one.

5.1.2 Directed Acyclic Graph (DAG)

The aforementioned software components are not independent. Instead, a data flow connects them and they function as a chain. We formally model this relationship for further analysis.

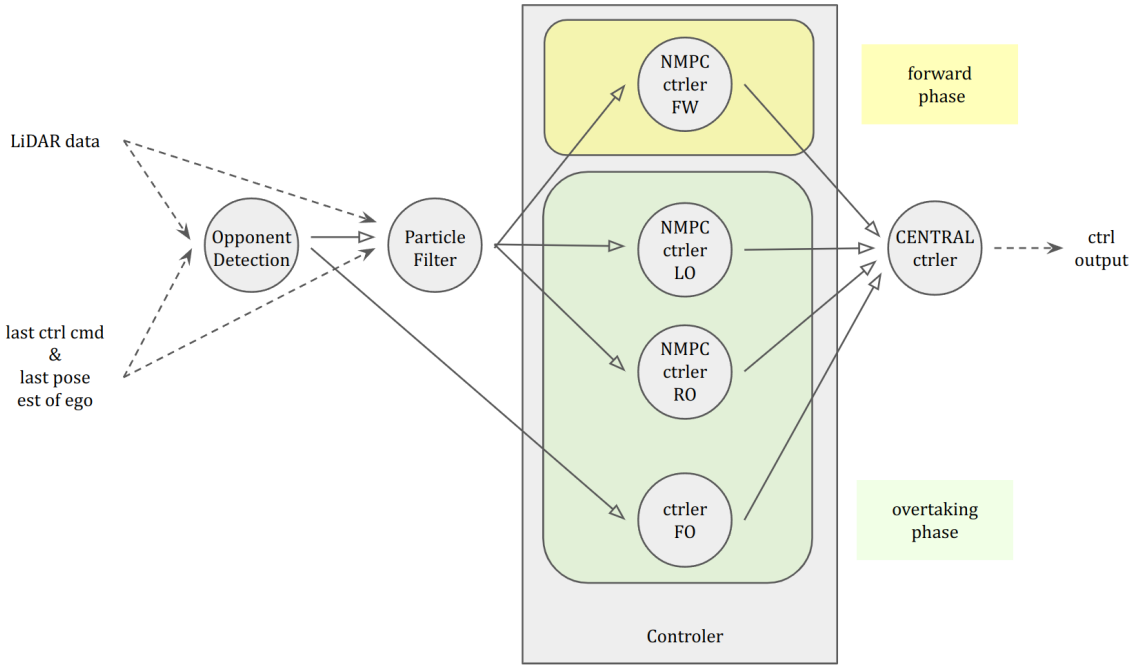

Figure 5.3: Directed Acyclic Graph (DAG) for modeling software components and data flow in the generic system architecture for head-to-head racing mode.

As shown in Fig. 5.3, each software component is considered as a node, and the chain with nodes is modeled as a Directed Acyclic Graph (DAG) $G = (V, E)$. Each node $v \in V$ is represented by a vertex which is characterized by the worst-case execution time (WCET) c. In the figure, the solid edge $(v, w) \in E$ means that the node v must be completed before the node w can start the execution. Dashed edges reflect the system's input and output, whereas solid edges demonstrate data dependencies between nodes. LiDAR data, the last control command, and the last pose estimation of the ego vehicle are inputs for the first vertex (i.e. the source) Opponent Detection (OD) and the second vertex Particle Filter (PF). OD passes information that is related to the opponent vehicle to PF and sends the relative

position to the controller FO. The vertex PF performs self-localization and provides this information to controllers, which is either NMPC controller FW in the forward phase (when two vehicles are far away from each other), or NMPC controller LO/RO and controller FO in the overtaking phase. The last vertex (i.e. the sink) "central controller" (CENTRAL) represents a simple program that aggregates results from individual controllers and sends the control signal to motors. We should mention that, in the overtaking phase, LO, RO, and FO can be parallelized using the fork-join method [124].

5.1.3 Hardware platforms

A variety of mainstream computing hardware, including CPUs, GPUs, field-program -mable gate arrays (FPGAs), application-specific integrated circuits (ASICs), and digital signal processors (DSPs), is available for autonomous vehicles to meet the needs of various types of computing workloads [125].

CPUs are used for general-purpose computation, which is relatively easy to program but consumes more energy and is less efficient than other devices for certain calculation tasks. GPUs are widely used for the acceleration of Deep Neural Networks and parallel image processing. NVIDIA DRIVE series is one of the currently most powerful System on Chip (SoC) containing multi-core CPUs and a many-core GPU. Such SoC is used as the main computation platform in several autonomous racing competitions, e.g. F1tenth competition [114] and Roborace [126]. FPGAs can be used for accelerating specific tasks, e.g. object tracking, to deliver a better per-watt performance [127]. ASICs can save even more energy and computational time for certain dedicated functions. One leading ASIC-based solution for autonomous driving is MobilEye EyeQ5. DSPs such as Texas Instruments' TDA can be used as programmable vision accelerators to perform pre-processing on video streams for computer vision tasks. These DSPs are directly connected to sensors and can greatly improve the efficiency of such tasks. The main advantage of DSP is its energy efficiency but its usage is limited to specific tasks, e.g. opponent detection, object tracking, etc.

In this work, we select NVIDIA Jetson TX2 as the testing platform since it is used in a quite large number of autonomous systems. It is actually an SoC that features a 256-core NVIDIA Pascal GPU, a Dual-Core NVIDIA Denver 2 64-bit CPU, and a Quad-Core ARM Cortex-A57 32-bit MPCore. It is less efficient than NVIDIA DRIVE series but more affordable for lightweight systems.

According to a survey on autonomous racing [5], the middleware Robot Operating System (ROS) is commonly used to simplify the deployment of algorithms and to leverage the existing code in the robotic community. ROS basically uses a publish/subscribe design pattern: tasks are executed on different ROS nodes that can receive/send messages from/to topics. An incoming message from the subscribed topic will initiate the callback function in the ROS node. Another important functionality provided in ROS is the Timer, which allows us to create a periodically triggered execution sequence. In the later experiment, we use ROS to deploy the DAG onto the selected hardware, i.e. NVIDIA Jetson TX2.

5.2 Task execution model

In this section, we introduce a task execution model for mapping the DAG $G = (V, E)$ that is proposed in section 5.1.2 onto m ($m \geq 1$) identical processor(s). The DAG is regarded as a recurrent task. Each execution instance of the DAG is called a *job*. We concentrate on the overtaking phase of the DAG because it features parallelizable components and is more representative of workloads in autonomous vehicle systems, whereas the forward phase of the DAG only consists of straightforward sequential nodes.

We define the problem as how to obtain low end-to-end latency (i.e. short time delay between the beginning of the source vertex of the DAG and the end of the sink vertex of the DAG) but also high control update rate (i.e. short update period) by exploiting the full potential of available processor(s). The end-to-end latency is noted as L. The control update rate is noted as R. We have $R = 1/T$, where T is the control period. A lower L can guarantee that the controller uses the most recent data. R determines how frequently the control command is refreshed. A higher R can ensure that the controller responds to the changing environment more actively.

We assume that the worst-case execution time (WCET) of each node has the following relationship: $c_{\text{FO}} + c_{\text{CENTRAL}} \ll c_{\text{OD}} + c_{\text{PF}} < c_{\text{LO}} = c_{\text{RO}}$. This assumption is reasonable and can be confirmed in the experiment in section 5.3. We define the WCET of the critical path as $C = c_{\text{OD}} + c_{\text{PF}} + c_{\text{LO/RO}} + c_{\text{FO}} + c_{\text{CENTRAL}}$. Without loss of generality, we simplify the problem by assuming that the DAG consists of only two parallelizable computation components with the WCET C. We define the term $degree_{DAG}$ that refers to the maximum number of tasks in the DAG that can be executed in parallel on an unbounded number of processors. We define another term *degree* that refers to the actual number of DAG components that are executed in parallel. In our case, $degree_{DAG} = 2$, meaning that there are at most 2 computation components in the DAG running in parallel. Moreover, since *degree* is no more than $degree_{DAG}$: $degree \in (1, \ldots, degree_{DAG})$, we have either $degree = 1$ or $degree = 2$.

In order to compute latency L without interference from the rest of the system, we assume that the DAG has the highest priority in the system.

5.2.1 $degree_{DAG} = 2$ and $degree = 1$

With $degree = 1$, two computation components in one job run after one another in order to minimize the latency as $L = 2C$. m jobs can be launched on m available processors with an overlap time. When operating at maximum efficiency, we have a uniform period T and $L = mT$. We calculate the control period and the control update rate as $T = 2C/m$ and $R = m/(2C)$.

We conclude that: the latency is fixed as $2C$, while the control period can be shortened (i.e. the control update rate can be increased) as the available processor number increases. Taking $m = 3$ as an example, we show in Fig. 5.4 that the DAG takes the full use of all 3 available processors with the control period $T = 2C/3$ and the update date rate $R = 3/(2C)$.

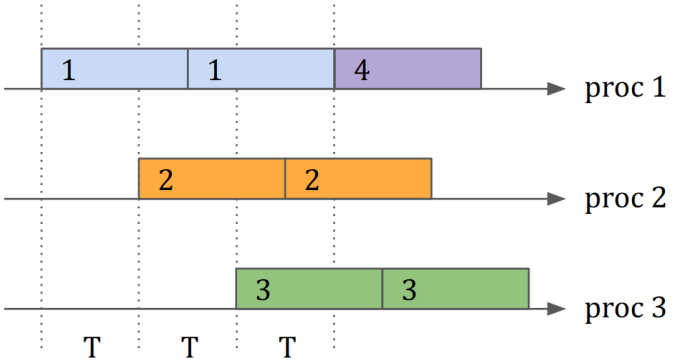

Figure 5.4: An example for $degree_{DAG} = 2$ and $degree = 1$, on 3 available processors. Numbers on rectangles represent the release order of jobs. DAG components belonging to the same job are marked with the same color.

5.2.2 $degree_{DAG} = degree = 2$

With $degree = 2$, when the available processor number m is **even**, i.e. $m\%2 = 0$, we can execute the two computation components strictly in parallel, which means that they have the same release time. It is actually equivalent to the execution of a single task on $m/2$ processors. The latency is thus fixed as $L = C$. Since we can launch multiple jobs at the same time, the control period is $T = C/(m/2) = 2C/m$, and the control update rate is $R = 1/T = m/(2C)$. An example is shown in Fig. 5.5 for the case $m = 4$, where $T = C/2$ and $R = 2/C$.

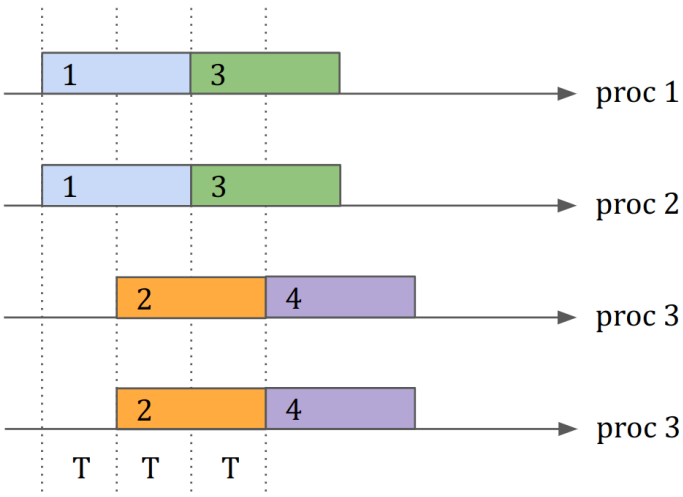

Figure 5.5: An example for $degree_{DAG} = 2$ and $degree = 2$, on 4 available processors. Numbers on rectangles represent the release order of jobs. DAG components belonging to the same job are marked with the same color.

When the available processor number m is **odd**, i.e. $m\%2 = 1$, the two computation components can not be executed strictly in parallel, otherwise, there will be one processor idle and wasted.

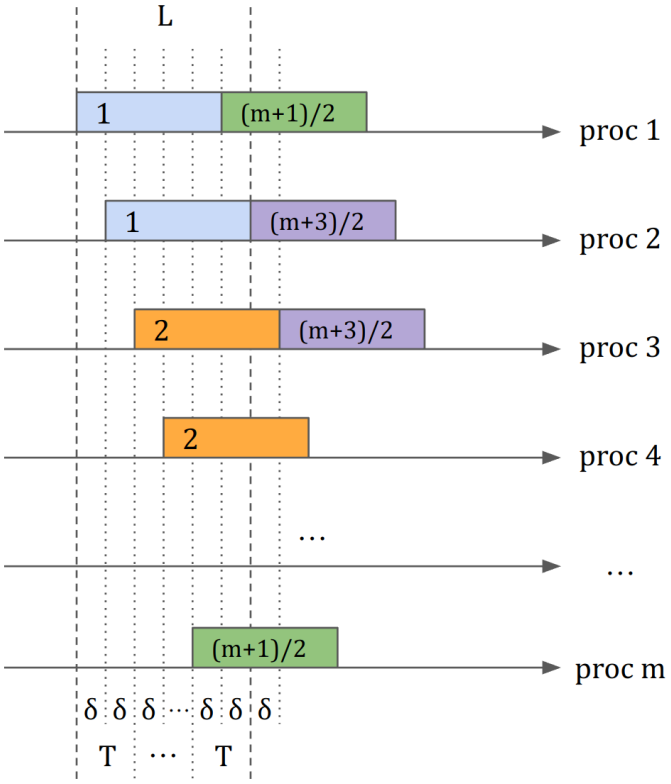

Figure 5.6: An example for $degree_{DAG} = degree = 2$, on m (m is odd) available processors. Numbers on rectangles represent the release order of jobs. DAG components belonging to the same job are marked with the same color.

Suppose that we assign jobs in order: assign the first job to processors 1 and 2, the second job to processors 3 and 4, ..., and the $(m-1)/2$-th job to processors $m-2$ and $m-1$. One computation component in $(m+1)/2$-th job will be assigned to processor m. Since we search for optimal performance, i.e. no idle time on processors, the second computation component in $(m+1)/2$-th job should be assigned to the first available processor. The delay between the launching time of the two components is noted as δ. Since the DAG is periodically launched, jobs share the same execution pattern. In other words, in every job, the two components are launched successively with a delay of δ. As shown in Fig. 5.6, ideally, the second component in $(m+1)/2$-th job should be immediately launched on processor 1 once job 1 is finished, which requires that $(m-1)/2 \cdot 2\delta + \delta = C$. Therefore, we can calculate the value of δ as $\delta = C/m$. The latency is $L = C + \delta = C + C/m$. The control period and the update rate are $T = 2\delta = 2C/m$ and $R = 1/T = m/(2C)$.

5.2.3 $degree_{DAG} = p$

Although in our case the DAG has only parallelism degrees 1 and 2, it will be intriguing to observe how a DAG performs with more parallelizable components, which has some practical applications. For instance, assume that there are variants

for left/right side overtaking strategy, one being safe but conservative and the other being unsafe but with superior performance, the number of parallelizable components in the DAG will increase as $degree_{DAG} = 4$. Another example is that, if there are $n, n > 1$ opponent vehicles, Algo. 1 LOROFO should be changed to account for at most $n+1$ homotopic overtaking trajectories rather than just two-side overtaking trajectories. It will result in $degree_{DAG} = n + 1$.

We suppose that there exists a DAG with $degree_{DAG} = p$ and the WCET for each computation component is C. In the following, We discuss two extreme cases: one using the minimum *degree* and the other using the maximum *degree*.

$\boldsymbol{degree} = 1$

In this case, tasks run in sequence. We can find that the latency is the accumulated WCET of each computation component: $L = pC$. Since m jobs can be executed on m available, we have the control period $T = pC/m$ and the update rate $R = 1/T = m/(pC)$.

$\boldsymbol{degree} = p$

In this case, we assume the use of the highest degree of parallelism, i.e. $degree = degree_{DAG} = p$, and suppose that $m \geq p$.

If $m\%p = 0$, it is equivalent to the execution of a single task on m/p processors. Therefore, we have $L = C$, $T = C/(m/p) = pC/m$, and $R = 1/T = m/(pC)$.

If $m\%p \neq 0$, we first define some integers as $N_1 = m\%p$, $N_2 = N_1 \cdot p$, $N_3 = m - N_2$. After assigning N_1 jobs on the first N_2 processors, the $(N_3 + 1)$-th component in the $(N_1 + 1)$-th job will wait for the next available processor with a delay of δ after the launching N_3-th component. The total accumulated delay from the beginning of the 1st job is $(N_2 + N_3) \cdot \delta$. Ideally, we should have $(N_2 + N_3) \cdot \delta = C$, i.e. let the $(N_3 + 1)$-th component in the $(N_1 + 1)$-th job immediately launch on processor 1 once the 1st component of the 1st job is completed.

We can calculate the value of δ as: $\delta = C/(N_2 + N_3) = C/m$. The latency is $L = C + (p - 1) \cdot \delta = C + (p - 1)C/m$. The control period and the update rate are $T = p \cdot \delta = pC/m$ and $R = 1/T = m/(pC)$. We show in Fig. 5.7 an example, in which $m = 5, degree_{DAG} = degree = 3, C = 10$. The latency is calculated as $L = C + (p - 1)C/m = 10 + 2 \cdot 10/5 = 14$, the control period is $T = pC/m = 3 \cdot 10/5 = 6$, and the update rate is $R = 1/T = 1/6$. All these values are confirmed in the figure.

To ensure real-time compliance, the response time T^{resp} should not exceed the deadline T^{ddl}: $T^{\text{resp}} \leq T^{\text{ddl}}$. As stated at the beginning of this section, the DAG has the highest priority in the system, which indicates that the following relationship exists in the suggested task execution model: $T^{\text{resp}} = L$. According to the analysis of the LOROFO algorithm in section 4.4.1, the deadline is estimated as $T^{\text{ddl}} = (d_{\min} - {v^{\text{ego}}_{\max}}^2/2a_{\max})/v^{\text{ego}}_{\max}$. In the experiment, we will use both predetermined and measured parameter values to confirm this real-time constraint $T^{\text{resp}} \leq T^{\text{ddl}}$.

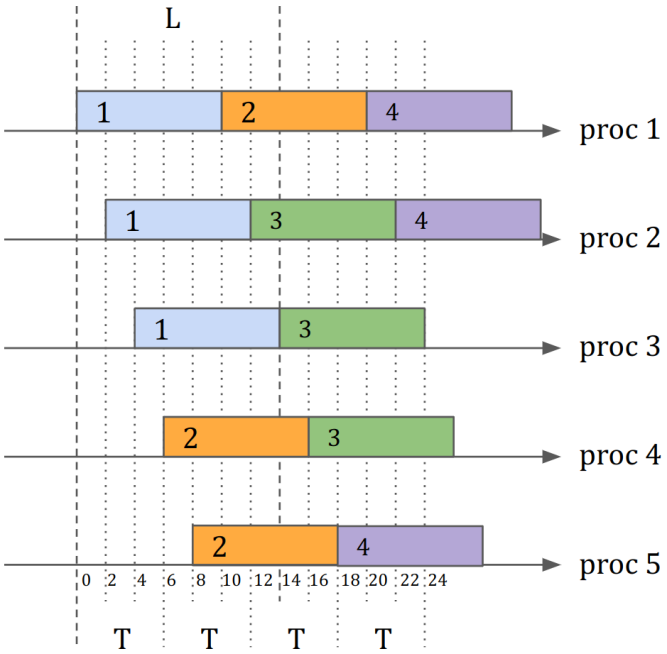

Figure 5.7: An example for $degree_{DAG} = degree = 3$, on 5 processors, $C = 10$. Numbers on rectangles represent the release order of jobs. DAG components belonging to the same job are marked with the same color.

5.2.4 Summary

Table 5.1: Comparison of latency/update rate under different configurations.

degree		L	R
1		$2C$	$m/(2C)$
2	$m\%2 = 0$	C	$m/(2C)$
	$m\%2 = 1$	$C + C/m$	$m/(2C)$

In Table 5.1, we summarize the latency L and the update rate R for the DAG under different configurations. We can find that, in our case, $degree = 2$ is always preferred since it can always deliver a shorter latency with the same control update rate as $degree = 1$. For the extended case that $degree = p$, we observe the same result: with the same number of m, a higher $degree$ is preferred for reducing the latency L.

5.3 Hardware-in-the-Loop simulation

In this section, we test the proposed task execution model in a Hardware-in-the-Loop simulation, in which the computation of the DAG is performed on the realistic embedded device and interacts with a simulation loop on the laptop.

5.3.1 Hardware and software configuration

All nodes in the DAG are implemented as ROS programs in C++ and performed on the Jetson TX2. The onboard operating system is a tailed version of Ubuntu (JetPack SDK 4.6 + Linux 4 Tegra 32.6.1) on which we use the *PREEMPT_RT* patch to enable preemptive behaviors. In order to achieve consistent performance and to maximally benefit from full computation resources, we disable CPU frequency scaling and enter the "performance" mode. The CPU and GPU frequencies are fixed to the maximum values: 2.0 GHz and 1.3 GHz, respectively. Linux system calls `sched_setscheduler`, `sched_setaffinity`, `pthread_setschedparam` and `pthread_setaffinity_np` are used for setting threads' priority and for assigning CPU cores.

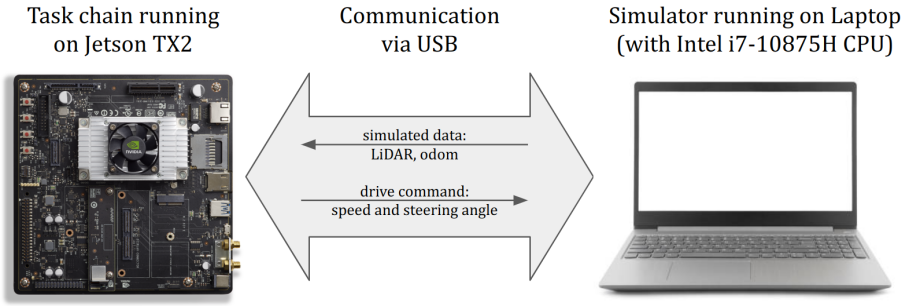

Figure 5.8: Experiment setup with a Hardware-in-the-Loop configuration.

As shown in Fig. 5.8, we run a simulator named f1tenth_gym_ros [114] on the laptop and interacts with nodes in the DAG that run on the Jetson TX2 via a USB cable. As a comparison, in the Hardware-in-the-Loop configuration in section 4.4.3, only the controller runs on Jetson TX2, and it is supplied with exact pose data of both vehicles. In this configuration, all nodes in the DAG run on Jetson TX2. The last control command and the last pose estimation of the ego vehicle, together with LiDAR data, are provided to these nodes for self-localization and opponent detection. To assess the online execution capability of the controller, this experiment is carried on in a simultaneous manner, where the calculation of tasks and the vehicle's dynamics evolution are performed concurrently. It is different from experiments in Chapter 3 and 4, which run in an alternate manner for isolating the effects of the controller from other factors

For the Particle Filter, we use the following configurations: particle number - 1000; motion dispersion in $x/y/\psi$ - $0.1[m]/0.1[m]/0.25[rad]$.

5.3.2 WCET measurement

We first measure the WCET of each node in the DAG over 100000 randomly generated samples on one single CPU core. The horizon length for the NMPC controller is selected as $N = 10$. An example of the execution time histogram is shown in Fig. 5.9 and the whole testing result is listed in Table 5.2. We notice a slightly different behavior on two available types of CPU on Jetson TX2: NVIDIA DENVER2 CPU

Table 5.2: Execution time of different nodes (in [ms]).

		OD	PF	NMPC		
				LO per iter	RO per iter	FW per iter
ARM A57	max	0.624	7.774	7.333	7.380	7.322
	99%*	0.399	6.612	2.928	2.929	2.822
	mean	0.331	5.643	1.710	1.694	1.688
NVIDIA DENVER2	max	1.732	13.879	11.270	11.159	11.298
	99%*	0.546	6.019	3.651	3.578	3.493
	mean	0.374	4.691	1.770	1.694	1.758

*99% means the value at the 99th percentile in the distribution

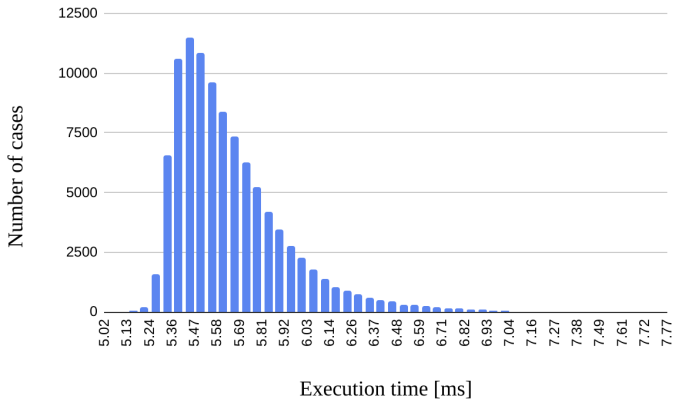

Figure 5.9: Histogram of particle filter's execution time for all 100000 samples (on an ARM A57 CPU of Jetson TX2)

core has better average performance than ARM A57 CPU core for the node v_{PF} while slightly worse performance for other nodes; the observed maximum execution time on DENVER2 CPU core is, in general, higher than A57 CPU core.

In the following experiment, we decide to deploy nodes in the DAG to ARM A57 CPU (featuring 4 cores in total) to make use of its computation efficiency. For the NMPC-related nodes, we use the 99th percentile value in the distribution to define an optimistic WCET for allocating resources in a way that maximizes efficiency and minimizes waste. In case the actual calculation time exceeds this optimistic WCET, the result from the previous calculation cycle will be reused. For the convenience of implementation, we round up the WCETs to integers in milliseconds. We have $c_{\text{OD}} = 1ms, c_{\text{PF}} = 8ms$. The maximum iteration number for QP sub-problems is set to $N_{\text{iter}}^{\max} = 10$, thus we have $c_{\text{NMPC LO or RO or FW}} = 3ms \cdot N_{\text{iter}}^{\max} = 30ms$. Nodes v_{FO} and v_{CENTRAL} are implemented with deterministic codes with an execution time of much less than $1ms$. We let $c_{\text{FO}} = c_{\text{CENTRAL}} = 0ms$.

The assumption made in section 5.2, $c_{\text{FO}} + c_{\text{CENTRAL}} \ll c_{\text{OD}} + c_{\text{PF}} < c_{\text{LO}} = c_{\text{RO}}$, is confirmed: $0 + 0 \ll 1 + 8 < 30$. The WCET C defined in section 5.2 can be calculated as $C = c_{\text{OD}} + c_{\text{PF}} + c_{\text{LO or RO}} + c_{\text{FO}} + c_{\text{CENTRAL}} = 1 + 8 + 30 + 0 + 0 =$

$39ms$. It is necessary to confirm the real-time constraint established in section 5.2.3: $T^{\text{resp}} \leq T^{\text{ddl}}$. We compute it with measured value of C and the assumption $m \geq p$: $T^{\text{resp}} = C + (p-1)C/m \leq 2C = 78[ms]$. According to the calculation in section 4.4.3, $T^{\text{ddl}} = 165[ms]$. The real-time requirement $T^{\text{resp}} \leq T^{\text{ddl}}$ is thus satisfied.

5.3.3 Implementation in Robot Operating System (ROS)

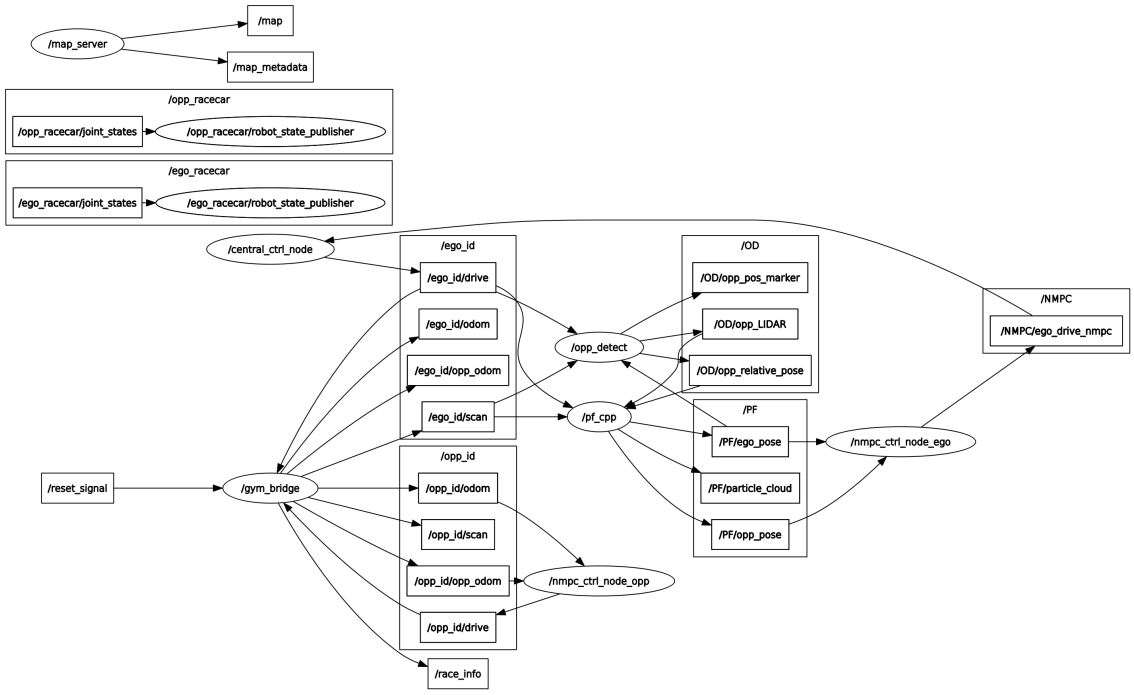

Figure 5.10: Visualization of ROS nodes/topics for the proposed generic architecture. Ovals: nodes. Boxes: topics. Arrows: data flow.

We present in this subsection how nodes in the DAG are implemented in Robot Operating System (ROS) and what data dependencies exist between them. As shown in Fig. 5.10, the simulator runs as the ROS node `gym_bridge` on the laptop to provide sensing information (odometry, LiDAR scan) and to collect the driving command from both vehicles. The implementation can be found in a code repository[1].

For the ego vehicle, the first node in the DAG is implemented as `opp_detect`, using LiDAR scan/the last driving command/the last pose estimation of the ego vehicle as input, and generating opponent-related information as output. The second node is implemented as `pf_cpp`. It takes LiDAR-related information and the last driving command as input and infers the pose of both vehicles as output. The node `nmpc_ctrl_node_ego` serves as the controller, which takes the vehicles' pose information as input to compute an optimal driving command. The node

[1] https://github.com/nanli42/ROS_impl_F1tenth

`central_ctrl_node` collects the driving command from the ego vehicle and sends it back to the simulator. These nodes are all performed on Jetson TX2.

For the opponent vehicle, the node `nmpc_ctrl_node_opp` runs on the host laptop as the controller. We suppose that it has direct access to the pose information of both vehicles.

Once the number of available processors m is known, we can calculate the control period T according to the selected parallelism degree. The ROS Timer is used for periodically launching DAG jobs on available processors with the time interval T.

5.3.4 Impact of parameters L and R

As discussed in section 5.2, a shorter latency L allows us to use more recent data from the environment, and a higher control update rate R enables us to react more quickly to the surroundings. We use a representative simulation example to demonstrate this property. In this simulation, the ego vehicle starts with the pose of $(x[m], y[m], \varphi[rad]) = (0, -1, 0)$; the opponent vehicle starts at $(6, 0, 0)$.

We decide to compare how the ego vehicle behaves under two configurations with a large difference in L and R: configuration 1 - with $degree = 1$ and $m = 1$, $L = T = 2C = 78[ms], R = 1/T = 12.8[Hz]$, which is actually equivalent to the case without scheduling; configuration 2 - with $degree = degree_{DAG} = 2$ and $m = 4$, $L = C = 39[ms], T = C/2 = 19.5[ms], R = 1/T = 51.2[Hz]$.

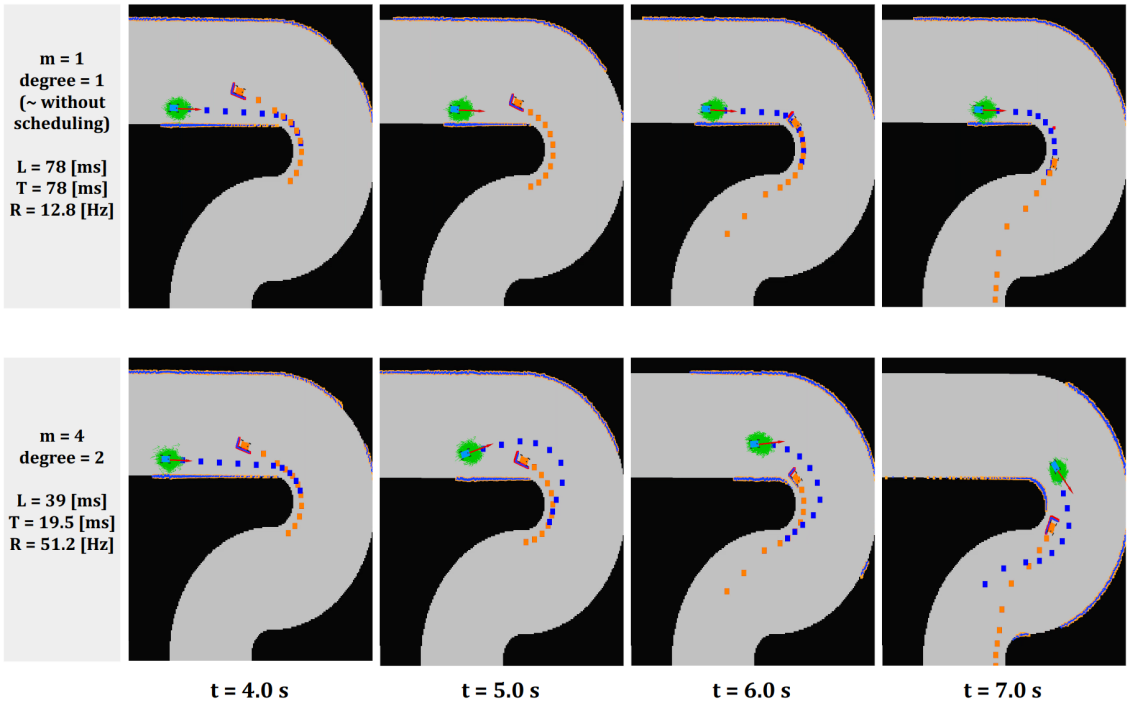

Figure 5.11: Screenshots of vehicles' pose at different time instants. Blue/orange rectangle: ego/opponent vehicle. Dot lines: planned trajectories. Green area: the collection of a large number of arrows representing the pose of each particle in PF.

As seen in Fig. 5.11, since the opponent vehicle is always ahead of the ego vehicle in the 4 given time instants, it is unaffected by the ego vehicle and has exactly the same trajectory in both configurations. Until $t = 4.0s$, the ego vehicle behaves the same in both configurations. Around $t = 5.0s$, with configuration 1, the ego vehicle keeps trying to right-side overtake (RO) the opponent vehicle and finally be "blocked". It turns to the following mode (FO) and recovers to use the NMPC controller later. Also around $t = 5.0s$, with configuration 2, the ego vehicle reacts more frequently to the opponent vehicle and realizes in advance that RO is infeasible. It has time to switch to left-side overtaking (LO) and finally avoids getting "blocked". By comparing the ego vehicle's location at time instant $t = 7.0s$, we can find that shorter L and higher R are advantageous.

5.4 Conclusion

In this chapter, we first studied a generic system architecture for autonomous racing problems, with a focus on software components and hardware platforms. A simple but efficient opponent detection algorithm was proposed. The standard particle filter algorithm was slightly modified to adapt to the presence of the dynamic opponent vehicle. The control algorithm was also adjusted to compensate for the latency. A DAG was set up to represent software components and the data dependencies between them. Overall, the proposed generic system architecture is representative of the various components and considerations involved in autonomous racing problems. While simple in design, it is also clear in structure and generic enough to be adaptable to numerous racing scenarii. This architecture can thus serve as a reference for researchers and engineers to build upon and develop other similar autonomous systems.

We addressed the challenge of efficiently mapping and scheduling tasks in a DAG onto hardware platforms for autonomous racing systems. The DAG has multiple degrees of parallelism, and the highest degree that may be used depends on the number of processors available. We came up with a task execution model to deal with different configurations for maximizing the utilization of parallelization. The real-time constraint $T^{\text{resp}} \leq T^{\text{ddl}}$ was confirmed to be satisfied in this model. We introduced the method for implementing the task execution model within the commonly used ROS framework, which serves as a guide for the development of other similar robotic systems. Finally, we tested it in a realistic Hardware-in-the-Loop simulation and demonstrated that, by exploring different parallelization configurations in the task execution model, the racing problem benefits from lower latency and higher control update rate.

CHAPTER 6

Conclusion and perspectives

6.1 Conclusion

In this dissertation, we presented time-optimal NMPC-based control approaches for the autonomous racing problem in both single-vehicle racing mode and head-to-head racing mode.

In Chapters 1 and 2, we reviewed the development of control systems in the field of autonomous passenger cars, based on which we discussed state-of-the-art control techniques in the emerging field of autonomous race cars. For this study, we selected NMPC as the base controller based on the following considerations. First of all, we can directly set the vehicle's progress time as the optimization objective function of NMPC, as well as explicitly handle the vehicle's dynamics and other potentially nonlinear constraints, which suits our needs for the racing scene. Furthermore, research results on NMPC, as a typical representative of optimization-based controllers, can serve as a reference for other similar control schemes. Finally, NMPC is extendable. For example, it can be readily integrated with MIP to solve the decision-making problem. In Chapter 2, we also discussed relevant tools and techniques, including the modeling of racetracks, the characterization of system dynamics using the ODE equation, and different approaches for solving NMPC, etc.

In Chapter 3, we investigated an issue caused by the high complexity of NMPC with long prediction horizons for the single-vehicle racing mode: in the classical NMPC framework, only the first step of the resulting optimal control series will be deployed; the NMPC recalculation is required for generating a new optimal control series for the next step; however, limited computation resources may prevent the recalculation from being completed in time, i.e. it may not have finished when the next step begins. We thus suggested a triggering-based recalculation method. To prevent the calculations from becoming obsolete, we used the first triggering condition to ensure that the vehicle's progress time is greater than the calculation time, i.e. ensuring the real-time execution of NMPC. We achieved this by reusing inter-

mediate NMPC calculation results for the entire prediction horizon, which could degrade lap time performance. To compensate for this effect, we introduced the second triggering condition to detect significant changes in track curvature that may cause performance degradation and that justifies the necessity of recalculation. In fact, the first triggering condition is related to the given time budget for guaranteeing the real-time feasibility of NMPC. The second triggering condition aims to minimize the use of computation resources while still achieving a satisfactory lap time performance. We gained inspiration from the behaviors of experienced human race car drivers: they make significant modifications to the vehicle's intended trajectory planning only when the vehicle moves into a track segment of a different type (e.g. from a straight segment into a curve, or in reverse). The intuition is that the system's computing burden can be lessened by reducing the number of NMPC recalculations, that is, only performing one calculation when necessary and reusing the result for multiple steps.

In Chapter 4, we explored techniques for the head-to-head racing mode, particularly those based on the time-optimal NMPC to produce optimal results, for encoding and producing safe and efficient overtaking maneuvers of race cars. One challenge we encountered was the necessity to depict the vehicle's footprint in an easy-to-handle form. We characterized the vehicle's footprint in the curvilinear coordinate system in the form of simple intervals and established the collision avoidance constraints. Another challenge was how to encode decisions at each prediction step and how to solve the resulting decision-making problem. We proposed an approach to represent the overtaking behavior as a series of integers using MIP. While this MIP-based method is effective in simulation, it is challenging to deploy in real-time embedded systems due to the performance limitations of current MIP solvers. We offered an alternative approach, i.e. LOROFO (Left-side Overtaking, Right-side Overtaking, and FOllowing) algorithm, working with a deterministic control period and a short prediction horizon, to reduce the complexity of the decision-making problem by considering the nature of overtaking behaviors: in a short term, overtaking typically occurs in successive steps and only on one side (either left or right). It is simple to set up and run online. It can also be effectively parallelized thanks to the presence of parallelizable components.

In order to enable the online execution of NMPC-based controllers with limited computation resources, as stated above, we proposed methods in Chapters 3 and 4 for reducing the computational workload, either by skipping recalculation steps with a triggering-based method in single-vehicle racing mode or using short prediction horizon and reduced decision-making series in head-to-head racing mode. However, when working on multi-core systems, we should make efficient use of abundant computation resources and explore parallelization to decrease end-to-end latency (i.e. allow us to use more recent data from the environment), improve the control update rate (i.e. enable us to react more quickly to the surroundings), and ultimately improve system performance. Therefore, we investigated in Chapter 5 how to build a task execution model to efficiently schedule tasks on real-time embedded systems with multi-cores. We developed a generic system architecture in the form of DAG that consists of modules for control (LOROFO method) and perception (opponent

detection and self-localization). This DAG has various degrees of parallelism and can be mapped onto hardware platforms using different configurations in the proposed task execution model. To meet the real-time requirement, the task execution model allocates time budgets to each DAG node based on their WCET. We tested the task execution model in a Hardware-in-the-Loop simulation, which demonstrated that an effective task execution model can optimize the utilization of parallelization on systems with multi-cores. This work can serve as a reference for modeling tasks and improving performance in similar autonomous systems.

6.2 Perspectives

There are several possible extensions for future work. Some of them concern enhancing the solver now in use, while others involve extending the use cases.

The solver performance of NMPC and MIP is one of the bottlenecks restricting them from operating to their full potential on real-time embedded systems. Although this thesis provided methods for realizing the online execution of NMPC-based control algorithms, faster solvers might enable us to employ even longer prediction horizons and allocate computation resources more flexibly. Given that many modern embedded systems are equipped with multi-core CPUs and GPUs, it would be instructive to investigate how solvers can benefit from working on such systems to help reduce computation time. In existing works, authors in [128] exploited the massive parallelism of GPUs to accelerate calculations of large-scale QPs (which can serve as a sub-problem solver for NMPC). Authors in [129] studied the potential of running branch-and-bound algorithms in parallel on GPUs for accelerating the calculation of MIP.

This thesis assumed two research scenarii: single-vehicle racing mode and head-to-head racing mode. It would be interesting to explore the scenario with the presence of $n, n \geq 2$ opponent vehicles. To model overtaking maneuvers of the ego vehicle in this case, the MIP-based method can be naturally extended by encoding trajectories of multiple opponent vehicles. However, it will become increasingly difficult to prevent the computation complexity from exploding as the number of opponent vehicles increases. In such a situation, the LOROFO algorithm should also be modified to take into account at most $n+1$ potential homotopic trajectories rather than only two-side overtaking trajectories. The resulting task chain will have a higher degree of parallelism, which the task execution model will need to take into account. In this dissertation, we made the assumption that the opponent vehicle would not actively obstruct or interfere with the ego vehicle. However, further investigation into active interactions between vehicles could be valuable for understanding realistic racing strategies. One promising research path is to take inspiration from game theory, as previously explored in works such as [130] and [131].

Another limitation of our present work is the definition of time budget. We used the WCET to allocate the time budget when building the task execution model. However, as we observed in experiments, the actual execution time is usually much shorter than the estimated WCET. One potential improvement is to assign differ-

ent time budgets according to the criticality level of tasks. This kind of system is called a Mixed-Criticality System [132], which guarantees that high criticality level components always fulfill their real-time requirements, while low criticality level components can have an adequate quality of service. For instance, the DAG discussed in this dissertation is on a high criticality level since its failure would result in catastrophic consequences for the vehicle's safety, while tasks such as data logging, diagnostic information transmitting, and non-essential sensor processing are considered on a lower criticality level. The WCET of high-criticality tasks can be set as different values for different scenarii, e.g. a lower value in a normal driving scene to conserve computation resources, and a higher value in an emergency braking scene to enable robust and reliable control. Since the consequences of missing deadlines are less severe for low-criticality tasks, in some cases, it may be appropriate to set their time budget as the average or expected execution time rather than a conservative WCET to free up resources for tasks with higher criticality levels.

APPENDIX A

Introduction (en Français)

A.1 Motivation et contexte

La recherche d'algorithmes de contrôle sophistiqués suscite un intérêt considérable depuis de nombreuses années. De nombreux efforts ont été déployés pour améliorer la qualité de service des systèmes autonomes. Malgré des progrès notables, il est encore nécessaire de poursuivre l'étude de leur application en temps réel. Cette thèse se concentre sur le contrôle en temps réel pour la conduite autonome et plus particulièrement la course de véhicules autonomes.

La conduite autonome est classée par la Société des ingénieurs automobiles [1] du niveau 0 (aucune automatisation de la conduite), au niveau 1 (assistance au conducteur), jusqu'au niveau 5 (automatisation complète de la conduite dans toutes les circonstances). Les systèmes avancés d'aide à la conduite [2] (Advanced Driver-Assistance Systems, ADAS, y compris le contrôle électronique de la stabilité, le contrôle de la stabilité en roulis, l'alerte de franchissement de ligne, le régulateur de vitesse adaptatif, les systèmes d'aide au stationnement, etc.) fournissent des fonctions de niveau 1, qui aident les conducteurs humains à conduire de manière sûre, efficace et avec une meilleure expérience utilisateur. La recherche sur l'ADAS se poursuit depuis des décennies. Le DARPA Urban Challenge [3] est une étape importante qui a considérablement fait évoluer le domaine de la conduite autonome. Depuis lors, des recherches connexes ont été massivement menées dans les milieux académiques et industriels. Bien que plusieurs entreprises aient mis sur le marché des systèmes de conduite autonome présentant des niveaux élevés d'automatisation, par exemple le système FSD (Full Self-Driving) de Tesla [4], la conduite de niveau 5 ne sera pas accessible au grand public avant une dizaine d'années, en raison de problèmes de réglementation et d'éthique, ainsi que de plusieurs difficultés techniques : l'incapacité à gérer des situations extrêmes inattendues, l'absence de compréhension sophistiquée des comportements implicites des différents acteurs impliqués dans le trafic routier, etc.

La course de véhicules autonomes [5] est une forme particulière de conduite autonome, dans laquelle le véhicule de course doit parcourir un tour de circuit le plus rapidement possible tout en évitant les collisions potentielles avec les limites de la piste ou les véhicules adverses. Comme le véhicule de course se déplace généralement à une vitesse élevée et nécessite des capacités de manipulation extrêmes (dérive active, direction précise, etc.), nous devons étudier les performances du système à ses limites physiques. Ces recherches sur les cas limites peuvent également contribuer à faire avancer l'étude de la conduite autonome des véhicules routiers [6]. D'autres domaines pertinents peuvent également bénéficier de l'étude de la course de véhicules autonomes. Par exemple, le problème de contrôle en temps optimal d'un véhicule de course, qui est soumise à des contraintes de saturation du moteur, de traction et qui suit une trajectoire contrainte, est analogue au problème de contrôle en temps optimal d'un bras robotisé qui se déplace sur une trajectoire sous contrainte de saturation de l'actionneur. Dans les deux cas, afin de réaliser les comportements désirés, le système de contrôle prend en compte les contraintes et les objectifs dans une structure de problème similaire. Du point de vue du monde académique, la course autonome est une plateforme de recherche idéale pour l'étude des systèmes cyber-physiques (Cyber-Physical Systems, CPS) [7], où les installations physiques (c'est-à-dire les véhicules de course) ont des dynamiques de système qui changent rapidement, et où le calcul est typiquement effectué sur des systèmes embarqués.

Le contrôle prédictif à base de modèle (Model Predictive Control, MPC) est une technique de contrôle de pointe pour commander une installation physique dans le cadre d'un horizon fuyant afin d'atteindre un objectif spécifié tout en satisfaisant un ensemble de contraintes. Parmi les différentes techniques de contrôle pour le problème de la course de véhicules autonomes, nous sommes particulièrement intéressés par le MPC pour les raisons ci-dessous :

1. Le MPC est l'une des méthodes à base de modèles les plus prometteuses pour prendre en compte des modèles dynamiques physiques précis et des contraintes complexes. Il est logique d'appliquer le MPC aux problèmes de course puisque le MPC peut intégrer efficacement nos connaissances préalables sur le véhicule de course et sur les contraintes physiques.

2. MPC est un représentant typique des contrôleurs basés sur l'optimisation qui, en général, fournissent des résultats optimaux au prix d'une complexité élevée. En étudiant le MPC sur un véhicule de course équipé d'unités de calcul embarquées, nous pouvons mieux comprendre comment concevoir et déployer efficacement des algorithmes basés sur l'optimisation de dispositifs aux ressources de calcul limitées.

3. MPC est extensible pour fonctionner en conjonction avec d'autres techniques. Par exemple, il peut coopérer avec la programmation mixte en nombres entiers (Mixed Integer Programming, MIP) pour les problèmes de prise de décision. Dans la littérature, il est combiné avec plusieurs autres outils pour renforcer ses capacités et améliorer ses performances, par exemple la théorie des jeux [8] pour permettre au MPC de réagir aux actions des adversaires, la théorie

de la viabilité [9] pour assurer la faisabilité récursive du MPC, la théorie de l'opérateur de Koopman [10] pour faciliter l'identification du système d'un véhicule dont la dynamique est inconnue dans le MPC, etc.

Dans cette étude, pour capturer précisément la dynamique du véhicule, formalisée à l'aide d'équations différentielles ordinaires (Ordinary Differential Equations, ODE), nous la représentons sous la forme de contraintes d'égalité non linéaires. Dans certaines circonstances, les contraintes d'inégalité dans le MPC, telles que les contraintes d'évitement de collision, peuvent également être non linéaires. Ce travail se concentre donc sur le MPC non linéaire (Nonlinear MPC, NMPC).

Le NMPC doit fonctionner *correctement* dans deux sens. D'une part, son horizon de prédiction doit être suffisamment long pour fournir un contrôle fiable. D'autre part, un NMPC avec un horizon de prédiction plus long nécessite plus de ressources de calcul. Il risque de produire un résultat obsolète, c'est-à-dire un résultat qui arrive après la date limite et qui est considéré comme inutile et incorrect. Étant donné que les comportements déterministes sont souhaitables dans les systèmes temps réel, le pire cas de temps d'exécution (Worst Case Execution Time, WCET) du NMPC doit être limité à une valeur raisonnable compte tenu des ressources limitées des systèmes embarqués.

Les contrôleurs, dont le NMPC, collaborent avec d'autres composants logiciels, tels que les modules de perception pour la localisation et la détection des adversaires. Ils fonctionnent comme une chaîne de tâches et partagent les ressources de calcul. Différents composants de calcul peuvent être parallélisés, et les processeurs modernes, tels que les CPU multi-cœurs et les GPU, ont la capacité de paralléliser ces tâches. Il est donc crucial de construire un modèle d'exécution des tâches afin de profiter au maximum des capacités de parallélisation du processeur pour diminuer la latence et augmenter le taux de rafraîchissement des commandes, ce qui améliore les performances du contrôleur et nous prémunit des comportements non déterministes.

A.2 Champ d'étude de la thèse

Dans cette section, nous définissons le champs d'étude de cette thèse. Ces travaux se limitent à la course de véhicules autonomes dans 2 modes : le mode de course à véhicule unique (similaire aux séances de qualification de Formule 1, dans lequel un seul véhicule de course vise à réaliser le meilleur temps de tour de circuit) et le mode de course un-contre-un (dans lequel deux véhicules de course s'affrontent sur le même circuit pour être le premier à franchir la ligne d'arrivée).

Dans les chapitres 3 et 4, qui couvrent la conception de l'algorithme du contrôleur, nous supposons que le contrôleur du véhicule peut accéder à son vecteur d'état (incluant la position) exact comme entrée de contrôle, ainsi qu'à la position et à la trajectoire prévue du véhicule adverse. Les deux véhicules sont supposés avoir une connaissance préalable de la courbure de la piste de course.

Dans le chapitre 5, qui concerne la conception du modèle d'exécution des tâches au niveau du système, nous supposons que des données LiDAR servent d'entrée de

contrôle au véhicule. Des modules de perception sont nécessaires pour la localisation du véhicule et la détection du véhicule adverse. De plus, la trajectoire prévue du véhicule adverse n'est pas fournie. L'expérience est réalisée dans une simulation « Hardware-in-the-Loop » (HiL), c'est-à-dire que les composants logiciels du véhicule sont exécutés sur un dispositif embarqué tandis que les autres composants de simulation sont effectués sur un ordinateur portable standard.

Dans cette thèse, nous traiterons les problèmes ci-dessous.

- Le NMPC nécessite généralement un temps de calcul relativement long, surtout lorsque l'horizon de prédiction est lointain. En revanche, le NMPC à horizon lointain donne généralement de meilleurs résultats. Nous cherchons à savoir comment rendre le NMPC à horizon lointain réalisable dans un budget de temps déterminé tout en réduisant les coûts de calcul et en ayant des performances satisfaisantes.

- La prise de décision, par exemple la décision de dépassement, est un élément crucial du problème de la course. La combinaison des techniques de prise de décision avec les cadres de contrôle existants reste une question ouverte. Nous discuterons de comment formuler le problème de dépassement sous la forme d'un MIP et de comment créer un contrôleur de dépassement simplifié mais réalisable en ligne.

- Un modèle d'exécution des tâches doit être correctement construit pour réduire la latence, augmenter le taux de rafraîchissement du contrôle et garantir que les tâches sont achevées avant leur échéance. Nous prenons en compte le parallélisme de la tâche et la capacité de parallélisation du processeur lors de la construction de ce modèle. Cela permet d'améliorer les performances et la fiabilité du contrôle.

Il existe encore des sujets intéressants qui ne sont pas inclus dans le champ de cette étude. Par exemple, la prise en compte de l'incertitude (qui peut provenir soit de l'environnement, soit de l'inadéquation du modèle de la dynamique du système) est significative pour le déploiement d'algorithmes sur des véhicules fonctionnant dans des circonstances imparfaites.

A.3 Plan de la thèse

La thèse est organisée de la manière décrite ci-dessous. Nous introduisons d'abord le contexte et l'état de l'art du problème de la course autonome. Les problèmes mentionnés dans la section 1.2 sont ensuite traités respectivement dans les différents chapitres.

- Chapitre 2, Autonomous race car control: state of the art, couvre les techniques de contrôle pour le problème de la course autonome. Le modèle de piste de course et la dynamique du véhicule sont discutés. Les principes du MPC et les techniques de résolution associées sont présentés. Nous introduisons le NMPC

à temps optimal qui sert de base aux chapitres suivants. Une discussion du problème de la course à un-contre-un donne un aperçu des problèmes de prise de décision dans le scénario de la course.

- Le chapitre 3, Autonomous racing in single-vehicle mode, se concentre sur la complexité du NMPC et présente le problème de temps de calcul élevé en mode course à véhicule unique. Nous proposons une solution basée sur le déclenchement pour satisfaire la contrainte du budget de temps et réduire la charge de calcul tout en maintenant des performances adéquates en termes de temps de tour de circuit.

  Contribution : La méthode proposée utilise deux conditions de déclenchement. L'une est associée à la contrainte du budget de temps et permet l'exécution du NMPC en temps réel. L'autre est inspirée du comportement des conducteurs humains en course. Elle cherche à allouer les charges de calcul en fonction de l'évolution de l'environnement, ce qui peut être considéré comme un modèle de calcul à la demande.

- Le chapitre 4, Autonomous racing in head-to-head mode, se concentre sur le problème de la course autonome en mode de course à un-contre-un. Nous proposons d'encoder les contraintes d'évitement des collisions sous la forme de MIP pour prendre des décisions de dépassement dans le cadre du NMPC dans le système de coordonnées curvilignes. Basé sur les observations de l'expérimentation avec la méthode MIP, une stratégie de dépassement alternative est proposée pour permettre l'exécution en ligne.

  Contribution : Nous fournissons une technique pour représenter la géométrie du véhicule, qui peut être utilisée pour établir des contraintes d'évitement de collision en coordonnées curvilignes. La combinaison de NMPC et de MIP fournit un moyen efficace de résoudre les problèmes de prise de décision dans le problème de la course autonome. Comme méthode alternative à MIP, l'algorithme LOROFO (Left-side Overtaking, Right-side Overtaking, and FOllowing) tente efficacement de prendre des décisions de dépassement groupées. Cet algorithme est réalisable en ligne et sert d'exemple pour montrer comment réduire la complexité des problèmes de prise de décision.

- Le chapitre 5, Task execution model for autonomous racing systems, présente tout d'abord une architecture système générique pour le véhicule de course autonome, y compris les composants logiciels, la chaîne de tâches et les plateformes matérielles. Un modèle d'exécution des tâches est ensuite proposé pour affecter les composants logiciels aux processeurs disponibles avec différents degrés de parallélisme. Ce modèle d'exécution des tâches vise à réduire la latence et à améliorer le taux de rafraîchissement des commandes. Une simulation HiL est réalisée pour démontrer comment le modèle d'exécution des tâches est mis en œuvre dans le système d'exploitation du robot (Robot Operating System, ROS) et comment il fonctionne.

Contribution : Dans la simulation, le modèle d'exécution des tâches est confirmé comme améliorant efficacement les performances du système dans le scénario de course. Il fournit un aperçu pour la construction de modèles d'exécution de tâches dans d'autres systèmes autonomes similaires.

Bibliography

[1] SAE International: On-Road Automated Driving (ORAD) committee. *Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles*, apr 2021.

[2] Adnan Shaout, Dominic Colella, and Selim Awad. Advanced driver assistance systems-past, present and future. In *2011 Seventh International Computer Engineering Conference (ICENCO'2011)*, pages 72–82. IEEE, 2011.

[3] Martin Buehler, Karl Iagnemma, and Sanjiv Singh. *The DARPA urban challenge: autonomous vehicles in city traffic*, volume 56. springer, 2009.

[4] Emil Talpes, Debjit Das Sarma, Ganesh Venkataramanan, Peter Bannon, Bill McGee, Benjamin Floering, Ankit Jalote, Christopher Hsiong, Sahil Arora, Atchyuth Gorti, et al. Compute solution for tesla's full self-driving computer. *IEEE Micro*, 40(2):25–35, 2020.

[5] Johannes Betz, Hongrui Zheng, Alexander Liniger, Ugo Rosolia, Phillip Karle, Madhur Behl, Venkat Krovi, and Rahul Mangharam. Autonomous vehicles on the edge: A survey on autonomous vehicle racing. *IEEE Open Journal of Intelligent Transportation Systems*, 2022.

[6] Johannes Betz, Alexander Wischnewski, Alexander Heilmeier, Felix Nobis, Tim Stahl, Leonhard Hermansdorfer, Boris Lohmann, and Markus Lienkamp. What can we learn from autonomous level-5 motorsport? In *9th International Munich Chassis Symposium 2018*, pages 123–146. Springer, 2019.

[7] Wayne Wolf. Cyber-physical systems. *Computer*, 42(03):88–89, 2009.

[8] Chanyoung Jung, Seungwook Lee, Hyunki Seong, Andrea Finazzi, and David Hyunchul Shim. Game-theoretic model predictive control with data-driven identification of vehicle model for head-to-head autonomous racing. *arXiv preprint arXiv:2106.04094*, 2021.

[9] Alexander Liniger and John Lygeros. Real-time control for autonomous racing based on viability theory. *IEEE Transactions on Control Systems Technology*, 27(2):464–478, 2017.

[10] Rongyao Wang, Yiqiang Han, and Umesh Vaidya. Deep koopman data-driven control framework for autonomous racing. In *Proc. Int. Conf. Robot. Autom.(ICRA) Workshop Opportunities Challenges Auton. Racing*, pages 1–6, 2021.

[11] Brian Paden, Michal Čáp, Sze Zheng Yong, Dmitry Yershov, and Emilio Frazzoli. A survey of motion planning and control techniques for self-driving urban vehicles. *IEEE Transactions on intelligent vehicles*, 1(1):33–55, 2016.

[12] Edsger W Dijkstra et al. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

[13] Nils J Nilsson. A mobile automaton: An application of artificial intelligence techniques. Technical report, Sri International Menlo Park Ca Artificial Intelligence Center, 1969.

[14] Steven M LaValle et al. Rapidly-exploring random trees: A new tool for path planning. Technical report, Iowa State University, 1998.

[15] Sertac Karaman and Emilio Frazzoli. Optimal kinodynamic motion planning using incremental sampling-based methods. In *49th IEEE conference on decision and control (CDC)*, pages 7681–7687. IEEE, 2010.

[16] Richard S Wallace, Anthony Stentz, Charles E Thorpe, Hans P Moravec, William Whittaker, and Takeo Kanade. First results in robot road-following. In *IJCAI*, pages 1089–1095, 1985.

[17] Brigitte d'Andréa Novel, Guy Campion, and Georges Bastin. Control of nonholonomic wheeled mobile robots by state feedback linearization. *The International journal of robotics research*, 14(6):543–559, 1995.

[18] Alessandro Rucco, Giuseppe Notarstefano, and John Hauser. Computing minimum lap-time trajectories for a single-track car with load transfer. In *2012 IEEE 51st IEEE Conference on Decision and Control (CDC)*, pages 6321–6326. IEEE, 2012.

[19] Luigi Cardamone, Daniele Loiacono, Pier Luca Lanzi, and Alessandro Pietro Bardelli. Searching for the optimal racing line using genetic algorithms. In *Proceedings of the 2010 IEEE Conference on Computational Intelligence and Games*, pages 388–394. IEEE, 2010.

[20] Maurizio Bevilacqua, Antonios Tsourdos, and Andrew Starr. Particle swarm for path planning in a racing circuit simulation. In *2017 IEEE International Instrumentation and Measurement Technology Conference (I2MTC)*, pages 1–6. IEEE, 2017.

[21] PE Wolfgang Kuhn. Methodology for the numerical calculation of racing lines and the virtual assessment of driving behavior for training circuits for the automobile industry. *Transportation research procedia*, 25:1416–1429, 2017.

[22] Paul A Theodosis and J Christian Gerdes. Generating a racing line for an autonomous racecar using professional driving techniques. In *Dynamic Systems and Control Conference*, volume 54761, pages 853–860, 2011.

[23] Francesco Braghin, Federico Cheli, Stefano Melzi, and Edoardo Sabbioni. Race driver model. *Computers & Structures*, 86(13-14):1503–1516, 2008.

[24] Daniel Patrick Kelly. *Lap time simulation with transient vehicle and tyre dynamics*. PhD thesis, Cranfield University Cranfield, 2008.

[25] Efstathios Velenis and Panagiotis Tsiotras. Minimum time vs maximum exit velocity path optimization during cornering. In *2005 IEEE international symposium on industrial electronics*, pages 355–360. Citeseer, 2005.

[26] JPM Hendrikx, TJJ Meijlink, and RFC Kriens. Application of optimal control theory to inverse simulation of car handling. *Vehicle System Dynamics*, 26(6):449–461, 1996.

[27] Daniele Casanova, Robin S Sharp, and Pat Symonds. Minimum time manoeuvring: The significance of yaw inertia. *Vehicle system dynamics*, 34(2):77–115, 2000.

[28] Grady Williams, Paul Drews, Brian Goldfain, James M Rehg, and Evangelos A Theodorou. Aggressive driving with model predictive path integral control. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1433–1440. IEEE, 2016.

[29] Gennaro Notomista, Mingyu Wang, Mac Schwager, and Magnus Egerstedt. Enhancing game-theoretic autonomous car racing using control barrier functions. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5393–5399. IEEE, 2020.

[30] Wilko Schwarting, Alyssa Pierson, Sertac Karaman, and Daniela Rus. Stochastic dynamic games in belief space. *IEEE Transactions on Robotics*, 37(6):2157–2172, 2021.

[31] Joseph Funke, Matthew Brown, Stephen M Erlien, and J Christian Gerdes. Collision avoidance and stabilization for autonomous vehicles in emergency scenarios. *IEEE Transactions on Control Systems Technology*, 25(4):1204–1216, 2016.

[32] John K Subosits and J Christian Gerdes. From the racetrack to the road: Real-time trajectory replanning for autonomous driving. *IEEE Transactions on Intelligent Vehicles*, 4(2):309–320, 2019.

[33] Tim Stahl, Alexander Wischnewski, Johannes Betz, and Markus Lienkamp. Multilayer graph-based trajectory planning for race vehicles in dynamic scenarios. In *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, pages 3149–3154. IEEE, 2019.

[34] Alexander Liniger, Alexander Domahidi, and Manfred Morari. Optimization-based autonomous racing of 1: 43 scale rc cars. *Optimal Control Applications and Methods*, 36(5):628–647, 2015.

[35] Matthew O'Kelly, Varundev Sukhil, Houssam Abbas, Jack Harkins, Chris Kao, Yash Vardhan Pant, Rahul Mangharam, Dipshil Agarwal, Madhur Behl, Paolo Burgio, et al. F1/10: An open-source autonomous cyber-physical platform. *arXiv preprint arXiv:1901.08567*, 2019.

[36] Aman Sinha, Matthew O'Kelly, Hongrui Zheng, Rahul Mangharam, John Duchi, and Russ Tedrake. Formulazero: Distributionally robust online adaptation via offline population synthesis. In *International Conference on Machine Learning*, pages 8992–9004. PMLR, 2020.

[37] Jeong hwan Jeon, Raghvendra V Cowlagi, Steven C Peters, Sertac Karaman, Emilio Frazzoli, Panagiotis Tsiotras, and Karl Iagnemma. Optimal motion planning with the half-car dynamical model for autonomous high-speed driving. In *2013 American control conference*, pages 188–193. IEEE, 2013.

[38] Oktay Arslan, Karl Berntorp, and Panagiotis Tsiotras. Sampling-based algorithms for optimal motion planning using closed-loop prediction. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4991–4996. IEEE, 2017.

[39] Stefano Feraco, Sara Luciani, Angelo Bonfitto, Nicola Amati, and Andrea Tonoli. A local trajectory planning and control method for autonomous vehicles based on the rrt algorithm. In *2020 AEIT International Conference of Electrical and Electronic Technologies for Automotive (AEIT AUTOMOTIVE)*, pages 1–6. IEEE, 2020.

[40] Jun Ni, Jibin Hu, and Changle Xiang. Envelope control for four-wheel independently actuated autonomous ground vehicle through afs/dyc integrated control. *IEEE Transactions on Vehicular Technology*, 66(11):9712–9726, 2017.

[41] HB Pacejka and IJM Besselink. Magic formula tyre model with transient properties. *Vehicle system dynamics*, 27(S1):234–249, 1997.

[42] Kirstin LR Talvala, Krisada Kritayakirana, and J Christian Gerdes. Pushing the limits: From lanekeeping to autonomous racing. *Annual Reviews in Control*, 35(1):137–148, 2011.

[43] Eugenio Alcalá, Vicenç Puig, Joseba Quevedo, and Ugo Rosolia. Autonomous racing using linear parameter varying-model predictive control (lpv-mpc). *Control Engineering Practice*, 95:104270, 2020.

[44] A Wischnewski, M Euler, S Gümüs, and B Lohmann. Tube model predictive control for an autonomous race car. *Vehicle System Dynamics*, 60(9):3151–3173, 2022.

[45] Jesús Velasco Carrau, Alexander Liniger, Xiaojing Zhang, and John Lygeros. Efficient implementation of randomized mpc for miniature race cars. In *2016 European Control Conference (ECC)*, pages 957–962. IEEE, 2016.

[46] Hilbert J Kappen. Linear theory for control of nonlinear stochastic systems. *Physical review letters*, 95(20):200201, 2005.

[47] Robin Verschueren, Mario Zanon, Rien Quirynen, and Moritz Diehl. Time-optimal race car driving using an online exact hessian based nonlinear mpc algorithm. *ECC 2016*, pages 141–147, 2006.

[48] Alexander Heilmeier, Maximilian Geisslinger, and Johannes Betz. A quasi-steady-state lap time simulation for electrified race cars. In *2019 Fourteenth International Conference on Ecological Vehicles and Renewable Energies (EVER)*, pages 1–10. IEEE, 2019.

[49] Dieter Schramm, Manfred Hiller, and Roberto Bardini. Vehicle dynamics. *Modeling and Simulation. Berlin, Heidelberg*, 151, 2014.

[50] Matthias Althoff and Gerald Würsching. CommonRoad: Vehicle Models, 2020.

[51] Juraj Kabzan, Miguel I Valls, Victor JF Reijgwart, Hubertus FC Hendrikx, Claas Ehmke, Manish Prajapat, Andreas Bühler, Nikhil Gosala, Mehak Gupta, Ramya Sivanesan, et al. Amz driverless: The full autonomous racing system. *Journal of Field Robotics*, 37(7):1267–1294, 2020.

[52] Achin Jain, Matthew O'Kelly, Pratik Chaudhari, and Manfred Morari. Bayesrace: Learning to race autonomously using prior experience. *arXiv preprint arXiv:2005.04755*, 2020.

[53] Robin Verschueren, Stijn De Bruyne, Mario Zanon, Janick V Frasch, and Moritz Diehl. Towards time-optimal race car driving using nonlinear mpc in real-time. In *53rd IEEE conference on decision and control*, pages 2505–2510. IEEE, 2014.

[54] Jacques Richalet, André Rault, JL Testud, and J Papon. Model predictive heuristic control: Applications to industrial processes. *Automatica*, 14(5):413–428, 1978.

[55] S Joe Qin and Thomas A Badgwell. A survey of industrial model predictive control technology. *Control engineering practice*, 11(7):733–764, 2003.

[56] Milan Vukov. Embedded model predictive control and moving horizon estimation for mechatronics applications. *PhD thesis*, 2015.

[57] S Sathiya Keerthi and Elmer G Gilbert. Optimal infinite-horizon feedback laws for a general class of constrained discrete-time systems: Stability and moving-horizon approximations. *Journal of optimization theory and applications*, 57(2):265–293, 1988.

[58] Hong Chen and Frank Allgöwer. A quasi-infinite horizon nonlinear model predictive control scheme with guaranteed stability. *Automatica*, 34(10):1205–1217, 1998.

[59] Mazen Alamir and Guy Bornard. Stability of a truncated infinite constrained receding horizon scheme: the general discrete nonlinear case. *Automatica*, 31(9):1353–1356, 1995.

[60] Lars Grüne, Jürgen Pannek, Martin Seehafer, and Karl Worthmann. Analysis of unconstrained nonlinear mpc schemes with time varying control horizons. *arXiv preprint arXiv:1203.6783*, 2012.

[61] David Angeli, Rishi Amrit, and James B Rawlings. Receding horizon cost optimization for overly constrained nonlinear plants. In *Proceedings of the 48h IEEE Conference on Decision and Control (CDC) held jointly with 2009 28th Chinese Control Conference*, pages 7972–7977. IEEE, 2009.

[62] Matthias A Müller and Frank Allgöwer. Robustness of steady-state optimality in economic model predictive control. In *2012 IEEE 51st IEEE Conference on Decision and Control (CDC)*, pages 1011–1016. IEEE, 2012.

[63] Lars Grüne. Economic receding horizon control without terminal constraints. *Automatica*, 49(3):725–734, 2013.

[64] Lars Grüne and Marleen Stieler. Asymptotic stability and transient optimality of economic mpc without terminal conditions. *Journal of Process Control*, 24(8):1187–1196, 2014.

[65] Moritz Diehl, Hans Joachim Ferreau, and Niels Haverbeke. Efficient numerical methods for nonlinear mpc and moving horizon estimation. In *Nonlinear model predictive control*, pages 391–417. Springer, 2009.

[66] Friedrich Pfeiffer and Rainer Johanni. A concept for manipulator trajectory planning. *IEEE Journal on Robotics and Automation*, 3(2):115–123, 1987.

[67] Yiqi Gao, Andrew Gray, Janick V Frasch, Theresa Lin, Eric Tseng, J Karl Hedrick, and Francesco Borrelli. Spatial predictive control for agile semi-autonomous ground vehicles. In *Proceedings of the 11th international symposium on advanced vehicle control*, pages 1–6, 2012.

[68] Hans Georg Bock and Karl-Josef Plitt. A multiple shooting algorithm for direct solution of optimal control problems. *IFAC Proceedings Volumes*, 17(2):1603–1608, 1984.

[69] Rien Quirynen, Boris Houska, Mattia Vallerio, Dries Telen, Filip Logist, Jan Van Impe, and Moritz Diehl. Symmetric algorithmic differentiation based exact hessian sqp method and software for economic mpc. In *53rd IEEE Conference on Decision and Control*, pages 2752–2757. IEEE, 2014.

[70] Florian Kehrle, Janick V Frasch, Christian Kirches, and Sebastian Sager. Optimal control of formula 1 race cars in a vdrift based virtual environment. *IFAC Proceedings Volumes*, 44(1):11907–11912, 2011.

[71] Moritz Diehl. *Real-time optimization for large scale nonlinear processes*. PhD thesis, Heidelberg University, 2001.

[72] Richard H Byrd, Mary E Hribar, and Jorge Nocedal. An interior point algorithm for large-scale nonlinear programming. *SIAM Journal on Optimization*, 9(4):877–900, 1999.

[73] Jorge Nocedal and Stephen J Wright. *Numerical optimization*. Springer, 1999.

[74] Boris Houska, Hans Joachim Ferreau, and Moritz Diehl. Acado toolkit: An open-source framework for automatic control and dynamic optimization. *Optimal Control Applications and Methods*, 32(3):298–312, 2011.

[75] Hans Joachim Ferreau, Christian Kirches, Andreas Potschka, Hans Georg Bock, and Moritz Diehl. qpoases: A parametric active-set algorithm for quadratic programming. *Mathematical Programming Computation*, 6(4):327–363, 2014.

[76] Gianluca Frison, Hans Henrik Brandenborg Sørensen, Bernd Dammann, and John Bagterp Jørgensen. High-performance small-scale solvers for linear model predictive control. In *2014 European Control Conference (ECC)*, pages 128–133. IEEE, 2014.

[77] Lorenz T Biegler and Victor M Zavala. Large-scale nonlinear programming using ipopt: An integrating framework for enterprise-wide dynamic optimization. *Computers & Chemical Engineering*, 33(3):575–582, 2009.

[78] Moritz Diehl, H Georg Bock, Johannes P Schlöder, Rolf Findeisen, Zoltan Nagy, and Frank Allgöwer. Real-time optimization and nonlinear model predictive control of processes governed by differential-algebraic equations. *Journal of Process Control*, 12(4):577–585, 2002.

[79] Andrea Zanelli, Alexander Domahidi, Juan Jerez, and Manfred Morari. Forces nlp: an efficient implementation of interior-point methods for multistage nonlinear nonconvex programs. *International Journal of Control*, 93(1):13–29, 2020.

[80] Shilp Dixit, Umberto Montanaro, Mehrdad Dianati, David Oxtoby, Tom Mizutani, Alexandros Mouzakitis, and Saber Fallah. Trajectory planning for autonomous high-speed overtaking in structured environments using robust mpc.

IEEE Transactions on Intelligent Transportation Systems, 21(6):2310–2323, 2019.

[81] Alexander Liniger, Alexander Domahidi, and Manfred Morari. Optimization-based autonomous racing of 1: 43 scale rc cars. *ArXiv abs/1711.07300*, 2017.

[82] Maximilian Kloock, Patrick Scheffe, Lukas Botz, Janis Maczijewski, Bassam Alrifaee, and Stefan Kowalewski. Networked model predictive vehicle race control. *IEEE Intelligent Transportation Systems Conference (ITSC)*, 2019.

[83] Alexander Liniger and J. Lygeros. A noncooperative game approach to autonomous racing. *IEEE Transactions on Control Systems Technology*, 28:884–897, 2020.

[84] Yunlong Song, HaoChih Lin, Elia Kaufmann, Peter Dürr, and Davide Scaramuzza. Autonomous overtaking in gran turismo sport using curriculum reinforcement learning. In *2021 IEEE international conference on robotics and automation (ICRA)*, pages 9403–9409. IEEE, 2021.

[85] Arthur Richards and Jonathan How. Mixed-integer programming for control. In *Proceedings of the 2005, American Control Conference, 2005.*, pages 2676–2683. IEEE, 2005.

[86] Fabio Molinari, Nguyen Ngoc Anh, and Luigi del Re. Efficient mixed integer programming for autonomous overtaking. *2017 American Control Conference (ACC)*, pages 2303–2308, 2017.

[87] Johan Karlsson, Nikolce Murgovski, and Jonas Sjöberg. Comparison between mixed-integer and second order cone programming for autonomous overtaking. *2018 European Control Conference (ECC)*, pages 386–391, 2018.

[88] Christina Miller, Christian Pek, and Matthias Althoff. Efficient mixed-integer programming for longitudinal and lateral motion planning of autonomous vehicles. *2018 IEEE Intelligent Vehicles Symposium (IV)*, pages 1954–1961, 2018.

[89] Rudolf Reiter, Martin Kirchengast, Daniel Watzenig, and Moritz Diehl. Mixed-integer optimization-based planning for autonomous racing with obstacles and rewards. *IFAC-PapersOnLine*, 54(6):99–106, 2021.

[90] Michael R. Bussieck and Stefan Vigerske. Minlp solver software. In *Wiley Encyclopedia of Operations Research and Management Science*. John Wiley & Sons, Ltd, 2011. lastest online version 2014.

[91] LLC Gurobi Optimization. Gurobi optimizer reference manual, 2018.

[92] Christian Bliek1ú, Pierre Bonami, and Andrea Lodi. Solving mixed-integer quadratic programming problems with ibm-cplex: a progress report. In *Proceedings of the twenty-sixth RAMP symposium*, pages 16–17, 2014.

[93] Kipp Martin. Tutorial: Coin-or: Software for the or community. *Interfaces*, 40(6):465–476, 2010.

[94] Brian Borchers and John E Mitchell. An improved branch and bound algorithm for mixed integer nonlinear programs. *Computers & Operations Research*, 21(4):359–367, 1994.

[95] Nan Li, Eric Goubault, Laurent Pautet, and Sylvie Putot. Autonomous racecar control in head-to-head competition using mixed-integer quadratic programming. In *Opportunities and Challenges with Autonomous Racing, ICRA*, 2021.

[96] Matthew G Earl and Raffaello D'andrea. Iterative milp methods for vehicle-control problems. *IEEE Transactions on Robotics*, 21(6):1158–1167, 2005.

[97] John Bellingham, Arthur Richards, and Jonathan P How. Receding horizon control of autonomous aerial vehicles. In *Proceedings of the 2002 American control conference (IEEE Cat. No. CH37301)*, volume 5, pages 3741–3746. IEEE, 2002.

[98] Arthur Richards, Tom Schouwenaars, Jonathan P How, and Eric Feron. Spacecraft trajectory planning with avoidance constraints using mixed-integer linear programming. *Journal of Guidance, Control, and Dynamics*, 25(4):755–764, 2002.

[99] Zhongqi Sun, Li Dai, Kun Liu, Dimos V Dimarogonas, and Yuanqing Xia. Robust self-triggered mpc with adaptive prediction horizon for perturbed nonlinear systems. *IEEE Transactions on Automatic Control*, 64(11):4780–4787, 2019.

[100] Ning He, Lipeng Qi, Ruoxia Li, and Yuesheng Liu. Design of a model predictive trajectory tracking controller for mobile robot based on the event-triggering mechanism. *Mathematical Problems in Engineering*, 2021, 2021.

[101] Moritz Diehl, Hans Georg Bock, and Johannes P Schlöder. A real-time iteration scheme for nonlinear optimization in optimal feedback control. *SIAM Journal on control and optimization*, 43(5):1714–1736, 2005.

[102] Boris Houska, Hans Joachim Ferreau, and Moritz Diehl. An auto-generated real-time iteration algorithm for nonlinear mpc in the microsecond range. *Autom.*, 47:2279–2285, 2011.

[103] Paul T Boggs and Jon W Tolle. Sequential quadratic programming. *Acta numerica*, 4(1):1–51, 1995.

[104] Rodrigo Longhi Guimarães, André Schneider de Oliveira, João Alberto Fabro, Thiago Becker, and Vinícius Amilgar Brenner. Ros navigation: Concepts and tutorial. In *Robot Operating System (ROS)*, pages 121–160. Springer, 2016.

[105] Julius Ziegler, Philipp Bender, Thao Dang, and Christoph Stiller. Trajectory planning for bertha: A local, continuous method. In *2014 IEEE intelligent vehicles symposium proceedings*, pages 450–457. IEEE, 2014.

[106] Jinxiang Wang, Yongjun Yan, Kuoran Zhang, Yimin Chen, Mingcong Cao, and Guodong Yin. Path planning on large curvature roads using driver-vehicle-road system based on the kinematic vehicle model. *IEEE Transactions on Vehicular Technology*, 71(1):311–325, 2021.

[107] Jun Zeng, Bike Zhang, and Koushil Sreenath. Safety-critical model predictive control with discrete-time control barrier function. In *2021 American Control Conference (ACC)*, pages 3882–3889. IEEE, 2021.

[108] Bruno Brito, Boaz Floor, Laura Ferranti, and Javier Alonso-Mora. Model predictive contouring control for collision avoidance in unstructured dynamic environments. *IEEE Robotics and Automation Letters*, 4(4):4459–4466, 2019.

[109] Oscar de Groot, Bruno Brito, Laura Ferranti, Dariu Gavrila, and Javier Alonso-Mora. Scenario-based trajectory optimization in uncertain dynamic environments. *IEEE Robotics and Automation Letters*, 6(3):5389–5396, 2021.

[110] Stephen Boyd, Stephen P Boyd, and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[111] Ionela Prodan, Sorin Olaru, Cristina Stoica, and Silviu-Iulian Niculescu. Collision avoidance and path following for multi-agent dynamical systems. In *ICCAS 2010*, pages 1930–1935, 2010.

[112] Jayanth Bhargav, Johannes Betz, Hongrui Zehng, and Rahul Mangharam. Deriving spatial policies for overtaking maneuvers with autonomous vehicles. In *International Conference on COMmunication Systems & NETworkS (COMSNETS)*, pages 859–864. IEEE, 2022.

[113] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, Andrew Y Ng, et al. Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, page 5, 2009.

[114] Matthew O'Kelly, Hongrui Zheng, Dhruv Karthik, and Rahul Mangharam. F1tenth: An open-source evaluation environment for continuous control and reinforcement learning. In *NeurIPS 2019 Competition and Demonstration Track*, pages 77–89. PMLR, 2020.

[115] Hugh Durrant-Whyte and Tim Bailey. Simultaneous localization and mapping: part i. *IEEE robotics & automation magazine*, 13(2):99–110, 2006.

[116] Wolfgang Hess, Damon Kohler, Holger Rapp, and Daniel Andor. Real-time loop closure in 2d lidar slam. In *2016 IEEE international conference on robotics and automation (ICRA)*, pages 1271–1278. IEEE, 2016.

[117] Leiv Andresen, Adrian Brandemuehl, Alex Honger, Benson Kuan, Niclas Vödisch, Hermann Blum, Victor Reijgwart, Lukas Bernreiter, Lukas Schaupp, Jen Jen Chung, et al. Accurate mapping and planning for autonomous racing. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4743–4749. IEEE, 2020.

[118] Zirui Zang, Renukanandan Tumu, Johannes Betz, Hongrui Zheng, and Rahul Mangharam. Winning the 3rd japan automotive ai challenge–autonomous racing with the autoware. auto open source software stack. *arXiv preprint arXiv:2206.00770*, 2022.

[119] Georgios Zamanakos, Lazaros Tsochatzidis, Angelos Amanatiadis, and Ioannis Pratikakis. A comprehensive survey of lidar-based 3d object detection methods with deep learning for autonomous driving. *Computers & Graphics*, 99:153–181, 2021.

[120] Dieter Fox, Wolfram Burgard, Frank Dellaert, and Sebastian Thrun. Monte carlo localization: Efficient position estimation for mobile robots. *AAAI/IAAI*, 1999(343-349):2–2, 1999.

[121] Fredrik Gustafsson. Particle filter theory and practice with positioning applications. *IEEE Aerospace and Electronic Systems Magazine*, 25(7):53–82, 2010.

[122] Michael Montemerlo, Sebastian Thrun, and William Whittaker. Conditional particle filters for simultaneous mobile robot localization and people-tracking. In *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No. 02CH37292)*, volume 1, pages 695–701. IEEE, 2002.

[123] Thinal Raj, Fazida Hanim Hashim, Aqilah Baseri Huddin, Mohd Faisal Ibrahim, and Aini Hussain. A survey on lidar scanning mechanisms. *Electronics*, 9(5):741, 2020.

[124] Karthik Lakshmanan, Shinpei Kato, and Ragunathan Rajkumar. Scheduling parallel real-time tasks on multi-core processors. In *2010 31st IEEE Real-Time Systems Symposium*, pages 259–268. IEEE, 2010.

[125] Shaoshan Liu, Jie Tang, Zhe Zhang, and Jean-Luc Gaudiot. Computer architectures for autonomous driving. *Computer*, 50(8):18–25, 2017.

[126] Leonhard Hermansdorfer, Johannes Betz, and Markus Lienkamp. Benchmarking of a software stack for autonomous racing against a professional human race driver. In *2020 Fifteenth International Conference on Ecological Vehicles and Renewable Energies (EVER)*, pages 1–8. IEEE, 2020.

[127] Vamsi Boppana, Sagheer Ahmad, Ilya Ganusov, Vinod Kathail, Vidya Rajagopalan, and Ralph Wittig. Ultrascale+ mpsoc and fpga families. In *2015 IEEE Hot Chips 27 Symposium (HCS)*, pages 1–37. IEEE, 2015.

[128] Michel Schubiger, Goran Banjac, and John Lygeros. Gpu acceleration of admm for large-scale quadratic programming. *Journal of Parallel and Distributed Computing*, 144:55–67, 2020.

[129] Wang Xi, Li Dewei, and Xi Yugeng. A mixed-integer quadratic programming solver based on gpu. In *2015 34th Chinese Control Conference (CCC)*, pages 2686–2691. IEEE, 2015.

[130] Alexander Liniger and John Lygeros. A noncooperative game approach to autonomous racing. *IEEE Transactions on Control Systems Technology*, 28(3):884–897, 2019.

[131] Mingyu Wang, Zijian Wang, John Talbot, J Christian Gerdes, and Mac Schwager. Game-theoretic planning for self-driving cars in multivehicle competitive scenarios. *IEEE Transactions on Robotics*, 37(4):1313–1325, 2021.

[132] Alan Burns and Robert Davis. Mixed criticality systems-a review. *Department of Computer Science, University of York, Tech. Rep*, pages 1–69, 2013.

ECOLE
DOCTORALE

Titre : Contrôle temps réel et efficace pour la course autonome

Mots clés : véhicule autonome, contrôle prédictif, prise de décision, temps réel, système robotique

Résumé : Récemment, le domaine du véhicule autonome a connu d'énormes progrès. En étudiant le défi de la course autonome, une forme spéciale de conduite autonome, nous cherchons à mieux comprendre comment les véhicules peuvent être contrôlés efficacement en temps réel pour gérer des situations dynamiques et complexes. Nous développons une approche basée sur le contrôle prédictif de modèle non linéaire (NMPC), une technique de contrôle avancée, qui permet d'obtenir un temps de progrès optimal du véhicule tout en tenant compte de la dynamique non linéaire du système et des contraintes temporelles liées aux obstacles. Pour faire face à la présence d'un véhicule adverse, nous combinons le contrôle prédictif NMPC avec la programmation mixte en nombres entiers (MIP) afin d'effectuer des manœuvres de dépassement sûres et efficaces. Cependant, il est difficile de mettre en œuvre le contrôle prédictif NMPC sur des dispositifs embarqués en raison de sa complexité de calcul élevée. Une des difficultés consiste à assurer l'exécution en temps réel du contrôleur, ce qui nécessite un respect strict des budgets d'exécution et des échéances de temps. Une autre difficulté consiste à rendre le contrôle efficace, et donc à préserver les performances quasi-optimales du système. Pour le mode de course à véhicule unique, nous proposons une approche de recalcul sur un horizon de plusieurs pas, calcul qui est déclenché en fonction d'événements spécifiques. Une des conditions de déclenchement vise à s'assurer que les contraintes temps réel de budget temps seront respectées. L'autre condition de déclenchement sert à réduire le temps de calcul tout en maintenant des performances quasi-optimales en termes de temps de tour de circuit. Pour le mode de course à un-contre-un, nous proposons un algorithme compatible avec une exécution en ligne comme alternative à l'encodage MIP. Il regroupe efficacement des décisions de dépassement et les exécute à une fréquence de contrôle déterministe pour répondre aux exigences de temps réel. Au travers d'une architecture générique, nous prenons également en compte d'autres composants logiciels que le contrôleur, tels que les algorithmes de détection des adversaires et de localisation, qui constituent collectivement un graphique acyclique dirigé (DAG). Un modèle d'exécution des tâches est proposé pour affecter les composants du DAG aux processeurs disponibles avec différents degrés de parallélisme. Il réduit la latence, augmente le taux d'actualisation de la commande et, finalement, améliore la performance du système. En résumé, cette thèse fournit un ensemble de mécanismes visant à une mise en œuvre efficace d'un contrôle en temps réel dans des systèmes autonomes.

Title : Real-Time and Efficient Control for Autonomous Racing

Keywords : autonomous vehicle, model predictive control, decision-making, real-time, robotic system

Abstract : Recently, the autonomous driving domain has made tremendous advancements. By investigating the challenge of autonomous racing, a special form of autonomous driving, we seek to better understand how vehicles could be efficiently controlled in real-time settings for handling intricate dynamic situations. We develop an approach based on Nonlinear Model Predictive Control (NMPC), a cutting-edge control technique, that can attain the optimal progress time of the vehicle while accounting for nonlinear system dynamics and obstacle-related time-varying constraints. To deal with the presence of an opponent vehicle, we combine NMPC with Mixed Integer Programming (MIP) for encoding safe and efficient overtaking maneuvers. However, it is challenging to implement NMPC on embedded devices due to its high calculation complexity. One concern is ensuring real-time execution of the controller, which necessitates strict adherence to the time budget restriction and rigorous compliance with deadlines. Another problem is managing to make the control efficient, which calls for the maintenance of an adequate level of system performance. We propose a multi-step recomputation approach for the single-vehicle race mode, which is triggered based on specific events. One of the triggering conditions aims at ensuring that the real-time budget constraints are respected. The other triggering condition serves for reducing computational time while retaining quasi-optimal lap time performance. For head-to-head racing mode, we propose an algorithm as an online feasible alternative to MIP encoding. It efficiently aggregates overtaking decisions and schedules them at a deterministic control frequency to meet real-time requirements. In a generic system architecture, we also take into account other software components besides the controller, such as opponent detection and self-localization algorithms, which collectively constitute a Directed Acyclic Graph (DAG). To assign DAG components to available processors with varying degrees of parallelism, we propose a task execution model which decreases the latency, increases the control update rate, and eventually enhances the system performance. In summary, this thesis provides a set of mechanisms aimed at an efficient implementation of real-time control in autonomous systems.

Institut Polytechnique de Paris
91120 Palaiseau, France