Technische Universität München

Institute of Flight System Dynamics

Nonlinear Model Predictive Control for an Autonomous Racecar

–

Nichtlineare modelprädiktive Regelung für ein autonomes Rennfahrzeug

Masterarbeit

Author: Francisco Sevilla

Matriculation Number: 03621157

Supervisors: Benedikt Grüter, M.Sc.

Prof. Dr.-Ing. Florian Holzapfel

July 2018

Statutory Declaration

I, Francisco Sevilla, declare on oath towards the Institute of Flight System Dynamics of Technische Universität München, that I have prepared the present Master thesis independently and with the aid of nothing but the resources listed in the bibliography.

This thesis has neither as-is nor similarly been submitted to any other university.

Garching, 31 July 2018

Francisco Sevilla

Kurzfassung

In diesem Projekt wurde ein NMPC-Algorithmus (Nonlinear Model Predictive Control) für das Rapid-Prototyping dieser Art von Reglern in einer Simulationsumgebung programmiert. Basierend auf FALCON.m, die Optimalsteuerungstoolbox für MATLAB, entwickelt am Institut für Flugsystemdynamik der Technischen Universität München, kann der in diesem Projekt implementierte Algorithmus eine generische Zielfunktion annehmen und jedes dynamische System verwenden, das mit differential-algebraischen Gleichungen modelliert ist. Während der Entwicklung des NMPC-Algorithmus wurden in Simulink verschiedene dynamische Systeme mit steigendem Komplexitätsgrad modelliert, um den Controller zu testen. Als Beispiel für ein hochkomplexes System wurde der Regler für das zeitoptimale Manövrieren eines autonomen Formula-Student-Rennwagens getestet. In der Simulation konnte der NMPC-Regler eine vergleichbare Rundenzeit erreichen wie das reale Fahrzeug, das von einem menschlichen Fahrer auf der gleichen Strecke gesteuert wurde.

Abstract

In this project, a Nonlinear Model Predictive Control (NMPC) algorithm was programmed for rapid-prototyping of this kind of controllers in a simulation environment. Based on FALCON.m, the Optimal Control toolbox for MATLAB developed at the Institute for Flight System Dynamics of the Technical University of Munich, the algorithm implemented in this project can take a generic objective function and use any dynamic system modelled with differential algebraic equations. During the development of the NMPC algorithm, different dynamic systems with increasing levels of complexity were modelled in Simulink to test the controller. As an example of a highly complex system, the controller was tested for time-optimal maneuvering of an autonomous Formula Student racecar. In simulation, the NMPC controller was able to achieve a comparable laptime to that of the real-life vehicle piloted by a human driver on the same track.

Table of Contents

List of Figures

List of Tables

Table of Acronyms

Acronym	Description
AD	Autonomous Driving
ADAS	Advanced Driver Assistance Systems
ADS	Automated Driving System
DAE	Differential Algebraic Equation
DDT	Dynamic Driving Task
ESP	Electronic Stability Program
ERK4	Explicit 4^{th}-order Runge-Kutta method
FSG	Formula Student Germany
MPC	(Linear) Model Predictive Control
NMPC	Nonlinear Model Predictive Control
OC	Optimal Control
OCP	Optimal Control Problem
ODD	Operational Design Domain
QP	Quadratic Programming
RTI	Real-Time Iteration scheme
SQP	Sequential Quadratic Programming
TUM	Technical University of Munich
ZOH	Zero-Order Hold

1 Introduction

Autonomous cars have been gaining more and more attention in the last years, with many automotive companies investing in their research [1]. Now, there are even racing competitions introducing autonomous driving into motorsports. One of them is Roborace [2], which is currently in the development and testing phase. Another one is Formula Student Driverless (FSD), which held its first event at the Formula Student Germany (FSG) in August 2017 [3, 4]. Both of these competitions require cars to maneuver autonomously around a track by steering the tires and by driving or braking each wheel independently.

One of the main difficulties of autonomous driving is how to control the vehicle dynamics of the car by steering, accelerating and braking in real-time so that the car follows a desired trajectory [5]. To maintain the desired path, it is not only necessary to control the car at each given time point, but the controller should also look ahead into the path, particularly at higher speeds [6]. This problem becomes even harder in motorsports where the lap-time should be minimized, and the desired path might not be given per se, but rather as left and right boundaries that allow to optimize the path between them. Nonlinear model predictive control (NMPC) is a very promising method to tackle this challenge.

The current chapter will provide a broader motivation for the project, as well as the state of the art concerning it. Also, the goals and contributions of this thesis will be addressed, and the last section explains how this document is structured.

1.1 Motivation

Self-driving technology has become an important topic for the automotive industry in recent years. This technology is expected to open new markets, create new products and reshape the automotive industry and its relationship to other businesses [7]. Some of the benefits that autonomous cars can bring with them include [8, 9]:

- **Road safety and reduction of car crashes**: A downtrend in the number of crashes and in the severity of crashes can already be seen with the implementation of driver assistance systems like the *Electronic Stability Program* (ESP) and the *Anti-Lock Brakes System* (ABS). As human error is the cause for 90% of car crashes, fully autonomous vehicles could prevent many of the crashes nowadays.
- **Congestion**: Autonomous vehicles can accelerate and brake automatically to maintain a constant speed and constant distance to the vehicle in front. This not only reduces traffic, but also increases the capacity of the road, since the distance between cars can be kept to a minimum. Vehicles connected between each other and to the traffic system have also a big potential to decrease inner-city congestion. Furthermore, a reduction in crashes would also reduce traffic jams.
- **Utilization of idle time**: While riding in an autonomous vehicle, passengers could engage in other productive activities. Römer et al. [7] estimate that autonomous vehicles could free up about 1.9 trillion minutes of idle time by 2030.
- **Land use**: Parking spaces are expected to decrease in number and size, especially in urban areas. Instead, spaces for pick-up and drop-off of passengers would be

necessary, but most of the freed-up space could be used for other purposes, for example as green-areas.

- **Environment**: Since most of the autonomous cars would be electric vehicles, their direct emissions of greenhouse gases would be zero. Autonomous vehicles have also a big potential to drive more efficiently than a human driver.

Most of the big automakers and automotive suppliers are investing in the research and development of self-driving technology. This can be appreciated in the number of patents published in this area [1]. Many of them have also either founded or acquired start-ups that should leap the development of self-driving vehicles, mostly specializing in software. A few examples of these companies are Autonomous Intelligent Driving (subsidiary of Audi) [10], Zenuity (founded by Volvo and Autoliv) [11] and Elektrobit (bought by Continental in 2015) [12]. Also, businesses in the computer and semiconductor industry, e.g. Intel [13] and NVIDIA [14], are developing both hardware and software dedicated for Advanced Driver Assistance Systems (ADAS) and Autonomous Driving (AD).

An important aspect for bringing autonomous driving vehicles into the streets will be consumer acceptance. A way to foster mass acceptance of these technologies could be through motorsports: showing the public that driverless vehicles can work under extreme conditions at high speeds could increase their confidence in this technology. Motorsport events could also boost the technological developments in this field.

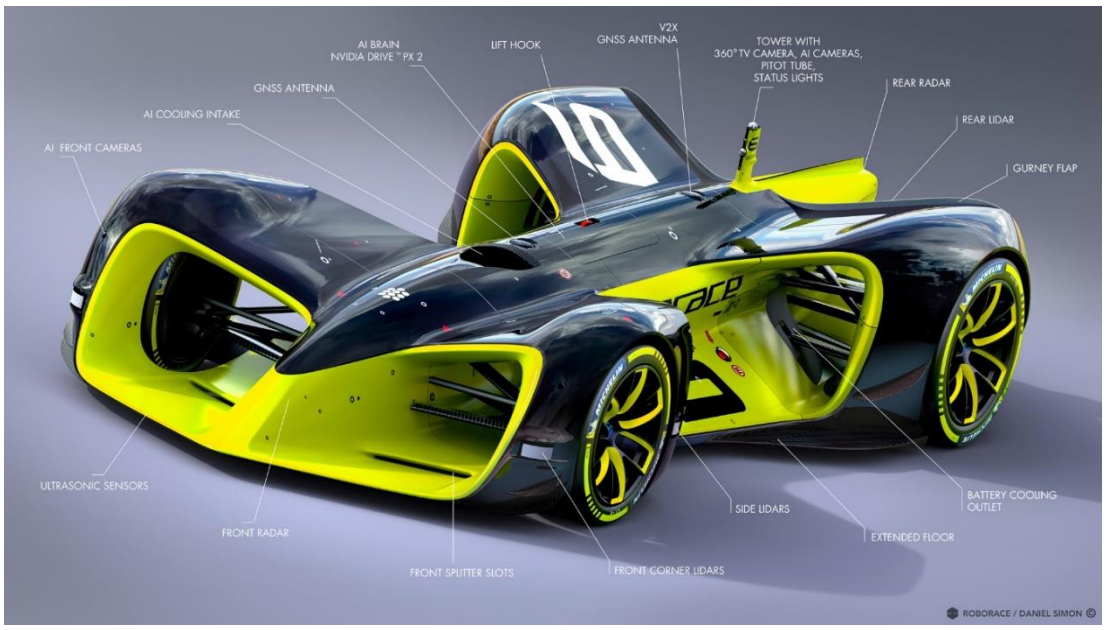

Figure 1-1: Roborace – Hardware Overview [15]

An example of an autonomous racing series is carried out by Roborace. This is a company that is creating one of the first motorsport events for self-driving vehicles [2], which will go with the same name as the company. Roborace offers a full-size All-Wheel-Driven electric vehicle as a platform for teams to deploy their self-driving software and race around a track. Currently, the company has been presenting their development car at the FIA's Formula E races, where the car drives autonomously, sometimes comparing the lap-times that the software achieved to ones set by a professional driver on the same vehicle.

Another motorsport event for autonomous cars, the Formula Student Driverless [3, 4]. As the name suggests, this competition is conceived for students. It allows students to design and build full scale racing cars and run them in international events against other universities. In this thesis, the racecar simulation presented in chapter 4 uses a model and parameters that correspond to a Formula Student vehicle.

The Formula Student Driverless competitions are composed of several disciplines that are divided into two categories: the dynamic and the static events. The dynamic disciplines include *Acceleration*, which is a 75 meters straight line, *Skidpad*, a figure 8 where the maximum lateral acceleration is tested, *Trackdrive*, a stint around a track set up with cones, and *Efficiency*, where the energy-use during the Trackdrive is scored. In every discipline, each of the autonomous cars drives alone on the track and the teams get points based on their lap-times. In the static disciplines, the teams have the chance to present a *Business Plan* and a *Cost Report* for the car, as well as an *Engineering Design* and an *Autonomous Design*, where the teams show their ideas, calculations and simulations, and get judged by experts in the field [16].

Figure 1-2: Formula Student car of the Technical University of Munich's team, TUfast

In 2017, in the first Formula Student Driverless event, the team of the Swiss Federal Institute of Technology in Zurich (ETH Zürich) completed the competition successfully. For motion planning and control, this team used software from *embotech AG* [17], a swiss start-up that specializes in embedded Nonlinear Model Predictive Control [18]. The following figure shows a plot where the prediction horizon of the NMPC algorithm can be appreciated (right) next to camera footage of the team's car driving on a typical Formula Student track (left) [19]. This example shows a successful implementation of a Nonlinear Model Predictive Control for the same application as in this project. However, the algorithm implemented in this thesis differs from the one provided by *embotech AG.*

Figure 1-3: Visualization of the NMPC of the Formula Student team of the ETH Zurich, AMZ [22]

As described in [5] and [6], motion planning and motion control are two of the key technologies that make autonomous driving possible. Planning is the decision-making process used to calculate the trajectory that brings the vehicle from a start location to an end location. Control refers to the actions that are taken by the vehicle's actuators to follow the planned trajectory. These two software components can be combined into one algorithm using, for example, Model Predictive Control.

Thus, for this project, a Nonlinear Model Predictive Control (NMPC) algorithm was implemented in MATLAB / Simulink to control a simulated Formula Student racecar. This implementation is based on the MATLAB toolbox for Optimal Control, FALCON.m, which was developed at the institute for Flight System Dynamics of the Technical University of Munich. Furthermore, the vehicle for which this controller was developed has the possibility to drive and brake each tire independently, see Appendix B. Therefore, since NMPC is able to handle systems with multiple inputs and complex constraints [20], it provides a good solution for this challenge.

1.2 State of the art

In this section, the state of the art concerning topics relevant to this thesis are presented. Subsection 1.2.1 gives an overview of the field of Autonomous Vehicles, introducing the six levels of Autonomous Driving and then presenting the different elements that compose an autonomous driving system. The second subsection enlists and explains several strategies of Nonlinear Model Predictive Control, later deriving a justification for the strategy that was implemented in this project, the Real-Time Iteration scheme. In 1.2.3, some examples of successful projects that employed NMPC in the automotive industry are introduced, also presenting applications that used RTI. Subsection 1.2.4 briefly presents two applications Optimal Control in motorsports, describing their relevance for this project.

1.2.1 Autonomous Vehicles

The Society of Automotive Engineers, SAE International, introduced a new standard, SAE J3016 [21], for autonomous cars in 2014. This standard, called "Taxonomy and Definitions for Terms Related to On-Road Motor Vehicle Automated Driving Systems", introduces new terminology for this field as well as the *six levels of driving automation*. Many of these definitions and their acronyms have already been widely adopted in the automotive industry and the research community. The most important include:

- **Dynamic Driving Task** (DDT): Operational and tactical functions required to operate a vehicle on the road, excluding the selection and scheduling of destinations.
- **Operational Design Domain** (ODD): Operating conditions under which an autonomous system is designed to function, for example environmental, geographical or time-of-day restrictions.
- **Minimal Risk Condition**: Condition to which a vehicle may be brought to reduce the risk of a crash when a DDT cannot be completed.
- **Automated Driving System** (ADS): Software and hardware that are together able to perform an entire DDT. This term is used to describe levels 3 to 5 of driving automation (see Table 1-1).
- **DDT Fallback**: Response of an ADS to reach a *minimal risk condition* due to a system failure or exit of the ODD.

The six levels of driving automation introduced in the SAE J3016 standard are summarized in Table 1-1. They go from *level 0*, which corresponds to no automation, to *level 5*, which is the level at which the vehicle can complete all driving tasks entirely on its own. The system required in a Formula Student car to be able to complete the competition must achieve level 4, as there cannot be a human driver in the car to monitor the environment or to react in case of failure. However, the ODD is limited to driving on a track with specific characteristics [16].

SAE level	Name	Narrative Definition	Execution of Steering and Acceleration/ Deceleration	*Monitoring* of Driving Environment	Fallback Performance of *Dynamic Driving Task*	System Capability *(Driving Modes)*
***Human driver* monitors the driving environment**						
0	**No Automation**	the full-time performance by the *human driver* of all aspects of the *dynamic driving task*, even when enhanced by warning or intervention systems	Human driver	Human driver	Human driver	n/a
1	**Driver Assistance**	the *driving mode*-specific execution by a driver assistance system of either steering or acceleration/deceleration using information about the driving environment and with the expectation that the *human driver* perform all remaining aspects of the *dynamic driving task*	Human driver and system	Human driver	Human driver	Some driving modes
2	**Partial Automation**	the *driving mode*-specific execution by one or more driver assistance systems of both steering and acceleration/ deceleration using information about the driving environment and with the expectation that the *human driver* perform all remaining aspects of the *dynamic driving task*	**System**	Human driver	Human driver	Some driving modes
***Automated driving system* ("system") monitors the driving environment**						
3	**Conditional Automation**	the *driving mode*-specific performance by an *automated driving system* of all aspects of the dynamic driving task with the expectation that the *human driver* will respond appropriately to a *request to intervene*	System	**System**	Human driver	Some driving modes
4	**High Automation**	the *driving mode*-specific performance by an automated driving system of all aspects of the *dynamic driving task*, even if a *human driver* does not respond appropriately to a *request to intervene*	System	System	**System**	Some driving modes
5	**Full Automation**	the full-time performance by an *automated driving system* of all aspects of the *dynamic driving task* under all roadway and environmental conditions that can be managed by a *human driver*	System	System	System	**All driving modes**

Table 1-1: Levels of Driving Automation for on-road vehicles [22]

An *Automated Driving System* is composed of multiple software elements. Cheng [5] divides these into 4 categories:

- **Environment Perception and Modelling**: Collect data from various sensors (camera, lidar, radar, etc.) and extract features like colors, edges and contours. These are then classified into objects, e.g. lanes, signs, vehicles and pedestrians. This software

component should also be able to track the objects it detects and, based on their movements, predict their future position to avoid collisions.

- **Localization and Map Building**: Based on the environment model and other information like geographic maps, create a local map around the vehicle and locate and track the vehicle's position in that map. A class of algorithms that combines these functions is called SLAM, which stands for Simultaneous Localization and Mapping.
- **Path Planning and Decision-Making**: Compute a trajectory from an initial location to a goal location avoiding both static and dynamic obstacles. Since the environment and the position of the vehicle in it are continuously changing, the path planning algorithm must constantly adapt to the new context.
- **Motion Control**: Control the vehicle's actuators in order to follow the trajectory calculated by the path planning algorithm. The motion control can be divided into two categories: longitudinal, corresponding to velocity and distance, and lateral, corresponding to lane keeping. However, the longitudinal and lateral motions are coupled through the characteristics of the tire and of the vehicle. Therefore, more advanced controllers handle the longitudinal and lateral dynamics simultaneously.

The focus of this thesis lies in path planning and motion control of an autonomous racecar using Nonlinear Model Predictive Control. NMPC is able to control both the longitudinal and the lateral dynamics of the car collectively (as shown in [23–27]), as well as to handle nonlinear constraints on the actuators and on the vehicle state. In the following subsection, different NMPC algorithms are introduced and compared.

1.2.2 Overview of NMPC Strategies

Model predictive control (MPC) is a type of feedback control that relies on a model of the system to predict its behavior, which can be used to optimize a cost function over a finite time horizon, as in an optimal control problem [28]. This type of controller is mostly used for trajectory tracking and stabilization of the system, and it can efficiently handle actuator saturations and other operational constraints. These features have made MPC a popular solution for slow systems in the process industry since the late 1970s. Nevertheless, applications for which a linear model is not accurate enough require the use of Nonlinear Model Predictive Control (NMPC). Although NMPC poses many challenges in terms of computational resources, especially for fast dynamic systems, this type of controller has been successfully implemented in many applications similar to this project [23–27, 29–32], and is expected to become more and more common [33, 34].

The basic idea of NMPC is to solve a nonlinear optimal control problem at each sampling iteration (see chapter 2 for a detailed explanation). Since solving this is a difficult task, there have been several proposals of techniques to perform these computations more efficiently. Camacho and Bordons introduce several techniques in [34] like Suboptimal NMPC, Use of Short Horizons, Feedback Linearization, etc. Furthermore, the NMPC algorithms can be categorized according to the method they use to solve the underlaying optimal control problem (sequentially/single-shooting or simultaneously/multiple-shooting) [35] or according to the type of solver they use for the optimization (interior-point or active-set) [36].

The so-called Real-Time Iteration scheme (RTI), presented in [33, 36–38], is an approach for NMPC that has been applied to many different fields. This method performs a Sequential Quadratic Programming (SQP) iteration at every sampling timestep, dividing the computation into a "*Preparation Phase*" and a "*Feedback Phase*" in order to include the most recent sensor

measurements into the optimization. Furthermore, it uses *initial value embedding* (see section 0) to correct the difference between the predicted state and the actual state of the system. This algorithm has been implemented into the ACADO Toolkit [39], which automatically generates C-code for a generic system using qpOASES [40] as quadratic problem solver.

The *Multistep, Newton-Type Control Strategy* [41] of Li and Biegler, proposed in 1989, also performs one Quadratic Programming (QP) step every sampling iteration. It is based, however, in a sequential discretization of the optimal control problem and assumes no model mismatch. This means that the actual state of the system is not incorporated into the algorithm, though the authors propose using parameter estimation to compensate for disturbances and uncertainties.

Another approach is the *Continuation/GMRES Method* proposed by Ohtsuka [42], which likewise takes only one optimization step at every sampling time and uses sequential discretization. Nevertheless, instead of using an SQP-method, it takes the inequality constraints into account by inserting a penalty term into the cost function (similar to an interior-point method). A variation of this method is presented in [43], where multiple-shooting discretization is used.

The *Advanced Step Controller* by Zavala and Biegler [44] is another algorithm for NMPC. The basic idea of this algorithm is to solve the optimal control problem of a future predicted state of the system till convergence, once the solution is ready, the predicted state is compared to the actual state and the solution is corrected by using sensitivity analysis methods.

In this project, the Real-Time Iteration scheme (RTI) was implemented. The reason for this were the many applications for which the RTI has been employed, especially the ones sharing similarities with this project [24, 26, 29–31]. Also, the RTI algorithm has been explained in detail in multiple sources, for example in [33] and [36–38], and is fairly simple to implement using FALCON.m as starting point. The structure of this algorithm is explained in section 2.4.

The optimization problem divides the NMPC algorithm into two further categories: Tracking NMPC and Economic NMPC. The most common use of NMPC is, as stated before, *tracking* of a predefined trajectory. In this case, the objective function of the optimization penalizes the "distance" of the system's state vector to the state vector of the reference trajectory. Economic NMPC generalizes the optimization problem, where the objective function can, in principle, be any quantity that relates to the problem [28]. Time-optimal driving of an autonomous racecar is an example of economic NMPC [31].

1.2.3 NMPC in Automotive Applications

Model Predictive Control, and also Nonlinear Model Predictive Control, have been researched for the use in road vehicles since the early 1990's. MPC has been applied to control several components of the car including engine, transmission, steering, suspension, energy management and thermal management. A survey of applications of MPC in the automotive industry can be found in [45], which also enumerates various applications for autonomous vehicles.

Falcone et al. [32] implemented two different tracking MPC algorithms to control the steering of an autonomous car. For both of the algorithms, a bicycle model of the car, in which the two tires of each axle are reduced to one to decrease the complexity of the model, and a Pacejka tire model [46] were used. Moreover, both approaches have only the differences of the controls ($\Delta \boldsymbol{u}$) as optimization variables. The first approach takes the nonlinear differential equations as

constraints and solves a full nonlinear problem at every timestep. The second approach linearizes the model at each sample time, using the current measurements as operating point for the linearization. Thus, the optimization problem is reduced to a quadratic problem. Both controllers were tested in simulation and experimentally.

The approach that solves the nonlinear problem was extended with a full vehicle model with wheel dynamics in [23] to also control braking. This controller was also tested in simulation. For experimental tests, the authors modified the controller and used a suboptimal MPC, in which the nonlinear vehicle model is linearized successively. Gao et al. [25] use the same setup as in [23] for obstacle avoidance. Here, a reference trajectory that violates the obstacles is given to the model. Thus, the MPC controller must find a trajectory that avoids these obstacles, which are provided to the algorithm as an additional cost function. In this paper, the authors suggest keeping a certain number of control inputs $\boldsymbol{u}$ at the end of the prediction horizon constant in order to reduce the number of optimization variables. In comparison to shortening the horizon length, this method still predicts the behavior of the states of the system, but for a Zero-Order Hold of the controls. They also propose a two level MPC algorithm, in which the higher level uses a point-mass model to calculate the new trajectory that avoids the obstacles, and the lower level performs the tracking task using the full vehicle model.

Katriniok et al. [27] extended the work in [32] to track a path reaching the traction limit of the tires. In this paper, the authors propose to linearize the system dynamics using the predicted states and controls as operating points. This method is very similar to the Real-Time Iteration scheme. However, it does not divide the algorithm into a “Preparation Phase” and a “Feedback Phase”. It also does not include the states in the optimization variables and therefore does not use initial value embedding to correct the prediction horizon. In all the models described above, the tire loads are assumed constant (compare to section 4.1.2).

The RTI algorithm has also been implemented for several applications regarding ground vehicles, particularly using the ACADO Toolkit. Frasch et al. [24] used this algorithm for obstacle avoidance of an autonomous vehicle employing a full vehicle model with wheel dynamics and a Pacejka tire model. This model also includes tire load transfer calculated with a first order lag. This application was extended in [26] to include a model of the vehicle suspension, so that the model has 15 states and 6 controls in total. In [24] and [26] the dynamics are discretized over space to simplify the handling of the obstacles.

Verschueren et al. [29–31] also used the RTI scheme to control model racecars around a track. Here, a bicycle model was used, and the dynamics of the model were also discretized over space. However, an economic objective function was employed in these publications, namely to minimize the lap-time. In [29] and [30], because of the spatial reformulation of the dynamics, the time was made a state and the optimization problem could be formulated as a tracking NMPC. Later in [31], the authors used an actual economic NMPC and employed the exact Hessian of the problem in their algorithm.

1.2.4 Optimal Control in Motorsport

As NMPC is based on the solution of an optimal control problem at each iteration, it may be remarked that several authors have applied both indirect and direct methods of optimal control to solve minimal time manoeuvring problems for race cars. Surveys of publications in this field can be found in [47] and [48].

Geiger [47] is of particular interest for this project. In their project, the author used the optimal control toolbox FALCON.m [49] to predict the lap times of a Formula Student racecar. In [48], van Koutrik uses a full vehicle model, but neglects the wheel dynamics, as is done for the NMPC in chapter 4 of this thesis. Moreover, in this project, a similar tire model was used as the one employed by van Koutrik.

1.3 Goals and Contribution

This section presents the goals that were set for this project, as well as the limitations of the product that resulted from it, Falcon NMPC. The contributions that this project makes are also outlined in this section.

The primary objective of this project is to implement a Nonlinear Model Predictive Control algorithm based on the Optimal Control Toolbox FALCON.m [49]. This control algorithm should be tested on different models with increasing levels of complexity. The performance of the controller should be evaluated based on conventional metrics, e.g. command response and disturbance response. Furthermore, the NMPC algorithm should be used to control a simulation model of an autonomous vehicle with full torque vectoring capabilities. As secondary objective, the NMPC algorithm is to be programmed in a way that allows it to handle a model of the generic system to be controlled and any cost function.

The NMPC implementation, which in this thesis will be referred to as *Falcon NMPC*, is programmed in MATLAB and provides an interface to Simulink with a so-called S-Function. The implementation is coded with object-oriented programming, allowing to use any optimal control problem that can be set up in FALCON.m, as described in Appendix A. The code also provides an interface for the user to create user-defined plots or to perform other tasks on the controller during simulation.

For this project, the Real-Time Iteration scheme [33] was implemented. This algorithm was tested for a cart with one degree of freedom (section 3.1), for an inverted pendulum on a cart (section 3.2) and for a double inverted pendulum on a cart (section 3.3). The model used in the NMPC of the double inverted pendulum includes an algebraic loop. Several experiments to tackle this algebraic loop were performed in the corresponding section.

Furthermore, an extension of the RTI method to handle economic NMPC, as in [50] and [31], was also implemented. In this extension, an exact Hessian of the optimal control problem is used, assuring positive-definiteness with so-called regularization methods (see section 2.2.3). This was evaluated with a point-mass model on a racetrack with a time-optimal objective function (section 3.4) and was finally applied for the full vehicle model in chapter 4. In these models, using the exact Hessian was also compared to using a constant hessian. Moreover, several adaptations of the BFGS Hessian approximation for NMPC (see section 2.2.3) were implemented and tested with the inverted pendulum (section 3.2) and with the point-mass model (section 3.4).

Falcon NMPC has, however, some limitations. The code cannot be compiled, and it cannot be autogenerated into C-code with MATLAB. Therefore, it cannot be used in a real-time target to control an actual plant. The reason for this is the fact that object-oriented programming was used in order to simplify the interaction for the user. To run an NMPC in an embedded system, the algorithm, as well as some parts of FALCON.m, would have to be reprogrammed. Also,

the S-Function of Falcon NMPC runs synchronously within the model. Thus, the simulation waits until both the *Feedback Phase* and the *Preparation Phase* of the RTI algorithm are completed before continuing the simulation, even if the sampling time of the NMC is larger than the step-size of the simulation. This means that the simulation might not run in real-time, even in cases where the timing of the controller would allow it in real-life.

Thus, Falcon NMPC can be used to quickly setup and test NMPC controllers in simulation. The algorithm presented in this thesis to control an autonomous racecar with full torque vectoring capabilities serves as a proof-of-concept for further development and implementation on real-time hardware. Future work will be discussed in chapter 5.

1.4 Structure of the thesis

This thesis is organized as follows:

Chapter 1 entails an introduction to the topic. In this chapter, a motivation for the project is presented, followed by an overview of the state of the art of different subjects that concern the project. Afterwards, the goals and the contribution of this thesis are portrayed.

Chapter 2 presents the necessary concepts for understanding how the Nonlinear Model Predictive Control algorithm implemented in this thesis works. As mentioned before, the implementation of the algorithm is based on the Optimal Control toolbox FALCON.m. This chapter also explains conceptually how this implementation was done. However, more information can be found in Appendix A.

Chapter 3 describes some experiments that were performed during the development of this project as well as the results and insights that were extracted from these experiments. For these tests, four different dynamic systems were considered in simulation: a one-degree-of-freedom point mass, an inverted pendulum on a cart, a double inverted pendulum on a cart and a two-degrees-of-freedom point mass with a time-optimal maneuvering objective.

Chapter 4 contains the core topic of this thesis: the NMPC implementation for a time-optimal control of an autonomous racecar. Both the simulation model and the model used for the NMPC are described in this chapter. Afterwards, the results of the simulation are compared to a lap driven by a human driver in the real-life racecar. The results are also compared to the theoretical optimum calculated using an Optimal Control method.

Chapter 5 gives a brief conclusion of the project and suggests topics for future work for extending the scope of this thesis. The literature sources used throughout this project are listed in References.

Finally, this document includes four appendixes. Appendix A presents the object-oriented implementation of the NMPC algorithm in MATLAB/Simulink. Appendix B describes the hardware of the real-life vehicle modelled in chapter 4. The method applied to generate the racing track from logged data is presented in Appendix C.

2 Nonlinear Model Predictive Control

In this chapter, the main topic of this thesis, Nonlinear Model Predictive Control (NMPC), is discussed. The chapter starts by introducing the general algorithm for NMPC and then progressively presents the necessary aspects for the NMPC algorithm implemented in this project, the Real-Time Iteration scheme (RTI).

NMPC is a class of controller algorithms for the feedback control of nonlinear systems. This type of controllers is based on the online solution of an optimization problem. The most common applications of NMPC are *tracking* and stabilization of the system. In this case, the task of the optimization problem is to determine the control inputs of the system so that its states follow a reference trajectory as good as possible. However, as will be discussed in section 2.4, the objective function of the problem could be virtually any quantity of the system, in which case the algorithm is known as *Economic* NMPC. Furthermore, as expressed in its name, NMPC is model-based, which means that a model of the plant is needed for the internal calculations of the algorithm.

The general implementation of a Nonlinear Model Predictive Controller in discrete time is structured as follows [36]:

1. Get the current state of the system
2. Predict and optimize the future behavior of the system on a limited window of discrete time steps (*horizon*)
3. Implement the first control input on the real actuators
4. Move the optimization horizon one step further and repeat from step 1.

This procedure is executed periodically every sampling time T_s. Due to the time horizon in step 2. and step 4., NMPC is also known as *receding horizon control*. This window contains N samples, so that the time horizon comprises the time $T_H = T_s \cdot (N-1)$.

It may be remarked that the version of NMPC for linear systems is simply known as Model Predictive Control (MPC). Most MPC applications lead to a convex optimization problem. However, the optimization problem in *Nonlinear* MPC applications is typically non-convex.

2.1 The Optimization Problem

As discussed before, Nonlinear Model Predictive Control is an optimization-based algorithm. The optimization problem in this algorithm is an Optimal Control Problem (OCP) of the form:

$$
\begin{aligned}
\min_{\boldsymbol{x},\boldsymbol{v},\boldsymbol{u}} \quad & \sum_{i=0}^{N-1} L(\boldsymbol{x}_i, \boldsymbol{v}_i, \boldsymbol{u}_i) + E(\boldsymbol{x}_N) \\
\text{subject to} \quad & \boldsymbol{x}_0 - \bar{\boldsymbol{x}}_n = \boldsymbol{0} \\
& \boldsymbol{x}_{i+1} - \boldsymbol{f}_d(\boldsymbol{x}_i, \boldsymbol{v}_i, \boldsymbol{u}_i) = \boldsymbol{0}, \qquad i = 0, \dots, N-1 \\
& \boldsymbol{g}(\boldsymbol{x}_i, \boldsymbol{v}_i, \boldsymbol{u}_i) = \boldsymbol{0}, \qquad i = 0, \dots, N-1 \\
& \boldsymbol{h}(\boldsymbol{x}_i, \boldsymbol{v}_i, \boldsymbol{u}_i) \leq \boldsymbol{0}, \qquad i = 0, \dots, N-1 \\
& \boldsymbol{r}(\boldsymbol{x}_N) \leq \boldsymbol{0}
\end{aligned}
\tag{2-1}
$$

where $\boldsymbol{x}_i$ denotes the differential states, $\boldsymbol{v}_i$ the algebraic states and $\boldsymbol{u}_i$ the controls of the system [36]. Representing the system in this form allows to cover systems given as Differential Algebraic Equations (DAE). Note that the system is provided in discrete form, where $\boldsymbol{f}_d(\boldsymbol{x}_i, \boldsymbol{v}_i, \boldsymbol{u}_i)$ contains the discrete system dynamics. Therefore, continuous time systems must be discretized using the sampling time T_s, see subsection 2.1.2. The index i denotes the values that correspond to the same timestep, where $\boldsymbol{x}_i$ is considered for the timesteps 0 to N and $\boldsymbol{v}_i$ and $\boldsymbol{u}_i$ for the timesteps 0 to $N-1$.

$L(\boldsymbol{x}_i, \boldsymbol{v}_i, \boldsymbol{u}_i)$ and $E(\boldsymbol{x}_N)$ are scalar objective functions known as stage cost and terminal cost respectively. The functions $\boldsymbol{g}(\boldsymbol{x}_i, \boldsymbol{v}_i, \boldsymbol{u}_i)$ and $\boldsymbol{h}(\boldsymbol{x}_i, \boldsymbol{v}_i, \boldsymbol{u}_i)$ include respectively the equality and inequality constraints of the system. Finally, the constraint $\boldsymbol{r}(\boldsymbol{x}_N) \leq \boldsymbol{0}$ is known as terminal constraint.

The task of the optimization problem in a NMPC is to determine the control values $\boldsymbol{\mu}(\bar{\boldsymbol{x}}_n)$, also known as feedback values, that minimize the given objective function for the current state $\bar{\boldsymbol{x}}_n$. The constraint $\boldsymbol{x}_0 - \bar{\boldsymbol{x}}_n = \boldsymbol{0}$ ensures that the initial state of the prediction $\boldsymbol{x}_0$ corresponds to the current state of the system. The following figure shows a representation of the NMPC algorithm. The values $\boldsymbol{x}_i^*$ and $\boldsymbol{u}_i^*$ represent the optimal state and control values determined by the optimization solver.

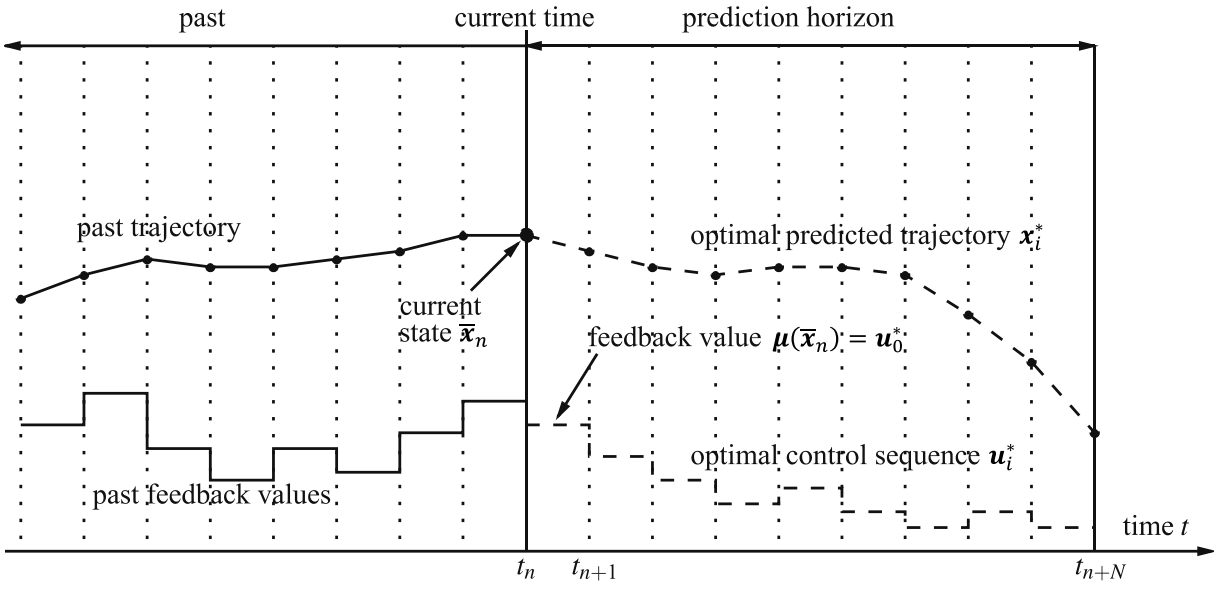

Figure 2-1: Representation of the NMPC algorithm [28]

In principle, there are two ways to solve an optimal control problem: using an *indirect method* or a *direct method*. For the indirect methods, the *Hamiltonian* of the problem is constructed and *Pontryagin's maximum principle* is used to solve the optimization. Moreover, indirect methods use continuous time dynamics. On the contrary, direct methods use discretized dynamics equations as in the problem in (2-1). In this thesis, FALCON.m was used, which is a MATLAB toolbox for solving optimal control problems with direct methods. The benefits of a direct approach compared to an indirect solution, will be clarified in the next subsection.

2.1.1 FALCON.m

FALCON.m [49] is a MATLAB toolbox developed at the institute for Flight System Dynamics of the Technical University of Munich. This toolbox provides the possibility to build and solve Optimal Control Problems using a direct method.

In general, an OCP consists of an objective function, a set of differential equations, a set of control inputs, a set of parameters and an optional set of constraints. Using the notation presented in the documentation of FALCON.m [49], an optimal control problem can be mathematically formulated as:

$$\min_{\boldsymbol{u},\boldsymbol{p}} J(\boldsymbol{x}(t),\boldsymbol{u}(t),\boldsymbol{p}) \tag{2-2}$$

where $\boldsymbol{x}(t)$ are the state variables, $\boldsymbol{u}(t)$ the controls and $\boldsymbol{p}$ the parameters of the system. These are subject to the physical constraints of the system given by the differential equations (2-3).

$$\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}(t),\boldsymbol{u}(t),\boldsymbol{p}) \tag{2-3}$$

Furthermore, $\boldsymbol{x}(t)$, $\boldsymbol{u}(t)$ and $\boldsymbol{p}$ are limited by lower and upper bounds:

$$\boldsymbol{x}_{\mathrm{lb}} \leq \boldsymbol{x}(t) \leq \boldsymbol{x}_{\mathrm{ub}} \tag{2-4}$$

$$\boldsymbol{u}_{\mathrm{lb}} \leq \boldsymbol{u}(\boldsymbol{t}) \leq \boldsymbol{u}_{\mathrm{ub}} \tag{2-5}$$

$$\boldsymbol{p}_{lb} \leq \boldsymbol{p} \leq \boldsymbol{p}_{\mathrm{ub}} \tag{2-6}$$

Additionally, the optimal control problem may be subject to nonlinear constraints of the form:

$$\boldsymbol{g}_{\mathrm{lb}} \leq \boldsymbol{g}(\boldsymbol{x}(t),\boldsymbol{u}(t),\boldsymbol{p}) \leq \boldsymbol{g}_{\mathrm{ub}} \tag{2-7}$$

These nonlinear constraints can be imposed over the entire time interval (path constraints) or only at specified time points t_j (point constraints). It can be noted that if the lower bound lb and the upper bound ub are set to the same value, equations (2-4) to (2-7) correspond to equality constraints.

The time interval considered in the OCP is $[t_0 \;\; t_f]$, where both t_0 and t_f can be either optimized as parameters or set fixed. In general, the functional J to be minimized in equation (2-1) is called a *Bolza cost function* and has the form:

$$J(\boldsymbol{x}(t),\boldsymbol{u}(t),\boldsymbol{p}) = m\big(\boldsymbol{x}(t_j),\boldsymbol{u}(t_j),\boldsymbol{p}\big) + \int_{t_0}^{t_f} l(\boldsymbol{x}(\tau),\boldsymbol{u}(\tau),\boldsymbol{p})\, d\tau \tag{2-8}$$

In this function, the expression $m\big(\boldsymbol{x}(t_j),\boldsymbol{u}(t_j),\boldsymbol{p}\big)$ is also referred to as a *Mayer cost function* and the integral term is also called a *Lagrange cost function*. It may be remarked that the Mayer cost function may contain state values and control values for multiple time points t_j.

Additionally, FALCON.m offers the possibility of constructing multiphase optimal control problems. By using this functionality, each of the phases may receive a different set of parameters, path constraints, Lagrange cost functions and even a different model.

Direct Methods

As mentioned before, FALCON.m uses a *direct method* to solve the OCP. In the direct methods, the trajectories of the states and controls of the system are discretized first. This reduces the infinite dimension of time of the trajectories to a finite one. Afterwards, a parametric optimization problem with the following Lagrange function can be built:

$$\mathcal{L}(\boldsymbol{z}, \boldsymbol{\lambda}) = J(\boldsymbol{z}) + \boldsymbol{\lambda}^{\mathrm{T}} \cdot \boldsymbol{h}(\boldsymbol{z}) \tag{2-9}$$

where the optimization variables $\boldsymbol{z}$ include the discretized state variables $\boldsymbol{x}_i$, the discretized control variables $\boldsymbol{u}_i$ and the parameters $\boldsymbol{p}$. In the equation above, $\boldsymbol{\lambda}$ is the vector of the Lagrange multipliers and $\boldsymbol{h}(\boldsymbol{z})$ represents the equality and inequality constraints, which include the limitations (2-4)-(2-6) as well as the point and path constraints (2-7) and the so called *defects*, which arise from the physical constraints (2-3), see Figure 2-3. The OCP is therefore transformed into a large parametric optimization problem.

The direct methods can be divided into multiple sorts. Here, the multiple shooting method and the collocation method are introduced. Their main difference is the number of discretized timepoints for the state variables that are included as optimization variables. FALCON.m offers the possibility of using either of the two methods.

In both methods, the control history must be discretized on a grid, which is called the *control grid*. This discretization parametrizes the control inputs $\boldsymbol{u}(t)$ by, in general, piecewise polynomial functions. To be able to evaluate the path and point constraints (2-7) in a finite number of points, the trajectories of the state variables are also discretized into the so-called *integration grid*. For simplicity, these grids are usually set equal. The following figure shows an overview of the discretization grids for the multiple shooting and collocation methods.

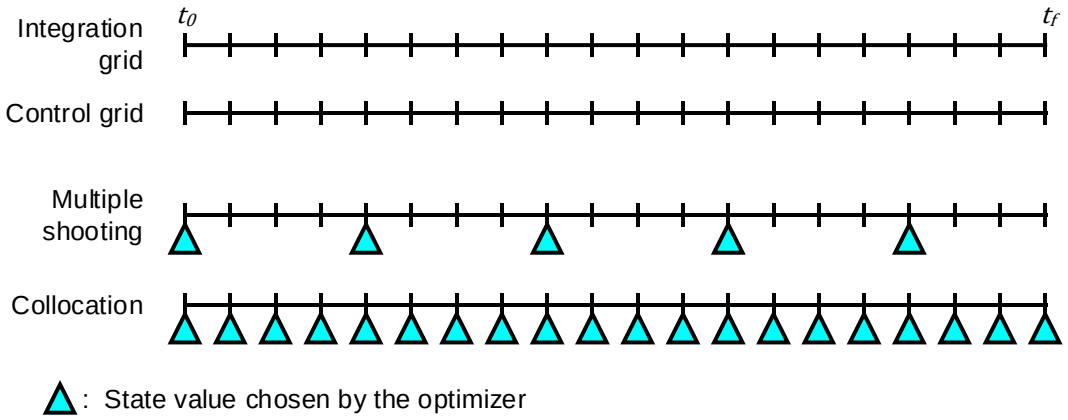

Figure 2-2: Multiple shooting vs. collocation

Multiple shooting

The multiple shooting method lets the optimizer choose the state vector at multiple points in the integration grid. These are marked by blue triangles in Figure 2-2. As stated above, the controls have to be optimized for every discretization point in the control grid. Therefore, the optimization vector $\boldsymbol{z}$ may be constructed as follows

$$\boldsymbol{z} = [t_f \quad x_0 \quad u_0 \quad u_1 \quad u_2 \quad x_3 \quad u_3 \quad u_4 \quad u_5 \quad x_6 \quad u_6 \quad u_7 \quad u_8 \quad \ldots] \tag{2-10}$$

Presented here is the case of a model with a single state variable and a single control and t_f as the only parameter that is optimized. As can be seen, in the multiple shooting method, only the values of the state variables x_0, x_3, x_6, ... can be changed directly by the optimizer. These states are called the *multiple shooting nodes*. Their values and the history of the controls are used to calculate the state variables for each point in the integration grid. This is done by numerical integration with explicit methods like the Euler forward method or another Runge-Kutta method [51]. Exemplary results of this integration can be seen in Figure 2-3.

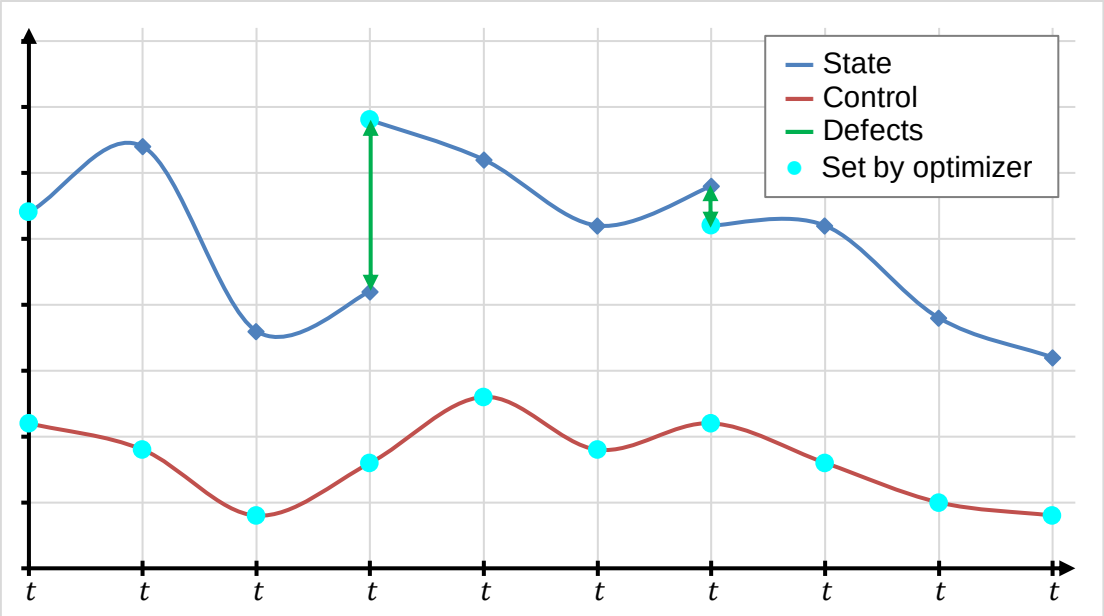

Figure 2-3: Multiple shooting defects

In this graph, the state (blue line) is integrated from t_0 till t_3, where it is again set by the optimizer. The difference between the result of the integration and the value set by the optimizer is called a *defect* and is introduced to the problem as a constraint that must vanish. This constraint is shown in the following equation, where $x_{i,end}$ represents the value of a state at the end of the integration from one multiple shooting node to the next, and $x_{i+1,ini}$ represents the value of the same state at the next multiple shooting node:

$$defect = x_{i,\mathrm{end}} - x_{i+1,\mathrm{ini}} \stackrel{!}{=} 0 \tag{2-11}$$

The values of the multiple shooting nodes and the values of the controls have a strong influence on the trajectory of the model. Therefore, if the system is unstable or if nonlinearities are involved, short integration intervals show superior local convergence [36]. Thus, the simulation is divided into multiple integrations to make the optimization process more robust. This comes at the cost of a larger optimization vector.

A special case of multiple shooting discretization where only the stating time t_0 is used as shooting node is called *single shooting*. In this case, only the initial state, the discretized controls and the parameters can be set directly by the optimization solver.

Collocation

As seen in Figure 2-2, the collocation method allows the optimizer to set the values of the state vector at each point in the integration grid. Thus, the optimization vector $\boldsymbol{z}$ for a model with one state and one control may look like:

$$\boldsymbol{z} = [t_f \quad \boldsymbol{x_0} \quad u_0 \quad \boldsymbol{x_1} \quad u_1 \quad \boldsymbol{x_2} \quad u_2 \quad \boldsymbol{x_3} \quad u_3 \quad \boldsymbol{x_4} \quad u_4 \quad \boldsymbol{x_5} \quad u_5 \quad \ldots] \tag{2-12}$$

Therefore, instead of having several integration steps, only one integration step is necessary for each discretization point. This makes the collocation method much more robust than the multiple shooting method, because instabilities in the integration are avoided to a large extent. Moreover, since the next step is known in advance, because it is set by the optimizer, implicit integration methods can be used, for example the trapezoidal method. It is also possible to interpolate the state trajectories between two points in the integration grid by Lagrange Polynomials, using Legendre/Radau integration methods [52]. The parameters of the polynomials would in this case also be part of the optimization variables, which further improves the convergence rate and reliability of the algorithm. However, collocation methods results in a larger optimization vector and more constraints than multiple shooting methods.

Using FALCON.m for NMPC

Now that FALCON.m and the basic principles behind direct optimal control have been introduced, the connection to the optimal control problem used in NMPC, equation (2-1), can be detailed. For more information on the actual implementation see Appendix A.

First, the time interval of the OCP must be fixed to represent the prediction horizon $[0\,,\ T_H]$. Furthermore, since the NMPC algorithm is executed periodically, the control grid should be discretized taking the sampling time T_s into account. Moreover, the integration grid is kept equal to the control grid.

Using discretized controls and state values, the Bolza cost function (2-8) shows similarities to the cost function in (2-1). The terminal cost can be represented by a Mayer cost function using the final state values $\boldsymbol{x}_N = \boldsymbol{x}(T_H)$. Moreover, in FALCON.m, the integral of the Lagrange cost function is evaluated using a trapezoidal approximation:

$$\int_0^{T_H} l(\boldsymbol{x}(\tau), \boldsymbol{u}(\tau), \boldsymbol{p})\, d\tau \approx \frac{1}{2} \cdot \sum_{i=0}^{N-1} \left(l(\boldsymbol{x}_i, \boldsymbol{u}_i, \boldsymbol{p}) + l(\boldsymbol{x}_{i+1}, \boldsymbol{u}_{i+1}, \boldsymbol{p}) \right) \tag{2-13}$$

This term can be reformulated to:

$$\frac{1}{2} \cdot l(\boldsymbol{x}_0, \boldsymbol{u}_0, \boldsymbol{p}) + \sum_{i=1}^{N-1} \left(l(\boldsymbol{x}_i, \boldsymbol{u}_i, \boldsymbol{p}) + l(\boldsymbol{x}_{i+1}, \boldsymbol{u}_{i+1}, \boldsymbol{p}) \right) + \frac{1}{2} \cdot l(\boldsymbol{x}_N, \boldsymbol{u}_N, \boldsymbol{p}) \tag{2-14}$$

Therefore, the original objective function in (2-1) can be represented equivalently in FALCON.m by adding the term $\frac{1}{2} \cdot l(\boldsymbol{x}_0, \boldsymbol{u}_0, \boldsymbol{p}) + \frac{1}{2} \cdot l(\boldsymbol{x}_N, \boldsymbol{u}_N, \boldsymbol{p})$ as Mayer costs.

The constraints for the discrete system dynamics in (2-1) are comparable to the defects described above. However, the controls that are actually implemented on the actuators of the system, are assumed constant during each timestep, i.e. a Zero Order Hold (ZOH) is assumed

(see Figure 2-1). This should be considered in the discretization of the dynamics, which will be covered in the next subsection.

In FALCON.m, the algebraic variables $\boldsymbol{v}_i$ can be regarded as controls that are subject to some algebraic equality constraints $\boldsymbol{g}(\boldsymbol{x}_i, \boldsymbol{v}_i, \boldsymbol{u}_i) = \boldsymbol{0}$, see section 3.3 for an example. Therefore, the algebraic variables will be neglected for the rest of this chapter. Furthermore, all equality and inequality constraints in the problem (2-1) can be represented as path or point constraints (2-7) in FALCON.m.

The parameters $\boldsymbol{p}$ are not directly part of the OCP (2-1) and are therefore not considered in this project. However, a constant parameter set from outside of the NMPC could be represented as a state x with no dynamics, i.e. $\dot{x} = 0$. This parameter would thus be part of the initial value embedding, discussed in section 0.

2.1.2 Discretization and Algorithmic Differentiation

As mentioned before, the continuous time model of the system (2-3) needs to be discretized in order to be handled numerically in the optimization. This is done by numerical integration methods, also known as *integrators [36]*. These methods can be categorized into two groups: *explicit*, which only use the previous state values, and *implicit*, which require information about future state values.

The optimization solver requires also the derivatives of the numerical integration. The calculation of these derivatives is known as *algorithmic differentiation* of the integrators. It is important to note that, if an exact Hessian is used for the optimization (see section 2.2.3), the second order derivatives will also be needed. For this project, two explicit numerical integrators were implemented for FALCON.m: The *Forward Euler* method and the *4th-order Runge-Kutta (ERK4)* method.

Forward Euler

The Forward Euler method has the form:

$$\boldsymbol{x}_{i+1} = \boldsymbol{x}_i + T_s \cdot \boldsymbol{f}(\boldsymbol{x}_i, \boldsymbol{u}_i) \tag{2-15}$$

where $\boldsymbol{x}_i$ and $\boldsymbol{x}_{i+1}$ represent the current and the next state respectively, $\boldsymbol{f}(\boldsymbol{x}_i, \boldsymbol{u}_i)$ is the vector-valued function that contains the system dynamics and T_s is the stepsize for the integration, which is set equal to the sampling time in an NMPC scheme. The defect constraints can therefore be formulated as:

$$\boldsymbol{def}_i = \boldsymbol{x}_{i+1} - \boldsymbol{x}_i - T_s \cdot \boldsymbol{f}(\boldsymbol{x}_i, \boldsymbol{u}_i) = \boldsymbol{0} \tag{2-16}$$

with the derivatives:

$$\begin{aligned} \nabla_{\boldsymbol{x}_{i+1}} \boldsymbol{def}_i &= \boldsymbol{I} \\ \nabla_{\boldsymbol{x}_i} \boldsymbol{def}_i &= -\boldsymbol{I} - T_s \cdot \nabla_{\boldsymbol{x}_i} \boldsymbol{f}(\boldsymbol{x}_i, \boldsymbol{u}_i) \\ \nabla_{\boldsymbol{u}_i} \boldsymbol{def}_i &= -T_s \cdot \nabla_{\boldsymbol{u}_i} \boldsymbol{f}(\boldsymbol{x}_i, \boldsymbol{u}_i) \end{aligned} \tag{2-17}$$

The second derivatives are provided by:

$$\begin{aligned}
\nabla^2_{\boldsymbol{x}_{i+1}\boldsymbol{x}_{i+1}} \boldsymbol{def}_i &= \mathbf{0} \\
\nabla^2_{\boldsymbol{x}_i\,\boldsymbol{x}_i} \quad \boldsymbol{def}_i &= -T_s \cdot \nabla^2_{\boldsymbol{x}_i\,\boldsymbol{x}_i} \boldsymbol{f}(\boldsymbol{x}_i, \boldsymbol{u}_i) \\
\nabla^2_{\boldsymbol{u}_i\,\boldsymbol{u}_i} \quad \boldsymbol{def}_i &= -T_s \cdot \nabla^2_{\boldsymbol{u}_i\,\boldsymbol{u}_i} \boldsymbol{f}(\boldsymbol{x}_i, \boldsymbol{u}_i) \\
\nabla^2_{\boldsymbol{x}_i\,\boldsymbol{u}_i} \quad \boldsymbol{def}_i &= -T_s \cdot \nabla^2_{\boldsymbol{x}_i\,\boldsymbol{u}_i} \boldsymbol{f}(\boldsymbol{x}_i, \boldsymbol{u}_i) \\
\nabla^2_{\boldsymbol{u}_i\,\boldsymbol{x}_i} \quad \boldsymbol{def}_i &= \nabla^2_{\boldsymbol{x}_i\,\boldsymbol{u}_i} \boldsymbol{def}_i{}^{\mathrm{T}}
\end{aligned} \tag{2-18}$$

Explicit 4th-order Runge-Kutta

With the explicit 4th-order Runge-Kutta method, the defects and their derivatives can be calculated by the following algorithm [36]:

Algorithm 1: Explicit 4th-order Runge-Kutta (ERK4)

Input: $\boldsymbol{x}_i$ and $\boldsymbol{u}_i$

$a_1 = 0\,;\ a_2 = {}^1\!/_2\,;\ a_3 = {}^1\!/_2\,;\ a_4 = 1$

$\boldsymbol{k}_0 = \mathbf{0}\,;\ \boldsymbol{k}_{x,0} = \mathbf{0}\,;\ \boldsymbol{k}_{u,0} = \mathbf{0}$

for $j = 1, \dots, 4$

$$\begin{aligned}
\boldsymbol{k}_j &= \boldsymbol{f}\left(\boldsymbol{x}_i + a_j T_s \boldsymbol{k}_{j-1}, \boldsymbol{u}_i\right) \\
\boldsymbol{k}_{x,j} &= \nabla_{\boldsymbol{x}_i} \boldsymbol{f}\left(\boldsymbol{x}_i + a_j T_s \boldsymbol{k}_{j-1}, \boldsymbol{u}_i\right) \cdot \left(\boldsymbol{I} + a_j T_s \boldsymbol{k}_{x,j-1}\right) \\
\boldsymbol{k}_{u,j} &= \nabla_{\boldsymbol{u}_i} \boldsymbol{f}\left(\boldsymbol{x}_i + a_j T_s \boldsymbol{k}_{j-1}, \boldsymbol{u}_i\right) + \nabla_{\boldsymbol{x}_i} \boldsymbol{f}\left(\boldsymbol{x}_i + a_j T_s \boldsymbol{k}_{j-1}, \boldsymbol{u}_i\right) \cdot a_j T_s \boldsymbol{k}_{u,j-1}
\end{aligned}$$

end for

$$\boldsymbol{def}_i = \boldsymbol{x}_{i+1} - \boldsymbol{x}_i - {}^{T_s}\!/_6 \cdot \left(\boldsymbol{k}_1 + 2\boldsymbol{k}_2 + 2\boldsymbol{k}_3 + \boldsymbol{k}_4\right) = \mathbf{0}$$

$$\begin{aligned}
\nabla_{\boldsymbol{x}_{i+1}} \boldsymbol{def}_i &= \boldsymbol{I} \\
\nabla_{\boldsymbol{x}_i} \quad \boldsymbol{def}_i &= -\boldsymbol{I} - {}^{T_s}\!/_6 \cdot \left(\boldsymbol{k}_{x,1} + 2\boldsymbol{k}_{x,2} + 2\boldsymbol{k}_{x,3} + \boldsymbol{k}_{x,4}\right) \\
\nabla_{\boldsymbol{u}_i} \quad \boldsymbol{def}_i &= -{}^{T_s}\!/_6 \cdot \left(\boldsymbol{k}_{u,1} + 2\boldsymbol{k}_{u,2} + 2\boldsymbol{k}_{u,3} + \boldsymbol{k}_{u,4}\right)
\end{aligned}$$

This algorithm must be repeated for the timesteps $i = 0, \dots, N-1$. The second derivatives could also be calculated in a similar fashion. However, the second derivatives are used when building the *Hessian* of the Lagrange function (2-9), which includes a term of the form $\boldsymbol{\lambda}^T \nabla^2_{\boldsymbol{z}} \boldsymbol{h}(\boldsymbol{z})$, where $\boldsymbol{\lambda}$ is the vector of the Lagrange multipliers and $\boldsymbol{h}(\boldsymbol{z})$ represents the equality constraints (including the defects) and the active inequality constraints. Therefore, for an accurate result of the Hessian, the Lagrange multipliers $\boldsymbol{\lambda}$ corresponding to the defects would also need to be propagated with a similar recursion to the one in Algorithm 1. These Lagrange multipliers converge to the *adjoint variables* of the continuous system. A proper algorithm for the Hessian calculation can be found in [50], its implementation was, however, out of the scope of this thesis.

It may be remarked that the control grid in FALCON.m includes the control inputs at the end of the time horizon $\boldsymbol{u}_N$ (in the case of algebraic variables also $\boldsymbol{v}_N$). These values are not included in the problem (2-1), because Zero Order Hold of the controls is assumed. Therefore, these values should be removed from the optimization variables.

Trapezoidal

As a final comment, the *Trapezoidal* method, which is an implicit integrator, is the default discretization method of FALCON.m. Implicit integrators show a superior performance for stiff systems and are simple to implement for Optimal Control Problems [36]. However, the Trapezoidal method (equation (2-19)) does not assume Zero Order Hold of the controls. This can be appreciated by the fact that $\boldsymbol{u}_{i+1}$ is used:

$$\boldsymbol{x}_{i+1} = \boldsymbol{x}_i + \frac{T_s}{2} \cdot \left(\boldsymbol{f}(\boldsymbol{x}_i, \boldsymbol{u}_i) + \boldsymbol{f}(\boldsymbol{x}_{i+1}, \boldsymbol{u}_{i+1}) \right) \tag{2-19}$$

To tackle this issue when using the Trapezoidal method, the following equation to calculate the control inputs $\boldsymbol{\mu}(\overline{\boldsymbol{x}}_n)$ was implemented in this project:

$$\boldsymbol{\mu}(\overline{\boldsymbol{x}}_n) = \frac{1}{2} \cdot (\boldsymbol{u}_0^* + \boldsymbol{u}_1^*) \tag{2-20}$$

where $\boldsymbol{u}_0^*$ and $\boldsymbol{u}_1^*$ are the optimal control values of the first two timesteps. Furthermore, since $\frac{1}{2} \cdot \boldsymbol{u}_1^*$ is applied to the actuators "prematurely", an additional constraint $\boldsymbol{u}_0 = \boldsymbol{u}_1^*$ is inserted to the OCP. This approach was tested with two simulation models and will be discussed further in sections 3.1 and 3.3.

2.1.3 Shift – Online Initialization

To solve the Optimal Control Problem (2-1) in a fast and reliable manner, an adequate *initial guess* should be provided for the optimization [33]. Since the prediction horizon is "shifted" periodically every sampling time T_s, then an obvious initial guess is to use the values of the previous optimal solution:

$$\begin{aligned} \boldsymbol{x}_i^{\text{guess}} &= \boldsymbol{x}_{i+1}^*, \quad i = 0, \dots, N-1 \\ \boldsymbol{u}_i^{\text{guess}} &= \boldsymbol{u}_{i+1}^*, \quad i = 0, \dots, N-2 \end{aligned} \tag{2-21}$$

The values for the last timestep $\boldsymbol{x}_N^{\text{guess}}$ can be calculated by forward simulation using an explicit integration method:

$$\boldsymbol{x}_N^{\text{guess}} = \boldsymbol{f}_d\left(\boldsymbol{x}_{N-1}^{\text{guess}}, \boldsymbol{u}_{N-1}^{\text{guess}}\right) \tag{2-22}$$

This ensures that the defect constraints are satisfied. For selecting the last control values $\boldsymbol{u}_{N-1}^{guess}$, several strategies are possible:

- If there is a known control law $\boldsymbol{\mu}\left(\boldsymbol{x}_{N-1}^{\text{guess}}\right)$ that stabilizes the system for the given system state, this control law could be used: $\boldsymbol{u}_{N-1}^{\text{guess}} = \boldsymbol{\mu}\left(\boldsymbol{x}_{N-1}^{\text{guess}}\right)$
- A simpler approach is to copy the previous control values: $\boldsymbol{u}_{N-1}^{\text{guess}} = \boldsymbol{u}_{N-2}^{\text{guess}} = \boldsymbol{u}_{N-1}^*$

An initial guess of the Lagrange multipliers $\boldsymbol{\lambda}^{guess}$ can also be provided to many optimization solvers. As for the optimization variables $\boldsymbol{x}_i$ and $\boldsymbol{u}_i$, providing an initial guess of the Lagrange multipliers can boost the performance of the algorithm. Furthermore, as mentioned before, the Lagrange Multipliers are needed for calculating the exact Hessian of the OCP. For their initial guess, the Lagrange multipliers corresponding to the defects and the path constraints can also be shifted as in (2-21). For their last timestep, these Lagrange multipliers can be either copied

from the previous one or set to zero. For the terminal constraint, the best initial guess is the one corresponding to the last optimal solution.

2.1.4 Stability

Proving stability of nonlinear systems, including closed-loop systems, can be a difficult task. This is especially true if a complex controller, like NMPC, is used. Here, a simple proof for a closed-loop system using Tracking NMPC is provided [36].

Here, it is assumed that the system is provided with a steady-state and feasible setpoint $\boldsymbol{x}^{\mathrm{ref}}$, i.e. $f(\boldsymbol{x}^{\mathrm{ref}}, \boldsymbol{u}^*) = \mathbf{0}$. Through a transformation, this setpoint and the controls $\boldsymbol{u}^*$ can be set to zero without loss of generality. A typical objective function for Tracking NMPC is $J(\boldsymbol{x}, \boldsymbol{u}) = \boldsymbol{x}^{\mathrm{T}}\boldsymbol{Q}\boldsymbol{x} + \boldsymbol{u}^{\mathrm{T}}\boldsymbol{R}\boldsymbol{u}$ with positive definite matrices $\boldsymbol{Q}$ and $\boldsymbol{R}$. Furthermore, an equality terminal constraint $\boldsymbol{x}_N = \boldsymbol{x}^{\mathrm{ref}} = \mathbf{0}$ is included in the problem to ensure that the system reaches the setpoint. At the timestep t_n the solution $\boldsymbol{\mu}(\bar{\boldsymbol{x}}_n)$ of the Optimal Control Problem is implemented to the system actuators. Provided that there are no model mismatch and no disturbances, the prediction $\boldsymbol{x}_1^*$ matches the new actual state $\bar{\boldsymbol{x}}_{n+1}$. Therefore, the shifted prediction horizon with $\boldsymbol{u}_{N-1} = \mathbf{0}$ and $\boldsymbol{x}_N = \mathbf{0}$ is already the new optimal solution according to the objective function $J(\boldsymbol{x}, \boldsymbol{u})$. This would be repeated every timestep T_s until the setpoint is reached. In that case, the system is and stays in steady-state, as the objective function is at the (global) minimum.

More detailed stability proofs, also without the terminal equality constraint and for Economic NMPC, can be found in [28].

2.2 Newton-Type Optimization

In this section, the Newton-type optimization solvers will be introduced briefly. An Optimal Control Problem can be transcribed to a nonlinear optimization problem. Hence, it must be solved using *Nonlinear Programming* (NLP). A nonlinear problem has the following form:

$$\begin{aligned} \min_{\boldsymbol{z}} \quad & f(\boldsymbol{z}) \\ \text{subject to} \quad & \boldsymbol{g}(\boldsymbol{z}) = \mathbf{0} \\ & \boldsymbol{h}(\boldsymbol{z}) \leq \mathbf{0} \end{aligned} \tag{2-23}$$

Here, the function $f(\boldsymbol{z})$ is called the *objective* or *cost function*, $\boldsymbol{g}(\boldsymbol{z})$ and $\boldsymbol{h}(\boldsymbol{z})$ are respectively the equality and inequality constraints. All $f(\boldsymbol{z})$, $\boldsymbol{g}(\boldsymbol{z})$ and $\boldsymbol{h}(\boldsymbol{z})$ must be twice continuously differentiable. First-order necessary conditions for a local minimizer $\boldsymbol{z}^*$ are [36, 53]:

$$\begin{aligned} \nabla_{\boldsymbol{z}}\, \mathcal{L}(\boldsymbol{z}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*) &= \mathbf{0} \\ \boldsymbol{g}(\boldsymbol{z}^*) &= \mathbf{0} \\ \boldsymbol{h}(\boldsymbol{z}^*) &\leq \mathbf{0} \\ \boldsymbol{\mu}^* &\geq \mathbf{0} \\ \mu_i^*\, h_i(\boldsymbol{z}^*) &= 0, \quad \text{for all } \boldsymbol{h}(\boldsymbol{z}) \end{aligned} \tag{2-24}$$

where the *Lagrange function* of this problem is formulated as (compare to equation (2-9)):

$$\mathcal{L}(\boldsymbol{z}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = f(\boldsymbol{z}) + \boldsymbol{\lambda}^{\mathrm{T}}\, \boldsymbol{g}(\boldsymbol{z}) + \boldsymbol{\mu}^{\mathrm{T}}\, \boldsymbol{h}(\boldsymbol{z}) \tag{2-25}$$

with the derivative:

$$\nabla_{\boldsymbol{z}} \mathcal{L}(\boldsymbol{z}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = \nabla_{\boldsymbol{z}} f(\boldsymbol{z}) + \nabla_{\boldsymbol{z}} \boldsymbol{g}(\boldsymbol{z})\, \boldsymbol{\lambda} + \nabla_{\boldsymbol{z}} \boldsymbol{h}(\boldsymbol{z})\, \boldsymbol{\mu} \tag{2-26}$$

Here, $\boldsymbol{\lambda}$ and $\boldsymbol{\mu}$ are called the *Lagrange multipliers* or *dual variables*. The conditions (2-24) are also known as the Karush-Kuhn-Tucker (KKT) conditions of optimality. Second-order necessary conditions can be found in [53].

Quadratic Programming (QP) solvers solve a special case of nonlinear problems, *quadratic problems*. These problems have a quadratic cost function and $\boldsymbol{g}(\boldsymbol{z})$ and $\boldsymbol{h}(\boldsymbol{z})$ are affine. Therefore, a quadratic problem has the form:

$$\begin{aligned} \min_{\boldsymbol{z}} \quad & \boldsymbol{c}^{\mathrm{T}}\boldsymbol{z} + \frac{1}{2}\boldsymbol{z}^{\mathrm{T}}\, \mathrm{B}\, \boldsymbol{z} \\ \text{subject to} \quad & \boldsymbol{A}\boldsymbol{z} - \boldsymbol{b} = \boldsymbol{0} \\ & \boldsymbol{C}\boldsymbol{z} - \boldsymbol{d} \leq \boldsymbol{0} \end{aligned} \tag{2-27}$$

As mentioned before, this section is only a brief introduction to Newton-type optimization. Topics like line-search algorithms, merit functions or Trust-Region methods will not be discussed here. These and other subjects are treated in detail in [53], for example.

2.2.1 Equality Constrained Optimization

The main idea of the Newton-type optimization methods, in the case of no inequality constraints, is to use Newton's method to find points that solve the nonlinear KKT conditions [36]:

$$\begin{aligned} \nabla_{\boldsymbol{z}} \mathcal{L}(\boldsymbol{z}^*, \boldsymbol{\lambda}^*) &= 0 \\ \boldsymbol{g}(\boldsymbol{z}^*) &= 0 \end{aligned} \tag{2-28}$$

Starting with an initial guess $\boldsymbol{w}_0$, *Newton's method* can be employed to find the zero-crossings of the vector-valued function $\boldsymbol{F}(\boldsymbol{w})$ by iterating:

$$\boldsymbol{w}_{k+1} = \boldsymbol{w}_k - \left(\frac{\partial \boldsymbol{F}(\boldsymbol{w}_k)}{\partial \boldsymbol{w}_k} \right)^{-1} \boldsymbol{F}(\boldsymbol{w}_k) \tag{2-29}$$

For an equality constrained optimization, this algorithm looks as follows:

Algorithm 2: Newton-type equality constrained optimization

Input: initial guess $\boldsymbol{z}_0$ and $\boldsymbol{\lambda}_0$, tolerance ϵ

$k = 0$

while $\|\nabla_{\boldsymbol{z}} \mathcal{L}(\boldsymbol{z}_k, \boldsymbol{\lambda}_k)\| \geq \epsilon$ or $\|\boldsymbol{g}(\boldsymbol{z}_k)\| \geq \epsilon$

get $\boldsymbol{z}_{k+1}$ and $\boldsymbol{\lambda}_{k+1}$ from:

$$\begin{bmatrix} \boldsymbol{z}_{k+1} \\ \boldsymbol{\lambda}_{k+1} \end{bmatrix} = \begin{bmatrix} \boldsymbol{z}_k \\ 0 \end{bmatrix} - \begin{bmatrix} \nabla^2_{\boldsymbol{z}\boldsymbol{z}} \mathcal{L}(\boldsymbol{z}_k, \boldsymbol{\lambda}_k) & \nabla_{\boldsymbol{z}} \boldsymbol{g}(\boldsymbol{z}_k) \\ \nabla_{\boldsymbol{z}} \boldsymbol{g}^T(\boldsymbol{z}_k) & 0 \end{bmatrix}^{-1} \begin{bmatrix} \nabla_{\boldsymbol{z}} f(\boldsymbol{z}_k) \\ \boldsymbol{g}(\boldsymbol{z}_k) \end{bmatrix} \tag{2-30}$$

$k = k + 1$

end while

The Hessian of the Lagrange function $\mathrm{H}_k = \nabla^2_{\boldsymbol{zz}}\, \mathcal{L}(\boldsymbol{z}_k, \boldsymbol{\lambda}_k)$ in equation (2-30) might be difficult to compute, therefore, it is usually approximated. Methods for approximating the Hessian will be covered in section 2.2.3. The matrix

$$\begin{bmatrix} \nabla^2_{\boldsymbol{zz}}\, \mathcal{L}(\boldsymbol{z}_k, \boldsymbol{\lambda}_k) & \nabla_{\boldsymbol{z}} \boldsymbol{g}(\boldsymbol{z}_k) \\ \nabla_{\boldsymbol{z}} \boldsymbol{g}^T(\boldsymbol{z}_k) & 0 \end{bmatrix} \tag{2-31}$$

is known as the KKT matrix and is in general symmetric indefinite.

It can be shown that one iteration of Algorithm 2 is equivalent to solving the following quadradic problem, see [36]:

$$\begin{aligned} \min_{\boldsymbol{z}} \quad & \nabla_{\mathbf{z}} f(\boldsymbol{z}_k)^{\mathrm{T}}\, (\boldsymbol{z} - \boldsymbol{z}_k) + \frac{1}{2} (\boldsymbol{z} - \boldsymbol{z}_k)^{\mathrm{T}}\, \mathrm{H}_k\, (\boldsymbol{z} - \boldsymbol{z}_k) \\ \text{subject to} \quad & \boldsymbol{g}(\boldsymbol{z}_k) + \nabla_{\mathbf{z}} \boldsymbol{g}(\boldsymbol{z}_k)^{\mathrm{T}}\, (\boldsymbol{z} - \boldsymbol{z}_k) = 0 \end{aligned} \tag{2-32}$$

2.2.2 Handling Inequality Constraints

For solving a nonlinear problem with inequality constraints, the *complementarity* conditions must be taken into account:

$$\begin{aligned} \boldsymbol{h}(\boldsymbol{z}^*) &\leq \mathbf{0} \\ \boldsymbol{\mu}^* &\geq \mathbf{0} \\ \mu_i^*\, h_i(\boldsymbol{z}^*) &= 0, \quad \text{for all } \boldsymbol{h}(\boldsymbol{z}) \end{aligned} \tag{2-33}$$

These KKT conditions are non-smooth, and there are different methods to deal with them. These methods will be presented here.

Interior-Point methods

The idea of the interior-point methods is to smoothen the complementarity conditions by inserting a constant $\tau > 0$. The complementarity conditions are then replaced by:

$$\mu_i^*\, h_i(\boldsymbol{z}^*) + \tau = 0, \quad \text{for all } \boldsymbol{h}(\boldsymbol{z}) \tag{2-34}$$

It can be noted that this equality is equivalent to the hyperbola $\mu_i^* = -\frac{\tau}{h_i(\boldsymbol{z}^*)}$. Therefore, by observing that $\frac{\tau}{h_i(\boldsymbol{z}^*)} \cdot \nabla_{\boldsymbol{z}} h_i(\boldsymbol{z}) = \tau\, \nabla_{\boldsymbol{z}} \log\big(-h_i(\boldsymbol{z})\big)$, this new complementarity condition can be interpreted as replacing the inequality constraints with an additional term in the cost function (see the first KKT condition in (2-24) and equation (2-26)):

$$\begin{aligned} \min_{\boldsymbol{z}} \quad & f(\boldsymbol{z}) - \tau \sum \log\big(-h_i(\boldsymbol{z})\big) \\ \text{subject to} \quad & \boldsymbol{g}(\boldsymbol{z}) = \mathbf{0} \end{aligned} \tag{2-35}$$

Normally, the optimization is started with a large value τ, which is then reduced as the iterations advance. It can be noted that the new objective function tends towards infinity when $h_i(\boldsymbol{z}) \to 0$. Therefore, even for a small τ this objective function prevents the inequality constraints from being violated.

Sequential Quadratic Programming

Sequential Quadratic Programming (SQP) is based on the quadratic problem interpretation (2-32). In each iteration, this approach solves an inequality constrained QP problem that is obtained by linearizing the constraint functions and approximating the objective function by a quadratic function:

$$\begin{aligned} \min_{\boldsymbol{z}} \quad & \nabla_{\mathbf{z}} f(\boldsymbol{z}_k)^{\mathrm{T}} (\boldsymbol{z} - \boldsymbol{z}_k) + \frac{1}{2} (\boldsymbol{z} - \boldsymbol{z}_k)^{\mathrm{T}} \mathrm{H}_k (\boldsymbol{z} - \boldsymbol{z}_k) \\ \text{subject to} \quad & \boldsymbol{g}(\boldsymbol{z}_k) + \nabla_{\mathbf{z}} \boldsymbol{g}(\boldsymbol{z}_k)^{\mathrm{T}} (\boldsymbol{z} - \boldsymbol{z}_k) = 0 \\ & \boldsymbol{h}(\boldsymbol{z}_k) + \nabla_{\mathbf{z}} \boldsymbol{h}(\boldsymbol{z}_k)^{\mathrm{T}} (\boldsymbol{z} - \boldsymbol{z}_k) \leq 0 \end{aligned} \tag{2-36}$$

The method to solve the quadratic problem depends however on the implementation. QP solvers usually use either an interior-point method or an active set method (described next).

Active Set methods

The idea in active set methods is that, if the set of active inequality constraints (the inequality constraints where $h_i(\boldsymbol{z}) \stackrel{!}{=} 0$), i.e. the *active set*, is known, then one can directly solve an equality constrained optimization. Thus, these methods iteratively refine their guess of the active set, often called the *working set*, and solve an equality constrained optimization every iteration.

This equality constrained optimization is particularly easy to solve in the case of affine inequality constraints, as in quadratic problems of the form (2-27). Moreover, an active set method can be easily warm-started, if there is a series of related problems to be solved. This is the case for Sequential Quadratic Programming and for problems in the context of model predictive control [36].

2.2.3 About the Hessian

In this subsection, a few remarks on the Hessian H_k of the Lagrange function will be made. As mentioned before, the Hessian is necessary to solve an optimization problem with a Newton-type method, regardless of the approach used to handle the inequality constraints. For a general inequality constrained nonlinear optimization, the Hessian is defined as:

$$\mathrm{H}_k \coloneqq \nabla^2_{\boldsymbol{zz}} \mathcal{L}(\boldsymbol{z}_k, \boldsymbol{\lambda}_k, \boldsymbol{\mu}_k) \tag{2-37}$$

see equation (2-26). However, the calculation of this matrix is sometimes difficult, and it is therefore often approximated by a matrix $B_k \approx \mathrm{H}_k$, which will be discussed in this subsection.

In general, the Hessian is symmetric and indefinite. Note that in a quadratic problem (2-27) the eigenvalues of the Hessian $\mathrm{H}_k = \mathrm{B}$ make the problem convex or non-convex, i.e. if it is possible to solve the problem in polynomial time [36]. If B is positive semi-definite ($\mathrm{B} \succcurlyeq 0$), then the problem is convex and can be solved, and if B is positive definite ($\mathrm{B} \succ 0$), then the problem is strictly convex and always has a unique minimizer $\boldsymbol{z}^*$.

It may be remarked that for an Optimal Control Problem of the form (2-1), the Lagrange function is separable [52]:

$$\mathcal{L}(\boldsymbol{z}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = \sum_{i=0}^{N} \mathcal{L}_i(\boldsymbol{z}_i, \boldsymbol{\lambda}, \boldsymbol{\mu}) \tag{2-38}$$

Therefore, the Hessian has a block-diagonal form:

$$\mathrm{H} = \nabla^2_{\boldsymbol{zz}} \mathcal{L}(\boldsymbol{z}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = \begin{bmatrix} \nabla^2_{\boldsymbol{z}_0 \boldsymbol{z}_0} \mathcal{L}_0(\boldsymbol{z}_0, \boldsymbol{\lambda}, \boldsymbol{\mu}) & 0 & 0 & 0 \\ 0 & \nabla^2_{\boldsymbol{z}_1 \boldsymbol{z}_1} \mathcal{L}_1(\boldsymbol{z}_1, \boldsymbol{\lambda}, \boldsymbol{\mu}) & 0 & 0 \\ 0 & 0 & \dots & 0 \\ 0 & 0 & 0 & \nabla^2_{\boldsymbol{z}_N \boldsymbol{z}_N} \mathcal{L}_N(\boldsymbol{z}_N, \boldsymbol{\lambda}, \boldsymbol{\mu}) \end{bmatrix} \tag{2-39}$$

The KKT matrix also has a similar form. Note that, for the sake of readability, the iteration index k has been omitted above. In this thesis, the Hessian blocks are represented by $\mathrm{H}_{k,i} = \nabla^2_{\boldsymbol{z}_i \boldsymbol{z}_i} \mathcal{L}_i(\boldsymbol{z}_{k,i}, \boldsymbol{\lambda}_k, \boldsymbol{\mu}_k)$ for $i = 0, \dots, N$. It is important to remark that $\mathcal{L}_0(\boldsymbol{z}_0, \boldsymbol{\lambda}, \boldsymbol{\mu})$, $\mathcal{L}_N(\boldsymbol{z}_N, \boldsymbol{\lambda}, \boldsymbol{\mu})$ and $\mathcal{L}_i(\boldsymbol{z}_i, \boldsymbol{\lambda}, \boldsymbol{\mu}) \; \forall \; i = 1, \dots, N-1$ have a different structure, due to the initial value constraints and the terminal constraints.

Exact Hessian

An obvious way of obtaining the Hessian is to calculate the second derivative of the Lagrange function directly: $\mathrm{B}_k = \mathrm{H}_k = \nabla^2_{\boldsymbol{zz}} \mathcal{L}(\boldsymbol{z}_k, \boldsymbol{\lambda}_k, \boldsymbol{\mu}_k)$. For this, *algorithmic differentiation* is used, as introduced in section 2.1.2.

Also, instead of calculating the Hessian directly, it could also be calculated by finite differences. However, the accuracy of finite differences may be poor, especially if finite differences is also used to calculate the first derivatives [36].

Since the exact Hessian is in general indefinite, it needs to be *regularized*, i.e. one needs to get a positive semidefinite approximation of the exact Hessian, to be able to solve the nonlinear problem. This is important for both interior-point and SQP methods [54].

Regularization methods

Regularization or convexification methods are used on the Hessian matrix, to make the optimization problem convex, and thus be able to solve it. In this project, two regularization methods presented in [54] were implemented, $\mathrm{project}$ and mirror:

$$\mathrm{B}_{k,i} = \mathrm{project}\big(\mathrm{H}_{k,i}\big) \coloneqq V_i \; [\; \max(\epsilon, D_i) \;]\; V_i^{-1} \tag{2-40}$$

$$\mathrm{B}_{k,i} = \mathrm{mirror}\big(\mathrm{H}_{k,i}\big) \coloneqq V_i \; [\; \max(\epsilon, \mathrm{abs}(D_i)) \;]\; V_i^{-1} \tag{2-41}$$

where $\mathrm{H}_{k,i} = V_i \; D_i \; V_i^{-1}$ is the eigenvalue decomposition of each Hessian block. This means that the eigenvalues of $\mathrm{H}_{k,i}$ are displayed on the diagonal matrix D_i , and V_i is only a transformation matrix. In (2-40) and (2-41) the $\max$ function is performed elementwise to the diagonal matrix D_i and ϵ is a small positive number.

Therefore, these regularization methods ensure that the Hessian becomes positive definite, by making all of its eigenvalues positive. In $\mathrm{project}$ the negative eigenvalues are “projected” to a

small positive value. In mirror their absolute value is taken into account so that their magnitude stays the same.

In [54], another convexification method is presented, which shows benefits in the context of OCPs. This procedure was however not implemented in this thesis. The authors in [31] also suggest that the regularization methods above can be applied to the full-Hessian or to the *condensed* Hessian (see subsection 2.2.5), instead of to each block separately.

Gauss-Newton

The Gauss-Newton method can be used for a special class of optimization problems that have an objective function of the form $f(\boldsymbol{z}) = \frac{1}{2}\|\boldsymbol{R}(\boldsymbol{z})\|_2^2$ [36]. This kind of objective function is given in many applications, for example in least-squares parameter fitting or in *tracking NMPC*.

One can see that the quadratic approximation of this objective function at iterate $\boldsymbol{z}_k$ results in a convex quadratic function:

$$\begin{aligned} \frac{1}{2}\|\boldsymbol{R}(\boldsymbol{z})\|_2^2 \;&\approx\; \frac{1}{2}\left\|\boldsymbol{R}(\boldsymbol{z}_k) + \nabla_{\boldsymbol{z}}\boldsymbol{R}(\boldsymbol{z}_k)^{\mathrm{T}}(\boldsymbol{z}-\boldsymbol{z}_k)\right\|_2^2 \;= \\ \underbrace{\frac{1}{2}\boldsymbol{R}(\boldsymbol{z}_k)^{\mathrm{T}}\boldsymbol{R}(\boldsymbol{z}_k)}_{=const.} &+ (\boldsymbol{z}-\boldsymbol{z}_k)^{\mathrm{T}}\underbrace{\nabla_{\boldsymbol{z}}\boldsymbol{R}(\boldsymbol{z}_k)\,\boldsymbol{R}(\boldsymbol{z}_k)}_{=\nabla_{\boldsymbol{z}} f(\boldsymbol{z}_k)} + \frac{1}{2}(\boldsymbol{z}-\boldsymbol{z}_k)^{\mathrm{T}}\underbrace{\nabla_{\boldsymbol{z}}\boldsymbol{R}(\boldsymbol{z}_k)\,\nabla_{\boldsymbol{z}}\boldsymbol{R}(\boldsymbol{z}_k)^{\mathrm{T}}}_{=:\,\mathrm{B}_k}(\boldsymbol{z}-\boldsymbol{z}_k) \end{aligned} \tag{2-42}$$

This equation can be compared to the quadratic problem (2-36) of an SQP iteration. It can be noted, that the Gauss-Newton Hessian approximation $\mathrm{B}_k \coloneqq \nabla_{\boldsymbol{z}}\boldsymbol{R}(\boldsymbol{z}_k)\,\nabla_{\boldsymbol{z}}\boldsymbol{R}(\boldsymbol{z}_k)^{\mathrm{T}}$ does not depend on the Lagrange multipliers, and therefore it does not include the second order derivatives of the constraints. Equation (2-43) shows the comparison of the Gauss-Newton approximation to the exact Hessian:

$$\begin{aligned} \nabla_{\boldsymbol{zz}}^2\mathcal{L}(\boldsymbol{z},\boldsymbol{\lambda},\boldsymbol{\mu}) \;&=\; \nabla_{\boldsymbol{z}}\boldsymbol{R}(\boldsymbol{z_k})\,\nabla_{\boldsymbol{z}}\boldsymbol{R}(\boldsymbol{z}_k)^{\mathrm{T}} \;+\; \sum R_j(\boldsymbol{z})\,\nabla_{\boldsymbol{zz}}^2 R_j(\boldsymbol{z_k}) \;+\; \sum \lambda_j\,\nabla_{\boldsymbol{zz}}^2 g_j(\boldsymbol{z}) \\ &=\; \mathrm{B}_k \;+\; O(\|\boldsymbol{R}(\boldsymbol{z}_k)\|) \;+\; O(\|\boldsymbol{\lambda}\|) \end{aligned} \tag{2-43}$$

BFGS

Another important approach for approximating the Hessian of the Lagrange function is with the Broyden–Fletcher–Goldfarb–Shanno (BFGS) formula:

$$\mathrm{B}_{k+1} = \mathrm{B}_k + \frac{\mathrm{B}_k\,\boldsymbol{s}_k\,\boldsymbol{s}_k^{\mathrm{T}}\,\mathrm{B}_k}{\boldsymbol{s}_k^{\mathrm{T}}\,\mathrm{B}_k\,\boldsymbol{s}_k} + \frac{\boldsymbol{y}_k\,\boldsymbol{y}_k^{\mathrm{T}}}{\boldsymbol{y}_k^{T}\,\boldsymbol{s}_k} \tag{2-44}$$

$$\text{with} \quad \boldsymbol{s}_k = \boldsymbol{z}_{\mathrm{k+1}} - \boldsymbol{z}_k \quad \text{and} \quad \boldsymbol{y}_k = \nabla_{\boldsymbol{z}}\,\mathcal{L}(\boldsymbol{z}_{k+1},\boldsymbol{\lambda}_{k+1},\boldsymbol{\mu}_{k+1}) - \nabla_{\boldsymbol{z}}\,\mathcal{L}(\boldsymbol{z}_k,\boldsymbol{\lambda}_k,\boldsymbol{\mu}_k) \tag{2-45}$$

In this formula, the Hessian approximation B_{k+1} is calculated using information from the last iteration k [53]. B_0 must be initialized with a positive definite matrix.

The BFGS formula delivers a positive definite approximation of the Hessian, provided that the *curvature condition* $\boldsymbol{s}_k^{\mathrm{T}}\,\boldsymbol{y}_k > 0$ is satisfied. This condition may not always hold. Therefore, Nocedal and Wright [53] suggest two options:

1. Skip the update, i.e. $\mathrm{B}_{k+1} = \mathrm{B}_k$, in case the curvature condition is not satisfied. One can reinitialize the Hessian approximation if too many updates have been skipped

2. Replace $\boldsymbol{y}_k$ in (2-44) with $\boldsymbol{r}_k = \theta_k \boldsymbol{y}_k + (1-\theta_k)\,\mathrm{B}_k\,\boldsymbol{s}_k$, where θ_k is calculated as in (2-46) with the *damping* factor $\delta \in]0;1[$.

$$\theta_k = \begin{cases} 1 & \text{if } \boldsymbol{s}_k^{\mathrm{T}}\,\boldsymbol{y}_k \geq \delta\,\boldsymbol{s}_k^{\mathrm{T}}\,\mathrm{B}_k\,\boldsymbol{s}_k \\ \dfrac{(1-\delta)\,\boldsymbol{s}_k^{\mathrm{T}}\,\mathrm{B}_k\,\boldsymbol{s}_k}{\boldsymbol{s}_k^{\mathrm{T}}\,\mathrm{B}_k\,\boldsymbol{s}_k - \boldsymbol{s}_k^{\mathrm{T}}\,\boldsymbol{y}_k} & \text{otherwise} \end{cases} \tag{2-46}$$

In the context of NMPC, Quirynen et al. [50] mention that it is not recommendable to use the BFGS Hessian approximation. They argue that a jump in the states $\bar{x}_0$ (for example due to disturbances) may cause a poor approximation and many BFGS updates may be needed to recover a good convergence rate. However, BFGS is the default Hessian approximation method in embotech's FORCES PRO [55], and was therefore implemented and tested in this project. FORCES PRO also has the peculiarity that each Hessian block $\mathrm{B}_{k,i} \approx \mathrm{H}_{k,i}$ is updated individually. It should be remarked that, to match the new $\boldsymbol{z}_{k+1}$ and $\nabla_{\boldsymbol{z}}\,\mathcal{L}(\boldsymbol{z}_{k+1}, \boldsymbol{\lambda}_{k+1}, \boldsymbol{\mu}_{k+1})$, a shifting procedure may have to be performed on the previous ones.

2.2.4 Survey of Optimization Solvers

In this subsection, some newton-type optimization solvers will be listed. All of these were considered to be implemented in Falcon NMPC. However, only IPOPT and qpDUNES have been applied as part of this thesis.

Interior-Point

IPOPT is a well-known NLP solver based on an interior-point method. It can solve problems of the form [56]:

$$\begin{aligned} \min_{\boldsymbol{z}} \quad & f(\boldsymbol{z}) && \text{(objective function)} \\ \text{subject to} \quad & \boldsymbol{g}_{\mathrm{lb}} \leq \boldsymbol{g}(\boldsymbol{z}) \leq \boldsymbol{g}_{\mathrm{ub}} && \text{(nonlinear constraints)} \\ & \boldsymbol{z}_{\mathrm{lb}} \leq \boldsymbol{z} \leq \boldsymbol{z}_{\mathrm{ub}} && \text{(upper/lower bounds)} \end{aligned} \tag{2-47}$$

This solver also allows to pass an exact Hessian of the problem, and for quadratic problems the Hessian can be fixed so that it is only evaluated once. IPOPT is the default solver for FALCON.m, and was therefore the solver that was employed the most during this project.

FORCES PRO is another nonlinear solver based on an interior-point method. This solver is tailored for nonlinear optimal control problems with horizon length N:

$$\begin{aligned} \min_{\boldsymbol{z}} \quad & \sum_{i=0}^{N-1} f_i\,(\boldsymbol{z}_i, \boldsymbol{p}_i) && && \text{(nonlinear separable objective)} \\ \text{subject to} \quad & \boldsymbol{z}_0 = \boldsymbol{z}_{\mathrm{init}} && && \text{(initial equality)} \\ & \boldsymbol{z}_{i+1} = \boldsymbol{f}_d(\boldsymbol{z}_i, \boldsymbol{p}_i) && \forall\, i = 0, \dots, N-1 && \text{(inter-stage equality)} \\ & \boldsymbol{z}_N = \boldsymbol{z}_{\mathrm{final}} && && \text{(final equality)} \\ & \boldsymbol{z}_{lb} \leq \boldsymbol{z} \leq \boldsymbol{z}_{ub} && && \text{(upper/lower bounds)} \\ & \boldsymbol{g}_{lb} \leq \boldsymbol{g}(\boldsymbol{z}, \boldsymbol{p}) \leq \boldsymbol{g}_{ub} && && \text{(nonlinear constraints)} \end{aligned} \tag{2-48}$$

where $\boldsymbol{z}_{\mathrm{init}}$ and $\boldsymbol{z}_{\mathrm{final}}$ are respectively the initial and final boundary conditions for the states and the controls and $\boldsymbol{p}_k$ are parameters that can be changed in real-time. Furthermore,

FORCES PRO offers the possibility to compile and download the code to an embedded platform.

Sequential Quadratic Programming

SNOPT is an SQP solver, in which the QP iterations use an active-set approach. The nonlinear problem may have the form [57]:

$$\begin{aligned} \min_{\boldsymbol{z}} \quad & f(\boldsymbol{z}) && \text{(nonlinear objective)} \\ \text{subject to} \quad & \boldsymbol{g}_{\text{lb}} \leq \boldsymbol{g}(\boldsymbol{z}) \leq \boldsymbol{g}_{\text{ub}} && \text{(nonlinear constraints)} \\ & \boldsymbol{a}_{\text{lb}} \leq \boldsymbol{A}\,\boldsymbol{z} \leq \boldsymbol{a}_{\text{ub}} && \text{(linear constraints)} \\ & \boldsymbol{z}_{\text{lb}} \leq \boldsymbol{z} \leq \boldsymbol{z}_{\text{ub}} && \text{(upper/lower bounds)} \end{aligned} \tag{2-49}$$

This solver is also interfaced with FALCON.m, however it has not been implemented to solve the NMPC optimal control problem during this project.

Quadratic Programming

qpOASES is an active-set QP solver, that can use known information about the solutions of previous problems to speed-up the solution of the current problem [40]. This *parametric active-set* method is of special interest for Model Predictive Control applications. qpOASES takes problems of the form:

$$\begin{aligned} \min_{\boldsymbol{z}} \quad & \frac{1}{2}\,\boldsymbol{z}^{\mathrm{T}}\,\mathrm{H}\,\boldsymbol{z} + \boldsymbol{f}^{\mathrm{T}}\boldsymbol{z} && \text{(quadratic objective)} \\ \text{subject to} \quad & \boldsymbol{a}_{\text{lb}} \leq \boldsymbol{A}\,\boldsymbol{z} \leq \boldsymbol{a}_{\text{ub}} && \text{(linear constraints)} \end{aligned} \tag{2-50}$$

Although this solver can be provided with sparse matrices, it works best for *dense* (non-sparse) problems. Therefore, the authors suggest to use *condensing* (see next subsection) as a preprocessing step, in the case of having an Optimal Control Problem.

qpDUNES is a QP solver especially designed for MPC problems of the form [58]:

$$\begin{aligned} \min_{\boldsymbol{z}} \quad & \sum_{i=0}^{N-1} \left(\frac{1}{2}\boldsymbol{z}_i^{\mathrm{T}}\,\mathrm{H}_i\,\boldsymbol{z}_i + \boldsymbol{f}_i^{\mathrm{T}}\,\boldsymbol{z}_i \right) && \text{(quadratic separable objective)} \\ \text{subject to} \quad & \boldsymbol{z}_{i+1} = \boldsymbol{C}_i\,\boldsymbol{z}_i + \boldsymbol{c}_i \qquad \forall\, i = 0, \dots, N-1 && \text{(linear inter-stage equality)} \\ & \boldsymbol{d}_{i,\text{lb}} \leq \boldsymbol{D}_i\,\boldsymbol{z} \leq \boldsymbol{d}_{i,\text{ub}} \quad \forall\, i = 0, \dots, N-1 && \text{(linear intra-stage constraints)} \end{aligned} \tag{2-51}$$

This solver can only handle strictly convex problems, therefore all H_i must be positive definite ($\mathrm{H}_i \succ 0 \quad \forall\, i = 0, \dots, N$). Since the Lagrange function of an OCP is separable (see equation (2-38)), qpDUNES works by solving the dual problem:

$$\max_{\boldsymbol{\lambda}} \sum_{i=0}^{N-1} f_i^*(\lambda)$$

$$\text{where} \quad f_i^*(\lambda) \coloneqq \min_{\boldsymbol{z}_i} \quad \frac{1}{2}\boldsymbol{z}_i^{\mathrm{T}}\,\mathrm{H}_i\,\boldsymbol{z}_i + \left(\boldsymbol{f}_i^{\mathrm{T}} + \begin{bmatrix} \boldsymbol{\lambda}_i \\ \boldsymbol{\lambda}_{i+1} \end{bmatrix}^{\mathrm{T}} \begin{bmatrix} -\boldsymbol{E}_i \\ \boldsymbol{C}_i \end{bmatrix} \right) \boldsymbol{z}_i + \boldsymbol{\lambda}_{i+1}^{\mathrm{T}}\,\boldsymbol{c}_i \tag{2-52}$$

$$\text{subject to} \quad \boldsymbol{d}_{i,\text{lb}} \leq \boldsymbol{D}_i\,\boldsymbol{z} \leq \boldsymbol{d}_{i,\text{ub}}$$

$$\text{with} \quad \boldsymbol{E}_i = [\boldsymbol{I} \quad \boldsymbol{0}]$$

The advantage of this algorithm is that the subproblems $f_i^*(\lambda)$ are independent and can therefore be solved in parallel, using for example qpOASES. It can be remarked that the blocks of two or more timesteps i could be combined into one, thus varying the number of subproblems [59]. qpDUNES was implemented and tested in this project, see section 0.

2.2.5 Condensing

Condensing is a preprocessing procedure performed on an Optimal Control Problem to reduce the size of the optimization. This method is only introduced briefly, since it was not implemented as part of this thesis.

If one regards a discretized OCP as in (2-1) and linearizes it for a minor iteration of an SQP method, one gets equality constraints of the form:

$$\Delta \boldsymbol{x}_{i+1} = \boldsymbol{a}_i + \boldsymbol{A}_i \Delta \boldsymbol{x}_i + \boldsymbol{B}_i \Delta \boldsymbol{u}_i \tag{2-53}$$

The idea is to eliminate all states $\Delta \boldsymbol{x}_i$ from the optimization variables, as they are known through forward integration using the equation above (provided that the start $\Delta \boldsymbol{x}_0$ and the controls $\Delta \boldsymbol{u}_i$ are known). Hence only the controls are left as optimization variables and the KKT matrix becomes dense (lower triangular). The Lagrange multipliers corresponding to the constraints (2-53) that are eliminated from the problem can also be obtained in a postprocessing step. This method is particularly beneficial if the number of states is large compared to the number of controls. Efficient algorithms for condensing can be found in [60]. It is important to remark that using a condensing method differs from applying a single shooting method in the OCP, since more information is taken into account in the optimization.

2.3 Parametric Nonlinear Optimization

Parametric optimization is a field of nonlinear optimization that deals with evaluating the impact that a small perturbation in a parameter p has on the optimal solution $\boldsymbol{z}^*$ and the corresponding value of the objective function $J(\boldsymbol{z}^*)$. The methods presented here will be used later to manage the discrepancies between the predicted initial state $\boldsymbol{x}_0$ (after shifting) and the measured (or estimated) current state $\bar{\boldsymbol{x}}_n$.

A generic parametric NLP has the form:

$$\begin{aligned} \min_{\boldsymbol{z}} \quad & f(\boldsymbol{z}, p) \\ \text{subject to} \quad & \boldsymbol{g}(\boldsymbol{z}, p) = \boldsymbol{0} \\ & \boldsymbol{h}(\boldsymbol{z}, p) \leq \boldsymbol{0} \end{aligned} \tag{2-54}$$

where p is a fixed parameter in the optimization.

2.3.1 Sensitivity Analysis

Post-optimal sensitivity analysis or *parameter sensitivity analysis* was first introduced by Fiacco [61], who uses the implicit function theorem to show the differentiability of the solution $\boldsymbol{z}^*$ of a nonlinear optimization problem. This principle was then applied to optimal control theory by Büskens [62]. The main idea of sensitivity analysis is to express the influence that a

perturbation of a parameter p has on the optimal optimization variables $\boldsymbol{z}$ and the Lagrange multipliers $\boldsymbol{\lambda}$ and $\boldsymbol{\mu}$ by a Taylor series expansion:

$$\begin{aligned}\boldsymbol{z}(p) &\approx \boldsymbol{z}_0 + \frac{\partial \boldsymbol{z}}{\partial p}(p_0)\cdot(p-p_0)\\ \boldsymbol{\lambda}(p) &\approx \boldsymbol{\lambda}_0 + \frac{\partial \boldsymbol{\lambda}}{\partial p}(p_0)\cdot(p-p_0)\\ \boldsymbol{\mu}(p) &\approx \boldsymbol{\mu}_0 + \frac{\partial \boldsymbol{\mu}}{\partial p}(p_0)\cdot(p-p_0)\end{aligned} \tag{2-55}$$

In the equations above, p_0 denotes the unperturbed parameter, also called the reference or nominal parameter. Furthermore, $\boldsymbol{z}_0 = \boldsymbol{z}^*(p_0)$, $\boldsymbol{\lambda}_0 = \boldsymbol{\lambda}^*(p_0)$ and $\boldsymbol{\mu}_0 = \boldsymbol{\mu}^*(p_0)$ represent the optimal solution vector and the Lagrange multipliers for the unperturbed parameter. The partial derivatives $\frac{\partial \boldsymbol{z}}{\partial p}(p_0)$, $\frac{\partial \boldsymbol{\lambda}}{\partial p}(p_0)$ and $\frac{\partial \boldsymbol{\mu}}{\partial p}(p_0)$ can be calculated using the *explicit formulae for the sensitivity differentials* as proposed by Fiacco:

$$\begin{bmatrix}\nabla^2_{\boldsymbol{zz}}\,\mathcal{L} & \nabla_{\boldsymbol{z}}\boldsymbol{g} & \nabla_{\boldsymbol{z}}\boldsymbol{h}_{\mathrm{a}}\\ \nabla_{\boldsymbol{z}}\boldsymbol{g}^{\mathrm{T}} & \mathbf{0} & \mathbf{0}\\ \nabla_{\boldsymbol{z}}\boldsymbol{h}_{\mathrm{a}}^{\mathrm{T}} & \mathbf{0} & \mathbf{0}\end{bmatrix}\begin{bmatrix}\frac{\partial \boldsymbol{z}}{\partial p}\\ \frac{\partial \boldsymbol{\lambda}}{\partial p}\\ \frac{\partial \boldsymbol{\mu}_{\mathrm{a}}}{\partial p}\end{bmatrix} + \frac{\partial}{\partial p}\begin{bmatrix}\nabla_{\boldsymbol{z}}\,\mathcal{L}\\ \boldsymbol{g}\\ \boldsymbol{h}_{\mathrm{a}}\end{bmatrix} = \mathbf{0} \tag{2-56}$$

where the functions $\nabla^2_{\boldsymbol{zz}}\,\mathcal{L}$, $\nabla_{\boldsymbol{z}}\boldsymbol{g}$, $\nabla_{\boldsymbol{z}}\boldsymbol{h}_{\mathrm{a}}$, $\nabla_{\boldsymbol{z}}\,\mathcal{L}$, $\boldsymbol{g}$ and $\boldsymbol{h}_{\mathrm{a}}$ are evaluated at $\boldsymbol{z}_0$, $\boldsymbol{\lambda}_0$, $\boldsymbol{\mu}_0$. This expression is derived from the necessary conditions for optimality (2-24). One can see that the left matrix is the KKT matrix of an SQP problem, thus computing the sensitivities of the problem is computationally cheap as this matrix is already constructed during the optimization.

This method is only applicable if all active inequality constraints (denoted here with $\boldsymbol{h}_{\mathrm{a}}$) are strictly active, i.e. the corresponding Lagrange multipliers $\boldsymbol{\mu}_{\mathrm{a}}$ are greater than zero. Furthermore, the active set must stay the same for the new parameter. Thus, the Lagrange multiplier corresponding to the inactive inequality constraints must remain zero, that is:

$$\boldsymbol{\mu}_{\tilde{\mathrm{a}}} = \mathbf{0} \tag{2-57}$$

In [63], Büskens and Maurer present a method to predict the maximum and minimum values that the parameter p can adopt without changing the active set. However, in NMPC the current state $\bar{\boldsymbol{x}}_0$ is in general changing continuously. Therefore, one cannot assume that the optimal solution does not change the active set. This is where the predictor-corrector path-following methods introduced in the next subsection come into use.

2.3.2 Predictor-Corrector Path-Following Methods

Path-following is one important tool of parametric optimization, especially for real-time applications. The idea is that the path that a parameter p makes, has a corresponding path in the solution space $(\boldsymbol{z}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*)$. Thus, the task is to find and follow the parameter p in the solution space.

By combining equations (2-55), (2-56) and (2-57) one arrives at the conditions [36]:

$$\begin{bmatrix} \nabla_{zz}^2 \mathcal{L} & \nabla_z \boldsymbol{g} & \nabla_z \boldsymbol{h}_\mathrm{a} \\ \nabla_z \boldsymbol{g}^\mathrm{T} & \mathbf{0} & \mathbf{0} \\ \nabla_z \boldsymbol{h}_\mathrm{a}^\mathrm{T} & \mathbf{0} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \Delta \boldsymbol{z} \\ \Delta \boldsymbol{\lambda} \\ \Delta \boldsymbol{\mu}_\mathrm{a} \end{bmatrix} + \frac{\partial}{\partial p} \begin{bmatrix} \nabla_z \mathcal{L} \\ \boldsymbol{g} \\ \boldsymbol{h}_\mathrm{a} \end{bmatrix} (p - \bar{p}) = \mathbf{0} \qquad \Delta \boldsymbol{\mu}_{\tilde{\mathrm{a}}} = \mathbf{0} \tag{2-58}$$

where $\Delta \boldsymbol{z} = \boldsymbol{z}(p) - \bar{\boldsymbol{z}}$, $\Delta \boldsymbol{\lambda} = \boldsymbol{\lambda}(p) - \bar{\boldsymbol{\lambda}}$, $\Delta \boldsymbol{\mu}_\mathrm{a} = \boldsymbol{\mu}_\mathrm{a}(p) - \bar{\boldsymbol{\mu}}_\mathrm{a}$ and $\Delta \boldsymbol{\mu}_{\tilde{\mathrm{a}}} = \boldsymbol{\mu}_{\tilde{\mathrm{a}}}(p) - \bar{\boldsymbol{\mu}}_{\tilde{\mathrm{a}}}$. Note that the index 0 has been replaced with a bar ($\bar{\square}$) to denote that these values are arbitrary and do not need to correspond to a converged solution.

Furthermore, the solution with $\bar{p}$ must satisfy the following equations (compare the KKT conditions (2-24)):

$$\begin{bmatrix} \nabla_{zz}^2 \mathcal{L} & \nabla_z \boldsymbol{g} & \nabla_z \boldsymbol{h}_\mathrm{a} \\ \nabla_z \boldsymbol{g}^\mathrm{T} & \mathbf{0} & \mathbf{0} \\ \nabla_z \boldsymbol{h}_\mathrm{a}^\mathrm{T} & \mathbf{0} & \mathbf{0} \end{bmatrix} \begin{bmatrix} 0 \\ \bar{\boldsymbol{\lambda}} \\ \bar{\boldsymbol{\mu}}_\mathrm{a} \end{bmatrix} + \begin{bmatrix} \nabla_z f \\ \boldsymbol{g} \\ \boldsymbol{h}_\mathrm{a} \end{bmatrix} = \mathbf{0} \qquad \boldsymbol{\mu}_{\tilde{\mathrm{a}}_0} = \mathbf{0} \tag{2-59}$$

Adding (2-58) and (2-59) together results in:

$$\begin{bmatrix} \nabla_{zz}^2 \mathcal{L} & \nabla_z \boldsymbol{g} & \nabla_z \boldsymbol{h}_\mathrm{a} \\ \nabla_z \boldsymbol{g}^\mathrm{T} & \mathbf{0} & \mathbf{0} \\ \nabla_z \boldsymbol{h}_\mathrm{a}^\mathrm{T} & \mathbf{0} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \Delta \boldsymbol{z} \\ \boldsymbol{\lambda}(p) \\ \boldsymbol{\mu}_\mathrm{a}(p) \end{bmatrix} + \begin{bmatrix} \nabla_z f \\ \boldsymbol{g} \\ \boldsymbol{h}_\mathrm{a} \end{bmatrix} + \frac{\partial}{\partial p} \begin{bmatrix} \nabla_z \mathcal{L} \\ \boldsymbol{g} \\ \boldsymbol{h}_\mathrm{a} \end{bmatrix} (p - \bar{p}) = \mathbf{0} \qquad \boldsymbol{\mu}_{\tilde{\mathrm{a}}}(p) = \mathbf{0} \tag{2-60}$$

One can observe that (2-60) are the KKT conditions corresponding to the quadratic problem

$$\begin{aligned} \min_{\Delta \boldsymbol{z}} \quad & \frac{1}{2} \Delta \boldsymbol{z}^\mathrm{T} \, \mathrm{H} \, \Delta \boldsymbol{z} + \nabla_z f^\mathrm{T} \, \Delta \boldsymbol{z} + (p - \bar{p})^\mathrm{T} \, \nabla_{zp}^2 \mathcal{L} \, \Delta \boldsymbol{z} \\ \text{subject to} \quad & \boldsymbol{g} + \nabla_z \boldsymbol{g}^\mathrm{T} \, \Delta \boldsymbol{z} + \nabla_\mathrm{p} \boldsymbol{g}^\mathrm{T} \, (p - \bar{p}) = 0 \\ & \boldsymbol{h} + \nabla_z \boldsymbol{h}^\mathrm{T} \, \Delta \boldsymbol{z} + \nabla_\mathrm{p} \boldsymbol{h}^\mathrm{T} \, (p - \bar{p}) \leq 0 \end{aligned} \tag{2-61}$$

where $\mathrm{H} = \nabla_{zz}^2 \mathcal{L}$, $\nabla_z f$, $\nabla_{zp} \mathcal{L}$, $\nabla_z \boldsymbol{g}$, $\nabla_z \boldsymbol{h}$, $\boldsymbol{g}$ and $\boldsymbol{h}$ are evaluated at $\bar{\boldsymbol{z}}$, $\bar{\boldsymbol{\lambda}}$, $\bar{\boldsymbol{\mu}}$ and $\bar{p}$. One can prove that the solution of this quadratic problem is a piecewise-linear approximation of the solution of the perturbed nonlinear problem [36]. This is depicted for an example in Figure 2-4. The grey lines show the exact solution in relation to p and the dashed lines the linear approximation for $\boldsymbol{z}^*(\bar{p})$ and $\mu^*(\bar{p})$.

Also, one can see that for $p \neq \bar{p}$ and $(\bar{\boldsymbol{z}}, \bar{\boldsymbol{\lambda}}, \bar{\boldsymbol{\mu}}) = (\boldsymbol{z}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*)$ the QP delivers a linear prediction for the parameter p, as in the previous subsection. For $p = \bar{p}$ and $(\bar{\boldsymbol{z}}, \bar{\boldsymbol{\lambda}}, \bar{\boldsymbol{\mu}}) \neq (\boldsymbol{z}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*)$ then the QP performs a corrective Newton-step towards the solution of the nonlinear problem $(\boldsymbol{z}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*)$ as in a regular SQP method. Hence, this QP is called a predictor-corrector method. In the equations above, the predictor part is marked in red and the corrector part in blue.

It can be remarked that these methods can be implemented in SQP and in interior-point frameworks. Since the interior-point methods smoothen the inequality constraints depending a parameter τ (see *Interior-Point methods* in section 2.2.2), then the performance of the predictor-corrector method depends strongly on τ. This is presented in detail in [36] and [64].

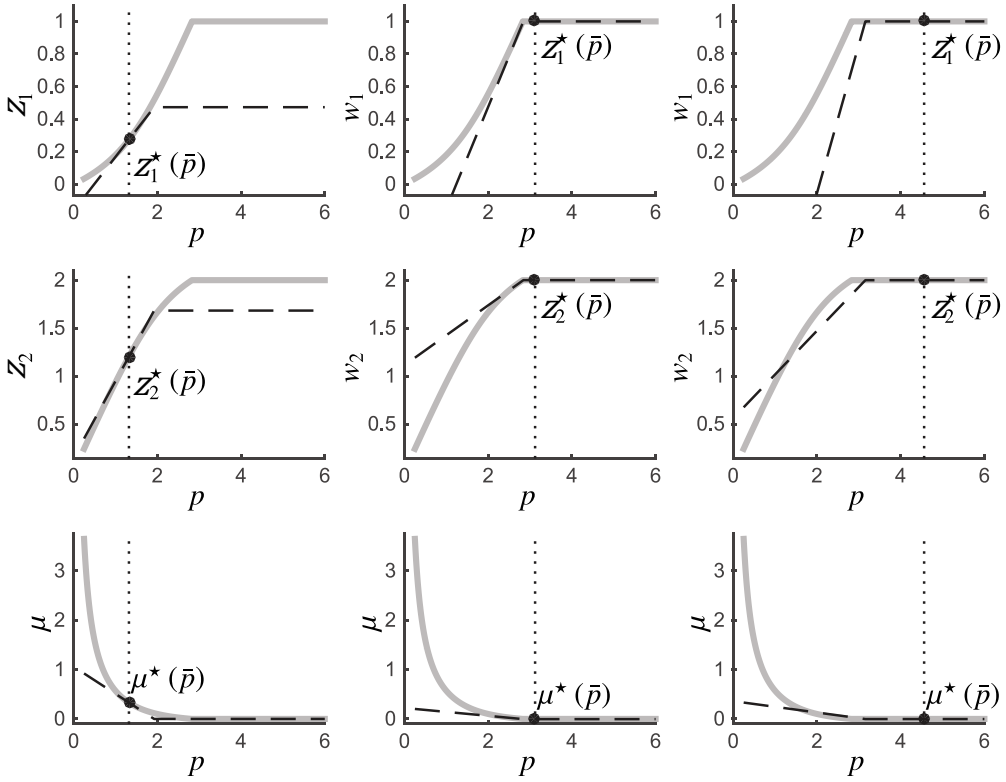

Figure 2-4: Visualization of predictor-corrector path-following methods [36].

2.3.3 Parametric Embedding

The idea of parametric embedding is to reformulate the parametric NLP (2-54), so that the parameter p only enters linearly in the constraints. This is achieved by inserting a new optimization variable θ:

$$\begin{aligned} \min_{\boldsymbol{z},\theta} \quad & f(\boldsymbol{z},\theta) \\ \text{subject to} \quad & \boldsymbol{g}(\boldsymbol{z},\theta) = \mathbf{0} \\ & \boldsymbol{h}(\boldsymbol{z},\theta) \leq \mathbf{0} \\ & p - \theta = 0 \end{aligned} \tag{2-62}$$

This means that the functions H, $\nabla_{\mathbf{z}} f$, $\nabla_{\boldsymbol{zp}}\, \mathcal{L}$, $\nabla_{\boldsymbol{z}} \boldsymbol{g}$, $\nabla_{\boldsymbol{z}} \boldsymbol{h}$, $\boldsymbol{g}$ and $\boldsymbol{h}$ in the predictor-corrector (2-61) are not evaluated using $\bar{p}$ but using θ. Therefore, θ is the "old parameter" with which the linearization for the quadratic problem (2-61) is done and p is the "new parameter" with which the QP is solved [65]. Applied to the path-following method (2-61), this results in:

$$\begin{aligned} \min_{\Delta \boldsymbol{z}} \quad & \frac{1}{2} \Delta \boldsymbol{z}^{\mathrm{T}} \, \mathrm{H} \, \Delta \boldsymbol{z} + \nabla_{\mathbf{z}} f^{\mathrm{T}} \, \Delta \boldsymbol{z} + \Delta \theta^{\mathrm{T}} \, \nabla_{zp}^{2} \, \mathcal{L} \, \Delta \boldsymbol{z} \\ \text{subject to} \quad & \boldsymbol{g} + \nabla_{\mathbf{z}} \boldsymbol{g}^{\mathrm{T}} \, \Delta \boldsymbol{z} + \nabla_{\mathrm{p}} \boldsymbol{g}^{\mathrm{T}} \, \Delta \theta = 0 \\ & \boldsymbol{h} + \nabla_{\mathbf{z}} \boldsymbol{h}^{\mathrm{T}} \, \Delta \boldsymbol{z} + \nabla_{\mathrm{p}} \boldsymbol{h}^{\mathrm{T}} \, \Delta \theta \leq 0 \\ & p - \theta - \Delta \theta = 0 \end{aligned} \tag{2-63}$$

It can be noted that the NMPC optimization problem (2-1) already has this form, taking the measured (or estimated) states $\bar{\boldsymbol{x}}_n$ as parameters. This is known as *initial value embedding* and is extremely beneficial in the Real-Time Iteration scheme, as it allows the algorithm to be divided into two phases.

2.4 The Real-Time Iteration Scheme

The Real-Time Iteration scheme, also known as RTI, is the Nonlinear Model Predictive Control algorithm that was implemented for this thesis. It was first introduced by Diehl et al. in [37]. Since then, this algorithm has become “one of the most successful and largely used approaches to fast NMPC” [33].

The basic idea of the algorithm is to solve the optimal control problem (2-1) with a path-following SQP framework in which only one quadratic problem per sampling time is solved. There is no *line-search*, thus a full Newton-step is taken between the SQP iterations. The sampling time must therefore be chosen so that all the necessary calculations for constructing the QP problem and solving the QP problem can be done during one sampling time. This sampling time must also be considered for the discretization of the Optimal Control Problem.

The path-following method (2-63) can thus be applied for the OCP (2-1) with parametric embedding of the initial states $\bar{\boldsymbol{x}}$, resulting in a QP problem of the form:

$$\begin{aligned} \min_{\Delta \boldsymbol{z}} \quad & \frac{1}{2} \Delta \boldsymbol{z}^{\mathrm{T}} \, \mathrm{H} \, \Delta \boldsymbol{z} + \nabla_{\mathbf{z}} f^{\mathrm{T}} \, \Delta \boldsymbol{z} \\ \text{subject to} \quad & \boldsymbol{g} + \nabla_{\mathbf{z}} \boldsymbol{g}^{\mathrm{T}} \, \Delta \boldsymbol{z} = 0 \\ & \boldsymbol{h} + \nabla_{\mathbf{z}} \boldsymbol{h}^{\mathrm{T}} \, \Delta \boldsymbol{z} \leq 0 \\ & \boldsymbol{x}_0 - \bar{\boldsymbol{x}}_n = \mathbf{0} \end{aligned} \tag{2-64}$$

The initial value embedding of $\bar{\boldsymbol{x}}_n$ allows to divide the RTI scheme into two phases, a *Preparation Phase*, where the QP problem is constructed, and a *Feedback Phase*, where the QP is solved. Finally, the RTI algorithm has the form:

Algorithm 3: Real-Time Iteration (RTI)

Preparation Phase (performed before t_n)

Input: previous NMPC solution, reference $\boldsymbol{x}^{\text{ref}}$

Perform *Shift* as in section 2.1.3 to get initial guess $(\boldsymbol{x}^{\text{guess}}, \boldsymbol{u}^{\text{guess}})$.
Evaluate H, $\nabla_{\boldsymbol{z}} f$, $\nabla_{\boldsymbol{z}} \boldsymbol{g}$, $\nabla_{\boldsymbol{z}} \boldsymbol{h}$, $\boldsymbol{g}$ and $\boldsymbol{h}$ using the initial guess
Perform all possible computations (e.g. condensing, matrix factorizations)
Build QP problem (2-64) omitting the equality constraints $\boldsymbol{x}_0 - \bar{\boldsymbol{x}}_n = \boldsymbol{0}$

Output: QP problem (2-64)

Feedback Phase (performed upon availability of $\bar{x}_n$)

Input: QP problem from Preparation Phase

Introduce the constraints $\boldsymbol{x}_0 - \bar{\boldsymbol{x}}_n = \boldsymbol{0}$
Solve QP problem
Apply full Newton-step $(\boldsymbol{x}^*, \boldsymbol{u}^*) \leftarrow (\boldsymbol{x}^{\text{guess}}, \boldsymbol{u}^{\text{guess}}) + (\Delta\boldsymbol{x}, \Delta\boldsymbol{u})$
Implement $\boldsymbol{\mu}(\bar{\boldsymbol{x}}_n) = \boldsymbol{u}_0^*$ on the system's actuators

Output: NMPC solution $(\boldsymbol{x}^*, \boldsymbol{u}^*)$

The reason for the separation in two phases is that the Feedback Phase is generally much shorter than the Preparation Phase. Depending on the application, the Feedback Phase can even be several orders of magnitude shorter than the Preparation Phase [36]. Therefore, in the Preparation Phase a guess is used for the calculation of all matrices necessary for the QP problem. The phase is taken from the previous prediction via shifting. Then the Feedback Phase is performed once the measurement of $\bar{x}_n$ is available, thus ensuring that the time between receiving the measurement and implementing the solution on the actuators is kept as short as possible. This is depicted in Figure 2-5.

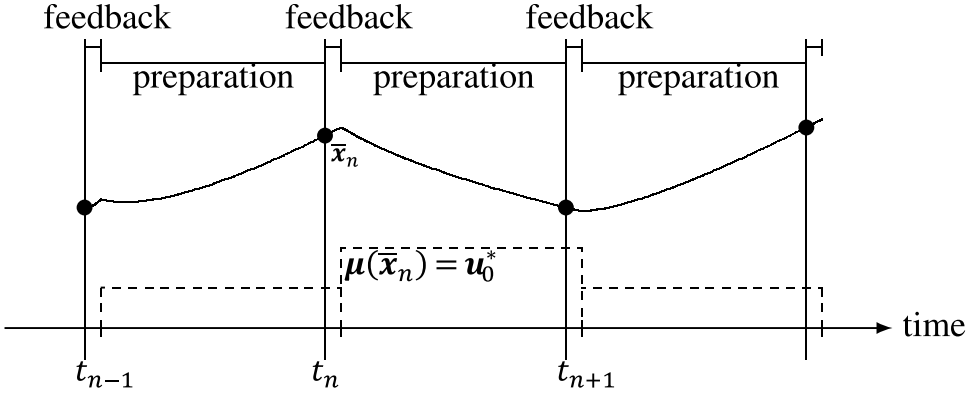

Figure 2-5: Preparation and Feedback phases of the RTI scheme [36]

Analyses on the error bounds and closed-loop stability of the RTI scheme can be found in [38] and [66]. In [33], Gros et al. describe the algorithm in detail and present several remarks, examples and comparisons relevant for an implementation of the algorithm.

RTI for tracking

As mentioned before, Nonlinear Model Predictive Control is usually employed for tracking applications, i.e. for following a reference trajectory. A commonly used objective function for this kind of applications is

$$\min_{\boldsymbol{x},\boldsymbol{u}} \sum_{i=0}^{N-1} \underbrace{\frac{1}{2}\begin{bmatrix}\Delta\boldsymbol{x}_i\\ \Delta\boldsymbol{u}_i\end{bmatrix}^{\mathrm{T}}\begin{bmatrix}\boldsymbol{Q} & \boldsymbol{S}\\ \boldsymbol{S}^T & \boldsymbol{R}\end{bmatrix}\begin{bmatrix}\Delta\boldsymbol{x}_i\\ \Delta\boldsymbol{u}_i\end{bmatrix}}_{L(\boldsymbol{x}_i,\boldsymbol{u}_i)} + \underbrace{\frac{1}{2}\,\Delta\boldsymbol{x}_N^{\mathrm{T}}\,\boldsymbol{P}\,\Delta\boldsymbol{x}_N}_{E(\boldsymbol{x}_N)} \tag{2-65}$$

with $\Delta\boldsymbol{x}_i = \boldsymbol{x}_i - \boldsymbol{x}_i^{\mathrm{ref}}$ and $\Delta\boldsymbol{u}_i = \boldsymbol{u}_i - \boldsymbol{u}_i^{\mathrm{ref}}$, where $\boldsymbol{x}^{\mathrm{ref}}$ is the reference for the states and $\boldsymbol{u}^{\mathrm{ref}}$ is the reference for the controls. The matrices $\boldsymbol{Q}$, $\boldsymbol{R}$, $\boldsymbol{S}$ and $\boldsymbol{P}$ become tuning parameters for the algorithm.

One can observe that the objective function has the same form as in (2-42) for the *Gauss-Newton* Hessian approximation. Therefore, a Gauss-Newton Hessian approximation is normally used with the RTI scheme, which significantly reduces the computational efforts compared to an exact Hessian algorithm. For the objective function (2-65) the Gauss-Newton Hessian approximation becomes constant and therefore, it must not be computed:

$$\mathrm{B}_k = \begin{bmatrix}\boldsymbol{Q} & \boldsymbol{S} & & & & \\ \boldsymbol{S}^T & \boldsymbol{R} & & & & \\ & & \ddots & & & \\ & & & \boldsymbol{Q} & \boldsymbol{S} & \\ & & & \boldsymbol{S}^T & \boldsymbol{R} & \\ & & & & & \boldsymbol{P}\end{bmatrix} \tag{2-66}$$

In the tests performed in this thesis, no reference for the controls was considered and the $\boldsymbol{S}$ matrix was set to zero. Therefore, the objective function was reduced to:

$$\min_{\boldsymbol{x},\boldsymbol{u}} \sum_{i=0}^{N-1} \underbrace{\frac{1}{2}\left(\left(\boldsymbol{x}_i - \boldsymbol{x}_i^{\mathrm{ref}}\right)^{\mathrm{T}}\boldsymbol{Q}\left(\boldsymbol{x}_i - \boldsymbol{x}_i^{\mathrm{ref}}\right) + \boldsymbol{u}_i^{\mathrm{T}}\,\boldsymbol{R}\,\boldsymbol{u}_i\right)}_{L(\boldsymbol{x}_i,\boldsymbol{u}_i)} + \underbrace{\frac{1}{2}\left(\boldsymbol{x}_N - \boldsymbol{x}_N^{\mathrm{ref}}\right)^{\mathrm{T}}\boldsymbol{P}\left(\boldsymbol{x}_N - \boldsymbol{x}_N^{\mathrm{ref}}\right)}_{E(\boldsymbol{x}_N)} \tag{2-67}$$

It is suggested to pre-compute a feasible reference trajectory $\boldsymbol{x}^{\mathrm{ref}}$, so that the NMPC only has to reject perturbations. However, it is also possible to let the controller plan the trajectory. In this case, it is beneficial to perform a procedure similar as the shifting in section 2.1.3, and slowly introduce the reference at the end of the prediction horizon. This allows the RTI scheme to converge before it needs to start moving [33].

Linear MPC vs. RTI

A Linear Model Predictive Control has the following optimal control problem [33]:

$$\begin{aligned}\min_{\boldsymbol{x},\boldsymbol{u}} \quad & \sum_{i=0}^{N-1} \frac{1}{2}\begin{bmatrix}\Delta\boldsymbol{x}_i\\ \Delta\boldsymbol{u}_i\end{bmatrix}^{\mathrm{T}}\begin{bmatrix}\boldsymbol{Q} & \boldsymbol{S}\\ \boldsymbol{S}^T & \boldsymbol{R}\end{bmatrix}\begin{bmatrix}\Delta\boldsymbol{x}_i\\ \Delta\boldsymbol{u}_i\end{bmatrix}\\ \text{subject to} \quad & \Delta\boldsymbol{x}_0 = \bar{\boldsymbol{x}}_n - \boldsymbol{x}_0^{\mathrm{ref}}\\ & \Delta\boldsymbol{x}_{i+1} = \boldsymbol{A}_i\Delta\boldsymbol{x}_i + \boldsymbol{B}_i\Delta\boldsymbol{u}_i + \boldsymbol{a}_i \quad i = 0,\dots,N-1\\ & \boldsymbol{C}_i\Delta\boldsymbol{x}_i + \boldsymbol{D}_i\Delta\boldsymbol{u}_i + \boldsymbol{h}_i \le 0 \quad i = 0,\dots,N-1\end{aligned} \tag{2-68}$$

In this case, the matrices $\boldsymbol{A}_i$, $\boldsymbol{B}_i$, $\boldsymbol{C}_i$ and $\boldsymbol{D}_i$, as well as the vectors $\boldsymbol{a}_i$ and $\boldsymbol{h}_i$ are calculated a priori offline and are therefore constant. These matrices are usually obtained from the linearization about the reference trajectory $(\boldsymbol{x}^{\mathrm{ref}}, \boldsymbol{u}^{\mathrm{ref}})$. The QP problem (2-68) is then solved at every sampling time.

The Real-Time Iteration scheme can be seen as an extension of Linear MPC, where the only difference is that these matrices are calculated online. The linearization is done at the guess provided by the last prediction instead of at the reference trajectory. However, one can observe that, if a Gauss-Newton Hessian approximation is used and the system's equations are evaluated at $(\boldsymbol{x}^{\mathrm{ref}}, \boldsymbol{u}^{\mathrm{ref}})$, then the RTI scheme delivers the same control law as a linear MPC.

RTI for economic NMPC

An economic NMPC application is an application for which the cost function of the OCP does not have the form (2-65). Therefore, the objective in economic NMPC is not necessarily to minimize the error between a reference trajectory and the actual state of system, but it can be to minimize any value concerning the system. This cost function can thus be represented by the objective function of problem (2-1):

$$\min_{\boldsymbol{x},\boldsymbol{u}} \sum_{i=0}^{N-1} L(\boldsymbol{x}_i, \boldsymbol{u}_i) + E(\boldsymbol{x}_N) \tag{2-69}$$

where $L(\boldsymbol{x}_i, \boldsymbol{u}_i)$ and $E(\boldsymbol{x}_N)$ can be any nonlinear function of the states and the controls [28]. Note however that the objective function is still divided into a stage cost and a terminal cost.

For an objective function of this form, it might not be possible to construct a Gauss-Newton approximation of the Hessian of the Lagrange function. Even if one can be constructed, it might not be a constant matrix. Therefore, when using an economic objective function, one must generally calculate a Hessian at each sampling time.

For a time-optimal driving problem, Verschueren et al. [31] suggest using an exact Hessian calculation with the $\mathrm{project}$ regularization method as presented in *Regularization methods* in section 2.2.3. In [67], the authors provide a method to formulate a tracking NMPC that is locally equivalent to the economic NMPC. This method allows to apply methods and analyses known for tracking NMPC.

In this thesis, the OCPs for the point mass model in section 3.4 and for the racecar model in chapter 4 use an economic cost function. Several experiments were performed with these models using different Hessian approximations. The results of these tests can be seen in the corresponding sections.

3 Preliminary Tests

This chapter presents a series of simulative experiments that were used during the development of the Nonlinear Model Predictive Control Toolbox, Falcon NMPC. Each of the models increases in complexity in comparison to the previous one, and each of the experiments tests a different part of the NMPC algorithm. All simulation models were built in Simscape, the MATLAB toolbox for physical systems simulation. At the end of the chapter, section 3.5 presents a summary of the different experiments and their results, which lead to the NMPC setup used for the racecar model in the next chapter.

3.1 1-DoF Cart

3.1.1 Modelling

Simulation model

The first model that was used to test the NMPC in this thesis is simply a mass $m_C = 0.1\ kg$ with one translational degree of freedom. In Simscape this is represented with a prismatic joint between the mass and the world frame. The gravitational field $g = 9.81\ m/s^2$ of the model is perpendicular to the joint and has therefore no impact on the movement of the mass. Additionally, a force $F = f(\mu) \cdot 10\ N$ acts on the mass, which depends on a function $f(\mu)$ of the control value μ. Figure 3-1 shows a schematic of this model.

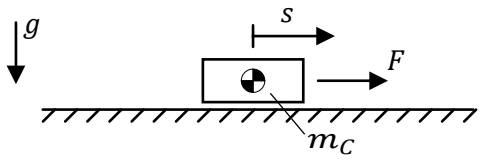

Figure 3-1: 1-DoF Cart – Schematic

NMPC model

The equations of motion for the model above are

$$\begin{aligned} \dot{s} &= v \\ \dot{v} &= \frac{f(\mu) \cdot 10\ N}{m_C} \end{aligned} \tag{3-1}$$

where s represents the mass' position and v its velocity. The control μ is restricted to values between -1 and 1. The function $f(\mu)$ will be describe below.

3.1.2 Results

In the tests with this model a Tracking NMPC was considered, for which the objective function (2-67) was used. The $\boldsymbol{Q}$, $\boldsymbol{R}$ and $\boldsymbol{P}$ matrices were set to:

$$\begin{aligned} \boldsymbol{Q} = \boldsymbol{P} &= \begin{bmatrix} 10 & 0 \\ 0 & 0.01 \end{bmatrix} \\ \boldsymbol{R} &= 0.1 \end{aligned} \tag{3-2}$$

The sampling time was set to $T_s = 0.05\ s$ and the horizon length to $T_H = 2\ s$. For these tests, the conventional RTI algorithm with Gauss-Newton Hessian approximation was used.

The reference trajectory for all tests was a pulsating function for the position s and constant zero for the velocity v. This reference was inserted at the end of the prediction horizon, as suggested in the paragraph *RTI for tracking* in section 2.4.

Trapezoidal discretization with vs. without approach (2-20)

Here, the function $f(\mu)$ was set to

$$f(\mu) = \mu \tag{3-3}$$

In the first test of the NMPC algorithm, a trapezoidal discretization method was used, where instead of applying the control μ using equation (2-20), it was set to the first value in the prediction horizon $\mu = \boldsymbol{u}_0^* = \mu_0^*$. The results of this test are shown in Figure 3-2.

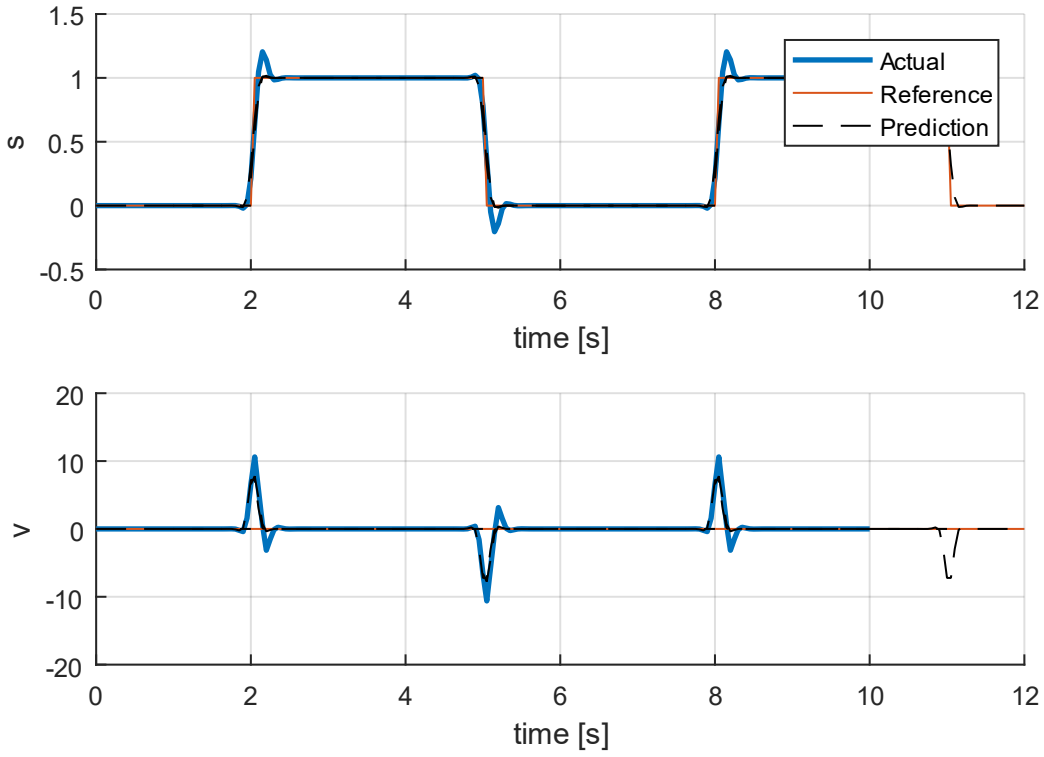

Figure 3-2: 1-DoF Cart – Trapezoidal discretization without approach (2-20)

As one can see, the actual position s shows an overshoot that is not present in the prediction. The reason for this is that with the trapezoidal discretization, the NMPC does not take a Zero Order Hold (ZOH) of the control μ into account. At an arbitrary time-point t_{n+1}, the NMPC predicts a velocity

$$v_{n+1} = v_n + \frac{T_s}{2} \cdot \left(\frac{\mu_n \cdot 10\ N}{m_C} + \frac{\mu_{n+1} \cdot 10\ N}{m_C} \right) \tag{3-4}$$

as can be seen from equation (2-19). However, taking into account the ZOH of the control outputted by the controller, the actual velocity can be calculated analytically by

$$v(t_{n+1}) = v(t_n) + \int_{t_n}^{t_{n+1}} \frac{\mu_n \cdot 10\ N}{m_C} \partial\tau = v(t_n) + \frac{\mu_n \cdot 10\ N}{m_C} \cdot (t_{n+1} - t_n)$$
$$= v(t_n) + \frac{\mu_n \cdot 10\ N}{m_C} \cdot T_s \quad \textbf{(3-5)}$$

Assuming that at the time point t_n the predicted velocity v_n equals the actual velocity $v(t_n)$, v_{n+1} and $v(t_{n+1})$ are in general not equal. However, if the approach (2-20) was used, one can see that v_{n+1} and $v(t_{n+1})$ would be identical:

$$v(t_{n+1}) = v(t_n) + \int_{t_n}^{t_{n+1}} \frac{{}^1\!/_2 \cdot (\mu_n + \mu_{n+1}) \cdot 10\ N}{m_C} \partial\tau = v(t_n) + \frac{1}{2} \cdot \frac{(\mu_n + \mu_{n+1}) \cdot 10\ N}{m_C} \cdot T_s \quad \textbf{(3-6)}$$

Since μ_{n+1} is outputted "prematurely", it is added as constraint to the OCP in the next timestep, as described in the *Trapezoidal* paragraph in section 2.1.2. The results applying this method are shown in Figure 3-3. As one can see, this method alleviates the problem with the overshoots and the actual trajectory follows the predicted one. Therefore, this method was implemented as default for Trapezoidal discretization in Falcon NMPC. However, as it will be shown next, this method does not work as well in case that the input enters nonlinearly into the system.

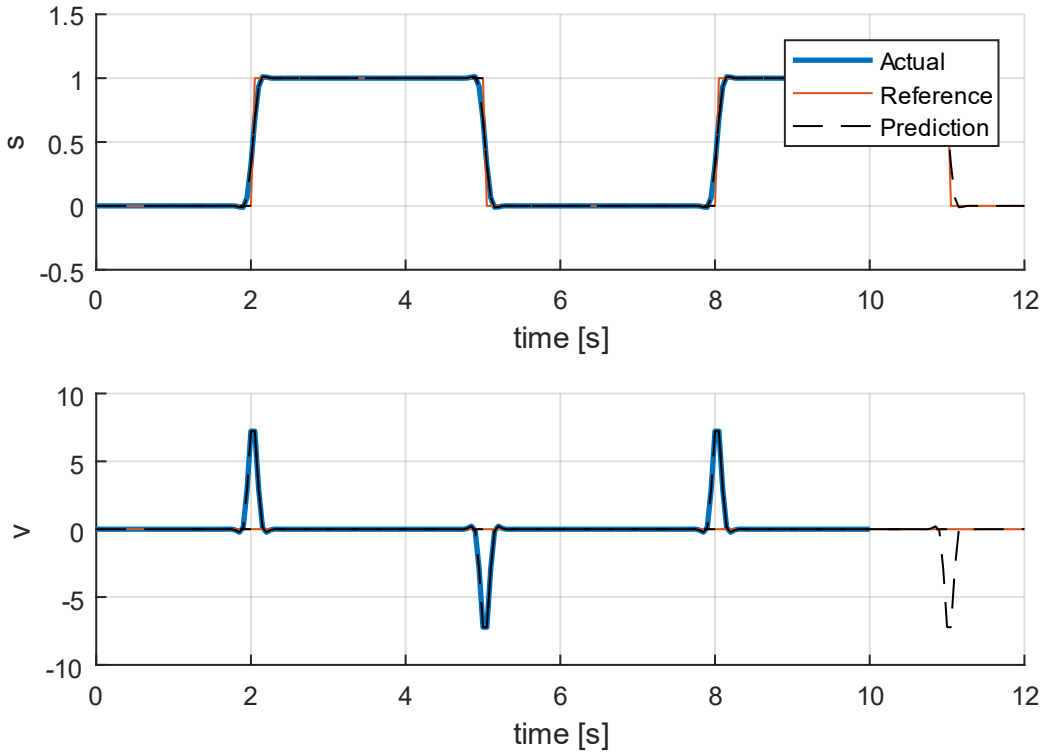

Figure 3-3: 1-DoF Cart – Trapezoidal discretization with approach (2-20)

Trapezoidal vs. ERK4 discretization

For this test, the function $f(\mu)$ was set to

$$f(\mu) = \mu + 0.5 \cdot \sin(2\pi \cdot \mu) \quad \textbf{(3-7)}$$

so that the system has a strongly nonlinear dependence on the input μ. Figure 3-4 shows the output of this function for $\mu \in [-1; 1]$.

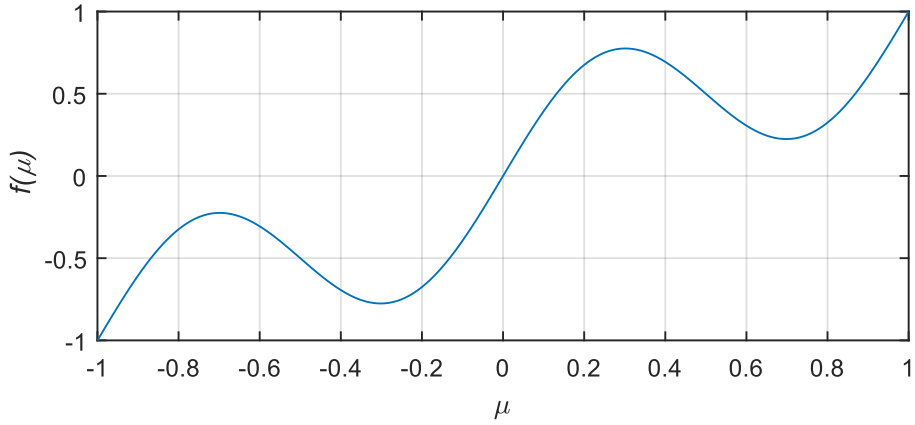

Figure 3-4: 1-DoF Cart – Nonlinear function $f(\mu)$

As can be seen in the figure below, the NMPC with Trapezoidal discretization does no longer manage to control the system for this setup. The reason for this is again that the NMPC does not deliver an accurate prediction because the ZOH of the input is not considered, not even with the approach used before (Figure 3-5). The results with equation (2-20) are similar to the ones applying a Zero-Order Hold, they are however not depicted here.

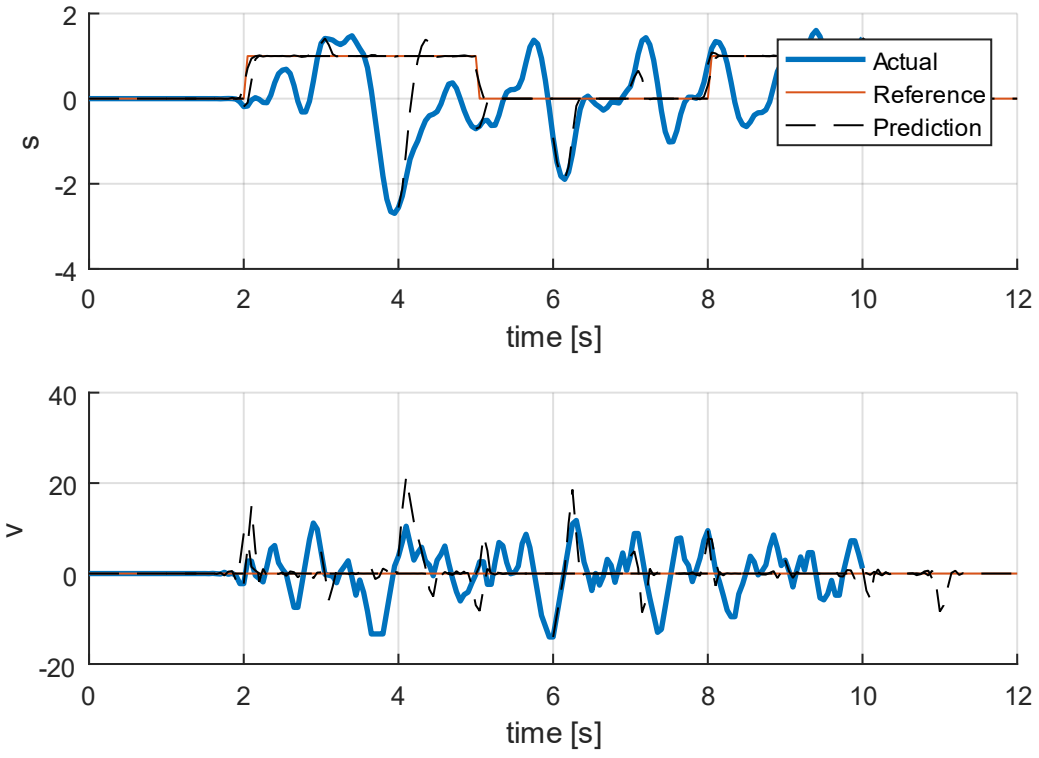

Figure 3-5: 1-DoF Cart – Trapezoidal discretization for nonlinear input function

Nevertheless, explicit Runge-Kutta methods do assume a Zero Order Hold of the control values. Therefore, running this same experiment with an explicit 4th-order Runge-Kutta (ERK4) method results in the plots shown in Figure 3-6. It can be appreciated that the actual states follow the prediction exactly and the performance of the algorithm is similar to that in Figure 3-3. It can be remarked that using a Forward Euler method delivers comparable results, although they are not presented here.

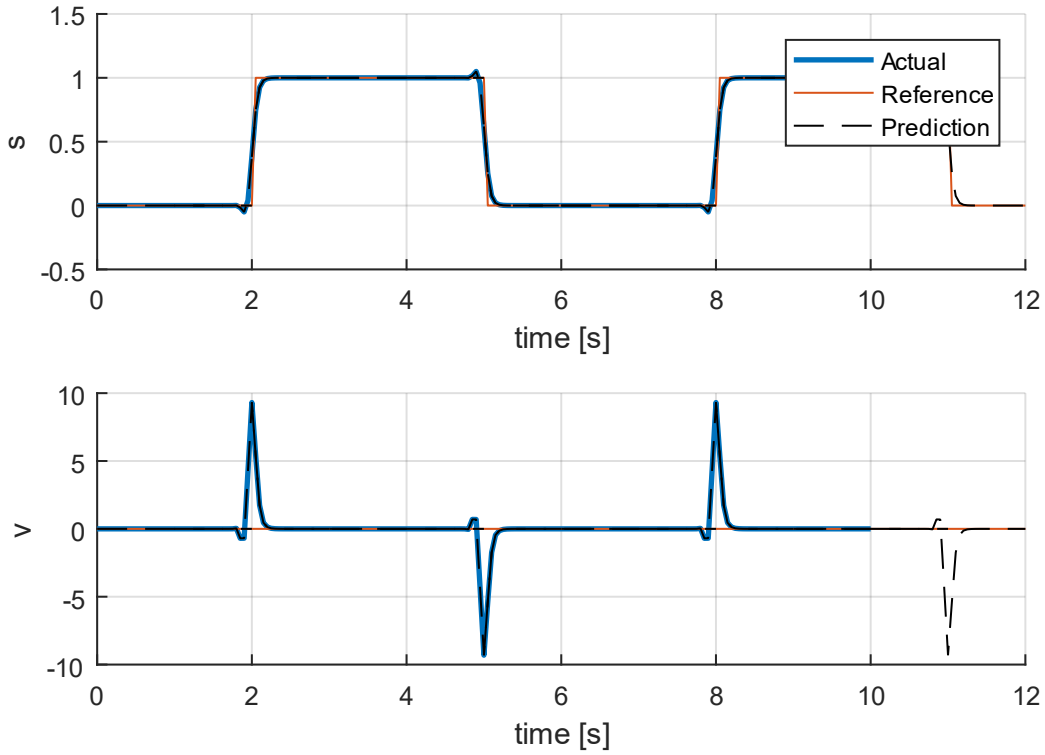

Figure 3-6: 1-DoF Cart – ERK4 discretization for nonlinear input function

Model mismatch

In the previous experiments the model used for the NMPC controller matched the simulation model exactly. To test the performance of the controller in case that the models have discrepancies, the simulation model was modified as follows:

- The mass was increased to $m_{\text{modif}} = 0.12\ kg$
- The prismatic joint was inclined 10°, so that the gravitational field influences the mass
- A damping of $k_d = 1\,\frac{N}{m/s}$ was introduced to the prismatic joint.

These changes are summarized in blue in the following schematic:

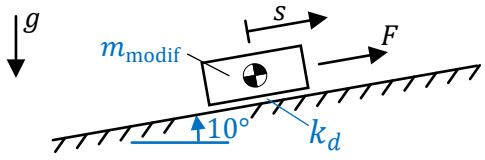

Figure 3-7: 1-DoF Cart – Schematic with model mismatch

In this model, the nonlinear function (3-7) was kept for the control values. Furthermore, an ERK4 discretization method was used. Figure 3-8 shows the results of this test. In this case, the predicted trajectory cannot be followed exactly, as seen clearly in the velocity plot. Nevertheless, the NMPC manages to control the system sufficiently well.

It may be remarked that, in the case that there is a model mismatch, a stationary control deviation may be found. In this example, a small control deviation is given due to the gravity acting on the mass. This control deviation minimizes the objective function, which is dependent

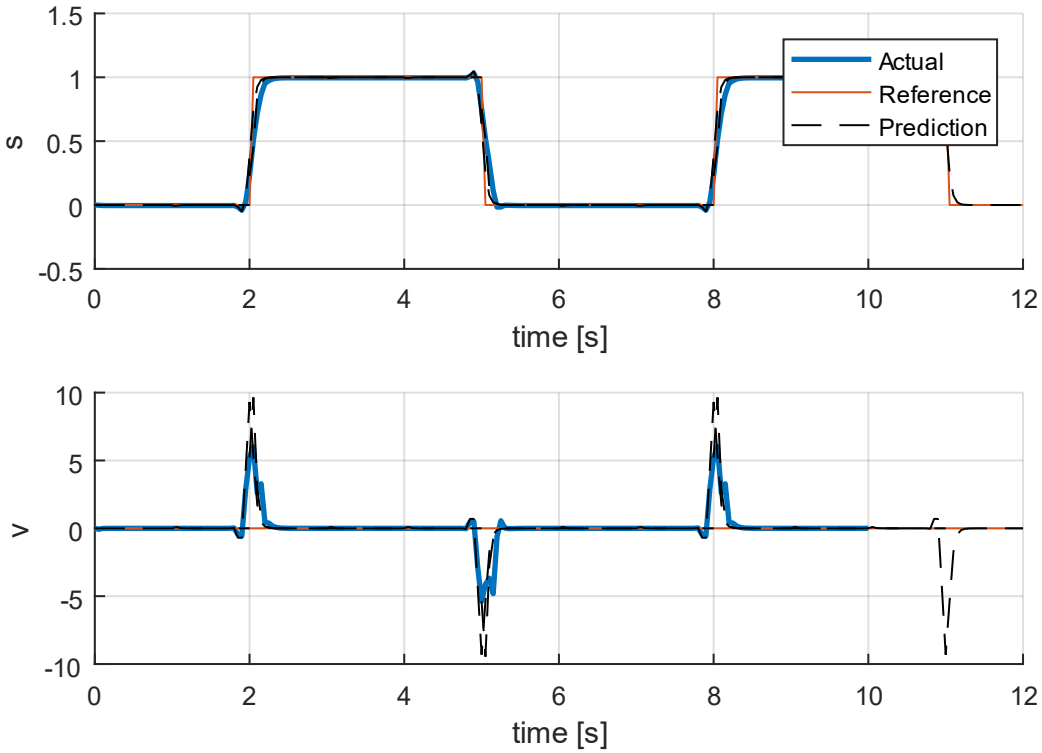

Figure 3-8: 1-DoF Cart – ERK4 discretization with model mismatch

on the deviation $(\boldsymbol{x}_i - \boldsymbol{x}_i^{\text{ref}})$ and on the control values $\boldsymbol{u}_i$. Therefore, a way to reduce this deviation is to make the values of $\boldsymbol{R}$ smaller. However, this can also affect the transient behavior of the system. Another method would be to add an integral controller on top of the NMPC. In the NMPC, this I-controller would be modelled as a state so that it can be considered via initial value embedding.

3.2 Inverted Pendulum

3.2.1 Modelling

Simulation model

Here, a typical system used in control theory, an *inverted pendulum on a cart* is presented. In Simscape, the 1-DoF cart of the last section was extended by another mass $m_P = 0.1\,kg$ connected to the cart with a revolute joint separated by a distance of $l = 0.5\,m$. The schematic is shown below. In this case, the force F is applied directly by the controller.

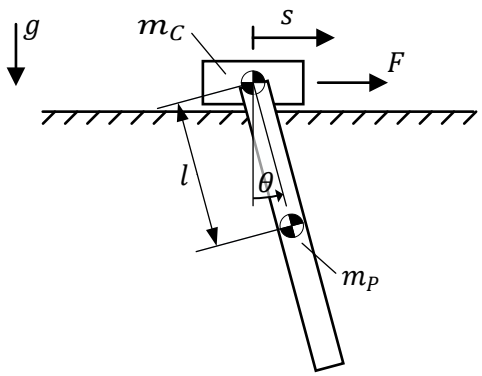

Figure 3-9: Inverted pendulum – Schematic

NMPC model

The equations of motion for this system can be derived using *Lagrangian equations* [68]. This method allows to formulate the equations of motion based on a set of independent *generalized coordinates* $\boldsymbol{q}$, in this case the displacement s and the angle θ. First, one calculates the total kinetic energy T and the total potential energy V of the system:

$$\begin{aligned} T &= \frac{1}{2} m_C \dot{s}^2 + \frac{1}{2} m_P \left(\left(\frac{d}{dt}(s + l\sin\theta) \right)^2 + \left(\frac{d}{dt}(-l\cos\theta) \right)^2 \right) \\ &= \frac{1}{2}(m_C + m_P)\,\dot{s}^2 + m_P l \dot{s}\dot{\theta}\cos\theta + \frac{1}{2} m_P l^2 \dot{\theta}^2 \\ V &= m_P g l \cos\theta \end{aligned} \tag{3-8}$$

Then the equations of motion are given by

$$\frac{d}{dt}\left(\frac{\partial T}{\partial \dot{\boldsymbol{q}}}\right) - \frac{\partial T}{\partial \boldsymbol{q}} + \frac{\partial V}{\partial \boldsymbol{q}} - \boldsymbol{Q}_{nc} = \boldsymbol{0} \tag{3-9}$$

where the non-conservative forces and torques are considered by

$$\boldsymbol{Q}_{nc} = \sum_{j_1} \frac{\partial \boldsymbol{r}_{j_1}}{\partial \boldsymbol{q}}^{\mathrm{T}} \boldsymbol{F}_{j_1} + \sum_{j_2} \frac{\partial \boldsymbol{\omega}}{\partial \dot{\boldsymbol{q}}}^{\mathrm{T}} \boldsymbol{M}_{j_2} \tag{3-10}$$

with the force $\boldsymbol{F}_{j_1}$ acting on the point $\boldsymbol{r}_{j_1}$ and the torque $\boldsymbol{M}_{j_2}$ acts on a body with the angular velocity $\boldsymbol{\omega}$. In planar coordinates, the position where the force F acts can be represented as $\boldsymbol{r}_F = [s \quad 0]^{\mathrm{T}}$ and the vector of the force is $\boldsymbol{F} = [F \quad 0]^{\mathrm{T}}$. Hence, using the Lagrangian equations, one gets:

$$\begin{aligned} &(m_C + m_P)\ddot{s} + m_P l \ddot{\theta}\cos\theta - m_P l \dot{\theta}^2 \sin\theta = F \\ &m_P l \ddot{s}\cos\theta - m_P l \dot{s}\dot{\theta}\sin\theta + m_P l^2 \ddot{\theta} + m_P l \dot{s}\dot{\theta}\sin\theta + m_P g l \sin\theta = 0 \end{aligned} \tag{3-11}$$

Solving these equations for $\ddot{s}$ and $\ddot{\theta}$ result in the equations of motion:

$$\ddot{s} = \frac{m_P l \dot{\theta}^2 \sin\theta + m_P g \sin\theta \cos\theta + F}{m_C + m_P \sin^2\theta}$$
$$\ddot{\theta} = -\frac{m_P l \dot{\theta}^2 \sin\theta \cos\theta + (m_C + m_P) g \sin\theta + F \cos\theta}{m_C l + m_P l \sin^2\theta} \quad \text{(3-12)}$$

The state vector of the system is thus $\boldsymbol{x} = [s \quad \dot{s} \quad \theta \quad \dot{\theta}]^{\mathrm{T}}$.

3.2.2 Results

The tests with this model also concerned a tracking application with the objective function (2-67). The $\boldsymbol{Q}$, $\boldsymbol{R}$ and $\boldsymbol{P}$ matrices were set to:

$$\boldsymbol{Q} = \boldsymbol{P} = \begin{bmatrix} 10 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 10 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$
$$\boldsymbol{R} = 0.01 \quad \text{(3-13)}$$

The sampling time was set to $T_s = 0.05\ s$ and the horizon length to $T_H = 2\ s$. ERK4 was employed as discretization method. In most of the tests, the conventional RTI algorithm with Gauss-Newton Hessian approximation was used. In the last experiment the Gauss-Newton Hessian approximation is compared to a BFGS and an exact Hessian algorithm.

The reference trajectory concerns a swing-up of the pendulum followed by a step for the position s, and then a swing-down of the pendulum staying on the same spot. This reference was inserted at the end of the prediction horizon, as for the previous simulation.

RTI vs. Converged Full OCP

This test regarded the optimality that the RTI algorithm reaches compared to solving the full nonlinear OCP till convergence at every sampling time. In Falcon NMPC, one can solve the

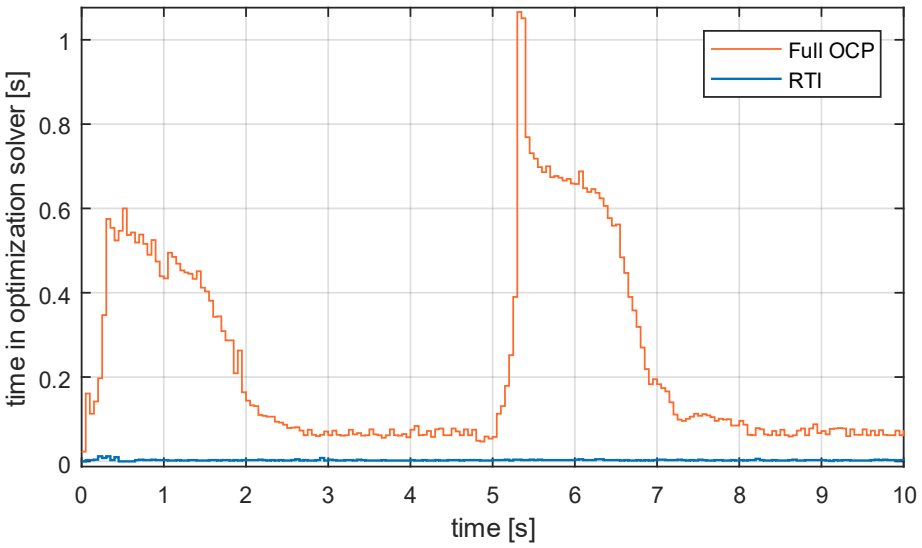

Figure 3-10: Inverted pendulum – RTI vs. converged full problem, optimization time

nonlinear OCP using the set_solveFullProblem method (see Appendix A). In both cases IPOPT was used as the optimization solver.

The Real-Time Iteration scheme linearizes the defect constraints at every sampling time. As one can appreciate form the equations of motion (3-12), these constraints are strongly nonlinear. However, the RTI algorithm performs just as well as solving the full OCP, see Figure 3-11. Furthermore, the quadratic problems can be solved in less than $0.01\ s$, whereas the full nonlinear problem may take up to $1.06\ s$ to get solved, see Figure 3-10.

It may be remarked that for solving the full OCP, the exact hessian was used (this is set when creating the FALCON.m problem), but solving with the default BFGS Hessian of IPOPT creates

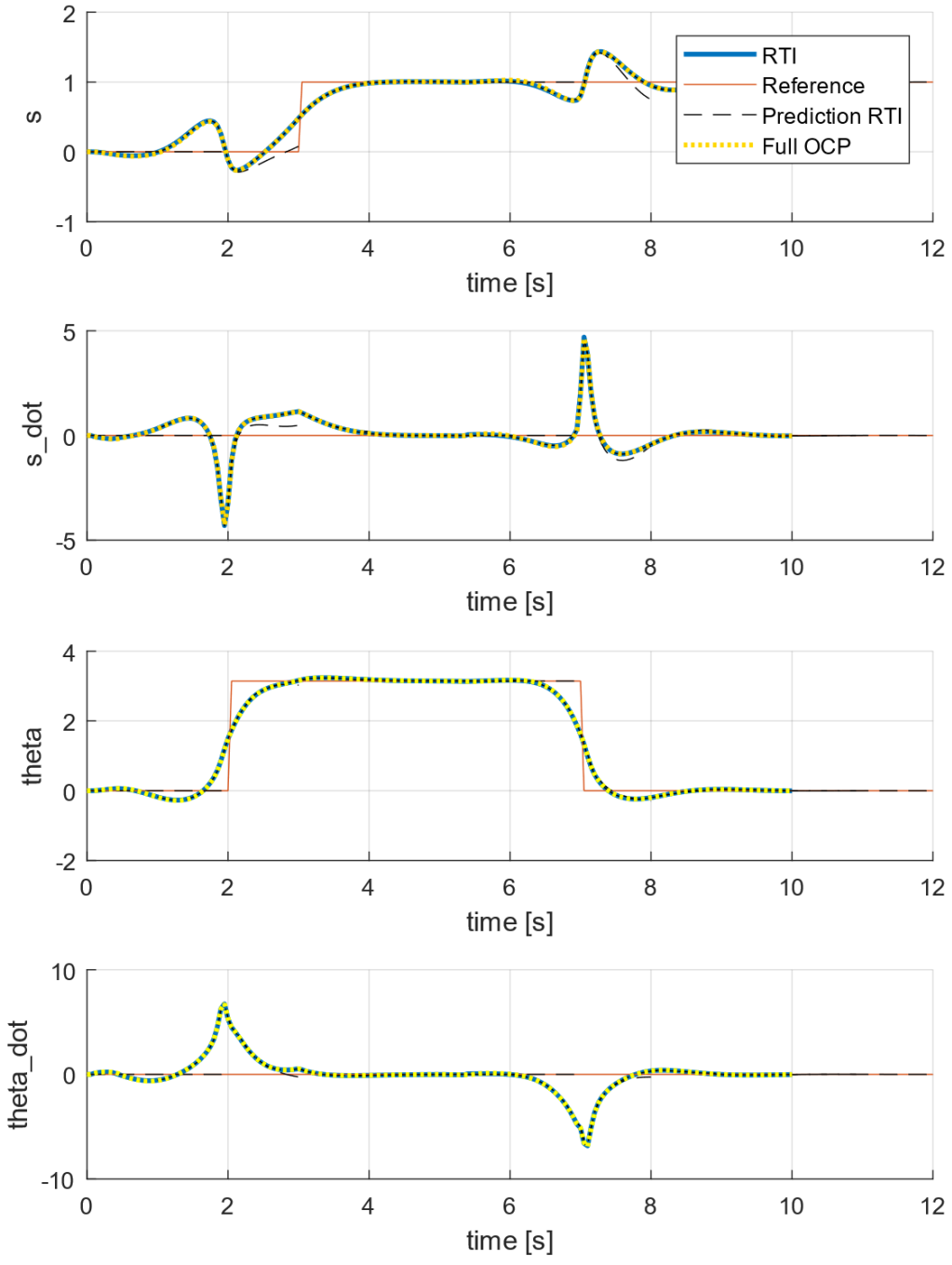

Figure 3-11: Inverted pendulum – RTI vs. converged full OCP, states

only sighlty worse results. For the full problem all constraints, the objective and the Hessian must be evaluated at every iteration of the optimization, which accounts for part of the time seen in Figure 3-10. On the other hand, for the RTI this is performed in the Preparation Phase. One can also see that, for the full OCP case, the planning of the swing-up and swing-down of the pendulum takes more computation time than the parts where the pendulum only has to be stabilized. However, solving the QP problems in the RTI algorithm always takes approximately the same computation time.

Gros et al. also present a similar test in [33] with comparable results. In this publication, other experiments can be found as well, for example, a variation of the sampling time, a variation of the horizon length, etc.

Infeasible and non-stationary setpoints

Here, the behavior of the RTI controller was tested for the case that non-stationary and infeasible setpoints are provided as reference. To test this, two constraints were added to the problem:

- The state s must be between $-1\,m$ and $1\,m$
- The horizontal position of the tip of the pendulum $x_{\text{tip}} = s + 2l\sin\theta$ must be greater than or equal to zero ($x_{tip} \geq 0$). This is a nonlinear path constraint

On top of this, for the swing-up of the pendulum, the reference for θ was set to 4 instead of π, so that this reference gives a non-stationary point. Also, the reference for s was set to $1.5\,m$ which is clearly infeasible for the first constraint described above. The results of this test can be appreciated in Figure 3-12.

In these plots one can see that:

- s is always less than $1\,m$, so that the first constraint is always satisfied, although the reference is greater
- θ is kept positive until the cart has advanced to $s = 1\,m$. This is done to satisfy the second constraint (compare to the swing-up in Figure 3-11)
- after the swing-up, θ stays at a value close to π. Although at the end of the prediction horizon the controller tries to get closer to the reference, this is never applied. The reason for this is that the reference for $\dot{s}$ and $\dot{\theta}$, which are zero, keep the system in a stationary state. For a different choice of the $\boldsymbol{Q}$ and $\boldsymbol{P}$ matrices, this might not be the case.

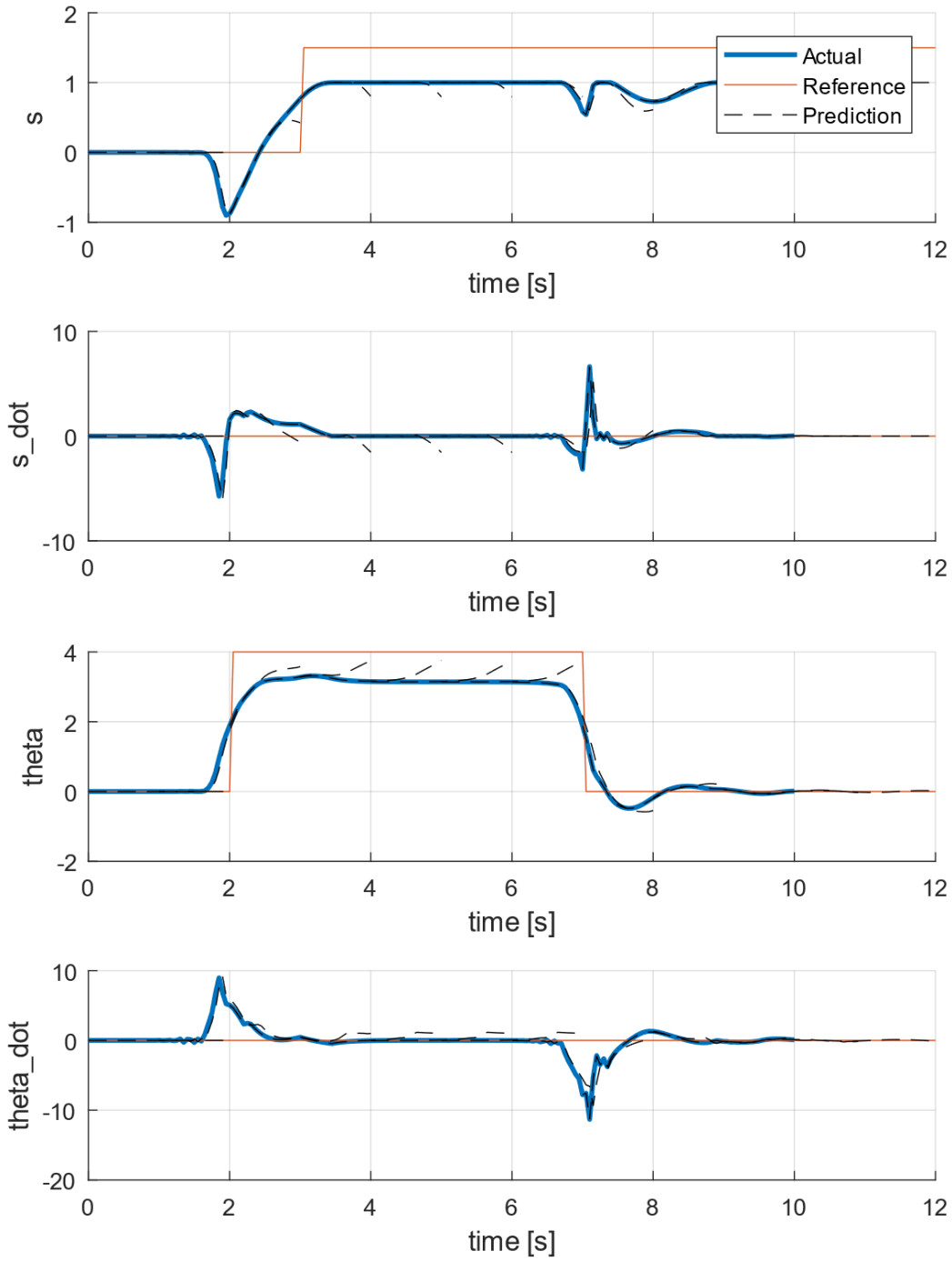

Figure 3-12: Inverted pendulum – RTI for non-stationary and infeasible setpoints

IPOPT vs. qpDUNES

With this model, the other optimization solver implemented in this thesis, qpDUNES was also tested. During different tests, it was found that qpDUNES did not manage to solve the quadratic problems appropriately when other constraints than the defect constraints become part of the active-set. Therefore, for the tests presented here, the constraints described above were removed.

Both solvers deliver the same results, not only for the trajectory planning for swing-ups, but also in the case of model mismatch or external perturbations (not depicted here). However, as one can see in Figure 3-13, the Feedback Phase, which is the phase in which the quadratic

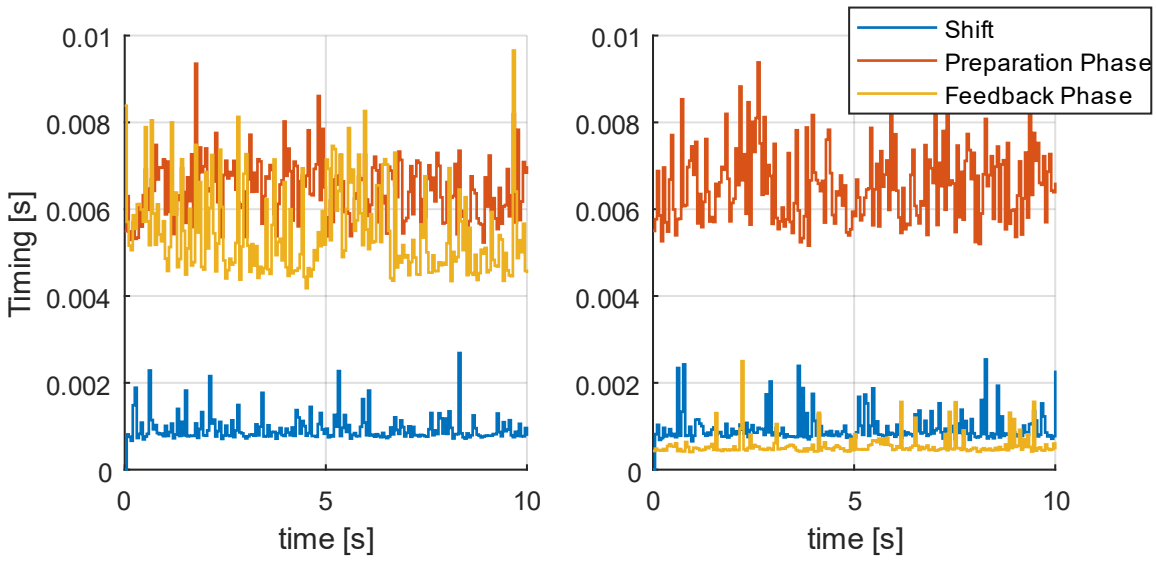

Figure 3-13: Inverted pendulum – IPOPT (left) vs. qpDUNES (right)

problem is solved, is much shorter with qpDUNES than with IPOPT. The average calculation time with qpDUNES is, for the tests performed in this thesis, an order of magnitude smaller than that with IPOPT. The main difference between the two, apart from their internal algorithms, is that IPOPT is made for solving general nonlinear problems and thus, it interfaces with MATLAB in every iteration to get new values of the objective function, constraints and their gradients. On the other hand, qpDUNES is tailored for QP problems of the form (2-27) and therefore, all of the gradients are constant, so that qpDUNES only receives one set of values at the start of the Feedback Phase.

Unfortunately, because of the issues described above, IPOPT was the solver that was used for the rest of this project. Nevertheless, as mentioned in the introduction, this project "only" delivers a module for *rapid prototyping* of NMPC algorithms that is only suitable for simulation. Therefore, the choice of using IPOPT instead of another solver does not constrain the scope of this thesis.

External perturbation

Next, it was tested how the RTI algorithm performed in the case of external perturbations. For this test, an external force of $0.5\ N$ was exerted on the center of mass m_P, perpendicular to the rotational degree of freedom θ, from second 4 until second 4.5 of the simulation. In Figure 3-14, it can be observed that at second 4 the system reacts to the external force by moving the cart in positive s direction, which was not taken into account in the prediction. This allows to keep the pendulum on top of the cart ($\theta \approx \pi$) until the external force is removed.

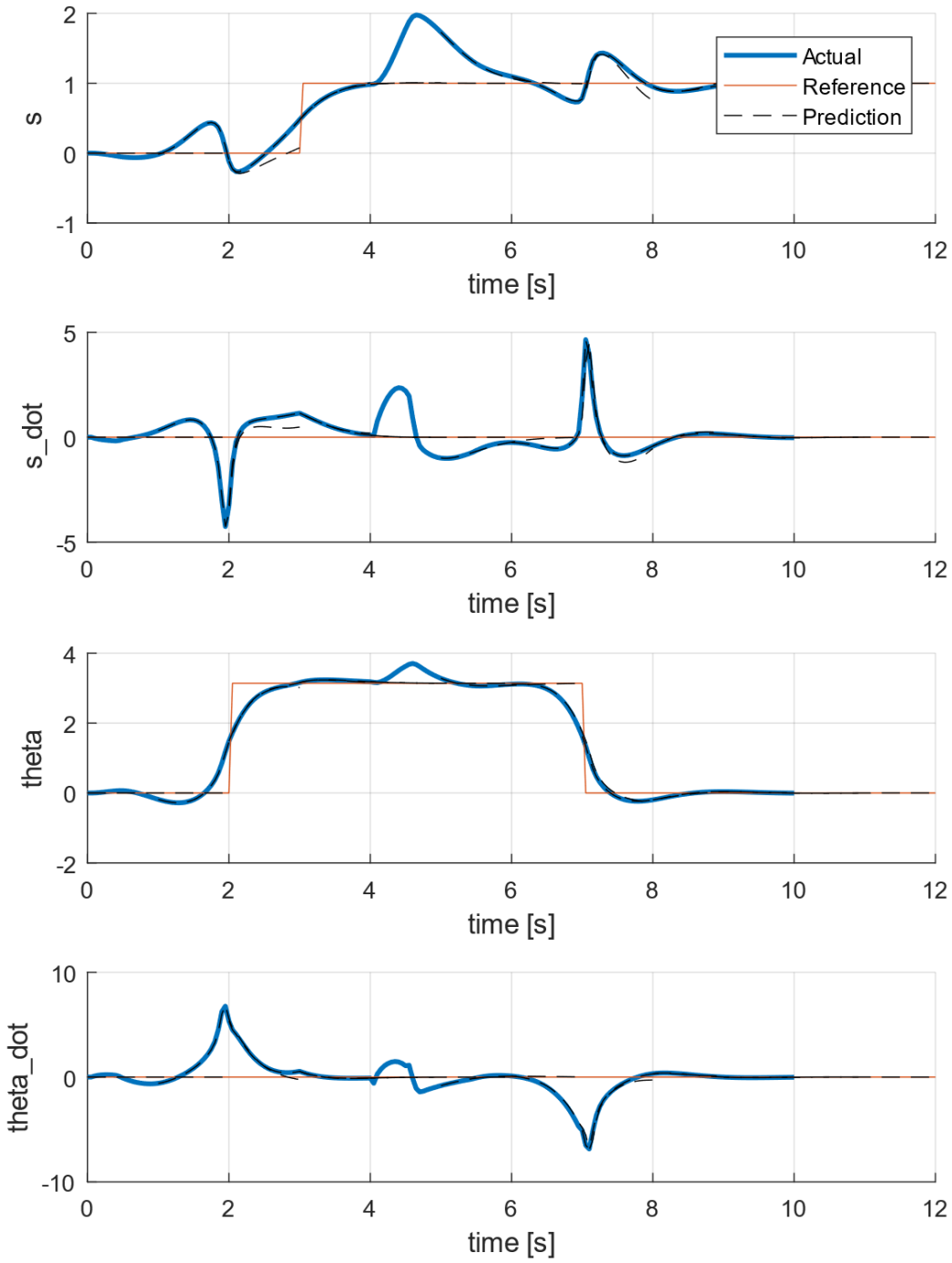

Figure 3-14: Inverted pendulum – External perturbation

Gauss-Newton vs. BFGS vs. Exact Hessian

It is known that the time-optimal objective function for the racecar problem in the next chapter is not a tracking application and thus the Gauss-Newton Hessian cannot be used for that problem. Therefore, the BFGS approximations and the regularization methods described in section 2.2.3 were implemented and tested with this model. For these tests, the discretization method was set to Forward Euler, since the exact Hessian has not been implemented for the ERK4. These tests were performed with the external perturbation described above.

It was found that the blockwise calculation of the BFGS approximation does not work stably for this model. This is true for both of the methods that keep the Hessian positive definite (see section 2.2.3), both with and without shifting the Hessian blocks. This is also true if the Lagrange multipliers are set to zero, so that the second derivatives of the constraints are neglected in the calculation.

Using the BFGS formula to calculate the whole Hessian, instead of blockwise, gives much better results. In this case, both the skipping method and the damping method for keeping the Hessian positive definite (see section 2.2.3) perform comparably well. It was found that performing a shift of the Hessian blocks is favorable, especially when the perturbation force acts on the system. Furthermore, as mentioned in section 2.2.3, the BFGS formula needs an initial matrix for the Hessian approximation. Providing a good guess for this matrix, for example the Gauss-Newton Hessian, improves the behavior of the closed-loop system significantly.

The exact Hessian also shows good results for this path-following and stabilization application. It must be noted that IPOPT performs a convexification (regularization of the Hessian) internally, so it is not necessary to provide a positive definite Hessian. However, using the $\mathrm{project}$ regularization method presented in section 2.2.3, slightly reduces the timing in the Feedback Phase. The mirror regularization method, however, destabilizes the system when the external force acts on the system. It may be remarked that the Lagrange multipliers should be shifted, as mentioned in section 2.1.3. For this procedure, it was found beneficial to set the Lagrange multipliers corresponding to the last defects to zero.

The figures in the next page depict the values in the Hessian for different methods during the swing-up (second 2) and during the perturbation (second 4.5). The color and the size of the dots represent the absolute value of the element in the matrix. These figures only show the values for the first three stages.

One can observe that the (constant) Gauss-Newton Hessian approximation differs from the others. This is because the Lagrange multipliers adopt values unequal to 0 during dynamic events, ensuring that the defects constraints are satisfied. In Figure 3-17, all the values in the diagonal have almost the same value. Especially the elements that correspond to the control values, for example $(5,5)$, are as big as the other values in the diagonal, which is not the case for the Gauss-Newton Hessian. This makes the system become sluggish and the BFGS approximation never converges to the exact Hessian. Initializing the BGFS algorithm with the Gauss-Newton Hessian improves the behavior of the system and the BFGS Hessian converges to the exact Hessian when the system is stationary. In Figure 3-19, one can see that the exact Hessian gets very big values during the external perturbation. This is because the Lagrange multipliers become larger when the system does not follow the prediction. For the BFGS Hessian, this is not the case, as it would take the algorithm more iterations to converge to the exact Hessian.

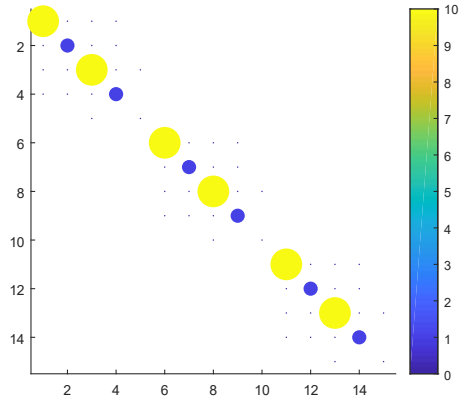

Figure 3-15: Inverted pendulum – Gauss-Newton Hessian (constant matrix)

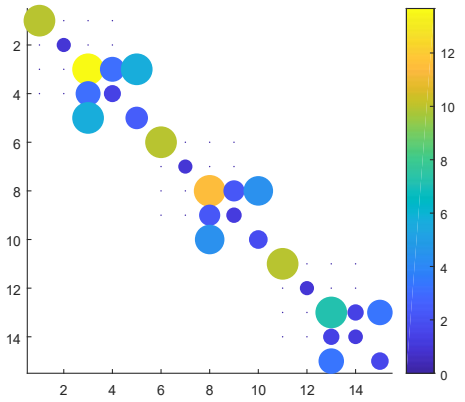

Figure 3-16: Inverted pendulum – Exact Hessian with project regularization (at swing-up)

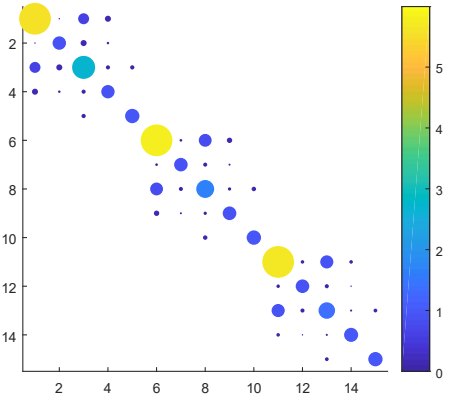

Figure 3-17: Inverted pendulum – BFGS Hessian with damping approach initialized with identity matrix (at swing-up)

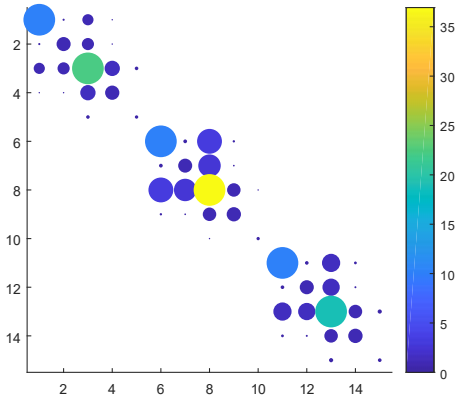

Figure 3-18: Inverted pendulum – BFGS Hessian with damping approach initialized with Gauss-Newton approximation (at swing-up)

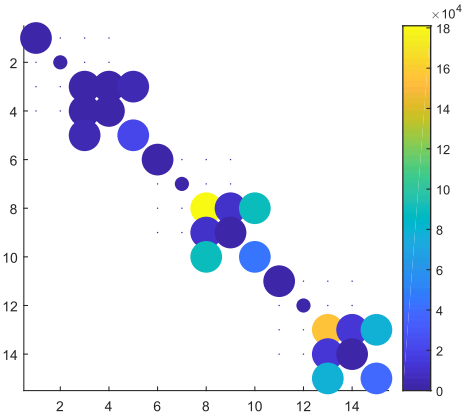

Figure 3-19: Inverted pendulum – Exact Hessian with project regularization (at perturbation)

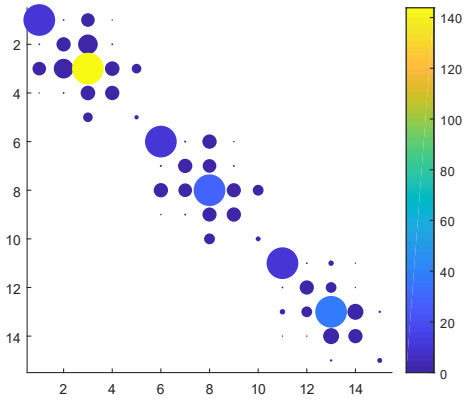

Figure 3-20: Inverted pendulum – BFGS Hessian with damping approach initialized with Gauss Newton approximation (at perturbation)

3.3 Double Inverted Pendulum

3.3.1 Modelling

Simulation model

The inverted pendulum of the previous section was then extended by a second pendulum with the mass $m_{P2} = 0.1\ kg$. In Simscape, this second mass is mounted with another revolute joint which is at a distance l from m_P and from m_{P2}. The figure below displays the schematic of this model.

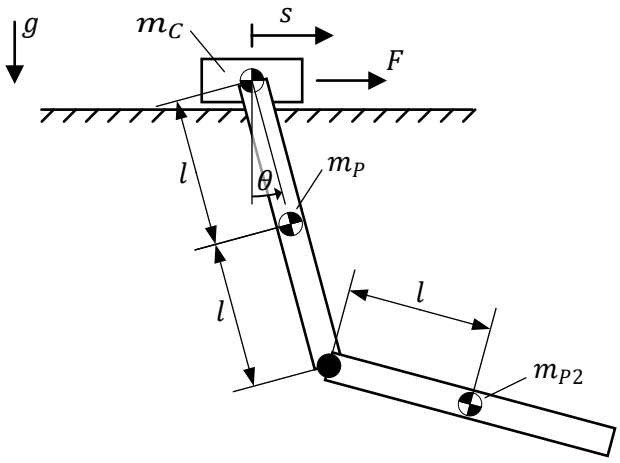

Figure 3-21: Double inverted pendulum – Schematic

NMPC model

The equations of motion of the system in Figure 3-21 can also be calculated analytically [69], however they are much more complex than the ones in the previous section (equations (3-12)). Furthermore, the objective of this simulation was to test the NMPC with an algebraic loop, as the racecar model also contains algebraic loops (see section 4.1.2). Therefore, the model for the NMPC was obtained based on the schematic in Figure 3-22.

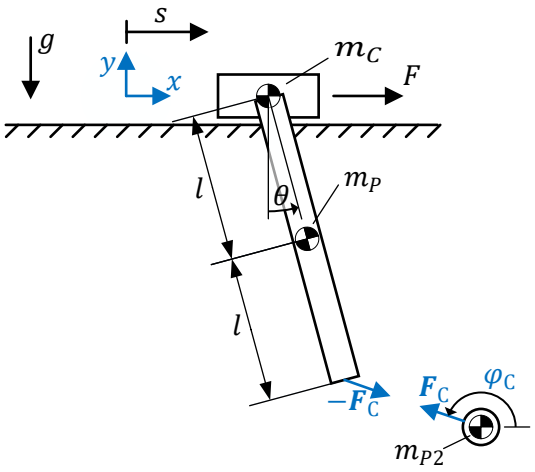

Figure 3-22: Double inverted pendulum – Schematic for NMPC model

Here, the mass m_{P2} is separated from the rest of the system and can move "freely" in the xy-plane. On this mass acts a constraining force $\boldsymbol{F}_C = F_C \begin{bmatrix} \cos\varphi_C \\ \sin\varphi_C \end{bmatrix}$ and an equal and opposite force acts on the tip of the pendulum. The angle φ_C can be calculated by

$$\tan\varphi_C = \frac{-y - 2l\cos\theta}{-x + s + 2l\sin\theta} \tag{3-14}$$

With the force $-\boldsymbol{F}_C$, the non-conservative forces become

$$\boldsymbol{Q}_{nc} = \underbrace{\begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}}_{\frac{\partial \boldsymbol{r}_F}{\partial \boldsymbol{q}}^{\mathrm{T}}} \begin{bmatrix} F \\ 0 \end{bmatrix} + \underbrace{\begin{bmatrix} 1 & 0 \\ 2l\cos\theta & 2l\sin\theta \end{bmatrix}}_{\frac{\partial \boldsymbol{r}_{F_C}}{\partial \boldsymbol{q}}^{\mathrm{T}}} \begin{bmatrix} -F_C\cos\varphi_C \\ -F_C\sin\varphi_C \end{bmatrix} \tag{3-15}$$

Augmenting these to the equations of motion (3-12) results in

$$\begin{aligned} \ddot{s} &= \frac{m_P l\dot{\theta}^2\sin\theta + m_P g\sin\theta\cos\theta + F - F_C\cos\varphi_C + 2F_C\cos\theta\cos(\varphi_C - \theta)}{m_C + m_P\sin^2\theta} \\ \ddot{\theta} &= -\frac{m_P l\dot{\theta}^2\sin\theta\cos\theta + (m_C + m_P)g\sin\theta + (F - F_C\cos\varphi_C)\cos\theta + 2F_C\left(\frac{m_C}{m_P} + 1\right)\cos(\varphi_C - \theta)}{m_C l + m_P l\sin^2\theta} \end{aligned} \tag{3-16}$$

The equations of motion of the point mass m_{P2} are trivially given by:

$$\begin{aligned} \ddot{x} &= \frac{F_C\cos\varphi_C}{m_{P2}} \\ \ddot{y} &= \frac{F_C\sin\varphi_C}{m_{P2}} - g \end{aligned} \tag{3-17}$$

The state vector of the system is thus $\boldsymbol{x} = [s \quad \dot{s} \quad \theta \quad \dot{\theta} \quad x \quad y \quad \dot{x} \quad \dot{y}]^{\mathrm{T}}$.

The constraining force F_C is inserted to the OCP as a control, so that it can be set by the optimization solver. However, a constraint must also be added, in order to maintain the distance l between the tip of the pendulum and m_{P2}. This constraint has the form:

$$l^2 - ((s + 2l\sin\theta - x)^2 + (-2l\cos\theta - y)^2) = 0 \tag{3-18}$$

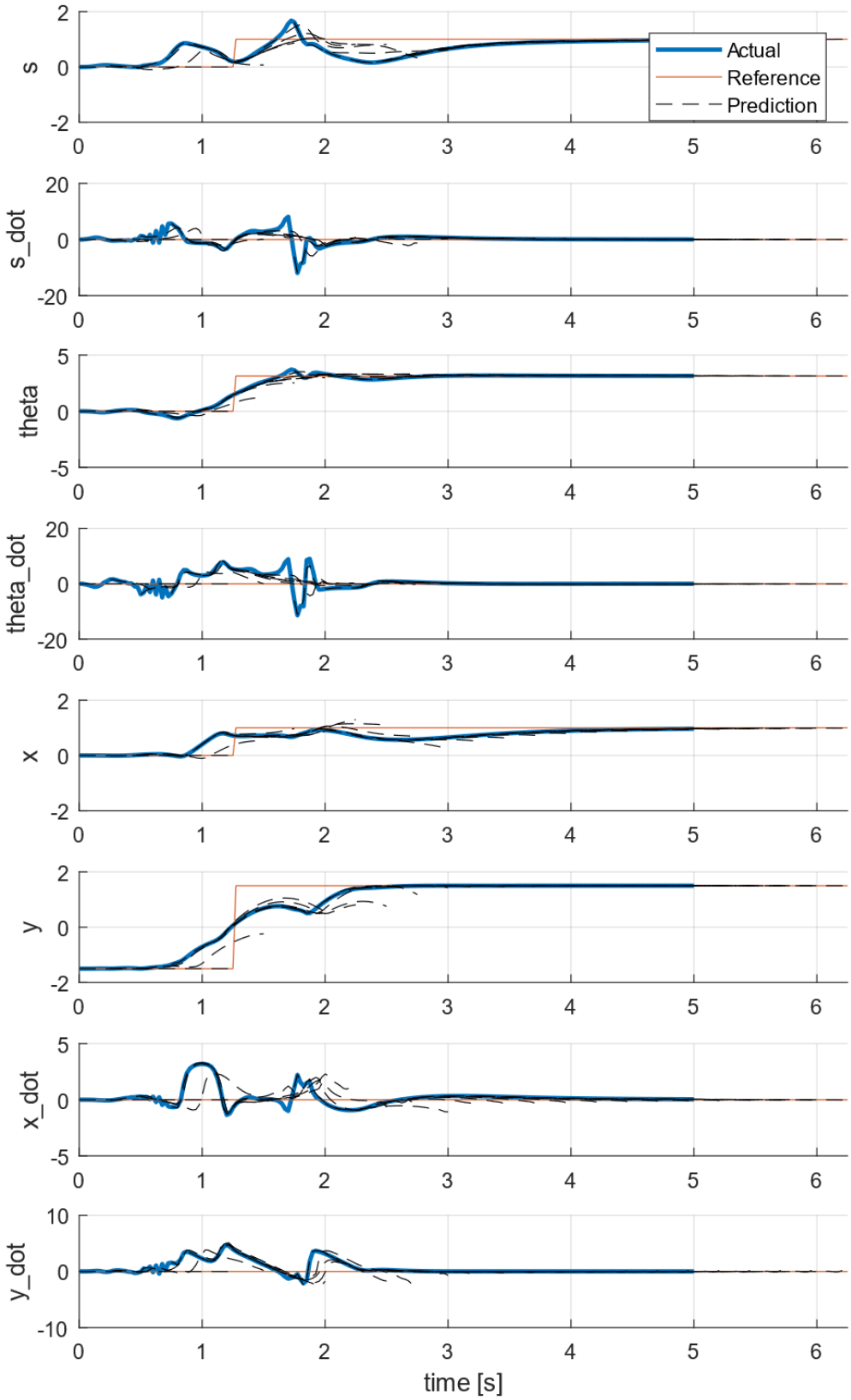

Figure 3-23: Double inverted pendulum – ERK4 discretization

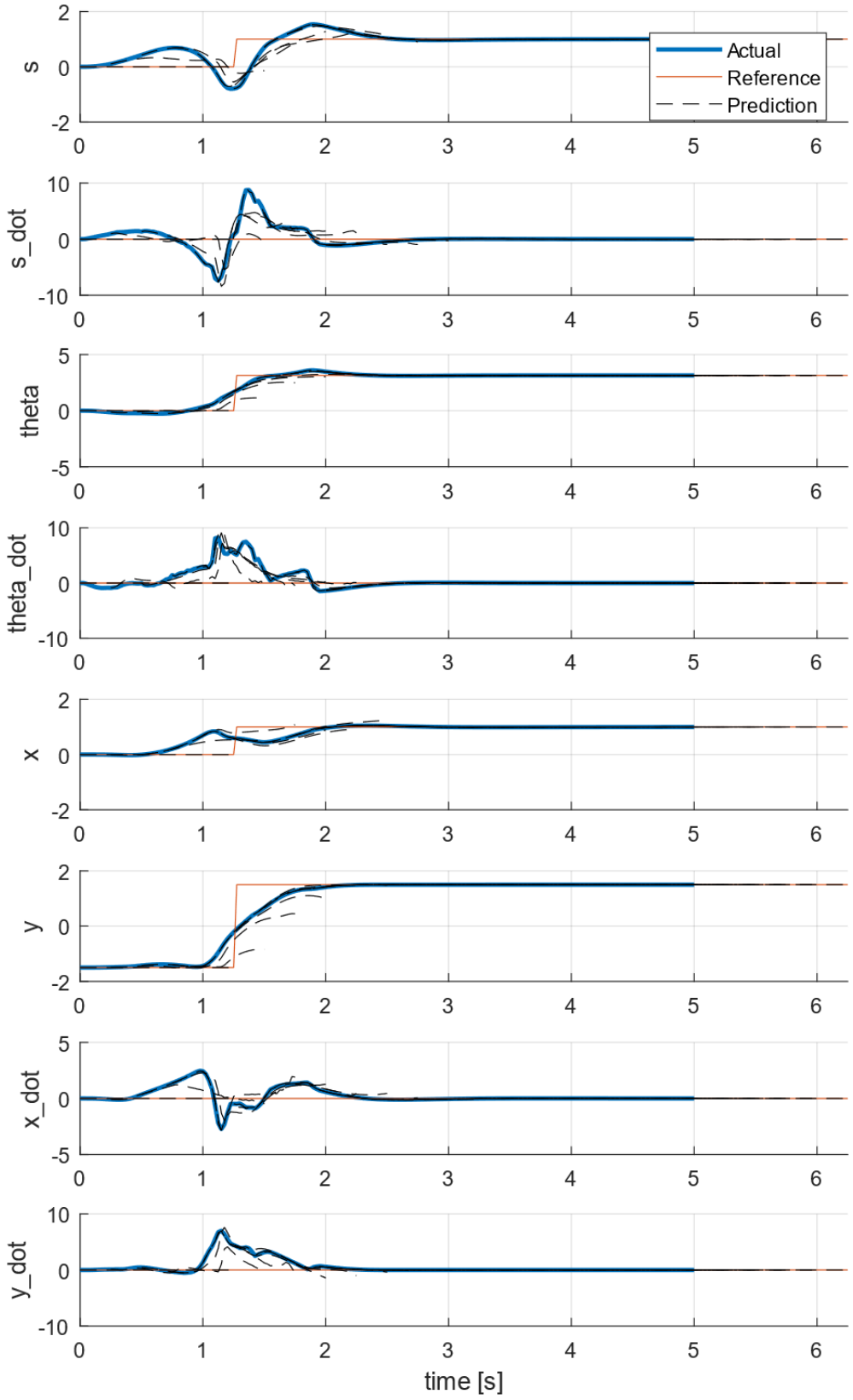

Figure 3-24: Double inverted pendulum – Trapezoidal discretization

3.3.2 Results

For the following tests with this model, the objective function (2-67) was also used, as it also concerned a tracking application. Therefore, the Gauss-Newton approximation of the Hessian could be employed. In this case, the $\boldsymbol{Q}$, $\boldsymbol{R}$ and $\boldsymbol{P}$ matrices were set to:

$$\boldsymbol{Q} = \boldsymbol{P} = \mathrm{diag}([10 \quad 0.01 \quad 10 \quad 1 \quad 0.1 \quad 10 \quad 1 \quad 1]^{\mathrm{T}})$$

$$\boldsymbol{R} = \begin{bmatrix} 0.01 & 0 \\ 0 & 0.0001 \end{bmatrix} \tag{3-19}$$

where the function $\mathrm{diag}(v)$ makes a square matrix with the elements of the vector v in its diagonal. The second nonzero value in $\boldsymbol{R}$ corresponds to the force $\boldsymbol{F}_C$ that closes the algebraic loop of the NMPC model.

The sampling time was set to $T_s = 0.025\ s$ and the horizon length to $T_H = 1.25\ s$. Note that the sampling time was halved compared to the one for the single inverted pendulum. This was done in order to cope with the more complex dynamics of this model. However, the horizon length was also reduced, so that the NMPC algorithm must plan the trajectory in a shorter time, although it has more iterations to perform the planning.

The reference trajectory concerns a swing-up of the pendulum while the position s is increased to $1\ m$. As before, this reference was inserted at the end of the prediction horizon.

ERK4 vs. Trapezoidal discretization

The figures in the previous pages show the performance of the RTI algorithm for the double inverted pendulum with two different integration methods: Explicit 4$^{\text{th}}$-order Runge-Kutta method (ERK4) and the Trapezoidal method. A similar test was performed with the 1-DoF Cart (see section 3.1.2). From those tests, the conclusion was that the ERK4 discretization outperformed the trapezoidal discretization, especially in the case of nonlinearities. However, with the double inverted pendulum, a new phenomenon becomes apparent: The algebraic loop makes this problem *stiff*, and therefore the NMPC controller performs better with the trapezoidal method.

A *stiff system* is one which has a "very stable" mode [36]. For example, the system $\dot{x} = -\lambda x$ with a very large $\lambda \gg 1$ is very stable. This can be seen in the exact solution of its differential equation $x(t) = x_0\, e^{-\lambda(t-t_0)}$, where x_0 is the initial condition of the system at time t_0. For $\lambda \gg 1$, the system decays very quickly to zero. For this kind of systems, implicit integration methods, like the trapezoidal method, perform much better than explicit ones.

An algebraic loop can be regarded as an infinitely fast mode of the system. Therefore, the NMPC model is *stiff*. One can see that with the trapezoidal method, the system reaches stationarity faster than with ERK4. As mentioned before, there exist implicit methods that assume Zero-Order Hold of the control values. Therefore, implementing one of these methods should be considered as future work for this project.

External perturbation

This system was also tested with an external perturbation. At second 5 of the simulation, a force of $1\ N$ is exerted on the mass m_{P2} perpendicularly to its rotational degree of freedom for

0.1 seconds. As can be seen in Figure 3-25 and Figure 3-26, the NMPC algorithm with the ERK4 discretization method performs similarly well as with the trapezoidal method. This is because the displacements stay relatively small, so that the linearization in the RTI method remains accurate.

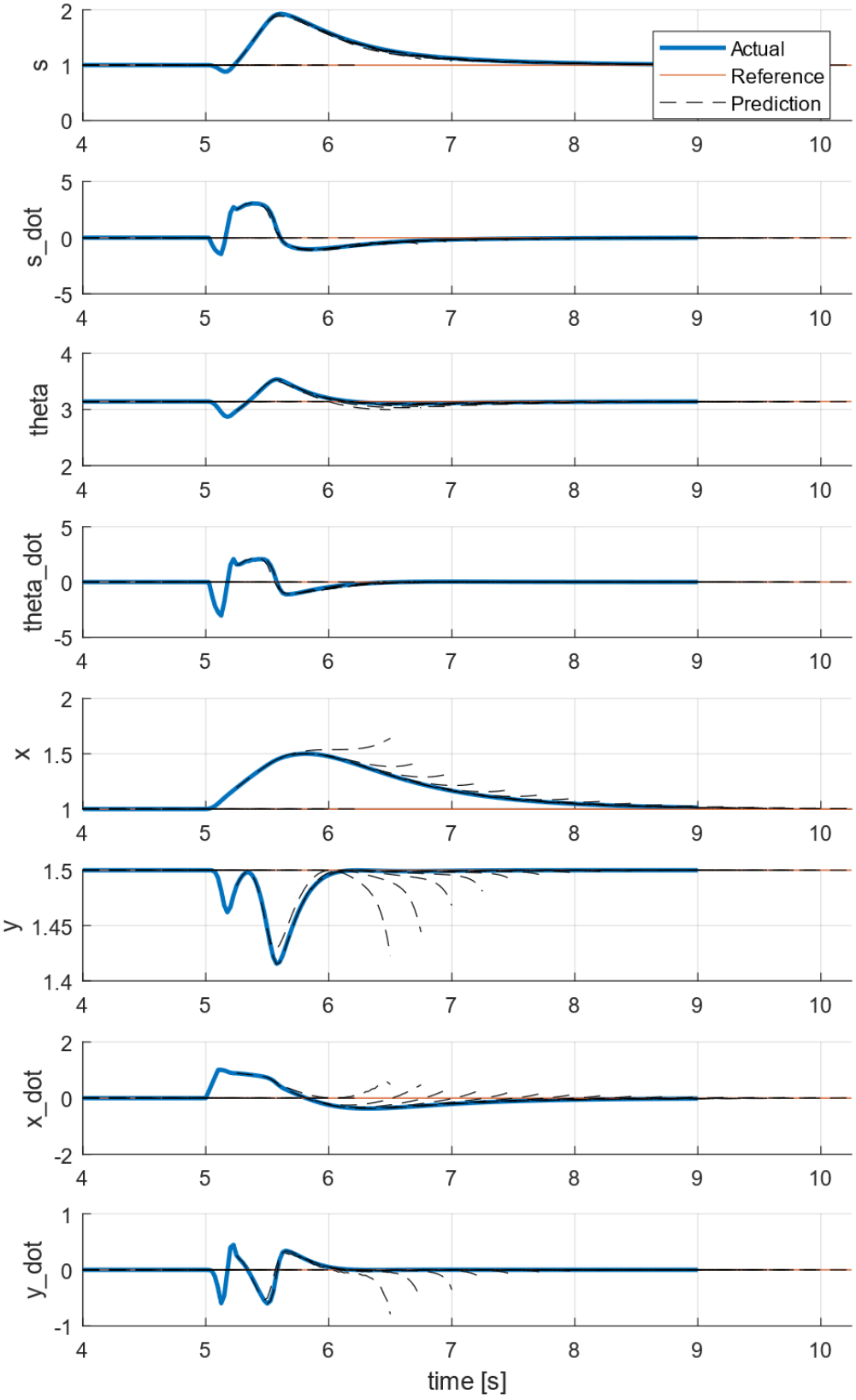

Figure 3-25: Double inverted pendulum – External perturbation with trapezoidal discretization

In the figure below, one can see some oscillations in the y-velocity of the mass m_{P2} ($\dot{y}$). These come from numerical errors resulting from the stiff system. However, the oscillations are not present in the actual system, as the algorithms used in Simscape are able to calculate the system states accurately.

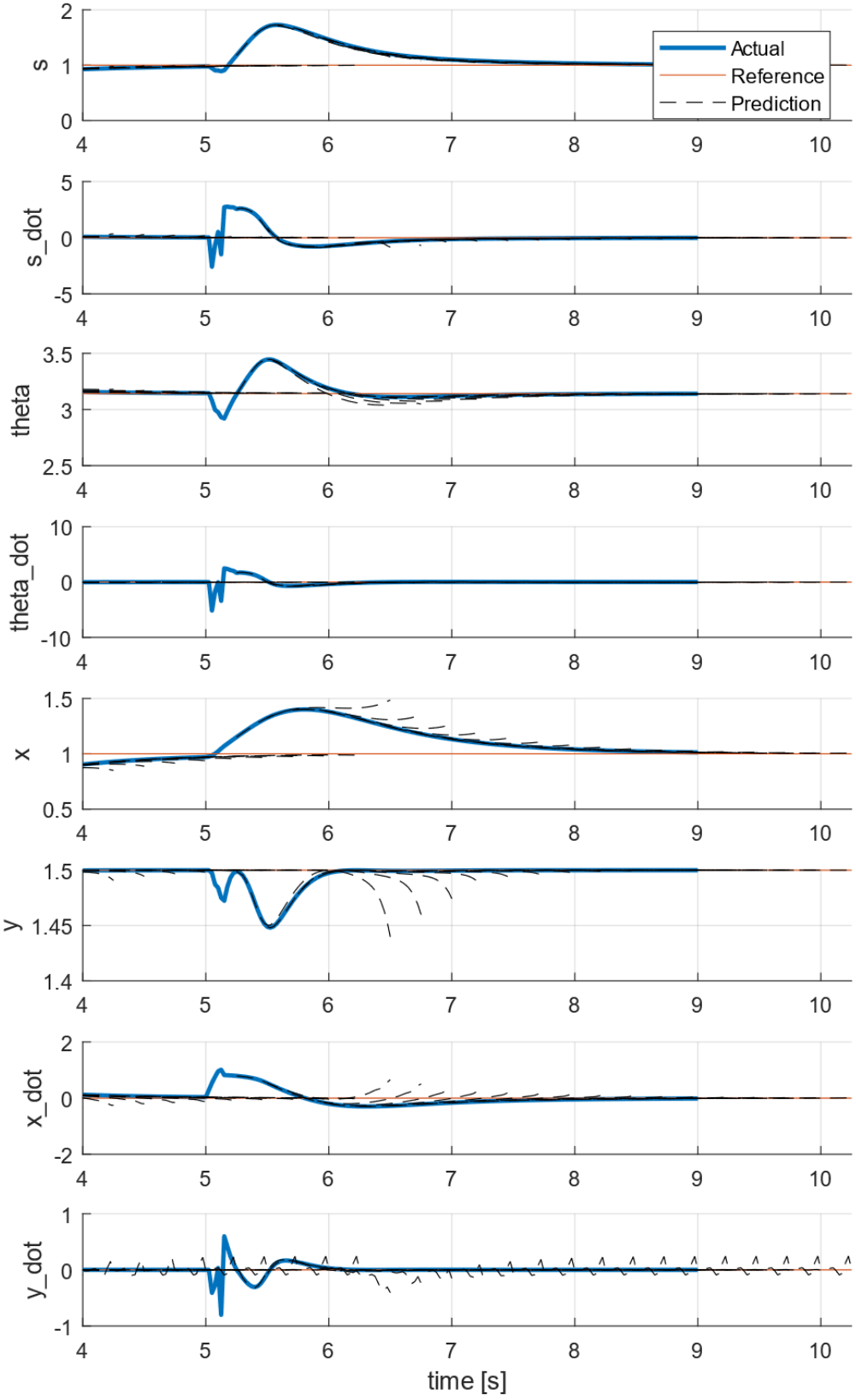

Figure 3-26: Double inverted pendulum – External perturbation with ERK4 discretization

3.4 Point Mass on Formula Student Track

Before trying the NMPC on the full autonomous racecar model, several tests were performed using a point mass model. This point mass model is not a vehicle model and therefore, it does not represent an actual car. However, it was used to gain insights about the Optimal Control Problem for time-optimal racing on a track.

3.4.1 Modelling

Simulation model

The simulation model concerns only a point mass m that can move on an xy-plane, i.e. it has two translational degrees of freedom. On this mass acts a damping force $\boldsymbol{D}$ opposite to the velocity $\boldsymbol{v}$ of the mass and an input force $\boldsymbol{F}$ at an angle ψ with respect to the x-axis. The absolute value of this force $F \in [-4\,kN\,;4\,kN]$ and the time derivative of its direction $\dot{\psi} \in [-\frac{\pi}{2} rad/_s\,;\,\frac{\pi}{2} rad/_s\,]$ are the inputs for this model. The value $\dot{\psi}$ can be compared to the yaw rate of the racecar model (see section 4.1.5), therefore ψ may be called the absolute yaw angle of the point mass. The damping force $\boldsymbol{D}$ is defined in Simscape in the joint for the point mass with a damping coefficient of $k_d = 25\,\frac{N}{m/_s}$.

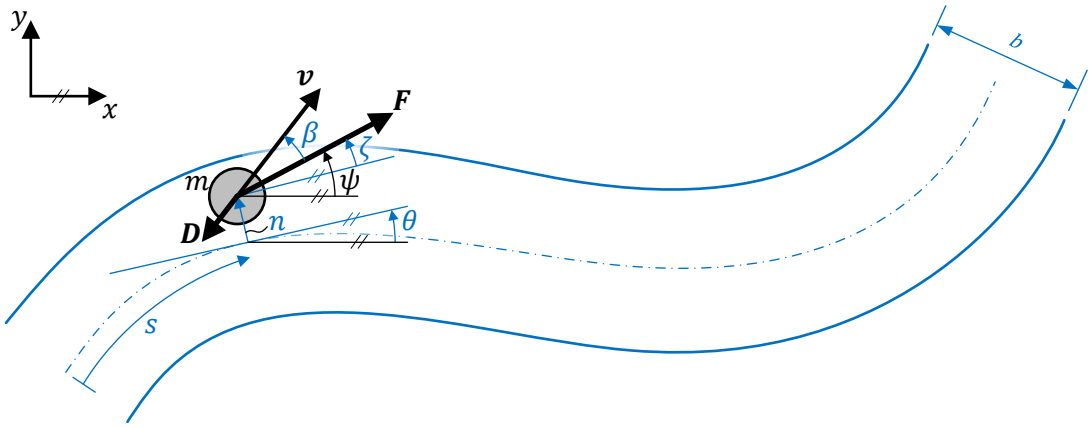

Figure 3-27: Point Mass – Schematic

To simplify the definition of the track-border constraints, the NMPC model is defined in track coordinates. The track is defined by the path coordinate s, the course angle $\theta(s)$ of the midline of the track and the track width b. In this project b was assumed constant and θ is defined by cubic splines and measured against the x-axis. For more information about how the track is constructed, see Appendix C. It must be remarked that the derivative of the course angle θ with respect to the path coordinate s gives the curvature $C = \frac{\partial\theta(s)}{\partial s}$ [70].

In order to provide the correct states to the NMPC, the states in the track-coordinate system must be calculated in the simulation model as well. These states are the path coordinate s, the

normal coordinate n, the relative yaw angle ζ and the slip angle β. The derivatives of s and n can be calculated by:

$$\dot{s} = \frac{v\cos(\operatorname{atan}\boldsymbol{v} - \theta)}{1 - n\,C}$$
$$\dot{n} = v\sin(\operatorname{atan}\boldsymbol{v} - \theta) \tag{3-20}$$

where v is the absolute value of the velocity vector $\boldsymbol{v}$ and $\operatorname{atan}\boldsymbol{v}$ is its angle with respect to the x-axis. Integrating these equations over time provides the actual values of s and n. The values of the course angle θ and the curvature C can be obtained by evaluating their splines definition using the path coordinate s.

The relative yaw angle ζ, which is the angle of the force $\boldsymbol{F}$ with respect to the tangential line of the midline of the track, and the slip angle β, which is the angle between the force vector $\boldsymbol{F}$ and the velocity vector $\boldsymbol{v}$, can be computed as:

$$\zeta = \psi - \theta$$
$$\beta = \operatorname{atan}\boldsymbol{v} - \psi \tag{3-21}$$

where the absolute yaw angle ψ is obtained by integration of the control $\dot{\psi}$.

NMPC model

The equations of motion for this model in the track-coordinate system are

$$\dot{s} = \frac{v\cos(\zeta + \beta)}{1 - n\,C}$$
$$\dot{n} = v\sin(\zeta + \beta)$$
$$\dot{v} = \frac{F}{m}\cos\beta - \frac{k_d\,v}{m} \tag{3-22}$$
$$\dot{\beta} = -\dot{\psi} - \frac{F}{m\,v}\sin\beta$$
$$\dot{\zeta} = \dot{\psi} - C\,\dot{s}$$

These equations can be compared to the equations of motion of a racing car in [47, 48, 70] and in the next chapter. The curvature C is obtained by evaluating its splines definition using the path coordinate s. Moreover, the power P that the force $\boldsymbol{F}$ exerts on the point mass is constrained to $\pm 80\ kW$. This power is obtained by

$$P = F\,v\cos\beta \tag{3-23}$$

3.4.2 Results

This is the first economic NMPC application tested in this thesis. The goal is to make the point mass go around the track in minimum time. Therefore, the objective function of the NMPC

Optimal Control Problem was formulated to maximize the distance that the point mass travels in its prediction horizon:

$$\min_{\boldsymbol{x},\boldsymbol{u}} -s_N \tag{3-24}$$

where s_N is the path coordinate at the end of the prediction horizon. Since the reference does not play a role in the optimization, it can be set to zero (or any other value). However, Falcon NMPC requires a grid of reference values, see Appendix A.

Since the differential equation for the slip angle $\dot{\beta}$ contains a division by the speed v, the NMPC cannot be started at standstill. Therefore, it is started when the speed is greater than $1\ {}^{m}/_{s}$. Before that, the controller outputs $F = 1000\ N$ and $\dot{\psi} = 0\ {}^{rad}/_{s}$. At the start, the initial guess for the optimization problem is a ramp from $0\,m$ to $100\,m$ for s, constant $10\ {}^{m}/_{s}$ for v and constant $1000\ N$ for F, all other values are zero.

The track in these tests consists of the Formula Student Germany track generated with logged data from TUfast's electric racecar *eb016*. More information on the import of the track can be found in Appendix C. The total length of the track is $1\,184$ meters and the width of the track was set to constant 4 meters. As the mass in Simscape has a diameter of $2\,m$, the normal coordinate n was limited to $\pm 1\ m$. Furthermore, the slip angle β was limited to $\pm\,{}^{1}/_{3}\,\pi$ and the relative yaw angle ζ to $\pm\,{}^{1}/_{2}\,\pi$.

It must be remarked that for the tests described next, the computation time for the solver (*IPOPT*) was limited to $0.5\ s$. In many timesteps, the solver did not converge properly, however its output was still used. Therefore, the results of these tests might not be reproduced exactly, but the conclusions following from them should still apply.

Modification on the Shift Procedure

For the first tests, Forward Euler was used as discretization method, as the tests were performed with an exact Hessian calculation. For these tests, no regularization method was used, so IPOPT performs the convexification of the quadratic problem. Furthermore, the horizon length was set to $T_H = 5\ s$ and the sampling time to $T_s = 0.05\ s$.

During the tests performed with this model, it was found that not setting the final path coordinate s_N^{guess} by forward simulation during the shift procedure (as suggested by equation (2-22)), but leaving it with its last value, results in significantly better performance of the NMPC:

$$s_N^{\text{guess}} = s_N^{*}\ \left(= s_{N-1}^{\text{guess}}\right) \tag{3-25}$$

Figure 3-28 shows the results of a simulation in which all the states were set by forward simulation and a simulation with the approach described above. Although the speed profile is not decisive for which result is better, one can see that with the approach (3-25) the lap is finished first. The reason for this is that without this approach the point mass "gets stuck" at several points in the track. This is also the case for all of the Hessian approximations implemented in this project. These simulations were repeated multiple times with similar results. Therefore, the approach (3-25) was used for the experiments described next.

At the time of writing of this thesis, the reason for why this approach improves the controller's performance is not entirely clear. However, it has to do with the fact that the gradient of the

objective function (3-24) and the gradient of the defect constraint for $\dot{s}$ in the last stage $(N-1) \rightarrow (N)$ point in the same direction, namely increasing s_N .

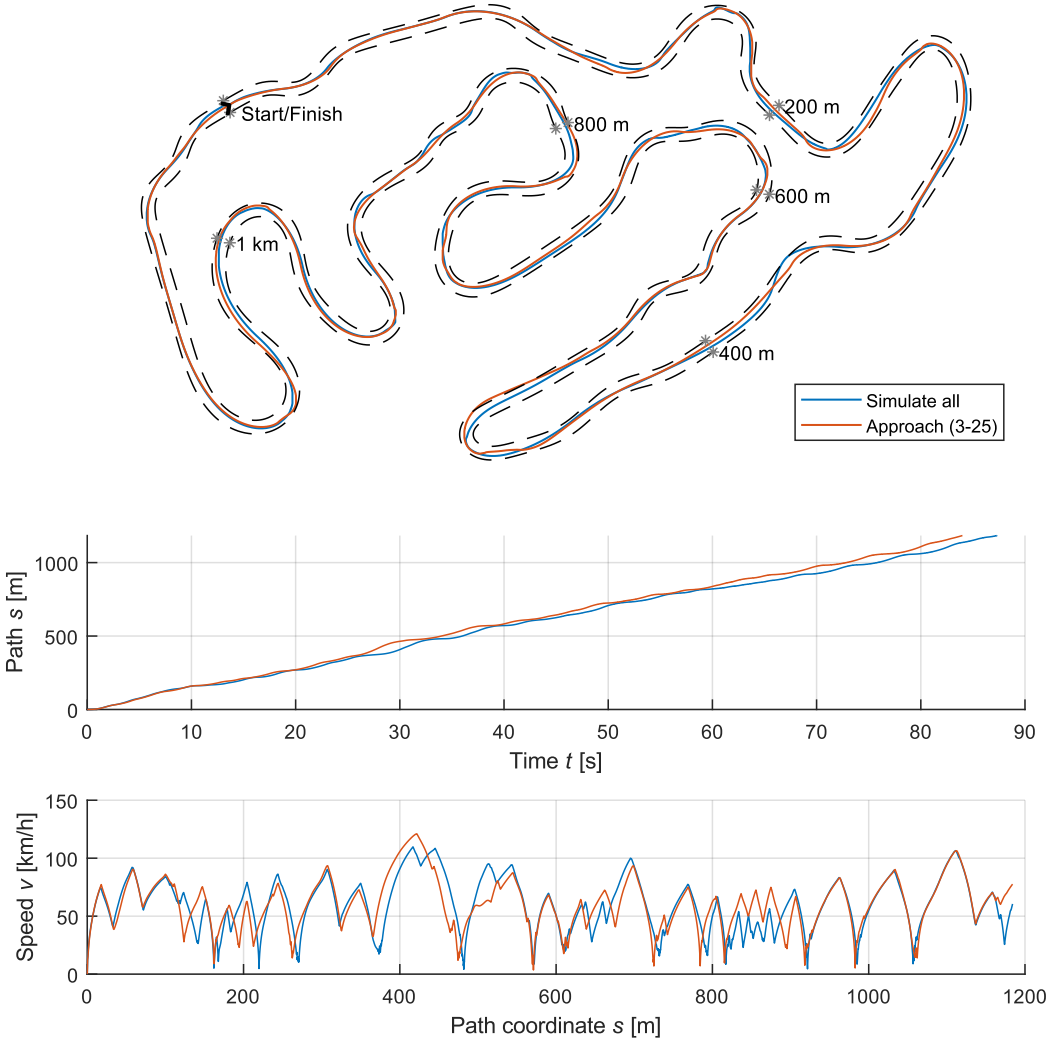

Figure 3-28: Point mass – Modification on the shift procedure

Hessian of the Objective Function vs. Hessian of the Lagrangian

In the following tests, the performance of the NMPC using a Hessian calculation with and without the second order derivatives of the constraints was compared. The second case (without the constraints) is equivalent to setting all Lagrange multipliers to zero, so that only the second order derivative of the objective function is considered. This was inspired by the fact that the Gauss-Newton approximation does not consider the second order derivatives of the constraints. For the tests described here, the horizon length was kept at $T_H = 5\ s$ and the sampling time at $T_s = 0.05\ s$. Forward Euler was used as discretization method.

Figure 3-29 shows the results of simulations using an exact Hessian calculation. The blue line considers the second order derivatives of the constraints (Hessian of the Lagrange function), the red line does not (Hessian of the objective function). In both cases, no regularization

method was used. In the figure, it can be regarded that both simulations have relatively similar results. In the speed profile, one can observe that both approaches perform equally good (the speed trajectory is almost identical) in several parts of the track. These are parts where the Lagrange multipliers have small values, because satisfying the constraints is not strongly concurrent with the objective function. The lap-time with both approaches is also almost the same.

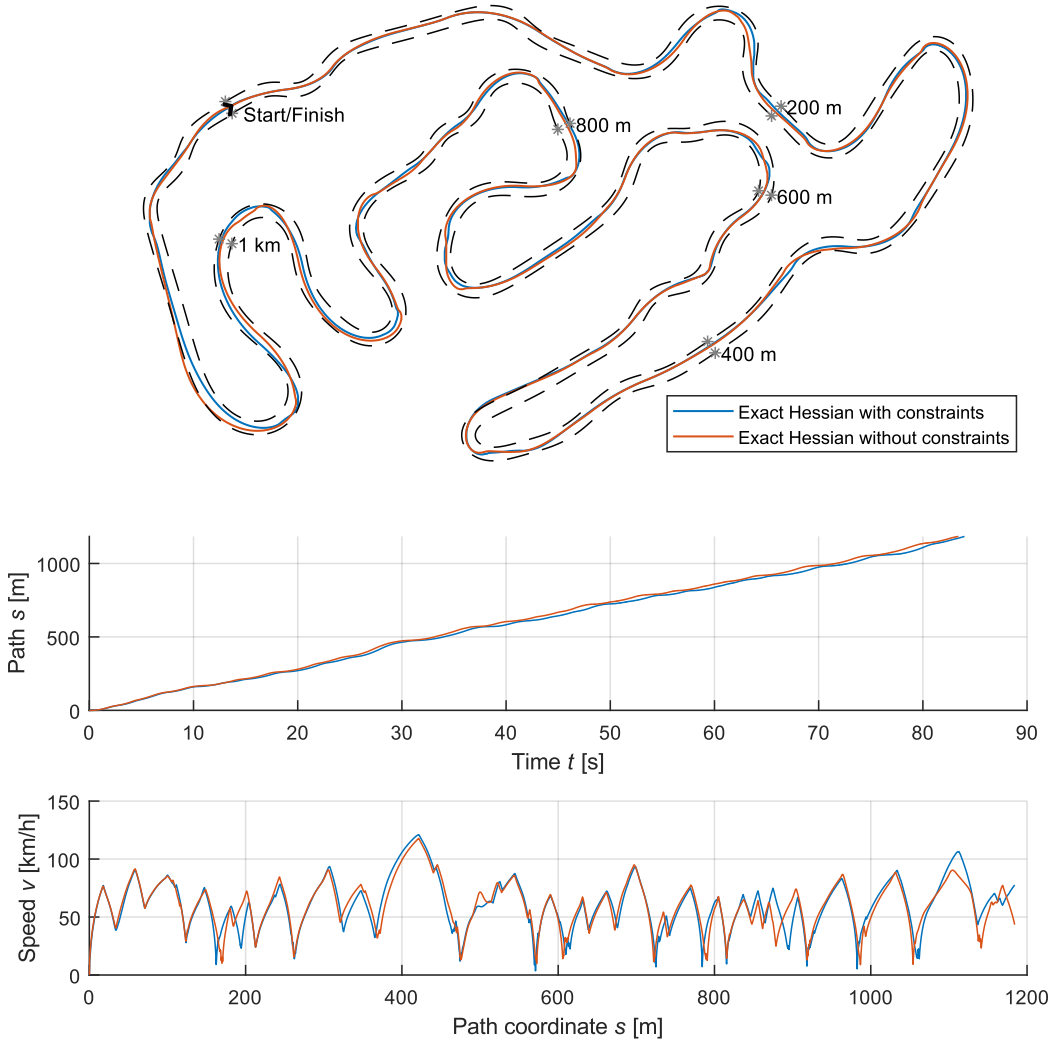

Figure 3-29: Point mass – Exact Hessian with and without second derivatives of the constraints

Similar simulations were performed using a BFGS Hessian approximation. The results are shown in Figure 3-30. These simulations show significantly poorer performance of the controller compared to the exact Hessian simulations shown above. However, it can be appreciated here as well, that in some parts of the track, both controllers give almost identical trajectories.

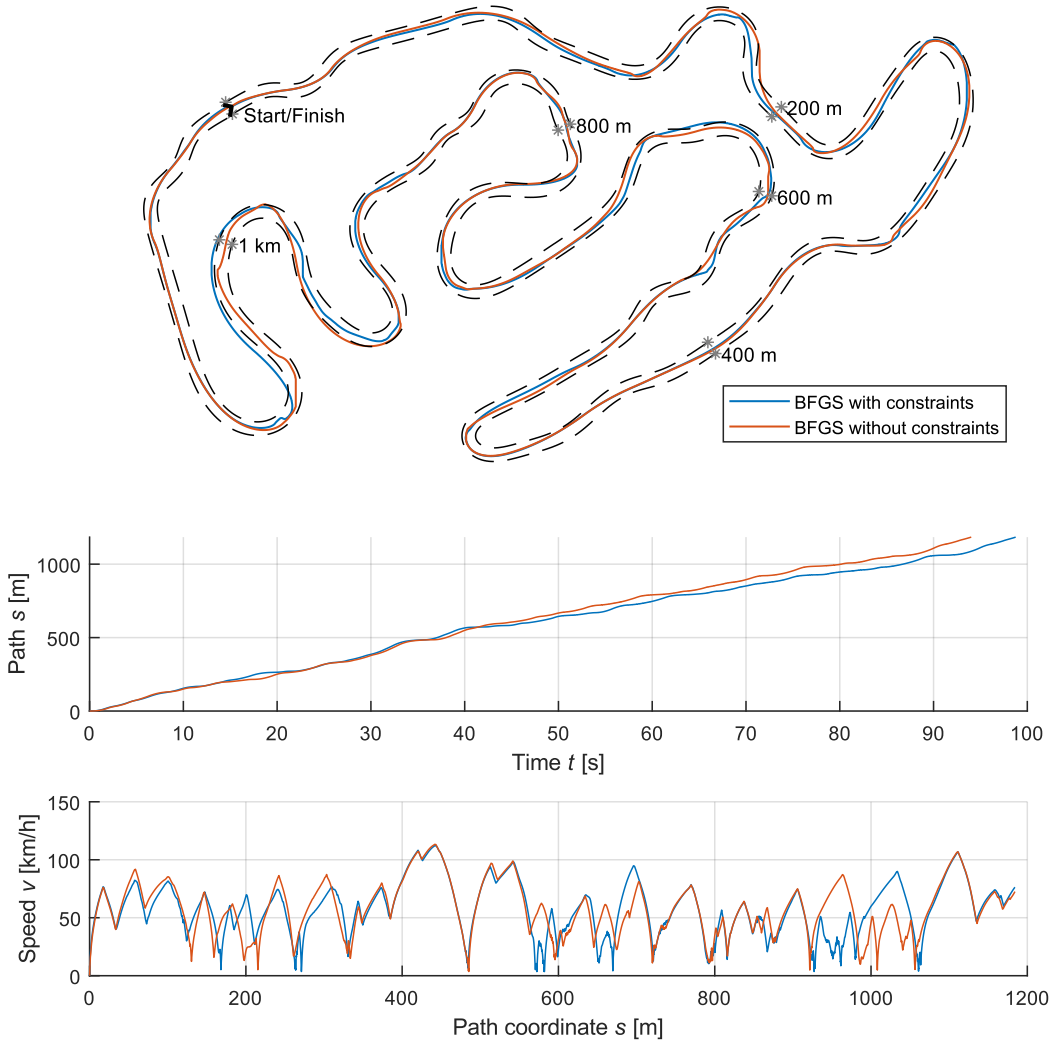

Figure 3-30: Point mass – BFGS Hessian with and without second derivatives of the constraints

Exact Hessian vs. Constant Hessian

Taking a closer look at the objective function of the Optimal Control Problem (3-24), one can see that it is a linear function $E(\boldsymbol{x}_N) = -s_N$. This means that its second derivative is zero. Therefore, if the second order derivatives of the constraints are not considered in the Hessian, the Hessian is a constant zero matrix. A zero matrix is not strictly positive definite and if it was not for the constraints, the minimization of the objective function would tend to $-\infty$.

Therefore, to try to help improve the optimization procedure, some simulations were performed with a constant Hessian. This constant Hessian would have small positive values in its diagonal, making it strictly positive definite. Note that doing this is equivalent to adding a tracking cost function of the form (2-65), where $\boldsymbol{Q}$, $\boldsymbol{R}$ and $\boldsymbol{P}$ have values only in their diagonal and the references for the states and for the controls are $\boldsymbol{x}_i^{\text{guess}}$ and $\boldsymbol{u}_i^{\text{guess}}$ respectively. As will be discussed in the next chapter, using this method provides good tuning parameters to improve the robustness of the NMC.

Figure 3-31 shows the comparison of a simulation with a constant (non-zero) Hessian and the exact Hessian without the second derivatives of the constraints, which is a constant zero matrix. For the constant Hessian, the $\boldsymbol{Q}$, $\boldsymbol{R}$ and $\boldsymbol{P}$ matrices were set to:

$$\boldsymbol{Q} = \boldsymbol{P} = \mathrm{diag}([10^{-16} \quad 10^{-16} \quad 10^{-16} \quad 10^{-16} \quad 10^{-16}]^{\mathrm{T}})$$

$$\boldsymbol{R} = \begin{bmatrix} 10^{-32} & 0 \\ 0 & 10^{-32} \end{bmatrix} \tag{3-26}$$

where the function $\mathrm{diag}(v)$ makes a square matrix with the elements of the vector v in its diagonal. For these tests, the horizon length was kept at $T_H = 5\,s$ and the sampling time at $T_s = 0.05\,s$. Forward Euler was used as discretization method.

One can see that, except for a small part at $s \approx 100\,m$, the speed profiles as well as the lap-times are almost identical. However, the path that the point mass follows with the constant Hessian is much smoother, see for example shortly before $s = 1\,km$ in the map. This is

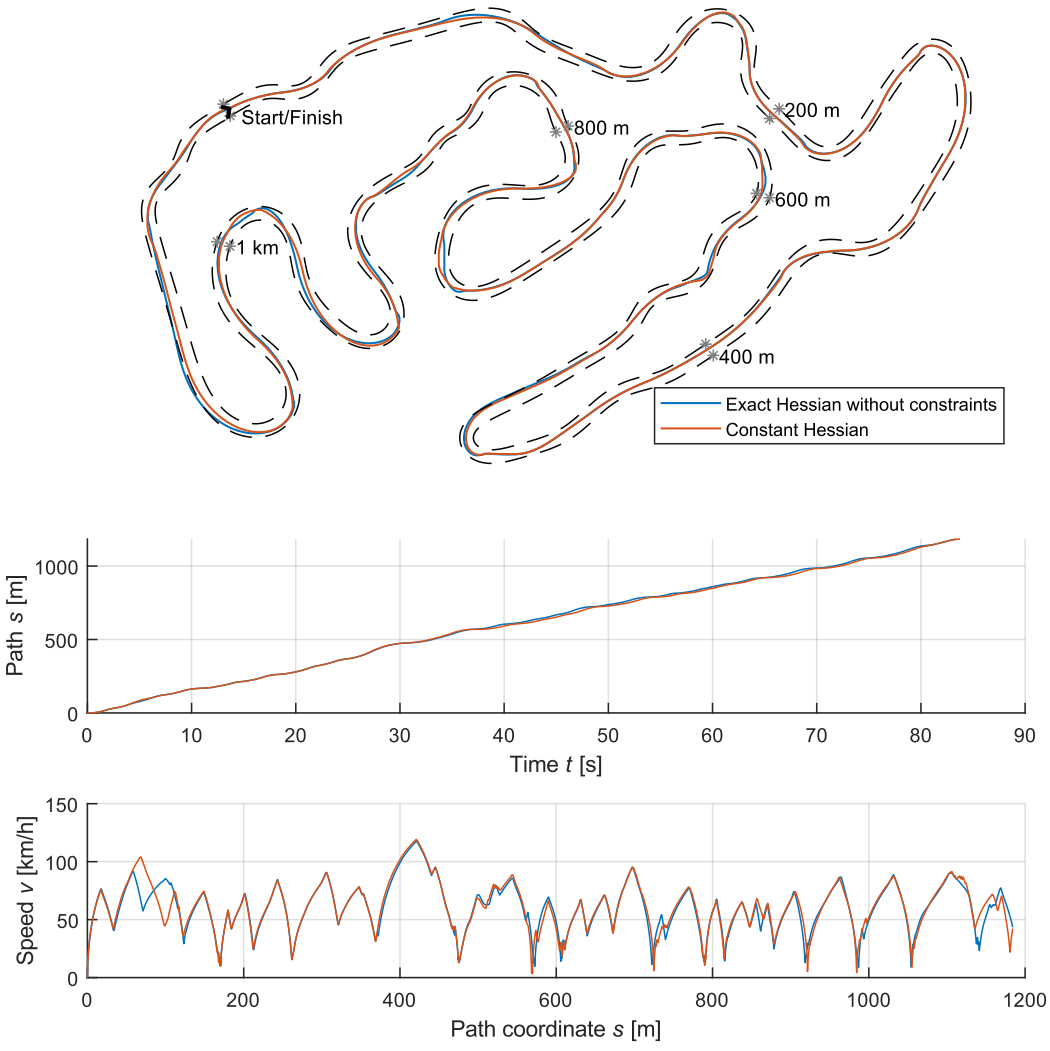

Figure 3-31: Point mass – Exact Hessian compared to Constant Hessian

because of the improved convergence of the optimization solver, as the Hessian is strictly positive definite.

Forward Euler vs. ERK4

The exact calculation of the second order derivatives of the explicit 4[th]-order Runge-Kutta (ERK4) discretization method were not implemented in this project. However, as a constant Hessian can be used without a big loss of performance (see the previous tests), the ERK4 discretization method can be compared to the Forward Euler.

The idea is to keep the NMPC using the Forward Euler method as before ($T_H = 5\ s$, $T_s = 0.05\ s$) but, with the ERK4 method, increase the sampling time to $T_s = 0.1\ s$. For this simulations the constant Hessian was kept as before, i.e. (3-26). Figure 3-32 shows the results.

One can see that the ERK4 outperforms the Forward Euler discretization method, although the sampling frequency of the latter is double the one of the ERK4. This also means that the outputs of the controller $\boldsymbol{\mu} = \boldsymbol{u}_1^*$ are kept constant (zero-order hold) for twice as long in the

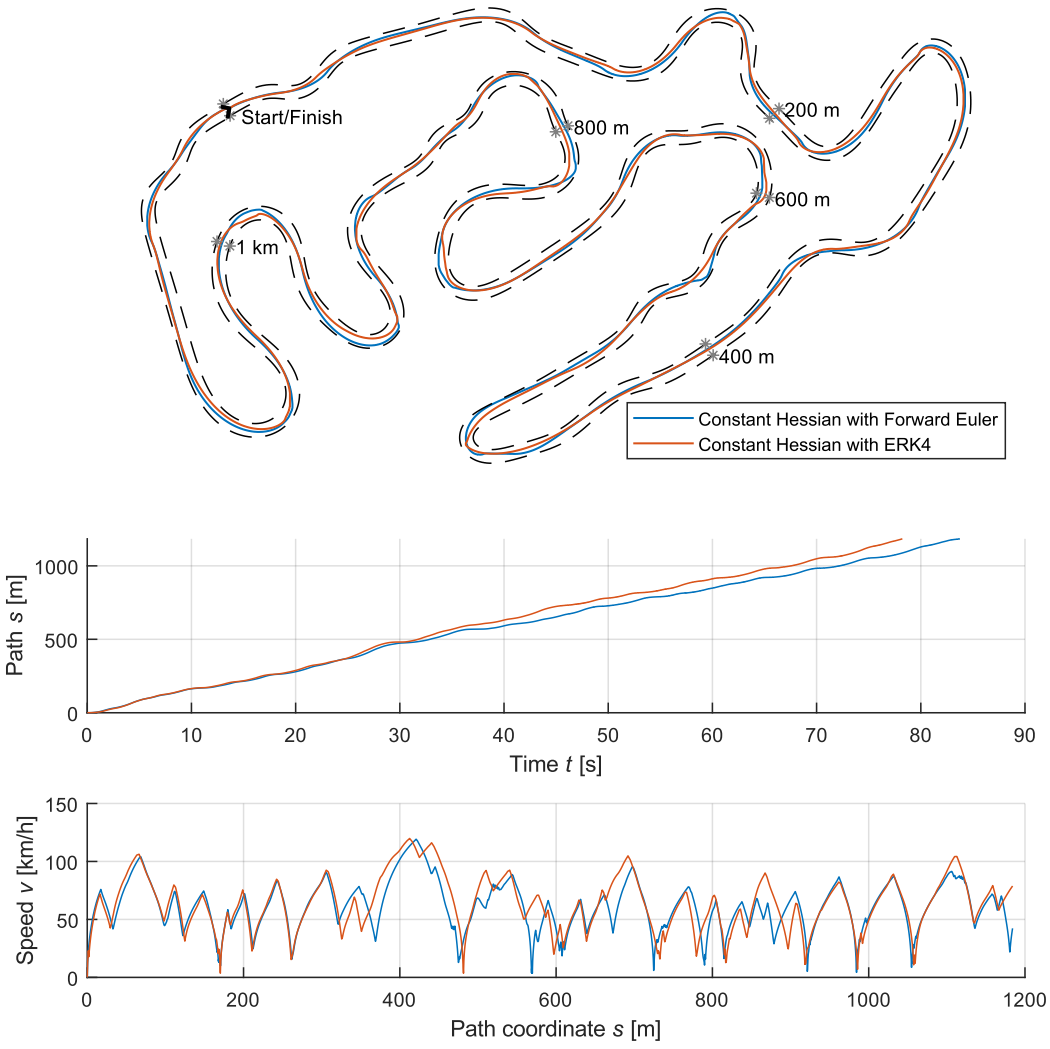

Figure 3-32: Point mass – Forward Euler compared to ERK4 with constant Hessian

ERK4 case. However, the reason why the ERK4 performs better is that its prediction is more accurate. This means that in the Feedback Phase the initial guess for the states $\boldsymbol{x}_1^{\text{guess}}$ matches the actual states $\bar{\boldsymbol{x}}$ better, so that the controller does not have to make large corrections.

As will be discussed in the next chapter, an ERK4 discretization method and a constant Hessian were used in the NMPC for the autonomous racecar model.

3.5 Summary

This section summarizes the preliminary tests performed with the NMPC algorithm during this project, as well as the conclusions following from these tests. With the 1-DoF Cart, the Inverted Pendulum and the Double Inverted Pendulum, tracking NMPC applications were regarded. An economic NMPC application was considered in section 3.4 with the point mass model.

Tracking NMPC

Table 3-1 provides a summary of the conclusions following from the preliminary tests concerning tracking NMPC applications:

Model	Test	Conclusions
1-DoF Cart	Trapezoidal discretization with vs. without approach (2-20)	The output $\boldsymbol{\mu}(\bar{\boldsymbol{x}}_n)$ of the NMPC algorithm is held constant until the next sampling time t_{n+1}. This is not considered with the trapezoidal discretization. Therefore, using approach (2-20) might improve the performance of the NMPC if trapezoidal discretization is used.
	Trapezoidal vs. ERK4 discretization	The trapezoidal method might not be robust in systems with nonlinearities, even using approach (2-20). The reasons are the same as described above. Therefore, a discretization method that considers that the output $\boldsymbol{\mu}(\bar{\boldsymbol{x}}_n)$ is held constant between sampling times usually gives better results. This is the case for the ERK4 discretization.
	Model mismatch	The NMPC controller gave a very good performance in this test. The NMPC algorithm minimizes the given objective function. Therefore, in the case of the cost (2-67), the $\boldsymbol{Q}$, $\boldsymbol{R}$ and $\boldsymbol{P}$ matrices can be used to tune the algorithm.
Inverted Pendulum	RTI vs. Converged Full OCP	If the horizon length and the sampling time are chosen correctly, the RTI scheme gives practically identical results as converging the full nonlinear OCP in every timestep. However, the RTI algorithm only takes a fraction of the time and its timing is more uniform.

(Continued in the next page…)

Table 3-1: Summary of preliminary tests on tracking NMPC applications

Model	Test	Conclusions
Inverted Pendulum (cont.)	Infeasible and non-stationary setpoints	The NMPC algorithm can handle infeasible and non-stationary setpoints as reference. It minimizes the given objective function. Therefore, in the case of the cost (2-67), the $\boldsymbol{Q}$, $\boldsymbol{R}$ and $\boldsymbol{P}$ matrices can be used to tune the algorithm. Although the RTI method linearizes all constraints, it also handles nonlinear constraints satisfactorily.
	IPOPT vs. qpDUNES	Using a dedicated QP solver like qpDUNES for solving the quadratic problems in every timestep of the RTI algorithm significantly reduces the time spent in the Feedback Phase of the algorithm. However, IPOPT is much more versatile and was therefore used for the rest of this thesis.
	External perturbation	The NMPC reacts satisfactorily to external perturbations. In this case also, the $\boldsymbol{Q}$, $\boldsymbol{R}$ and $\boldsymbol{P}$ matrices can be used to tune the performance algorithm.
	Gauss-Newton vs. BFGS vs. Exact Hessian	The different Hessian approximations were compared graphically. The BFGS approximation calculated blockwise did not work stably in this test. However, if the BFGS formula is used for the full Hessian gives good performance, especially if a shift procedure is performed and the Hessian approximation is initialized. Concerning the exact Hessian, the project regularization method gives good results. However, the mirror regularization method makes the system unstable in the case of external perturbations.
Double Inverted Pendulum	ERK4 vs. Trapezoidal discretization	The NMPC model includes an algebraic loop and is therefore stiff. Since implicit integration methods (like the trapezoidal) work better for stiff systems, the trapezoidal method with approach (2-20) gave better results than the ERK4 in these tests. However, in case of strong nonlinearities, using the trapezoidal method might make the system unstable, see the results with the 1-DoF Cart.
	External perturbation	The NMPC also reacts satisfactorily to external perturbations with this model. This was tested with the ERK4 and the trapezoidal discretization methods.

Table 3-2: Summary of preliminary tests on tracking NMPC applications (continued)

Economic NMPC

Here, the results and conclusions that followed from the tests concerning an economic NMPC application are summarized. The insights that were obtained in these tests will be applied for the autonomous racecar model in the next chapter.

Model	Test	Conclusions
Point Mass on Formula Student Track	Modification on the Shift Procedure	In this test, it was found that a slight modification on the shift procedure (before the Preparation Phase), significantly improves the performance and the robustness of the NMPC formulation for maximum progress in the defined time horizon. Note that this is not a general conclusion for any economic NMPC application. However, this modification will be applied for the autonomous racecar, as the objective function is identical.
	Hessian of the Objective Function vs. Hessian of the Lagrangian	These tests showed that neglecting the second-order derivatives of the constraints in the Hessian gives similar results as incorporating them in the QP problem. This means using the Hessian of the objective function instead of the Hessian of the Lagrange function. This was inspired by the fact that the Gauss-Newton approximation also neglects the second-order derivatives of the constraints. These tests were performed with an exact Hessian calculation and with a BFGS Hessian approximation.
	Exact Hessian vs. Constant Hessian	The exact Hessian of the objective function is a zero matrix. If this matrix is used for the QP problem in the RTI scheme, the quadratic problem is not strictly convex. Therefore, using a matrix with small positive numbers in its diagonal improves the convergence of the QP problem significantly. This also improves the performance of the controller.
	Forward Euler vs. ERK4	These tests compared the performance between the Forward Euler discretization method and the ERK4 method with a larger sampling time. Due to its better accuracy in the prediction, the ERK4 discretization method gave better results despite the sampling time being twice as big.

Table 3-3: Summary of preliminary tests on economic NMPC applications

4 Autonomous Racecar

This chapter contains the core of this thesis, namely the implementation of the Nonlinear Model Predictive Control algorithm for an autonomous racecar. As mentioned in the introduction, the characteristics and parameters of the vehicle presented here correspond to a Formula Student car, specifically the *eb016*. This is an electric all-wheel-driven racecar car of the Formula Student team of the Technical University of Munich. More details about this car can be found in Appendix B.

The first section of this chapter describes how the simulation model and the model used for the NMPC are built and how they differ from each other. A brief summary comparing both models can be found in subsection 4.1.7. Then, section 4.2 discusses how the NMPC was tuned and presents the results of the simulations.

4.1 Modelling

For both the simulation model and the NMPC model, the standard *vehicle coordinate system* (x_V, y_V, z_V) defined in the ISO 8855 [71] was used. The *vehicle reference point* used for this system is the center of gravity, simplifying the construction of the Newtonian equations of motion. In this system, the x_V-axis points forward, the y_V-axis to the left and the z_V-axis upward. This coordinate system is displayed in Figure 4-1.

Also relevant for this thesis is the *wheel coordinate system* (x_W, y_W, z_W), which is also defined in the ISO 8855. In this system, the x_W-axis is parallel to local plane of the road, the y_W-axis is the wheel rotation axis and the $x_W z_W$-plane is the midplane of the tire, so that z_W points upwards. In this thesis, the toe and camber angles as well as the Ackermann steering angle are neglected, and the road is assumed to be flat. This means that the z_V and the z_W axes of all wheels are always parallel and the x_W of both front tires are deflected only by the same steering angle δ_F with respect to x_V. The wheel coordinate system is used in this thesis to represent the forces and torques acting on each wheel. In the following sections, this system is also denoted by two letters corresponding to each wheel, for example FL for the front-left wheel.

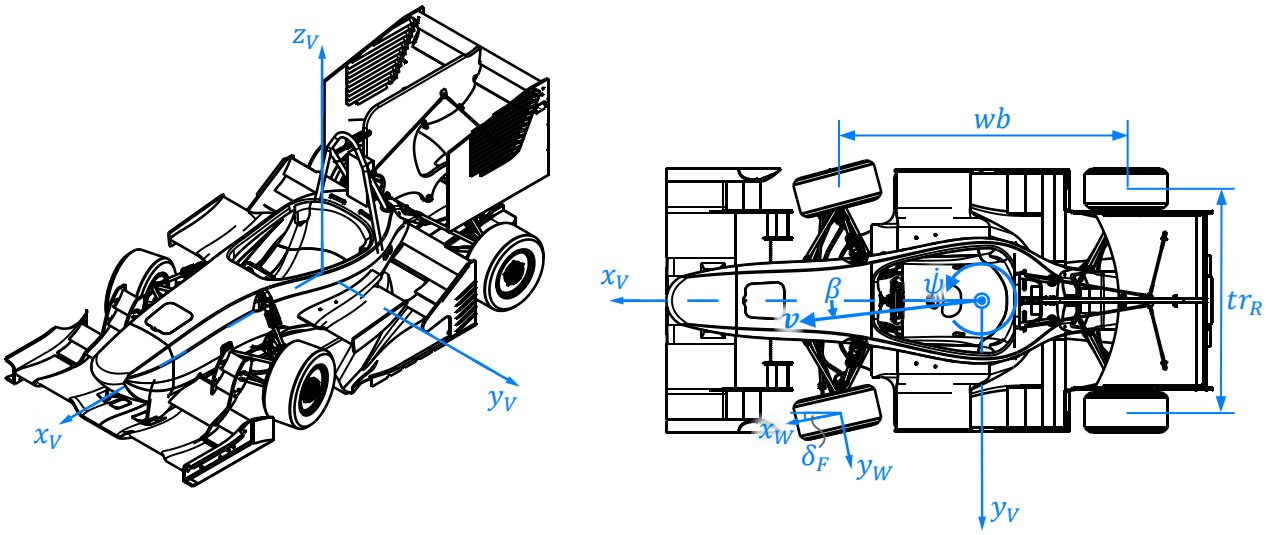

Figure 4-1: Racecar modelling – Coordinate system and states

Furthermore, the ISO 8855 also defines other values that are used in this thesis. The slip angle β is the angle between the velocity vector $\boldsymbol{v}$ of the center of gravity of the vehicle and the x_V-axis. The yaw angle ψ is the angle between the x_V-axis and the x_E-axis of an earth-fixed frame, which, like in this project, is usually assumed to be an inertial frame. The time derivative of this angle gives the angular velocity of the vehicle about its z_V-axis, which is known as its yaw rate $\dot{\psi}$. Note that the definitions in the norm are slightly different from the ones presented here. However, the pitch and roll motions of the vehicle as well as its translation in z_V-direction are neglected in this thesis, and thus the definitions presented here are equivalent to the ones in the norm.

It may be noted that the subsystems presented in the following subsections are ordered in the same sequence as they are calculated in the NMPC model.

4.1.1 Aerodynamic Forces

Simulation model

Over the last few years, Formula Student teams have spent lots of effort into developing high-downforce aerodynamic packages for their racecars. The motivation for this is to increase the aerodynamic grip of the car to achieve higher lateral (and longitudinal) accelerations and be able to drive at higher speeds in the corners. An example of these racecars is found in Figure 1-2. In this thesis, the aerodynamic forces were modelled as follows:

$$L = \frac{1}{2}\,\rho\, C_L A\, v^2$$
$$D = -\frac{1}{2}\,\rho\, C_D A\, v^2 \qquad \textbf{(4-1)}$$

where L is the lift (negative downforce) and D is the drag, $\rho = 1.225\,{}^{kg}/_{m^3}$ represents the air density, v the absolute value of the velocity of the car and $C_L A = -5.78\,m^2$ and $C_D A = 1.82\,m^2$ are the lift and drag coefficients respectively, multiplied by their reference surface area A. These last values are obtained by CFD simulations and validated on track with spring-travel and ride-height sensors with constant-speed tests and coasting tests.

The drag D acts opposite to the velocity vector $\boldsymbol{v}$. This means that in x_V-direction one obtains $D\cos\beta$ and in y_V-direction $D\sin\beta$. The lift L acts in z_V direction and is split between the front and rear axle by the aerodynamic balance $bal_A = 0.46$, see the next subsection. It may be remarked that the shifting of wheel loads due to the torque that the drag generates on the vehicle is already considered in this aerodynamic balance.

NMPC model

For the NMPC model, the aerodynamic forces are calculated identically as for the simulation model.

4.1.2 Wheel Loads

The vertical forces of the tires need to be calculated dynamically, as they are constantly changing depending on the state of the vehicle. For example, during a braking procedure the wheel loads are shifted to the front, while in a corner the outer wheels get more load than the

inner ones. This is known as *wheel load redistribution*. The wheel loads are important inputs for the tire model described in the next subsection.

Simulation model

For the simulation model, a steady-state wheel load redistribution calculation was used, thus neglecting the damping forces. A similar calculation is described in [72]. The inputs for this calculation are:

- the overall mass $m = 250\ kg$, which is the sum of the sprung mass m_{Spr} and the unsprung masses m_{Unspr} . It may be remarked that the overall mass used in this thesis includes the weight the driver. This is done to be able to compare the results of the NMPC algorithm to an actual driven lap, see section 4.2
- the unsprung mass $m_{Unspr} = 14\ kg$, which includes the mass of the wheel, the tire, the wheel hub, the upright, the (outboard) motor and gear box, and half of the mass of the suspension for each wheel. Thus, $m = m_{\mathrm{Spr}} + 4 \cdot m_{\mathrm{Unspr}}$.
- the distance $x_{\mathrm{CoG}} = 0.78\ m$ between the center of gravity (CoG) of the overall mass and center of the front axle
- the height $z_{\mathrm{CoG}} = 0.3\ m$ of the center of gravity (CoG) of the overall mass with respect to the road
- the height of the center of gravity of the unsprung masses, which for this calculation is assumed to be equal to the undeflected tire radius $r_{\mathrm{tire}} = 0.224\ m$
- the wheel base $wb = 1.55\ m$ and the front and rear track widths $tr_{\mathrm{F}} = 1.2\ m$ and $tr_{\mathrm{R}} = 1.2\ m$ (see Figure 4-1)
- the height of the roll axis at the front and rear axles $z_{\mathrm{RC,F}} = 0.044\ m$ and $z_{\mathrm{RC,R}} = 0.079\ m$, also known as front and rear roll centers
- the roll moment distribution $\Phi_{\mathrm{Roll}} = 0.6$, which is calculated considering the springs' and anti-roll-bars' (ARBs) stiffnesses.
- the lift L calculated by equation (4-1) and the aerodynamic balance bal_{A}
- the longitudinal and lateral accelerations a_x and a_y of the CoG of the overall mass
- the gravitational acceleration of the earth $g = 9.80665\ {}^{m}/_{s^2}$

First, the sprung mass acting on the front and rear axles in standstill can be calculated respectively by:

$$m_{\mathrm{Spr,F}} = m \cdot \frac{wb - x_{\mathrm{CoG}}}{wb} - 2 \cdot m_{\mathrm{Unspr}}$$
$$m_{\mathrm{Spr,R}} = m \cdot \frac{x_{\mathrm{CoG}}}{wb} - 2 \cdot m_{\mathrm{Unspr}} \qquad \textbf{(4-2)}$$

The height of the center of gravity of the sprung mass can then be computed as:

$$z_{\mathrm{CoG,Spr}} = \frac{m\, z_{\mathrm{CoG}} - 4\, m_{\mathrm{Unspr}}\, r_{\mathrm{tire}}}{m_{\mathrm{Spr,F}} + m_{\mathrm{Spr,R}}} \qquad \textbf{(4-3)}$$

The calculation of the wheel load at the front left tire is shown exemplarily in equation (4-4). The wheel loads at the other tires can be calculated analogically. It must be noted that wheel loads are contact forces and thus having a negative wheel load is physically impossible.

Therefore, in the simulation model the wheel loads are saturated to be greater than or equal to $0\ N$.

$$
\begin{aligned}
F_{\mathrm{z,FL}} = & \left(\frac{1}{2} m_{\mathrm{Spr,F}} + m_{\mathrm{Unspr}}\right) \cdot g && \text{(static load)} \\
& - m_{\mathrm{Spr,F}} \frac{z_{\mathrm{CoG,Spr}} - z_{\mathrm{RC,F}}}{tr_{\mathrm{F}}} (1 - \Phi_{\mathrm{Roll}})\, a_y && \text{(lateral redistr. acting on springs \& ARBs)} \\
& - m_{\mathrm{Spr,F}} \frac{z_{\mathrm{RC,F}}}{tr_{\mathrm{F}}} a_y && \text{(lateral redistr. acting directly on RC)} \\
& - 2\, m_{\mathrm{Unspr}} \frac{r_{\mathrm{tire}}}{tr_F}\, a_y && \text{(lateral redistr. due to unsprung mass)} \\
& - \frac{1}{2} m \frac{z_{\mathrm{CoG}}}{wb}\, a_x && \text{(longitudinal load redistr.)} \\
& - \frac{1}{2} L\, bal_{\mathrm{A}} && \text{(aerodynamic wheel load)}
\end{aligned}
\tag{4-4}
$$

It may be noted that this calculation creates an algebraic loop, since the wheel loads are necessary to calculate the tire forces and the tire forces are used to compute the longitudinal and lateral accelerations a_x and a_y, which are inputs for the wheel loads calculation. In a more detailed model, the wheel loads would concern a dynamic calculation considering the damper forces and the tire deflection. In this project, this algebraic loop was relaxed by a first-order lag of $0.001\ s$ on the wheel loads fed to the tire model.

NMPC model

For the NMPC model, the wheel loads calculation was simplified. Taking the wheel load at the front-left tire as an example, it is calculated as:

$$
\begin{aligned}
F_{\mathrm{z,FL}} = & \left(\frac{1}{2} m_{\mathrm{Spr,F}} + m_{\mathrm{Unspr}}\right) \cdot g && \text{(static load)} \\
& - m \frac{z_{\mathrm{CoG}}}{tr_{\mathrm{F}}} (1 - \Phi_{\mathrm{Roll}})\, a_y && \text{(lateral load redistr.)} \\
& - \frac{1}{2} m \frac{z_{\mathrm{CoG}}}{wb}\, a_x && \text{(longitudinal load redistr.)} \\
& - \frac{1}{2} L\, bal_{\mathrm{A}} && \text{(aerodynamic wheel load)}
\end{aligned}
\tag{4-5}
$$

This is the same calculation that is used in [47], [48] and [70]. These publications solve the algebraic loop inside the OCP. However, in this project it was found useful to relax the algebraic loop with a first-order lag on the lateral and longitudinal accelerations using the sampling time as time constant. Since a_x and a_y then become states, this allows to use the measurements of these values in the initial value embedding of the NMPC. Note that relaxing this algebraic loop is a method that can be found frequently in literature, for example in [24].

4.1.3 Tire Forces

A tire model is employed to represent the contact forces between the tires and the road. In this thesis, a tire model based on the TMeasy model proposed by Rill [73]. However, in this project the TMeasy model was modified to improve its accuracy and its suitability for Optimal Control applications, using the Pacejka's *similarity method* [46] for calculating the combined forces. Both the simulation model and the NMPC model use the TMeasy calculations, but some simplifications are made for the NMPC model in order to improve its performance. Tire

dynamics, for example the relaxation length, were, however, ignored in this thesis. It may be remarked that van Koutrik also used a TMeasy tire model in his work [48].

Simulation model

The TMeasy model, like many other tire models (e.g. Pacejka's Magic Formula [46]), is a semiempirical model based on observations concerning the movement of the thread particles of a tire. These observations lead to the unitless longitudinal slip s_x and lateral slip s_y definitions:

$$\begin{aligned} s_x &= \frac{r_E N - v_{x,W}}{r_E N} \\ s_y &= \frac{-v_{y,W}}{r_E N} \end{aligned} \tag{4-6}$$

where $v_{x,W}$ and $v_{y,W}$ are the components of the velocity of the wheel center in the corresponding wheel coordinate system, N is the wheel speed of the tire and r_E is its *effective roll radius*, also known as its *dynamic radius*, calculated by:

$$r_E = \lambda_r^{lin}\, r_0 + \left(1 - \lambda_r^{lin}\right) r_L \tag{4-7}$$

In this equation, λ_r^{lin} always assumes a value between zero and one. The dynamic tire radius is thus, according to the TMeasy model, a value between the unloaded radius $r_0 = r_{tire}$, which is the radius of the undeflected tire, and the loaded radius r_L , which is dependent on the wheel load:

$$r_L = r_0 - \frac{F_{z,W}}{c_z^{lin}} \tag{4-8}$$

where c_z^{lin} denotes the vertical stiffness of the tire. Note that the longitudinal and lateral slips are not defined for $N = 0\,{}^{rad}/_{s}$, a state that is given at standstill and at wheel lockage during braking. Therefore, in this thesis, the input N was saturated by a lower value of $1\,{}^{rad}/_{s}$.

The longitudinal and lateral slips are then normalized using $s_{M,x}^{lin}$ and $s_{M,y}^{lin}$ which are the slip values at which the tire reaches the longitudinal and lateral peak forces in pure longitudinal ($s_y = 0$) or pure lateral ($s_x = 0$) conditions respectively. The normalized longitudinal and lateral slips are denoted by σ_x and σ_y respectively and are computed as:

$$\begin{aligned} \sigma_x &= \frac{s_x}{s_{M,x}^{lin}} \\ \sigma_y &= \frac{s_y}{s_{M,y}^{lin}} \end{aligned} \tag{4-9}$$

The combined normalized slip σ is then calculated as the Euclidean norm of the components σ_x and σ_y :

$$\sigma = \sqrt{{\sigma_x}^2 + {\sigma_y}^2} \tag{4-10}$$

The longitudinal force $F_{x,W}$ and the lateral force $F_{y,W}$ are calculated by piecewise defined functions. The function for $F_{x,W}$ is given in equation (4-11), $F_{y,W}$ is defined analogically. Note that these forces are not calculated exactly as proposed by Rill [73], but Pacejka's *similarity method* [46] was employed to calculate the combined slip forces $F_{x,W}$ and $F_{y,W}$ separately. It may be remarked that $F_{x,W}$ and $F_{y,W}$ are given in the wheel coordinate system corresponding to each tire.

$$F_{x,W} = \begin{cases} \dfrac{dF_{0,x}^{quad}\, s_{M,x}^{lin}\, \sigma_x}{\sigma^2 + \left(\dfrac{s_{M,x}^{lin}}{F_{M,x}^{quad}}\, dF_{0,x}^{quad} - 2\right)\sigma + 1} & \text{for } \sigma \le 1 \\ \dfrac{\sigma_x}{\sigma}\left(F_{M,x}^{quad} - \left(F_{M,x}^{quad} - F_{S,x}^{quad}\right)\dfrac{(\sigma-1)^2}{\left(\dfrac{s_{S,x}^{lin}}{s_{M,x}^{lin}} - 1\right)^2}\left(3 - 2\cdot\dfrac{\sigma-1}{\dfrac{s_{S,x}^{lin}}{s_{M,x}^{lin}} - 1}\right)\right) & \text{for } 1 < \sigma \le \dfrac{s_{S,x}^{lin}}{s_{M,x}^{lin}} \\ \dfrac{\sigma_x}{\sigma}\, F_{S,x}^{quad} & \text{for } \sigma > \dfrac{s_{S,x}^{lin}}{s_{M,x}^{lin}} \end{cases} \tag{4-11}$$

Figure 4-2 shows an exemplary curve of $F_{x,W}$ in pure longitudinal slip conditions, i.e. $s_y = 0$. As mentioned before, the tire model has a strong dependency on the wheel load $F_{z,W}$. In the equations above, this dependency is represented by the superscripts $\square^{lin}$ and $\square^{quad}$, which denote a linear or a quadratic interpolation of the parameter depending on the wheel load.

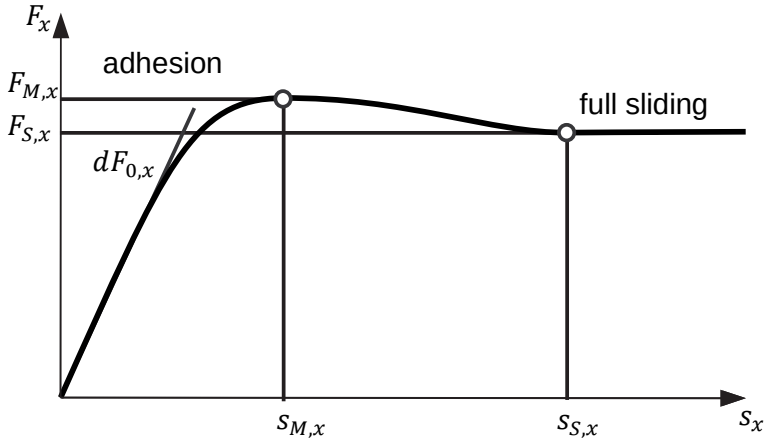

Figure 4-2: Tire forces – TMeasy parameters [74]

Each of these parameters is thus provided for two specified wheel loads $F_z^{(1)}$ and $F_z^{(2)}$ to perform the interpolation, where $F_z^{(2)} > F_z^{(1)}$. The linear interpolation is trivial. The quadratic interpolation is defined so that the parameter results in zero for zero wheel load [73]. For example, the maximum longitudinal force $F_{M,x}^{quad}$ is calculated by:

$$F_{M,x}^{quad} = F_{M,x}^{(1)}\frac{F_z}{F_z^{(1)}} + \frac{F_{M,x}^{(2)}\, F_z^{(1)} - F_{M,x}^{(1)}\, F_z^{(2)}}{F_z^{(1)}\, F_z^{(2)}} \cdot \frac{F_z{}^2 - F_z\, F_z^{(1)}}{F_z^{(2)} - F_z^{(1)}} \tag{4-12}$$

The tire parameters are fitted to flat-trac testbench data. During the fitting process, several conditions need to be taken into account, for example that:

- $F_z^{(2)}$ is greater than the maximum wheel load expected in the application of the tire model
- the linearly interpolated parameters are positive for $F_{z,W} = 0\ N$
- the curvature of the parabola of the quadratically interpolated parameters is negative, so that the interpolation has a degressive trend
- the focal point of the parabolas of the quadratically interpolated parameters is not before $F_z^{(2)}$, so that the function is always increasing

Furthermore, a factor $\lambda_\mu = 0.5$ is used to scale the tire model parameters to adapt the flat-trac testbench values to the friction achieved on an asphalt track. This scaling is performed as proposed by Rill [74]:

$$\begin{aligned} s_{M,x}^{lin} &\leftarrow \lambda_\mu\, s_{M,x}^{lin} & F_{M,x}^{quad} &\leftarrow \lambda_\mu\, F_{M,x}^{quad} & s_{S,x}^{lin} &\leftarrow \lambda_\mu\, s_{S,x}^{lin} & F_{S,x}^{quad} &\leftarrow \lambda_\mu\, F_{S,x}^{quad} \\ s_{M,y}^{lin} &\leftarrow \lambda_\mu\, s_{M,y}^{lin} & F_{M,y}^{quad} &\leftarrow \lambda_\mu\, F_{M,y}^{quad} & s_{S,y}^{lin} &\leftarrow \lambda_\mu\, s_{S,y}^{lin} & F_{S,y}^{quad} &\leftarrow \lambda_\mu\, F_{S,y}^{quad} \end{aligned} \tag{4-13}$$

NMPC model

The NMPC model uses the same tire forces calculation as the simulation model. However, the longitudinal slip s_x is used as input, so that it does not have to be calculated. The reason for this is that the wheel dynamics are neglected in the NMPC model, see next subsection. Also, the tire radius is assumed constant and the effective roll radius is set equal to the undeflected tire radius, i.e. $r_E = r_{\text{tire}}$.

Furthermore, path constraints are imposed on the NMPC so that the combined normalized slip σ of every tire is always less or equal to 0.95 . This helps to ensure that the tires are not operated above their traction limit, ensuring a stable performance of the system. Therefore, only the first part of the piecewise defined functions for $F_{x,W}$ and $F_{y,W}$ needs to be implemented, see equation (4-11). Moreover, the friction scaling parameter was reduced to $\lambda_\mu = 0.4$ for the prediction, so that the tire limit is not overestimated.

In Simscape, the $v_{x,W}$ and $v_{y,W}$ velocities of the tires can be directly obtained by a *Transform Sensor*. However, in the NMPC model, these values need to be calculated using the model's states and controls. For instance, the velocity components for the front-left wheel center denoted in the corresponding wheel coordinate system are computed by:

$$\begin{bmatrix} v_{x,FL} \\ v_{y,FL} \end{bmatrix} = \begin{bmatrix} \cos\delta_F & \sin\delta_F \\ -\sin\delta_F & \cos\delta_F \end{bmatrix} \begin{bmatrix} v_{x,V} - \dfrac{tr_F}{2}\,\dot{\psi} \\ v_{y,V} + x_{CoG}\,\dot{\psi} \end{bmatrix} \tag{4-14}$$

where $v_{x,V} = v\cos\beta$ and $v_{y,V} = v\sin\beta$ are the longitudinal and lateral velocity components of the center of gravity of the vehicle represented in the vehicle coordinate system.

4.1.4 Wheel Dynamics, Traction Control and Powertrain

Simulation model

In Simscape, the wheels are modelled as rotational inertia blocks connected to the overall mass by revolute joints. On each rotational inertia $I_W = 0.12\ kgm^2$ acts a total torque T_{Total} that is composed of three parts: the torque applied by each motor and gearbox T_{motor}, the torque resulting from the tire forces T_{tire} and the torque produced by the rolling resistance of the tire and bearings T_{roll} :

$$T_{\mathrm{Total}} = T_{\mathrm{motor}} + T_{\mathrm{tire}} + T_{\mathrm{roll}} \tag{4-15}$$

For this equation, the torques T_{tire} and T_{roll} are given by [74]

$$\begin{aligned} T_{\mathrm{tire}} &= -r_L \cdot F_x \\ T_{\mathrm{roll}} &= -F_z\, r_{\mathrm{tire}}\, (k_{R0} + k_{R1}\, v_x) \end{aligned} \tag{4-16}$$

where the parameters $k_{R0} = 0.053$ and $k_{R1} = 0.0013\ (^{m}/_{s})^{-1}$ are obtained by coasting tests of the actual vehicle.

The motor torques, however, are set by the inverters, which use a wheel speed controller to try to keep the tires below their traction limit, i.e. $\sigma \leq 1$. This is known as *traction control*. In the simulation model, the motor torque, which is the output of the traction control, is calculated by

$$T_{\mathrm{motor}} = T_{FF} + k_P(N_{cmd} - N) + buf \tag{4-17}$$

where buf is a buffer for the integral controller, which is set by

$$buf \overset{\mathrm{awu}}{\leftarrow} buf + k_I(N_{cmd} - N) \tag{4-18}$$

The torque T_{FF} is a feed-forward estimate for the traction control, N_{cmd} is the wheel speed command for the controller and N represents the actual wheel speed of the tire. The parameters of the controller are set to $k_P = 1\ Nm\left(^{rad}/_{s}\right)^{-1}$ and $k_I = 0.1\ Nm\left(^{rad}/_{s}\right)^{-1}$. An anti-windup procedure (awu) is used to limit the buffer of the integral controller. Furthermore, the output of the controller is saturated to the maximum and minimum torque that the motors and gearboxes can exert on each wheel, namely $\pm 400\ Nm$. As one can see, no braking force is modelled, as it is assumed that only the motors are used to decelerate the car by regenerative braking.

It may be remarked that the exact same controller is running on the actual *eb016*, see Appendix B. Both the feed-forward torque T_{FF} and the target wheel speed N_{cmd} need to be set by the NMPC algorithm.

NMPC model

The wheel spin dynamics are usually much faster than the vehicle longitudinal and lateral dynamics, with natural frequencies up to an order of magnitude larger at about $50\ Hz$ [75]. To represent these dynamics the NPMC model would need to have a sampling time of less than $0.02\ s$ or use an implicit solver. Therefore, the wheel dynamics are frequently neglected for Optimal Control applications, see [48]. Moreover, the traction control on the actual car runs at

a much higher frequency ($\sim 12\ kHz$), so that it can be assumed that the setpoint for the wheel speeds can be reached fast enough.

Therefore, the longitudinal slip s_x of each tire is used as control in the NMPC model. The feed-forward torque T_{FF} and the wheel speed setpoint N_{cmd} are then calculated as model outputs by:

$$\begin{aligned} T_{FF} &= F_x \cdot r_{\text{tire}} \\ N_{cmd} &= \frac{v_{x,W}}{r_{\text{tire}}\,(1 - s_x)} \end{aligned} \tag{4-19}$$

These *outputs* of the NMPC model are used as *controls* in the simulation model. Therefore, they need to be updated after the Feedback Phase of the RTI algorithm. In Falcon NMPC, this is done by setting the $\text{recalcModelOutputs}$ property to true, see Appendix A.

Each of the motors (together with the gearboxes) can set up to $\pm 400\ Nm$ of torque and has a peak mechanical power of $\pm 25\ kW$. These constraints are taken into account in the NMPC optimization problem, where the mechanical power is calculated as $P = T_{FF}\, N_{cmd}$. Furthermore, the maximum allowed electrical power output is $80\ kW$ [16]. Considering the efficiency of the powertrain (~ 0.8), the sum of the mechanical power of all motors is therefore limited to $P_{\text{Total}} = 64\ kW$ in the NMPC model.

4.1.5 Longitudinal and Lateral Dynamics

Simulation model

The longitudinal and lateral dynamics of the racecar model are represented by a planar joint in Simscape. Therefore, the vehicle has two translational and one rotational degree of freedom. The overall mass m of the vehicle, which is visualized by the vehicle's chassis, is attached to this joint. On this mass act the aerodynamic drag and the tire forces at their respective positions. Furthermore, the front tires are rotated by the steering angle δ_F.

The NMPC does not provide the steering angle δ_F directly, but its derivative $\dot{\delta}_F$. Therefore, in the simulation model, it must be integrated and passed to the NMPC as a state. This is done for two reasons: first, this provides a clean way to constrain the time derivative of the steering angle to a physically reasonable limit, second, this allows the steering angle to have a more continuous trend instead of being piecewise constant.

To be able to provide the NMPC with the correct state values, the slip angle needs to be calculated in the simulation model. This is done by:

$$\beta = \operatorname{atan} \boldsymbol{v}_V \tag{4-20}$$

where $\boldsymbol{v}_V$ denotes the velocity vector of the overall mass object represented in the vehicle coordinate system.

NMPC model

The equations of motion for the longitudinal and lateral dynamics of the vehicle are given by equation (4-21), compare to equations of motion in [47, 48, 70]. The absolute value of the velocity of the CoG of the overall mass is represented by v, the vehicle's slip angle by β and its yaw rate by $\dot{\psi}$.

$$\begin{aligned}\dot{v} &= \frac{F_x \cos\beta + F_y \sin\beta}{m} \\ \dot{\beta} &= \frac{F_y \cos\beta - F_x \sin\beta}{m\, v} - \dot{\psi} \\ \ddot{\psi} &= \frac{M_z}{I_{zz}}\end{aligned} \tag{4-21}$$

where F_x and F_y are the total longitudinal and lateral forces and M_z is the total yaw moment acting on the vehicle. Hence, $I_{zz} = 150\ kgm^2$ denotes the total yaw inertia of the vehicle.

The forces F_x and F_y and the torque M_z are calculated by:

$$\begin{bmatrix}F_x \\ F_y\end{bmatrix} = \begin{bmatrix}D\cos\beta \\ D\sin\beta\end{bmatrix} + \begin{bmatrix}\cos\delta_F & -\sin\delta_F \\ \sin\delta_F & \cos\delta_F\end{bmatrix}\begin{bmatrix}F_{x,FL} + F_{x,FR} \\ F_{y,FL} + F_{y,FR}\end{bmatrix} + \begin{bmatrix}F_{x,RL} + F_{x,RR} \\ F_{y,RL} + F_{y,RR}\end{bmatrix}$$

$$\begin{aligned}M_z = &\begin{bmatrix}-\frac{tr_F}{2} & x_{CoG}\end{bmatrix}\begin{bmatrix}\cos\delta_F & -\sin\delta_F \\ \sin\delta_F & \cos\delta_F\end{bmatrix}\begin{bmatrix}F_{x,FL} \\ F_{y,FL}\end{bmatrix} + \begin{bmatrix}\frac{tr_F}{2} & x_{CoG}\end{bmatrix}\begin{bmatrix}\cos\delta_F & -\sin\delta_F \\ \sin\delta_F & \cos\delta_F\end{bmatrix}\begin{bmatrix}F_{x,FR} \\ F_{y,FR}\end{bmatrix} \\ &+ \begin{bmatrix}-\frac{tr_R}{2} & wb - x_{CoG}\end{bmatrix}\begin{bmatrix}F_{x,RL} \\ F_{y,RL}\end{bmatrix} + \begin{bmatrix}\frac{tr_R}{2} & wb - x_{CoG}\end{bmatrix}\begin{bmatrix}F_{x,RR} \\ F_{y,RR}\end{bmatrix}\end{aligned} \tag{4-22}$$

As described before, the steering angle δ_F is a model state. Its time derivative $\dot{\delta}_F$ is thus a control value, which must be integrated as the differential equations of the other states. Furthermore, the longitudinal and lateral accelerations can be calculated by

$$\begin{aligned}a_x &= \frac{F_x}{m} \\ a_y &= \frac{F_y}{m}\end{aligned} \tag{4-23}$$

These are used to calculate the states of the accelerations used for the wheel loads calculation. As mentioned before, a first-order lag of the accelerations is employed for this purpose, see subsection 4.1.2.

4.1.6 Track Model

As for the point mass model presented in section 3.4 of this thesis, the position of the racecar is represented in track coordinates. This track coordinates are the path coordinate s, the normal coordinate n and the relative yaw angle ζ. As in section 3.4, the track is defined by the course angle $\theta(s)$ of its midline, which is built by cubic splines, and the constant track width $b = 4\ m$. The course angle $\theta(s)$ is measured against the inertial x_E-axis and its derivative with respect to the path coordinate s gives the curvature $C = \frac{\partial\theta(s)}{\partial s}$ of the track.

The figure below shows the definitions of the angles and track coordinates used in this thesis. It may be remarked that the track used in this project was imported from logged data of the actual vehicle. For more information on this, see Appendix C.

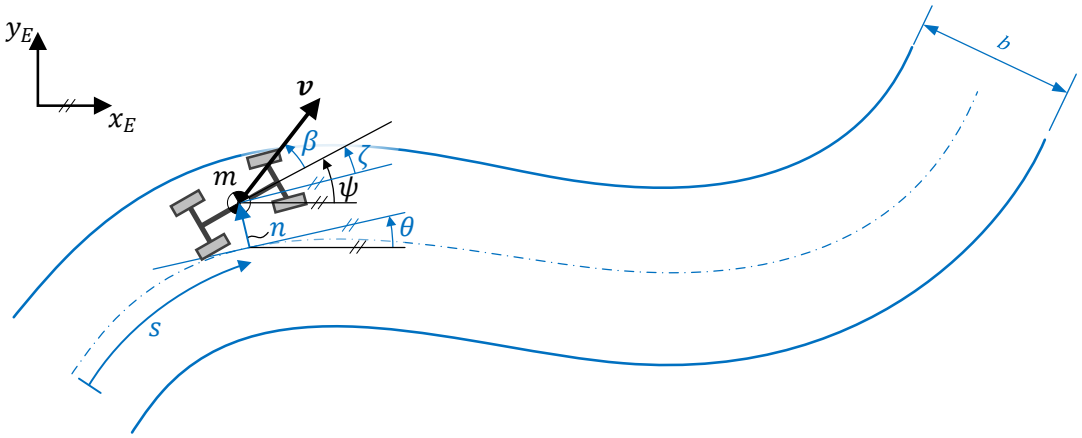

Figure 4-3: Racecar model – Definition of track values

Simulation model

The track coordinates s and n as well as the relative yaw angle ζ need to be computed in the simulation model to be able to pass them to the NMPC model. These are calculated as for the point mass model in section 3.4. The path coordinate s and the normal coordinate n are obtained by integration of their time derivatives:

$$\dot{s} = \frac{v\cos(\operatorname{atan}\boldsymbol{v}_E - \theta)}{1 - n\,C}$$
$$\dot{n} = v\sin(\operatorname{atan}\boldsymbol{v} - \theta) \qquad \textbf{(4-24)}$$

where v is the absolute value of the velocity vector $\boldsymbol{v}$ and $\operatorname{atan}\boldsymbol{v}_E$ is its angle with respect to the inertial x_E-axis. The values of the course angle θ and the curvature C can be obtained by evaluating their splines definition using the value of the path coordinate s.

The relative yaw angle can be obtained simply by:

$$\zeta = \psi - \theta \qquad \textbf{(4-25)}$$

where ψ is the absolute yaw angle of the vehicle given directly by the planar joint of the Simscape model.

NMPC model

The differential equations of the racecar's position in track coordinates and of the relative yaw angle are given by:

$$\dot{s} = \frac{v\cos(\zeta + \beta)}{1 - n\,C}$$
$$\dot{n} = v\sin(\zeta + \beta) \qquad \textbf{(4-26)}$$
$$\dot{\zeta} = \dot{\psi} - C\,\dot{s}$$

These differential equations can be compared to the ones used in [47, 48, 70].

4.1.7 Summary

This subsection offers a summary comparing the subsystems used in the racecar model in Simscape and in the FALCON.m model for the NMPC. This comparison is presented in Table 4-1.

Subsystem	Simulation model	NMPC model
Aerodynamic Forces	Lift and drag forces calculated from aerodynamic equations. Constant lift and drag coefficients and aerodynamic balance are used.	Identical to simulation model.
Wheel Loads	Steady-state calculation based on longitudinal and lateral accelerations, and aerodynamic lift and balance. Takes into account forces acting on roll centers.	Same as in simulation model. However, the roll center heights are neglected.
Tire Forces	Full TMeasy tire model using Pacejka's similarity method for the combined forces calculation. The friction scaling coefficient is set to $\lambda_\mu = 0.5$.	Same as in simulation model, but with constant tire radius r_{tire} . Furthermore, only the first part of the piecewise defined force functions is used, as the NMPC is constrained to work only below the traction limit of the tire. The friction scaling coefficient is lowered to $\lambda_\mu = 0.4$ so that the traction limit is not overestimated.
Wheel Dynamics, Traction Control and Powertrain	Wheel spin dynamics are modelled with revolute joints in Simscape. Traction control is implemented as a PI-controller of the wheel speed plus a feed-forward control. The same controller is implemented in the actual car, see Appendix B.	Wheel dynamics neglected. Therefore, the wheel speeds are obtained using the longitudinal slip s_x of each tire, which are control values of the NMPC model. This wheel speed is used as setpoint for the traction control. The feed-forward control torque is calculated using values of the tire model. This value is constrained in the NMPC to satisfy the peak torque and peak power limits of the motors and that of the overall system.
Longitudinal and Lateral Dynamics	Modelled with a planar joint in Simscape.	Modelled as differential equations of motion for the absolute speed v, the slip angle β and the yaw rate $\dot{\psi}$ of the vehicle
Track Model	The path coordinate s and the normal coordinate n are obtained through integration of differential equations. The relative yaw rate ζ is obtained by subtraction of the track course angle θ from the absolute yaw angle ψ.	The path coordinate s, the normal coordinate n and the relative yaw rate ζ are obtained from differential equations.

Table 4-1: Racecar models – Comparison of simulation model and NMPC model

Finally, the states and controls of the NMPC model are listed in Table 4-2.

	Symbol	Description	Min	Max
states	s	Path coordinate	$0\,m$	$\inf\ m$
	n	Normal coordinate	-1 m	$1\,m$
	ζ	Relative yaw angle	$-\pi/4\ rad$	$\pi/4\ rad$
	v	Speed	$1\ m/s$	$\inf\ m/s$
	β	Slip angle	$-\pi/2\ rad$	$\pi/2\ rad$
	$\dot{\psi}$	Yaw rate	$-2\ rad/s$	$2\ rad/s$
	a_x	Longitudinal acceleration	$-30\ m/s^2$	$30\ m/s^2$
	a_y	Lateral acceleration	$-30\ m/s^2$	$30\ m/s^2$
	δ_F	Steering angle	$-0.5\,rad$	$0.5\,rad$
controls	$\dot{\delta}_F$	Time derivative of steering angle	$-5\ rad/s$	$5\ rad/s$
	$s_{x,FL}$	Longitudinal slip of front-left tire	-0.15	0.15
	$s_{x,FR}$	Longitudinal slip of front-right tire	-0.15	0.15
	$s_{x,RL}$	Longitudinal slip of rear-left tire	-0.15	0.15
	$s_{x,RR}$	Longitudinal slip of rear-right tire	-0.15	0.15

Table 4-2: Racecar NMC model – States and controls

The outputs of the NMPC model are listed in Table 4-3. At this point it is reminded that the following values are used as input for the racecar simulation model:

- the time derivative of the steering angle $\dot{\delta}_F$, which is integrated in the simulation model to get the actual steering angle
- the wheel speed commands N_{cmd} , as setpoint for the PI part of the traction control algorithm
- the feed-forward torques T_{FF} , as feed-forward for the traction controller

All the other outputs are only used as nonlinear constraints for the system.

	Symbol	Description	Min	Max
outputs	σ_{FL}	Normalized combined slip of front-left tire	$-\inf$	0.95
	σ_{FR}	Normalized combined slip of front-right tire	$-\inf$	0.95
	σ_{RL}	Normalized combined slip of rear-left tire	$-\inf$	0.95
	σ_{RR}	Normalized combined slip of rear-right tire	$-\inf$	0.95
	$N_{cmd,FL}$	Wheel speed command for front-left wheel	$-\inf\ {}^{rad}/_{s}$	$\inf\ {}^{rad}/_{s}$
	$N_{cmd,FR}$	Wheel speed command for front-right wheel	$-\inf\ {}^{rad}/_{s}$	$\inf\ {}^{rad}/_{s}$
	$N_{cmd,RL}$	Wheel speed command for rear-left wheel	$-\inf\ {}^{rad}/_{s}$	$\inf\ {}^{rad}/_{s}$
	$N_{cmd,RR}$	Wheel speed command for rear-right wheel	$-\inf\ {}^{rad}/_{s}$	$\inf\ {}^{rad}/_{s}$
	$T_{FF,FL}$	Feed-forward torque of front-left wheel	$-400\ Nm$	$400\ Nm$
	$T_{FF,FR}$	Feed-forward torque of front-right wheel	$-400\ Nm$	$400\ Nm$
	$T_{FF,RL}$	Feed-forward torque of rear-left wheel	$-400\ Nm$	$400\ Nm$
	$T_{FF,RR}$	Feed-forward torque of rear-right wheel	$-400\ Nm$	$400\ Nm$
	P_{FL}	Mechanical power at front-left wheel	$-25\ kW$	$25kW$
	P_{FR}	Mechanical power at front-right wheel	$-25\ kW$	$25kW$
	P_{RL}	Mechanical power at rear-left wheel	$-25\ kW$	$25kW$
	P_{RR}	Mechanical power at rear-right wheel	$-25\ kW$	$25kW$
	P_{Total}	Total mechanical power	$-\inf\ kW$	$64\ kW$

Table 4-3: Racecar NMC model – Outputs

4.2 Results

As mentioned in the introduction, the time-optimal maneuvering of the racecar on the track is an economic NMPC application. In this case, the objective function for the Optimal Control Problem was formulated as for the point mass model in the previous chapter:

$$\min_{\boldsymbol{x},\boldsymbol{u}} -s_N \tag{4-27}$$

In this cost function the distance that the racecar travels in its prediction horizon is maximized. Thus, s_N represents the path coordinate at the end of the prediction horizon. As for the point mass model, the reference values for Falcon NMPC can be set to zero, as they do not take part in the problem.

In the NMPC model, the differential equation for the slip angle $\dot{\beta}$ contains a division by the speed v. Therefore, like for the point mass model, the NMPC algorithm cannot be started at

standstill. The speed v in the NMPC model is thus limited to a minimum of $1\,{}^{m}/_{s}$, see Table 4-2. The NMPC model is therefore started when the speed of the vehicle is above $2\,{}^{m}/_{s}$ and is turned off when the speed drops below $1\,{}^{m}/_{s}$. This hysteresis was added to avoid chattering. When the NMPC controller is switched off, the outputs to the racecar are:

$$\begin{aligned} \dot{\delta}_F &= 0\,{}^{rad}/_{s} \\ N_{cmd,FL} = N_{cmd,FR} &= N_{cmd,RL} = N_{cmd,RR} = 5\,{}^{rad}/_{s} \\ T_{FF,FL} = T_{FF,FR} &= 100\ Nm \\ T_{FF,RL} = T_{FF,RR} &= 150\ Nm \end{aligned} \tag{4-28}$$

This ensures that the vehicle accelerates forward to reach the necessary $2\,{}^{m}/_{s}$ to activate the NMPC.

The horizon length for the NMPC was set to $T_H = 2.5\ s$ and the sampling time to $T_s = 0.05\ s$, so that the Optimal Control Problem is composed of 50 stages. The approach (3-25), which sets the path coordinate s_N^{guess} after the shifting procedure of the RTI algorithm, was also used here. Furthermore, the optimization problems were solved using a constant Hessian as for the point mass model, setting the $\boldsymbol{Q}$, $\boldsymbol{R}$ and $\boldsymbol{P}$ matrices to:

$$\begin{aligned} \boldsymbol{Q} = \boldsymbol{P} &= 10^{-16} \cdot I^{9\text{x}9} \\ \boldsymbol{R} &= 10^{-2} \cdot I^{5\text{x}5} \end{aligned} \tag{4-29}$$

where I^{nxn} represents an identity matrix of n rows and n columns. As mentioned in section 3.4.2, using a constant Hessian for the optimization problem is equivalent to including a tracking cost function of the form (2-65), where the references for the states and for the controls are their initial guesses, i.e. their values after the shifting procedure. Therefore, the objective function becomes:

$$\begin{aligned} \min_{\boldsymbol{x},\boldsymbol{u}} -s_N + \sum_{i=0}^{N-1} \frac{1}{2} \Big(& \left(\boldsymbol{x}_i - \boldsymbol{x}_i^{\text{guess}}\right)^{\text{T}} \boldsymbol{Q} \left(\boldsymbol{x}_i - \boldsymbol{x}_i^{\text{guess}}\right) + \left(\boldsymbol{u}_i - \boldsymbol{u}_i^{\text{guess}}\right)^{\text{T}} \boldsymbol{R} \left(\boldsymbol{u}_i - \boldsymbol{u}_i^{\text{guess}}\right) \Big) \\ & + \left(\boldsymbol{x}_N - \boldsymbol{x}_N^{\text{guess}}\right)^{\text{T}} \boldsymbol{P} \left(\boldsymbol{x}_N - \boldsymbol{x}_N^{\text{guess}}\right) \end{aligned} \tag{4-30}$$

The $\boldsymbol{Q}$, $\boldsymbol{R}$ and $\boldsymbol{P}$ matrices can thus be used to tune the NMPC algorithm, penalizing the difference of the state and control values from the previous solution. During the tests with this model, it was found that setting the costs corresponding to the controls, i.e. the values in $\boldsymbol{R}$, to a relatively large value of 10^{-2} significantly improves the robustness of the algorithm. However, it must be noted that these values modify the cost function (4-27) considerably.

For the results presented next, a 4th-order explicit Runge-Kutta (ERK4) discretization was used. The track considered for these tests was imported from logged data of the actual vehicle, see Appendix C. Therefore, the logged data of the lap driven by a human pilot is compared to the results of the simulation using the NMPC algorithm to control the car. Afterwards, these results are compared to the theoretical optimum calculated with Optimal Control using the same model as in the NMPC algorithm.

NMPC vs. fastest Driven Lap

Figure 4-4 compares the performance of the NMPC algorithm in simulation against the real-life vehicle piloted by a human driver. The track corresponds to the *Endurance* event at the Formula Student Germany and the logged data corresponds to the fastest lap of TUfast's electric vehicle, the eb016, in the year 2016.

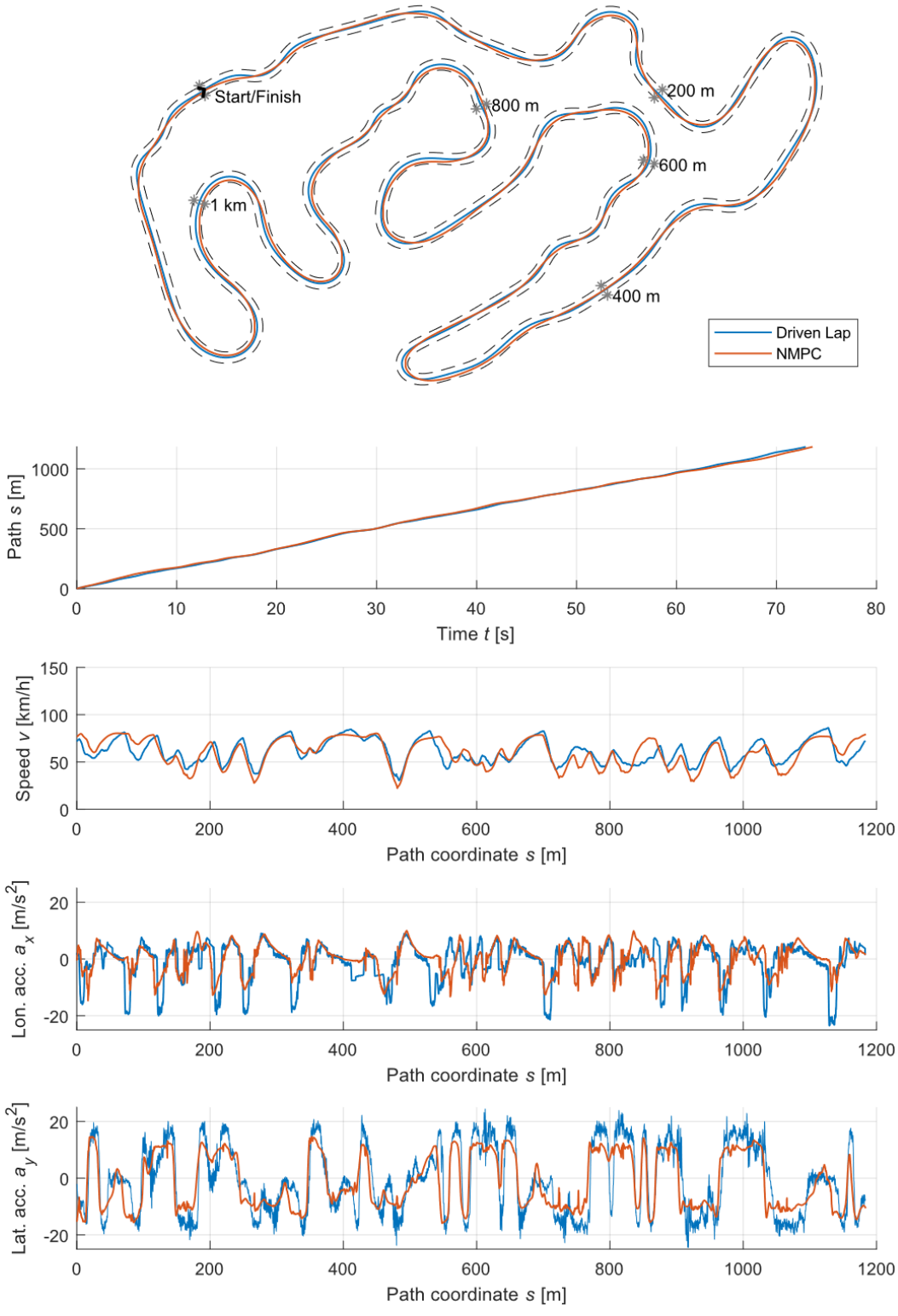

Figure 4-4: Racecar – NMPC vs. fastest Driven Lap

It must be remarked that the lap with the real-life vehicle described above was driven with the maximum electric power output set to $50\ kW$. Thus, for the comparison presented here, the value of the maximum mechanical power for the NMPC model was set to $P_{\mathrm{Total}} = 40\ kW$, considering the power losses of the powertrain (efficiency $\sim 80\%$).

As can be seen in the figure above, the laptime achieved with the NMPC algorithm is almost identical to the one achieved by the human driver, differing by less than 0.7 seconds in the $\sim 70\ s$ lap. However, the speed profiles show some discrepancies. These discrepancies are mainly the result of the import method of the track, which was generated using logged data, see Appendix C. This means that only the averaged driven line can be imported, which for this project was assumed to be the midline of the track. This is of course not the case and using this method reduces the curvature of the track in certain spots, for example in the chicane shortly after the starting line. This allows the simulation model to drive these curves at a higher velocity. Moreover, this also allows the NMPC algorithm to "cut the curves". Therefore, although the driven line is $1184\ m$ (as well as the midline and thus also the path coordinate s), the simulation model controlled by the NMPC travels only a total distance of $1152\ m$.

In the longitudinal acceleration plot, one can observe that the forward acceleration phases are matched quite well by the controller. However, the braking phases do not reach the same deceleration. This is due to the fact that the mechanical brakes were not modelled in this project, thus limiting the braking power to that that the electric motors can produce by regenerative braking.

The lateral acceleration achieved by the NMPC controller is qualitatively very close to that in the logged data. Nevertheless, its absolute value is about 80% smaller for most of the lap. The reason for this is that, in the NMPC model, the friction scaling factor is set to $\lambda_\mu = 0.4$, which is 80% of the value set for the simulation model. This is done to ensure that the traction limit of the tires is never overestimated. This would be the case in the first chicane, where both plots have almost identical values.

In conclusion, the NMPC algorithm only achieves the same laptime as the human driver, because the construction of the track creates inconsistencies in its curvature. If the exact same track would be used, the NMPC controller would be slightly slower. However, the speed and acceleration profiles achieved by the NMPC algorithm are already quite close to the ones achieved by the human driver.

NMPC vs. Optimal Control

Here, the performance of the NMPC algorithm to control the simulation model is compared to the actual optimum calculated with Optimal Control using the NMPC vehicle model. The results of this comparison are shown in Figure 4-5. The track is the as described before. For these results, the maximum mechanical power was elevated to the maximum allowed of $P_{\mathrm{Total}} = 64\ kW$.

However, the model used for the full Optimal Control Problem was slightly modified to incorporate more effects. First, the algebraic loop created by the wheel loads redistribution calculation was solved directly in the optimization problem. This was done using the accelerations as slack variables. Second, the calculation of the effective radii of the tires were included in the model.

As can be seen in the plots, the results show more similarity than compared to the real-life driven lap. This is because for both models, the track is identical, so that both can “cut the curves” similarly.

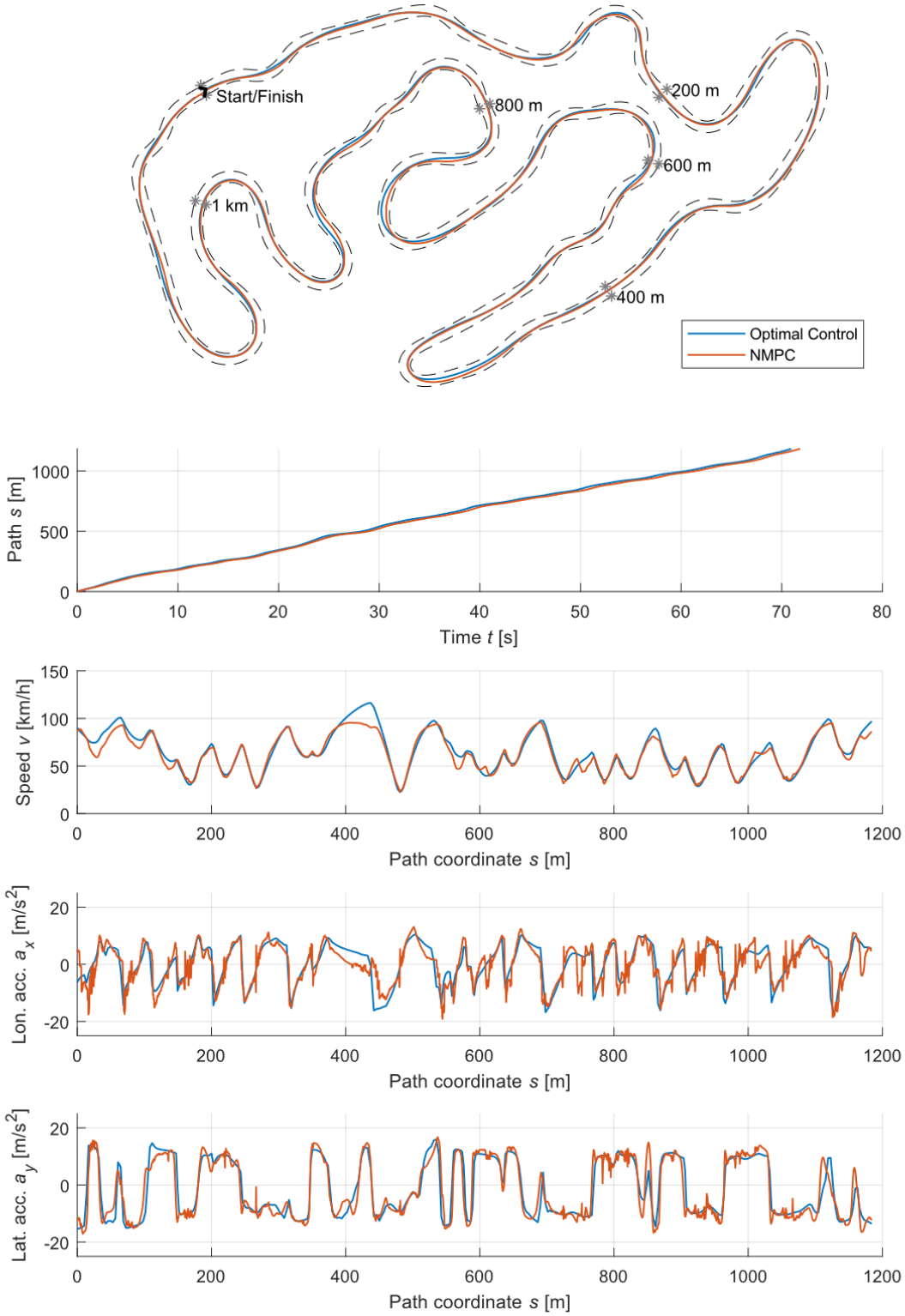

Figure 4-5: Racecar – NMPC vs. Optimal Control

However, the differences between the results are encountered because of three reasons:

- The NMPC algorithm uses a prediction horizon of $T_H = 2.5\,s$, while in the Optimal Control Problem the entire lap is optimized at once
- At every sampling time, the NMPC algorithm only solves an approximation of the nonlinear problem. This means that the optimum of the full nonlinear problem might not be reached for the control values $\boldsymbol{\mu}$ outputted by the NMPC algorithm.
- There are several discrepancies between the simulation model and the NMPC model, so that the prediction of the NMPC algorithm does not match the actual behavior of the simulation model exactly. This means that the controller needs to make corrections constantly.

This last point is the reason for the inconsistency in the speed profiles at the path coordinate $s = 400\,m$. In the simulation model, the effective roll radius r_E of the tires is taken into account, which is neglected in the NMPC model. Since the commanded wheel speed is calculated as

$$N_{cmd} = \frac{v_{x,W}}{r_{\text{tire}}\,(1 - s_x)} \tag{4-31}$$

(see section 4.1.4) and $r_{\text{tire}} > r_E$, the commanded wheel speed results too small. Therefore, the longitudinal slip in the simulation model is also smaller than predicted preventing the vehicle from accelerating. Beside this, the performance of the NMPC algorithm is very satisfactory.

5 Conclusions and perspective

5.1 Summary

In this thesis, a Nonlinear Model Predictive Control (NMPC) algorithm was implemented. Specifically, the Real-Time Iteration scheme [33] was programmed in object-oriented MATLAB and interfaced with Simulink. This was done based on FALCON.m, which is an Optimal Control toolbox for MATLAB that was developed at the Institute for Flight System Dynamics of the Technical University of Munich.

This project can therefore be regarded as an extension of FALCON.m for rapid-prototyping of NMPC controllers in simulation. The theoretical background of the NMPC algorithm implemented in this thesis is presented in chapter 2. During the development of the project, the algorithm was tested on several dynamic systems with increasing levels of complexity, which are presented in chapter 3. The results of the simulations performed with these systems finally lead to the configuration of the NMPC algorithm that was used for the control of the autonomous racecar model in chapter 4.

In chapter 4, a highly complex nonlinear racecar model was simulated and controlled by an NMPC algorithm. The controller uses a slightly simplified model of the system. The performance of the controller in simulation was satisfactory. The results of the simulation were compared to logged of the real-life vehicle on the same track. The results were also compared to the theoretical optimum calculated using Optimal Control.

5.2 Future work

In the following paragraphs, some recommendations for future work are listed:

Graphical User Interface

The NMPC implementation that resulted from this work is already quite versatile and user friendly, as this was one of the objectives of the thesis. However, a graphical user interface would further improve the user friendliness of the code. Since the NMPC is programmed in an object-oriented way, the implementation of a GUI should be straightforward. This could be done simply as a Simulink *Mask*.

Interfaces to other solvers

In this thesis, interfaces to the NLP solver IPOPT and the QP solver qpDUNES were implemented and tested. However, there exist other solvers that have very interesting properties for model predictive control, for example qpOASES. Furthermore, a condensing strategy [60] could be implemented to compare the performance of a condensed problem against the solving the full QP problem. This could be combined with a block-condensing strategy as proposed in [59].

Investigate relevance of Hessian Sparsity

For the experiments with the BFGS Hessian approximation and the regularization methods of the exact Hessian, it was assumed that the Hessian sparsity structure was the same as for the exact Hessian (without regularization). It should be investigated if the performance of these

algorithms gets affected if the Hessian sparsity structure is modified inside these algorithms. Furthermore, a regularization method that conserves the Hessian sparsity is proposed in [54]. This algorithm could also be implemented.

Other NMPC algorithms

Other NMPC algorithms could be made part of Falcon NMPC. For example, the *Advanced Step Controller* by Zavala and Biegler [44], which was briefly presented in section 1.2.2, is in interesting candidate. A similar method to this is used in *embotech*'s product [18]. The implementation of this algorithm should be straightforward, as it can be compared to solving the full nonlinear problem, as is done with the method `set_solveFullProblem` in Falcon NMPC, see Appendix A and example in section 3.2, and performing linear sensitivity analysis afterwards, which is already part of FALCON.m.

Code generation

The NMPC implementation in this project, Falcon NMPC, is currently only usable for rapid-prototyping in simulation, specifically in Simulink. FALCON.m is already able to generate C code for the models and constraints. Therefore, the generation of C code for the rest of the functions could be considered as future work. This C code could then be compiled to run on an embedded target.

On autonomous racecar control

For the control of the autonomous racecar, several methods could be investigated further. First, a Multilevel NMPC implementation is suggested, in which the trajectory planning and the trajectory tracking functions are separated. The trajectory planning could then be performed with a lower resolution, with spatial discretization as in [26, 31, 48, 70], and for a longer prediction horizon (or for the full track). For the trajectory planning, using other NMPC algorithms like the *Advanced Step Controller* by Zavala and Biegler [44] could be beneficial. For the trajectory tracking, an RTI scheme with constant Gauss-Newton Hessian could be used.

6 References

[1] H. Bardt, “Autonomes Fahren: Eine Herausforderung für die deutsche Autoindustrie,” (German), *IW-Trends Vierteljahresschrift zur empirischen Wirtschaftsforschung*, vol. 43, no. 2, pp. 39–55, 2016.

[2] FIA, *Formula E & Kinetik announce driverless support series*. United Kingdom, 2015.

[3] S. Hemer and S. Seewaldt, “Startklar für die Formula Student Driverless,” *Sonderprojekte ATZ/MTZ*, vol. 21, no. S5, pp. 12–17, 2016.

[4] F. Meier, “Formula Student Driverless — Autonom am Start,” *Sonderprojekte ATZ/MTZ*, vol. 22, no. S2, pp. 14–15, 2017.

[5] H. Cheng, *Autonomous intelligent vehicles: Theory, algorithms, and implementation*. London, New York: Springer, 2011.

[6] S. Pendleton *et al.,* “Perception, Planning, Control, and Coordination for Autonomous Vehicles,” *Machines*, vol. 5, no. 1, p. 6, 2017.

[7] M. Römer, S. Gaenzle, and C. Weiss, “How Automakers Can Survive the Self-Driving Era,” A.T. Kearney. [Online] Available: https://www.atkearney.com/automotive/article?/a/how-automakers-can-survive-the-self-driving-era. Accessed on: Jul. 11 2018.

[8] S. A. Bagloee, M. Tavana, M. Asadi, and T. Oliver, “Autonomous vehicles: Challenges, opportunities, and future implications for transportation policies,” *J. Mod. Transport.*, vol. 24, no. 4, pp. 284–303, 2016.

[9] S. Hörl, F. Ciari, and K. W. Axhausen, “Recent perspectives on the impact of autonomous vehicles,” 2016.

[10] Audi AG, *Audi mit wichtigen Weichenstellungen in herausforderndem Geschäftsjahr 2016*. Ingolstadt,Germany, 2018.

[11] Volvo Car Corporation, *Volvo Cars and Autoliv announce the launch of Zenuity*, 2017.

[12] Continental AG, *Continental Closes Acquisition of Elektrobit Automotive*. Hanover/Erlangen, 2015.

[13] Intel Corporation, *BMW Group, Intel and Mobileye Team Up to Bring Fully Autonomous Driving to Streets by 2021*. Munich, 2016.

[14] NVIDIA Corporation, *NVIDIA Announces World’s First Functionally Safe AI Self-Driving Platform*. CES, 2018.

[15] Roborace, *Media Assets: Roborace unveils tech capabilities on the production version of the car.* [Online] Available: https://roborace.com/media/. Accessed on: Jul. 11 2018.

[16] Formula Student Germany, “Formula Student Rules 2018,” 2018. [Online] Available: https://www.formulastudent.de/fileadmin/user_upload/all/2018/rules/FS-Rules_2018_V1.1.pdf. Accessed on: Jul. 14 2018.

[17] embotech AG, *ETH Team wins Formula Student Driverless competition using embotech Software*. 13.08.2017.

[18] A. Domahidi and J. Jerez, *FORCES Professional*.

[19] AMZFormulaStudent, *Autonomous Racing: AMZ Driverless with flüela*.

[20] L. T. Biegler, “Efficient Solution of Dynamic Optimization and NMPC Problems,” in *Nonlinear Model Predictive Control*, F. Allgöwer and A. Zheng, Eds., Basel: Birkhäuser Basel, 2000, pp. 219–243.

[21] *Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles*, 2018.

[22] SAE International, *Automated Driving: Levels of Driving Automation are defined in new SAE International Standard J3016*, 2014.

[23] P. Falcone, H. Eric Tseng, F. Borrelli, J. Asgari, and D. Hrovat, “MPC-based yaw and lateral stabilisation via active front steering and braking,” *Vehicle System Dynamics*, vol. 46, no. sup1, pp. 611–628, 2008.
[24] J. V. Frasch *et al.,* “An Auto-generated Nonlinear MPC Algorithm for Real-Time Obstacle Avoidance of Ground Vehicles,” in *Proceedings of the European Control Conference (ECC)*, 2013, pp. 4136–4141.
[25] Y. Gao, T. Lin, F. Borrelli, E. Tseng, and D. Hrovat, “Predictive Control of Autonomous Ground Vehicles With Obstacle Avoidance on Slippery Roads,” in *Proceedings of the ASME Dynamic Systems and Control Conference--2010: Presented at 2010 ASME Dynamic Systems and Control Conference, September 12-15, 2010 Cambridge, Mass., USA*, Cambridge, Massachusetts, USA, 2010, pp. 265–272.
[26] M. Zanon, J. V. Frasch, M. Vukov, S. Sager, and M. Diehl, “Model Predictive Control of Autonomous Vehicles,” in vol. 455, *Optimization and Optimal Control in Automotive Systems*, H. Waschl, I. Kolmanovsky, M. Steinbuch, and L. del Re, Eds., Cham: Springer International Publishing, 2014, pp. 41–57.
[27] A. Katriniok and D. Abel, “LTV-MPC approach for lateral vehicle guidance by front steering at the limits of vehicle dynamics,” in *2011 50th IEEE Conference on Decision and Control and European Control Conference: (CDC-ECC) ; 12 - 15 Dec. 2011, Orlando, FL, USA*, Orlando, FL, USA, 2011, pp. 6828–6833.
[28] L. Grüne and J. Pannek, *Nonlinear Model Predictive Control*. Cham: Springer International Publishing, 2017.
[29] R. Verschueren, “Design and Implementation of a Time-Optimal Controller for Model Race Cars,” KU Leuven, 2014.
[30] R. Verschueren, S. D. Bruyne, M. Zanon, J. V. Frasch, and M. Diehl, “Towards Time-Optimal Race Car Driving using Nonlinear MPC in Real-Time,” in *Proceedings of the IEEE Conference on Decision and Control (CDC)*, 2014, pp. 2505–2510.
[31] R. Verschueren, M. Zanon, R. Quirynen, and M. Diehl, “Time-optimal Race Car Driving using an Online Exact Hessian based Nonlinear MPC Algorithm,” in *Proceedings of the European Control Conference (ECC)*, 2016.
[32] P. Falcone, F. Borrelli, J. Asgari, H. E. Tseng, and D. Hrovat, “Predictive Active Steering Control for Autonomous Vehicle Systems,” *IEEE Trans. Contr. Syst. Technol.*, vol. 15, no. 3, pp. 566–580, 2007.
[33] S. Gros, M. Zanon, R. Quirynen, A. Bemporad, and M. Diehl, “From linear to nonlinear MPC: Bridging the gap via the real-time iteration,” *International Journal of Control*, vol. 20, no. 1, pp. 1–19, 2017.
[34] E. F. Camacho and C. Bordons, “Nonlinear Model Predictive Control: An Introductory Review,” in *Lecture notes in control and information sciences*, vol. 358, *Assessment and future directions of nonlinear model predictive control*, R. Findeisen, F. Allgower, and L. T. Biegler, Eds., Berlin: Springer, 2007, pp. 1–16.
[35] R. Findeisen and F. Allgöwer, *An Introduction to Nonlinear Model Predictive Control*, 2002.
[36] M. Diehl and S. Gros, *Numerical Optimal Control: (draft)*, 2017.
[37] M. Diehl *et al.,* “Real-time optimization and nonlinear model predictive control of processes governed by differential-algebraic equations,” *Journal of Process Control*, vol. 12, no. 4, pp. 577–585, 2002.
[38] M. Diehl, H. G. Bock, and J. P. Schlöder, “A Real-Time Iteration Scheme for Nonlinear Optimization in Optimal Feedback Control,” *SIAM J. Control Optim.*, vol. 43, no. 5, pp. 1714–1736, 2005.

[39] B. Houska, H. J. Ferreau, M. Vukov, and R. Quirynen, “ACADO Toolkit User’s Manual,”
[40] H. J. Ferreau, C. Kirches, A. Potschka, H. G. Bock, and M. Diehl, “qpOASES: A parametric active-set algorithm for quadratic programming,” *Math. Prog. Comp.*, vol. 6, no. 4, pp. 327–363, 2014.
[41] W. C. Li, L. T. Biegler, and C.-M. U. E. D. R. Center, *Multistep, Newton-type Control Strategies for Constrained, Nonlinear Processes*: Engineering Design Research Center, 1989.
[42] T. Ohtsuka, “A continuation/GMRES method for fast computation of nonlinear receding horizon control,” *Automatica*, vol. 40, no. 4, pp. 563–574, 2004.
[43] Y. Shimizu, T. Ohtsuka, and M. Diehl, “A real-time algorithm for nonlinear receding horizon control using multiple shooting and continuation/Krylov method,” *Int. J. Robust Nonlinear Control*, vol. 19, no. 8, pp. 919–936, 2009.
[44] V. M. Zavala and L. T. Biegler, “The advanced-step NMPC controller: Optimality, stability and robustness,” *Automatica*, vol. 45, no. 1, pp. 86–93, 2009.
[45] D. Hrovat, S. Di Cairano, H. E. Tseng, and I. V. Kolmanovsky, “The development of Model Predictive Control in automotive industry: A survey,” in *IEEE International Conference on Control Applications (CCA), 2012: Dubrovnik, Croatia, 3-5 Oct. 2012 ; [part of] 2012 IEEE Multi-Conference on Systems and Control (MSC)*, Dubrovnik, Croatia, 2012, pp. 295–302.
[46] H. B. Pacejka, I. Besselink, and H. B. T. a. v. d. Pacejka, *Tire and vehicle dynamics,* 3rd ed. Amsterdam, London: Butterworth-Heinemann, 2012.
[47] T. Geiger, “Fahrstrategieoptimierung für ein Rennfahrzeug,” Master of Science Thesis, Lehrstuhl für Fahrzeugtechnik, Technische Universität München, 2016.
[48] S. van Koutrik, “Optimal Control for Race Car Minimum Time Maneuvering,” Master of Science Thesis, Delft University of Technology, 2015.
[49] M. Rieck, M. Bittner, B. Grüter, J. Diepolder, “FALCON.m: User Guide,” Technische Universität München, Aug. 2016.
[50] R. Quirynen *et al.,* “Symmetric algorithmic differentiation based exact Hessian SQP method and software for Economic MPC,” in *IEEE 53rd Annual Conference on Decision and Control (CDC), 2014: 15-17 Dec. 2014, Los Angeles, California, USA*, Los Angeles, CA, USA, 2014, pp. 2752–2757.
[51] J. T. Betts, *Practical methods for optimal control using nonlinear programming*. Philadelphia, Pa.: Society for Industrial and Applied Mathematics, 2001.
[52] S. Gros, “Shooting & Direct Collocation,” 2017.
[53] J. Nocedal and S. J. Wright, *Numerical Optimization*. New York, NY: Springer Science+Business Media, LLC.; Springer e-books, 2006.
[54] R. Verschueren, M. Zanon, R. Quirynen, and M. Diehl, “A Sparsity Preserving Convexification Procedure for Indefinite Quadratic Programs Arising in Direct Optimal Control,” *SIAM J. Optim.*, vol. 27, no. 3, pp. 2085–2109, 2017.
[55] A. Domahidi and J. Jerez, *FORCES Professional.* Available: http://embotech.com/FORCES-Pro. Accessed on: Aug. 01 2018.
[56] Andreas Wächter, “Short Tutorial: Getting Started With Ipopt in 90 Minutes,” IBM T.J. Watson Research Center.
[57] P. E. Gill, W. Murray, and M. A. Saunders, “User’s Guide for SNOPT Version 7: Software for Large-Scale Nonlinear Programming,” Jun. 2008.
[58] J. V. Frasch, S. Sager, and M. Diehl, “A parallel quadratic programming method for dynamic optimization problems,” *Math. Prog. Comp.*, vol. 7, no. 3, pp. 289–329, 2015.

[59] D. Kouzoupis, R. Quirynen, J. V. Frasch, and M. Diehl, “Block Condensing for Fast Nonlinear MPC with the Dual Newton Strategy,” *IFAC-PapersOnLine*, vol. 48, no. 23, pp. 26–31, 2015.
[60] J. Andersson, “A General-Purpose Software Framework for Dynamic Optimization,” Dissertation, Faculty of Engineering Science, KU Leuven, Heverlee, Belgium, 2013.
[61] A. V. Fiacco, *Introduction to sensitivity and stability analysis in nonlinear programming*. New York: Academic Press, 1983.
[62] C. Büskens, *Optimierungsmethoden und Sensitivitätsanalyse für optimale Steuerprozesse mit Steuer- und Zustands-Beschränkungen*. Münster (Westfalen), Univ., Diss., 1998, 1998.
[63] C. Büskens and H. Maurer, “Sensitivity Analysis and Real-Time Optimization of Parametric Nonlinear Programming Problems,” in *Online Optimization of Large Scale Systems*, M. Grötschel, S. O. Krumke, and J. Rambau, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 3–16.
[64] S. Gros, “Parametric Optimization,” 2017.
[65] S. Gros, “From linear MPC to real-time NMPC,” 2017.
[66] R. Findeisen, H. G. Bock, M. Diehl, F. Allgöwer, and J. P. Schlöder, “Nominal stability of real-time iteration scheme for nonlinear model predictive control,” *IEE Proceedings - Control Theory and Applications*, vol. 152, no. 3, pp. 296–308, 2005.
[67] M. Zanon, S. Gros, and M. Diehl, “A tracking MPC formulation that is locally equivalent to economic MPC,” *Journal of Process Control*, vol. 45, pp. 30–42, 2016.
[68] D. Rixen, “Engineering Dynamics: Lecture Notes - Version 2.0 (draft),” Winter Semester 2016-2017.
[69] A. Bogdanov, “Optimal control of a double inverted pendulum on a cart,” *Oregon Health and Science University, Tech. Rep. CSE-04-006, OGI School of Science and Engineering, Beaverton, OR*, 2004.
[70] G. Perantoni and D. J.N. Limebeer, “Optimal control for a Formula One car with variable parameters,” *Vehicle System Dynamics*, vol. 52, no. 5, pp. 653–678, 2013.
[71] *Road vehicles -- Vehicle dynamics and road-holding ability -- Vocabulary*, ISO 8855:2011, 2011.
[72] M. Trzesniowski, *Rennwagentechnik*. Wiesbaden: Springer Fachmedien Wiesbaden, 2014.
[73] G. Rill, *Simulation von Kraftfahrzeugen*. Braunschweig: Vieweg, 1994.
[74] G. Rill, *Road vehicle dynamics: Fundamentals and modeling / Georg Rill*. Boca Raton, FL: CRC Press, 2012.
[75] R. Lot and N. Bianco, “The significance of high-order dynamics in lap time simulations,” in *The dynamics of vehicles on roads and tracks: Proceedings of the 24th Symposium of the International Association for Vehicle System Dynamics (IAVSD 2015), Graz, Austria, 17-21 August 2015*, M. Rosenberger, M. Plöchl, K. Six, and J. Edelmann, Eds., Leiden: CRC Press/Balkema, 2016, pp. 553–562.

Appendix A: NMPC Implementation

This appendix describes the implementation of the Nonlinear Model Predictive Control in object-oriented MATLAB code. First, the structure of the software files and folders is described. Afterwards, the MATLAB S-Function, which runs during the simulation in Simulink, is explained. Finally, the methods of the NMPC class that can be called by the user are listed and detailed.

The code contained in the CD attached to this thesis (see folder called '*Code*') is made up of the following folders:

- '*+falcon*' that are modified files of the FALCON.m version used in this project
- '*Falcon NMPC*' which contains all the files that correspond to the NMPC controller
- '*Cart*', '*InversePendulum*', '*DoublePendulum'*, '*PointMass*', '*Racecar*' which correspond to the experiments performed during this project and can be seen as examples

In '*Falcon NMPC*', one can find

- '*@NMPC*', which is a MATLAB class-definition folder for the NMPC objects
- '*+NMPC_Plots*' which is a MATLAB package folder containing different standard plots and a template for creating new plots
- '*NMPC_addLQR_Cost.m*' a function to add a new cost function of the form (2-67) to the problem
- '*NMPC_createLQR_Cost.m*' a function to create the cost function needed for the function above
- 'NMPC_createProblem_TEMPLATE.m' a template function to create a FALCON.m problem as is needed for the NMPC class
- '*NMPC_sfun.m*' the S-Function that runs while the Simulink simulation is running
- '*NMPC_SimulinkLibrary.slx*' a Simulink Library that contains one masked system that entails the S-Function mentioned above
- '*qpDUNES.mexw64*' the built function for the optimization solver qpDUNES from the repository https://github.com/jfrasch/qpDUNES.git
- '*qpDUNES_dev.mexw64*' the built function for the optimization solver qpDUNES from the repository https://github.com/qpDUNES/qpDUNES-dev.git
- '*qpDUNES_options.m*' a function that creates the default options for qpDUNES

The masked subsystem contained in '*NMPC_SimulinkLibrary.slx*' can be put inside a Simulink application. This mask requires a NMPC object as a parameter. During a simulation, the following procedure is performed:

1. Initialize NMPC, including a first Preparation Phase
2. Update the current state of the system
3. Perform the Feedback Phase and set current control values
4. Get next reference values and perform Shift
5. Perform Preparation Phase
6. If simulation continues go to 2., otherwise terminate

After each of these points, the plotting interface is called, so that the plots can be updated according to the step that is being performed.

The following list describes the methods of the NMPC class that can be employed by the user to change the settings of the NMPC object. For all of the methods that include a `Name,Value` pair with name `'force'`, a value `true` allows to run the method during simulation without bringing a pop-up. The default is `false`.

NMPC object constructor

```
obj = NMPC(Problem, T_s, T_H, varargin)
```

Inputs:

`Problem`: Either a function as in 'NMPC_createProblem_TEMPLATE.m' (recommended), which is called the ProblemConstructor, or a FALCON.m problem
`T_s`: Sampling time of NMPC in seconds
`T_H`: Horizon length in seconds
`varargin`: not yet implemented, can be left out

Outputs:

`obj`: NMPC object

Reconstruct Problem

```
NMPC(varargin)
```

Inputs:

`Name,Value`: `'force'`, boolean

Set Sampling Time

```
setSamplingTime(SamplingTime)
```

Inputs:

`SamplingTime`: New sampling time in seconds

Set Horizon Length

```
setHorizonLength(HorizonLength, varargin)
```

Inputs:

`HorizonLength`: New horizon length in seconds
`Name,Value`: `'force'`, boolean

Set Hessian approximation and regularization method

```
setHessApprox(HessApprox, varargin)
```

Inputs:

`HessApprox`: One of the following: `'Gauss-Newton'`, `'BFGS'`, `'BFGS-noConstraints'`, `'Exact-noConstraints'`, `'Exact'`, `'UserProvided'`
`Name,Value`: `'HessReg'`, one of the following: `'None'`,`'Project'`(default),`'Mirror'`
`'force'`, boolean

Set Optimization Solver

`setSolver(solverName, varargin)`

Inputs:

`solverName`: Either `'IPOPT'` or `'qpDUNES'`

`Name,Value`: `'maxIter'`, default: `100`
`'maxCPUTime'`, default: `0.1`
`'OptimalityTolerance'`, default: `1e-6`
`'PrintLevel'`, default: `0`
`'force'`, boolean

Set properties for BFGS Hessian approximation

`setBFGS(UpdateSize, PosDefStrategy, inp1, Shift, ReevalG, Init, varargin)`

Inputs:

`UpdateSize`: Either `'Block'` for blockwise updates or `'Full'` for an update of the entire Hessian

`PosDefStrategy`: Strategy for keeping the BFGS Hessian approximation positive definite. Must be either `'Damped'` or `'Skip'`

`inp1`: Damping factor for `'Damped'`, maximum number of skips before reinitializing for `'Skip'`

`Shift`: Boolean that defines if a blockwise shift of the Hessian, the Jacobian and the optimization variables of the last iteration should be performed before the update

`ReevalG`: Boolean that defines if the Jacobian should be reevaluated (since there are change because of the shift and the new references)

`Init`: Scalar, vector or matrix used for the blockwise initialization of the BFGS Hessian

`Name,Value`: `'force'`, Boolean

Set User-defined Hessian

`setUserHessian(Type, Size, HessFcn, HSparsity, varargin)`

Inputs:

`Type`: Either `'Constant'` for a constant Hessian or `'FunctionHandle'` for if Hessian is calculated by a function

`Size`: Either `'Block'` for blockwise updates or `'Full'` for an update of the entire Hessian

`HessFcn`: Either a numeric constant or a function handle to the function that calculate the Hessian

`HSparsity`: A sparse matrix defining the sparsity structure of the Hessian

`Name,Value`: `'force'`, boolean

Set Shift Strategy

`setShiftStrategy(Controls, States, Reference, LagrangeMultipliers, varargin)`

Inputs:

`Controls`: Either `'last'` or a numeric value or a cell-array with the same length as the control values containing `'last'` or numeric values as elements. A numeric value sets this value for the control at the end of the prediction horizon, `'last'` uses the last value.

`States`: Either `'last'` or `'simulate'` or a numeric value or a cell-array with the same length as the state values containing `'last'` or `'simulate'` or numeric values as elements. A numeric value sets this value for the state at the end of the prediction horizon, `'last'` uses the last value, `'simulate'` uses the result of a forward simulation of the system with an explicit 4th-order Runge-Kutta method.

`Reference`: Either `'one'` for setting the only the reference at the end of the prediction horizon or `'all'` for setting the entire prediction horizon

`LagrangeMultipliers`: Either

`Name,Value`: `'force'`, boolean

Switch for Solving Full Nonlinear Problem at every sampling time

`set_solveFullProblem(tf, varargin)`

Inputs:

`tf`: boolean

`Name,Value`: `'force'`, boolean

Switch to Set the State and Control values into the Falcon.Problem after the Feedback Phase

`set_save2Problem(tf, varargin)`

Inputs:

`tf`: boolean

`Name,Value`: `'force'`, boolean

Switch to Recalculate and get the model Outputs after the Feedback Phase

`set_recalcModelOutputs(tf, varargin)`

Inputs:

`tf`: boolean

`Name,Value`: `'force'`, boolean

Add New Plot

`addNewPlot(PlotName,PlotFcnHandle,PlotData,varargin)`

Inputs:

`PlotName`: String to name the plot, every plot must have a different name

`PlotFcnHandle`: Function handle to the plotting function. A template can be found in the '*+NMPC_Plots*' folder. However, the function can be saved anywhere in the MATLAB path, not necessarily in '*+NMPC_Plots*'.

`PlotData`: Constant data specific to the plot, it can be anything.

`Name,Value`: `'force'`, boolean

Remove Plot by name

`removePlot(PlotName,varargin)`

Inputs:

`PlotName`: Name of the plot to be removed
`Name,Value`: `'force'`, boolean

Remove All Plots

`removeAllPlots(varargin)`

Inputs:

`Name,Value`: `'force'`, boolean

Appendix B: About the Formula Student car – eb016

The car that the algorithm presented in this thesis was developed for is TUfast's eb016. TUfast is the Formula Student team of the Technical University of Munich. This team engineers and build an electric racecar every year since 2011 for the Formula Student competitions. The eb016 is the car built in the year 2016 and therefore TUfast's sixth electric car. It was also the TUfast's third all-wheel driven car and the first using outboard motors. This means that each tire is driven individually by one electric motor mounted directly at the wheel.

Each of the motors is controlled individually by the inverters, which set the output torque of the motors based on a wheel speed control. Therefore, each of the inverters receives four values every $10\,ms$: A feed-forward torque T_{FF}, a wheel speed command N_{cmd} and a lower and upper limit for the output torque $T_{lim,\mathrm{lo}}$ and $T_{lim,\mathrm{up}}$. In this thesis, these limits where neglected, assuming that they can be set at the minimum and maximum values of the motor respectively. The communication with the inverters is done over CAN-Bus. It must be noted that the values for T_{FF} and N_{cmd} calculated in this project, correspond to values at the wheel. Before they are passed to the inverters, the must be scaled using the ratio of the gearboxes ($i = 14.385$).

In this thesis, it is assumed that the state vector of the NMPC model is known exactly at every timestep of the simulation. The states of the NMPC model are listed in the table below:

Symbol	Name of the state	Sensor	EKF
s	Path coordinate	-	no
n	Normal coordinate	-	no
ζ	Relative yaw angle	-	no
v	Speed	Correvit, GPS	yes
β	Slip angle	Correvit	yes
$\dot{\psi}$	Yaw rate	MEMS	yes
a_x	Longitudinal acceleration	MEMS	yes
a_y	Lateral acceleration	MEMS	yes
δ_F	Steering angle	Potentiometer	no

Table 6-1: Racecar NMC model – States and corresponding sensor

This list also shows if a sensor that can directly measure the state value is included in the car. Furthermore, an Extended Kalman Filter (EKF) runs on the electronic control unit of the eb016. This algorithm is a model-based filter also used for state estimation. Therefore, the states extracted from the EKF are not only filtered, but they also provide a good estimate of the state in case of sensor failure. However, at the time of writing of this thesis, the EKF does not include the states for the position (s and n) and the heading (ζ) of the car and these values would have to be obtained in another way.

Appendix C: Track Import

The track used for the experiments with the point mass model in section 3.4 and for the tests with the autonomous racecar model in chapter 4, was imported using measured (and filtered) data. The data used here corresponds to the *Endurance* event of Formula Student Germany in the year 2016, specifically the laps driven by the first driver. It must be remarked that the midline of the track generated with this method corresponds to a driven line and not the midline of the actual track.

The main data used for the track generation is the distance, the speed, the yaw rate and the lateral acceleration, but other signals like the slip angle can also be used to improve the quality of the import. All of these signals were directly measured by sensors (see Appendix B) and most of them were filtered with a model based Extended Kalman Filter. The idea is to calculate the curvature of the track and then integrate it over distance to get the course angle. The curvature C can be calculated using the yaw rate $\dot{\psi}$ or the lateral acceleration a_y and the speed v of the car by

$$\begin{aligned} C_{\dot{\psi}} &= \frac{\dot{\psi}}{v} \\ C_{a_y} &= \frac{a_y}{v^2} \end{aligned} \tag{6-1}$$

These two values can then be averaged with a weight $w \in [0,1]$ to get $C = w\, C_{\dot{\psi}} + (1-w)\, C_{a_y}$. The integration over the distance to get the course angle θ must then be done numerically. This is done in by

$$\theta = \mathrm{cumsum}\big(C\ \mathrm{grad}(s)\big) + \theta_{\mathrm{start}} \tag{6-2}$$

where θ_{start} is the value of the course angle at the start of the track, s is the distance, the function $\mathrm{grad}(s)$ calculates the numerical gradient of s, i.e. $\partial s \approx \mathrm{grad}(s)$, and the function $\mathrm{cumsum}(z)$ creates a cumulative sum of the elements of vector z.

If the track is closed, as in the case of the Endurance event, the course angle θ should be $2\pi + \theta_{start}$ at the end of the lap. However, because of sensor noise and bias, this is not necessarily the case. Moreover, a closed lap should satisfy the following conditions for the 2D position $(x_{\mathrm{end}}, y_{\mathrm{end}})$ at the end of the lap:

$$\begin{aligned} x_{\mathrm{end}} &= \sum \cos\theta\ \ \mathrm{grad}(s) \stackrel{!}{=} 0 \\ y_{\mathrm{end}} &= \sum \sin\theta\ \ \mathrm{grad}(s) \stackrel{!}{=} 0 \end{aligned} \tag{6-3}$$

Therefore, the following algorithm is used to make these corrections:

Algorithm 4: Newton-type equality constrained optimization

Input: s , C

1. Calculate scaling factor for course angle: $sc = \sum C \ \mathrm{grad}(s)/2\pi$
2. Get first estimate of course angle: $\theta = sc \cdot \mathrm{cumsum}\big(C \ \mathrm{grad}(s)\big) + \theta_{\mathrm{start}}$
3. Calculate x_{end} and y_{end} with equation (6-3)
4. Calculate the corrected gradients ∂x and ∂y by

$$\partial x = \cos\theta \ \ \mathrm{grad}(s) - x_{\mathrm{end}}/\mathrm{numel}(\theta)$$

$$\partial y = \sin\theta \ \ \mathrm{grad}(s) - y_{\mathrm{end}}/\mathrm{numel}(\theta) \qquad \textbf{(6-4)}$$

   where $\mathrm{numel}(\theta)$ is the number of elements of θ.
5. Calculate the corrected course angle: $\theta = \mathrm{atan}\frac{\partial y}{\partial x}$

Finally, the coordinates (x, y) of the track can be computed by:

$$x = \mathrm{cumsum}\big(\cos\theta \ \ \mathrm{grad}(s)\big)$$

$$y = \mathrm{cumsum}\big(\sin\theta \ \ \mathrm{grad}(s)\big) \qquad \textbf{(6-5)}$$

Plotting these for the multiple laps in the data results in a plot as in

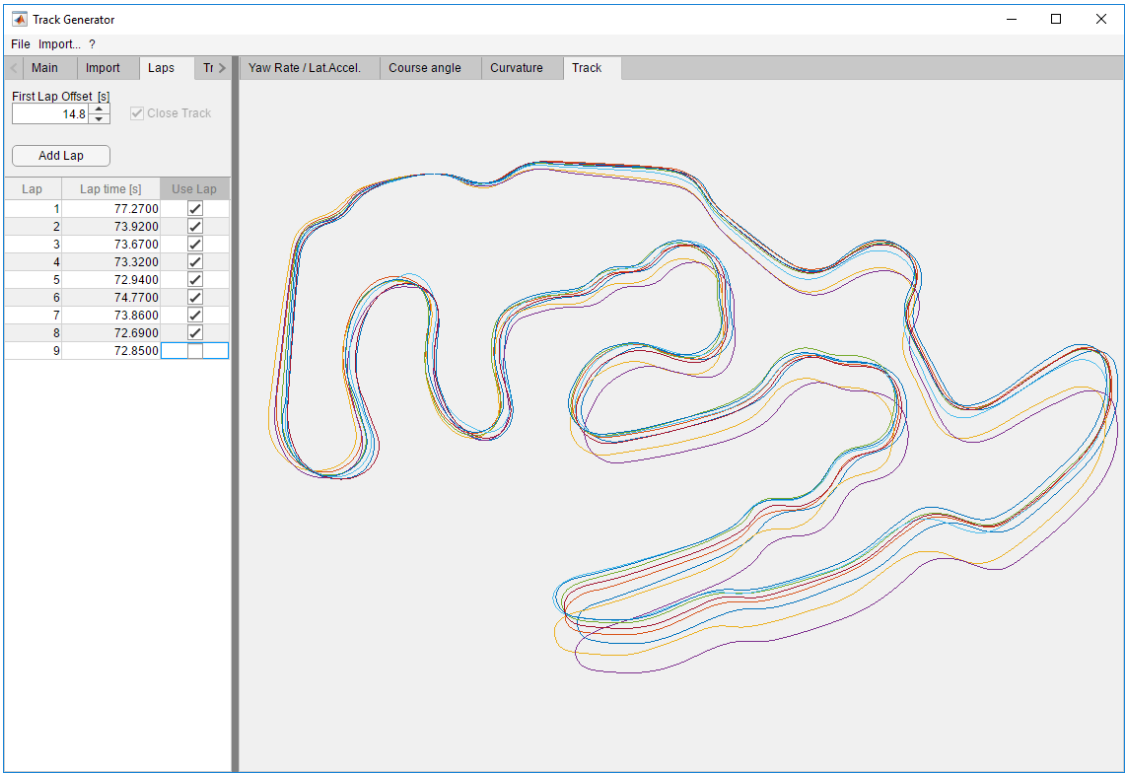

Figure 6-1: Track import – 2D coordinates

For the models in this project, the course angle θ and its derivative with respect to the track distance s, i.e. the curvature $C = \frac{\partial\theta}{\partial s}$, are relevant. Therefore, the course angle θ of all laps

together was fitted using cubic splines. This procedure was done using the SPLINEFIT function for MATLAB. The breaks for the splines were set manually. Figure 6-2 shows the start of the procedure, the black line shows the splines approximation and the asterisks (*) their breaks. Breaks are added and moved until the fit is satisfactory.

It must be noted that this procedure may create a small displacement of the coordinates $(x_{\mathrm{end}}, y_{\mathrm{end}})$ at the end of the track again, as is the case in this project. This displacement is unfortunately more complicated to correct and was neglected in this project.

The advantage of using splines to fit the data is that one gets a smooth and differentiable input for the models. For this project, these algorithms to build a track using logged data were implemented as a Graphical User Interface, called *Track Generator*.

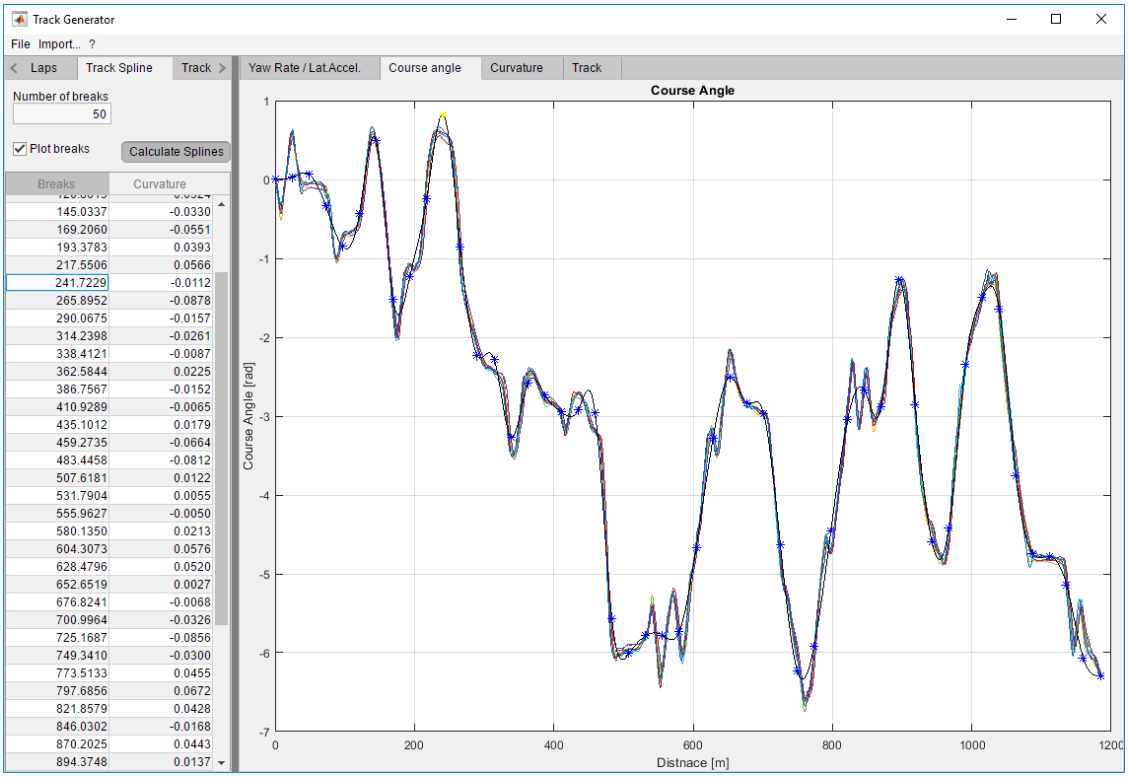

Figure 6-2: Track import – Course angle