KU LEUVEN

ARENBERG DOCTORAL SCHOOL
Faculty of Engineering Science

Embedded Model Predictive Control and Moving Horizon Estimation for Mechatronics Applications

Milan Vukov

Dissertation presented in partial
fulfillment of the requirements for the
degree of Doctor in Engineering Science

April 2015

Embedded Model Predictive Control and Moving Horizon Estimation for Mechatronics Applications

Milan VUKOV

Examination committee:
Prof. dr. ir. D. Vandermeulen, chair
Prof. dr. M. Diehl, supervisor
Prof. dr. ir. J. Swevers, co-supervisor
Dr. H. J. Ferreau, co-supervisor
(ABB Corporate Research)
Prof. dr. ir. H. Bruyninckx
Prof. dr. ir. M. Moonen
Prof. dr. ir. F. Logist
Prof. dr. J. Bagterp Jørgensen
(Technical University of Denmark)

Dissertation presented in partial
fulfillment of the requirements for
the degree of Doctor
in Engineering Science

April 2015

© 2015 KU Leuven – Faculty of Engineering Science
Uitgegeven in eigen beheer, Milan Vukov, Kasteelpark Arenberg 10, B-3001 Heverlee (Belgium)

Alle rechten voorbehouden. Niets uit deze uitgave mag worden vermenigvuldigd en/of openbaar gemaakt worden door middel van druk, fotokopie, microfilm, elektronisch of op welke andere wijze ook zonder voorafgaande schriftelijke toestemming van de uitgever.

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

Acknowledgments

I still feel quite strange in this, *new*, situation. Almost four and a half years ago, at the beginning of my PhD adventure, reaching the end seemed nearly impossible. Yet, after many hours of work, many ups and a bit more downs, many collaborations and help from a number of very fine people the adventure came to the end.

My first *official* meeting with my supervisor Moritz Diehl was in a park near Naamsepoort in Leuven. Quite atypical I would say. Later when I joined his group I realized environment in the group was atypical as well, full of flexibility. Moritz, I would like to thank you for a given opportunity to pursue a PhD degree in your group and sharing your vast enthusiasm with me. Moreover, I am grateful for your always positive attitude, no matter how difficult situations we encountered.

I would like to thank all the members of my PhD jury, prof. Jan Swevers, Hans Joachim Ferreau, prof. Herman Bruyninckx, prof. Marc Moonen, prof. Filip Logist and prof. John Bagterp Jørgensen, for their valuable comments and remarks given on my research and the PhD manuscript. In addition, I thank Joachim for guiding me in the early stages of my studies, occasionally reminding me of the bigger picture.

Funding by the EU projects FP7-EMBOCON, ERC HIGHWIND and Eurostars SMART is gratefully acknowledged.

Andrew Wagner and I spent countless hours in the kite lab. We used to work hard, but sometimes just *goofing off*. Thanks for helping me out with my problems, ideas and giving sometimes negative feedback that made me think even more. Besides working together on research activities, Greg Horn infected me with Python and taught me the American slang. Thanks for that! Thanks to Sébastien Gros sometimes chaotic trains of thoughts in my mind were put on the right tracks. It was my pleasure to work with Gianluca Frison. I wish I had met you earlier. Brainstorming sessions with Joel Andersson are highly appreciated. I have always enjoyed them very much.

I had a chance to spend quality time and collaborate with many smart people I met on the Arenberg campus. Here, I would like to express my gratitude to Hammad Ahmad, Max Bögli, Janick Frasch, Kurt Geebelen, Joris Gillis, Boris Houska, Tom Kraus, Adeleh Mohammadi, Reinhart Paelinck, Rien Quirynen, Michal Salaj and Mario Zanon.

A big thanks also goes to Alexander Domahidi for providing excellent tech support for his solvers. Moreover, I am grateful to Tobias Glück for sharing his pendulum models.

The increasing amount of administrative tasks was carried out with the efficient support from Veronique Cortens, Jacqueline De bruyn, Ida Tassens, Elsy Vermoesen and John Vos. Thank you very much for your efforts!

During my internship at AmpyxPower in 2013, I had a chance to experience life in a startup. I thank everyone there for making me feel welcomed and like a part of the team. Despite their busy schedules, Sören and Mat always managed to be helpful and enthusiastic. Thank you a lot guys!

The ex-Yu crew made my stay in Leuven even more memorable. I am indebted to Ivan and Bogdan in so many ways, particularly for the period when I just moved to Leuven. Moreover, I owe thanks, in no particular order, to: LićVu, TićMa, Mire, Paja, Ilija, Marija & Topke, Ljilja, Jelena, Milica & Danilo, Dušan, Anđelo, Gorana & Ivan, Ivana, Damjan, Marina & Bane.

My mother Dušanka, sister Jovana and father Ratomir have always encouraged me, shared their love and support. Thank you for being there for me and making those hundreds of kilometers between us look small. I would also like to thank my mother- and father-in-law, Nada and Petar, for their generous help in the final stage of my PhD.

I can hardly imagine finishing this adventure without my family, my wife Snežana and our son Nikola. Snežana, thank you for *just* listening to my stories about technical & research related *silly stuff*, though I knew most of them were boring and uninteresting to you. Knowing how much *hard for maintenance* I was, you did a really good job being my voice of reason, patient and always encouraging. Thank you for believing in me, thank you for your limitless love. *Ti si moj ludi kamen spoticanja.*

Milan Vukov
Leuven, April 2015

Abstract

The concepts of Model Predictive Control (MPC) and Moving Horizon Estimation (MHE) received widespread acceptance in both industry and academia. This is due to the ability to explicitly define objectives and constraints in the framework of dynamic optimization. Those key facts eventually lead to improved control performance. Progress in the area of optimization algorithms and computational hardware in the last two decades have extended the applicability of numerical optimization to mechatronics applications. In particular, the applicability was extended for small-scale systems with time constants in micro- and millisecond range. Following the success convex quadratic programming (QP) solvers made in linear MPC, the ideas have been extended for nonlinear MPC and MHE.

This thesis aims to further reduce the gap between academia and industry. With optimized software for nonlinear MPC and MHE and extended problem formulations we can efficiently handle complex nonlinear systems, possibly working under nonlinear constraints. We present recent extensions to the ACADO Code Generation Tool (CGT). Once specified, the problem structure is exploited offline by the tool that generates the tailored code optimized for execution in real-time environments. We demonstrate the strength of the newly developed features of the tool in numerical simulations and two real-world applications.

Our numerical simulations show readiness to effectively treat problems on both short and long horizons. For the systems with a few states and few controls solution times in the microsecond range are observed. Our largest test case involves an MPC formulation comprising 33 states, 3 controls, and a prediction horizon of 50 steps. This test case comprises 1800 optimization variables and is possible to solve on modern hardware in under 50 milliseconds. For another test problem where long prediction horizons are the necessity, we observe solution times less than 4 milliseconds in a test case with the horizon of 150 steps.

The first experimental study is the application of nonlinear MPC and MHE to a laboratory scale overhead crane. Here we present computational performance of two generations of the ACADO CGT. Using the original implementation of the tool and only an MPC controller in the first control scenario, we achieved execution times close to 1 millisecond. With the recently optimized code, we attained nearly the same execution times, now with both nonlinear MHE and the MPC in the loop. In addition, with the more optimized code we reached average runtimes for the nonlinear MPC three times faster than with the original implementation.

The aim of the second real-world application is to validate the computational performance of the auto-generated MHE and MPC solvers on an experimental setup for rotational start-up of an airborne wind energy system. The system model describes complex nonlinear dynamics comprising 27 differential states, 1 algebraic state and 4 controls. The results confirm that nonlinear MPC formulation with more than 1500 optimization variables is solved in just less than 5 milliseconds reducing the total feedback time to below 10 milliseconds.

Beknopte samenvatting

Model predictieve controle (MPC) en bewegende horizonschatting (MHE) kenden een doorbraak in zowel de industrie als in de academische wereld. De mogelijkheid van dynamische optimalisatie om een doelfunctie en beperkingen expliciet te specifiëren, ligt aan de basis van hun populariteit. Beide mogelijkheden dragen bij tot een verbeterde controleperformantie. De afgelopen twintig jaar kenden de domeinen van optimalisatie-algoritmes enerzijds en computationele hardware anderzijds een grote progressie zodat toepassing van numerieke optimalisatie ook in mechatronische systemen mogelijk werd. In het bijzonder werd toepassing op kleinschalige systemen met tijdsconstanten in de orde van micro- of milliseconden mogelijk. Geïnspireerd op het succes van solvers voor convex kwadratische programma's (QP), werden de ideeën voor lineaire MPC uitgebreid voor niet-lineaire MPC en MHE.

Deze thesis heeft tot doel om de tweespalt tussen industriële praktijk en de academische wereld verder te verkleinen. Met geoptimaliseerde software voor niet-lineaire MPC en MHE, en met alternatieve probleemformuleringen kunnen we op een efficiënte manier complexe niet-lineaire systemen behandelen, met inbegrip van eventueel niet-lineaire beperkingen. Concreet presenteren we recente uitbreidingen op de ACADO Code Generation Tool (CGT). Na de probleemspecificatie wordt de structuur van het probleem offline uitgebuit door deze tool die met deze informatie code genereert die geoptimaliseerd is voor uitvoering op real-time platformen. De sterkte van de uitbreidingen op deze tool demonstreren we met numerieke simulaties en ook met twee uitgewerkte toepassingen op reële systemen.

Onze numerieke simulaties geven aan dat probleemstellingen met zowel korte als lange tijdshorizon efficiënt kunnen opgelost worden. Voor systemen met enkele toestanden en control-inputs nemen we oplossingstijden in de orde van microseconden waar. De meest grootschalige test-case beschouwd in dit werk is een systeem met 33 toestanden, 3 controle-inputs en een tijdshorizon van 50 stappen. Voor deze test-case met 1800 optimalisatie-veranderlijken is een oplossingstijd van 50 milliseconden gerealiseerd op moderne hardware. Voor een andere test-case die een lange tijdshorizon vereist, is een oplossingstijd van 4 milliseconden gerealiseerd voor een tijdshorizon van lengte 150.

Een eerste experimentele studie van niet-lineaire MPC en MHE is doorgevoerd op een portaalkraan op laboratoriumschaal. We geven hierbij de computationele performantie van twee generaties van ACADO CGT. De eerste generatie kent een oplossingstijd van ongeveer 1 milliseconde voor MPC op zich. Met de recent verbeterde generatie komen we uit op eenzelfde oplossingstijd, maar nu voor de combinatie van niet-lineaire MPC en MHE. De MPC op zich komt met de verbetering een factor 3 sneller uit dan de eerste generatie van de tool.

In een tweede experimentele toepassing gaan we de computationele performatie na van automatisch gegenereerde MHE en MPC solvers na voor een toestel voor rotationele opstart van een alternatief windenergie-systeem. Het systeemmodel beschrijft complexe niet-lineaire dynamica met 27 toestanden, 1 algebraïsche beperking, en 4 controle-inputs. De resultaten bevestigen dat een MPC formulering met ruim 1500 optimalisatie-veranderlijken in 5 milliseconden kan worden opgelost, waarmee de totale tijd voor een feedback-stap onder de 10 milliseconden komt.

Abbreviations

AWE	Airborne wind energy
CGT	Code generation tool
DAE	Differential algebraic equations
DAQ	Data acquisition
FLOP	Floating point operation
IMU	Inertial measurement unit
IPM	Interior point method
KKT	Karush-Kuhn-Tucker (first order optimality conditions)
LAS	Line angle sensor
LTI	Linear time-invariant
MCU	Microcontroller
MHE	Moving horizon estimation
MPC	Model predictive control
NLP	Nonlinear program
NMPC	Nonlinear model predictive control

OCP	Optimal control problem
ODE	Ordinary differential equations
QP	Quadratic program
RTI	Real-time iteration scheme
SQP	Sequential quadratic programming

List of Symbols

N	Number of control intervals
n_{b}	Number of bounds
N_{c}	Number of control intervals in an MPC formulation
n_{c}	Number of affine constraints
N_{e}	Number of control intervals in an MHE formulation
$n_{\mathrm{qp,it}}$	Number of QP solver iterations
n_u	Number of control inputs
n_{v}	Number of optimization variables
n_x	Number of differential states
n_z	Number of algebraic states

Contents

Abstract **v**

Contents **xiii**

List of Figures **xvii**

List of Tables **xxi**

1 Introduction **1**

1.1 Motivation 1

1.2 Contributions and Overview 7

2 Fast Nonlinear Model Based Predictive Control and Estimation **13**

2.1 Preliminaries 15

2.1.1 Methods for Discretization of Optimal Control Problems 17

2.1.2 Nonlinear Programming 21

2.2 Fast Nonlinear Model Predictive Control 24

2.2.1 Real-time Iterations for Nonlinear MPC 25

2.3 Fast Nonlinear Moving Horizon Estimation 29
2.3.1 Problem Formulation 30
2.3.2 Real-time iterations for Nonlinear MHE 33
2.4 The Big Picture: Real-time Iterations in Closed Loop with Nonlinear MPC and MHE 35
2.5 Building Blocks for Fast Nonlinear MPC and MHE 36

3 Tailored Quadratic Programming Solvers 39
3.1 Active-set Quadratic Programming Solvers 42
3.2 Condensing Procedures 45
3.2.1 Classical Condensing 48
3.2.2 $\mathcal{O}(N^2)$ Condensing 59
3.2.3 $\mathcal{O}(N^2)$ Factorization of the Condensed Hessian 64
3.3 Structure Exploiting Interior Point Method Solvers 69
3.4 The Dual Newton Strategy 72
3.5 Numerical Simulations 76
3.5.1 Chain of Masses Connected by Springs 77
3.5.2 Double and Triple Pendulums 79
3.5.3 Performance Profiles 81
3.5.4 Results 82
3.6 Conclusions 95

4 The ACADO Code Generation Tool 97
4.1 Features 99
4.2 Interfaces to QP solvers 101
4.3 Structure of the Exported Solver 102

4.4 Real-world Applications 103

5 Real-time Control of an Overhead Crane 105

5.1 Experimental setup and Dynamic Model 106

5.1.1 Experimental Setup 106

5.1.2 Dynamic Model 107

5.2 Control Architectures 110

5.2.1 Scenario 1 110

5.2.2 Scenario 2 113

5.3 Experimental Results 116

5.3.1 Scenario 1 116

5.3.2 Scenario 2 121

5.4 Conclusions 126

6 Real-time Control of an Airborne Wind Energy System 127

6.1 Experimental Setup 130

6.1.1 Hardware 130

6.1.2 Software 135

6.2 Control Architecture 143

6.2.1 Modeling 144

6.2.2 Estimator 148

6.2.3 Controller 152

6.3 Experimental Results 157

6.4 Conclusions 170

7 Conclusions and Outlook 173

7.1 Directions for Future Research 175

Bibliography **177**

Curriculum Vitae **191**

List of Publications **193**

List of Figures

1.1 The Model Predictive Control and Moving Horizon Estimation frameworks. 3

2.1 Division into preparation and feedback step. 35

3.1 A chain of masses connected with springs. 77

3.2 Maximum execution times of four different NMPC solvers for a chain of masses with up to three masses in the chain. 82

3.3 Maximum execution times of four different NMPC solvers for a chain of masses with four and five masses in the chain. 83

3.4 Maximum number of QP iterations for different NMPC solvers in the chain of masses benchmark. 85

3.5 Maximum feedback times for the chain of masses benchmark. . 87

3.6 Performance profiles for the chain of masses benchmark. 88

3.7 Maximum execution times of different NMPC controllers for the double (DP) and the triple (TP) pendulums. 89

3.8 Maximum number of iterations for different NMPC solvers in the pendulum benchmark. 90

3.9 Conditioning within the condensing based solvers in the pendulum benchmark. 91

3.10 Maximum feedback times for the pendulum benchmark. 92

3.11 Performance profiles for different NMPC solvers in the pendulum benchmark. 93

4.1 Structure of an exported MHE/MPC solver. 103

5.1 Schematic of the experimental overhead crane. 106

5.2 Relation between measured angle α and pendulum angle deflection θ. 109

5.3 Point-to-point motions in the first scenario: states and controls. . 117

5.4 Point-to-point motions in the first scenario: performance of the nonlinear MPC. 119

5.5 Rejection of an external disturbance force applied to the hanging mass in the first scenario: states and controls. 120

5.6 Point-to-point motions in the second scenario: states and outputs. 121

5.7 Point-to-point motions in the second scenario: controls. 122

5.8 Point-to-point motions in the second scenario: computational performance of the MHE and the NMPC. 125

6.1 An airborne wind energy concept. 128

6.2 The experimental setup. 131

6.3 A schematic representation of the experimental setup. 132

6.4 A schematic representation of the carousel control hardware. . . 133

6.5 A schematic representation of control hardware inside the plane. 134

6.6 Simplified software organization. 137

6.7 Simplified deployment of the real-time software components. . 140

6.8 The schematic representation of the carousel. 144

6.9 Position of the plane obtained in closed-loop experiments. . . . 158

6.10 Aerodynamic forces and torques estimated in closed-loop experiments. 159

6.11 Carousel speed $\dot{\delta}$, torque u_{t} and torque slew rate u_{st} obtained in closed-loop experiments. 160

6.12 Euler angles obtained in closed-loop experiments. 161

6.13 Control surfaces′ angles and corresponding slew rates obtained in closed-loop experiments. 162

6.14 Performance indicators for the MHE obtained in closed-loop experiments. 163

6.15 Estimated invariants defined in the equation (6.6). 164

6.16 Performance indicators and execution times for the NMPC obtained in closed-loop experiments while using the HPMPC QP solver. 166

6.17 Performance indicators and execution times for the NMPC obtained in closed-loop experiments while using the condensing based approach and qpOASES QP solver. 168

6.18 Samples of timing diagrams obtained in closed loop experiments.169

List of Tables

3.1 Number of optimization variables for the sparse and the condensed QP. 46

3.2 Computational cost of Algorithm 3.1. 51

3.3 Computational cost of Algorithm 3.3. 52

3.4 Computational cost of Algorithm 3.4. 53

3.5 Computational cost of Algorithm 3.5. 54

3.6 Computational cost of Algorithm 3.6. 57

3.7 Computational cost of Algorithm 3.7. 57

3.8 Computational cost of Algorithm 3.8. 58

3.9 Computational costs of Algorithms 3.9 and 3.10. 62

3.10 Computational costs of Algorithms 3.11 and 3.12. 64

3.11 Computational costs of Algorithms 3.13 and 3.14. 68

3.12 Relevant dimensions for the chain of masses benchmark. 79

3.13 Relevant dimensions for the pendulum benchmark. 80

3.14 Percentages of the RTI scheme spent in the feedback phase. 94

5.1 Estimate of the model parameters. 108

5.2 Execution times of the auto-generated NMPC. 118
5.3 Execution times for the auto-generated MHE and NMPC algorithms. 123

6.1 Synchronization of the real-time software components. 139
6.2 Measurement information. 141
6.3 Standard deviations for measurements and disturbances. 150
6.4 Dimensions relevant to the MHE formulation (6.15). 150
6.5 Dimensions relevant to the NMPC formulation (6.18). 154

1

Introduction

1.1 Motivation

The task of operating a system is typically referred to as a *control problem* and requires the design of a controller. The controller is an algorithm that steers the system to a desired state. For example, a controller can serve for the regulation of the water level in a basin, or for the movement of a welding tool at the end of a robot arm along a trajectory. A solution of the control problem can be separated in two contexts: *offline* and *online*. The offline context refers to a system which is nonoperational, i.e. not running. Design of the controller is one of the tasks that is always done offline. Once the controller is designed it needs to be deployed to serve its purpose: to control the system. Assuming the system is running, once the controller is turned on, it executes within the *online* context. Implementations on digital computers require the controller to be repeatedly executed as the system is operated. Typically the controller is executed periodically. We define this time interval as the *control period*.

Model based control design is a technique that further requires development of a mathematical model of the system – a set of equations that describe the dynamics of the system. The developed model enables one to simulate the system and predict system's evolution for arbitrary control inputs. Looking from the perspective on *when* to calculate the controls, two approaches can be distinguished: *open-loop* and *closed-loop*.

The open-loop approach proposes to calculate all control actions offline. The controls are pre-calculated based on some insights about the system dynamics or, e.g. by using a model of the system. As a result, a sequence of control actions is obtained. Once online, the controller takes the control actions from the sequence, one control action per control period, and applies it to the system to accomplish the prescribed task. If the model used for the design is of insufficient quality and/or the system is under disturbances the approach might produce unsatisfactory results or even fail.

Another methodology is to control the system in *closed loop*. Based on a *feedback* signal, the controller calculates the control actions online at each time step. Every system meant to be controlled has a number of *outputs* that can be sensed with appropriate sensors. For example, a sensor can monitor the water level in a basin, or a joint angle in a robot arm. The number of outputs is usually limited, and sometimes additional efforts are needed to reconstruct the state of the system. The task of an *estimator* is to reconstruct the state based on data from the sensors. In a simple scenario, the estimator is an algorithm that is executed prior to execution of the controller with the aim to estimate the state of the system. The estimated state is *fed back* to the controller and the controller calculates the control action based on the estimated state. As with the controller, the estimator design is done offline, while the execution of the estimator is within the online context. In comparison to the open-loop approach, the closed-loop one has much higher potential to cope with *model-plant miss-match*[1] and to react to disturbances. Control design often involves the design of an estimator, thus in the text that follows we are not explicitly going to refer to estimator design but indicate when necessary.

Designing a controller one tries to fulfill some *objectives*, while in turn satisfying some *constraints*. For example, an objective might be to maintain the water level in a basin, while keeping the tap angle between certain minimum and maximum values. Yet another example of an objective is to keep the welding tool on a certain path. Furthermore, while moving the welding tool, we might want to penalize excessive and aggressive movements of the robot arm that lead to direct energy savings and prolonged lifetime of the equipment. Robot joints are typically actuated by electric motors, which have limits on available torque, speed, and angle of rotation. An efficient control design *is* supposed to take all those constraints into account. Control design in the framework of

---

[1] In chemical industry terminology, systems are usually referred to as *plants* or *processes*.

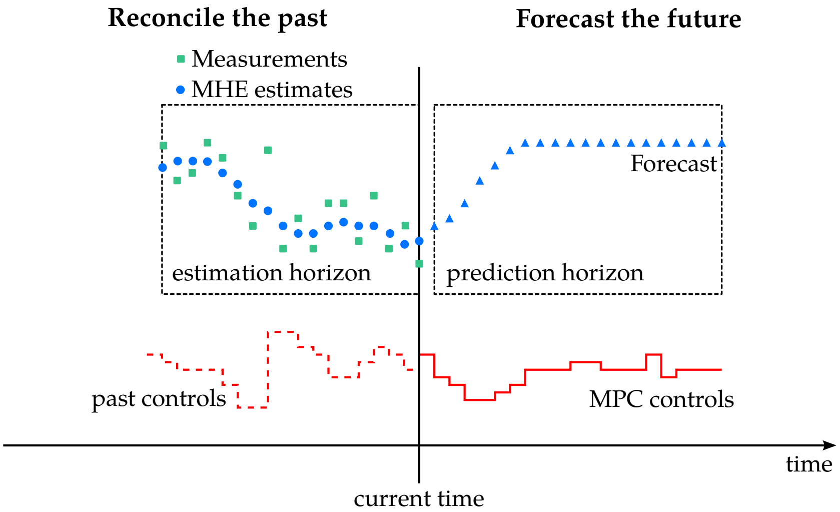


Figure 1.1: The Model Predictive Control and Moving Horizon Estimation frameworks[2].

dynamic optimization allows one to explicitly formulate objectives and constraints. Explicit inclusion of constraints avoids safeguarding logic that is necessary for some other approaches.

A mathematical formulation of a control problem within the dynamic optimization framework that *directly* takes into account objectives, system dynamics and constraints is called the *Optimal Control Problem (OCP)* formulation. In a similar fashion a name for an estimator formulation can be coined, but is uncommon. An OCP can be solved offline, yielding a sequence of optimal control actions that can be applied to the system in open-loop. Another approach is to solve an OCP repeatedly while the process is running. At each time step one OCP is solved, yielding an optimal solution. In the closed-loop setting there is also a need for an algorithm that solves the estimation problem that yields an optimal state estimate that is fed to the controller. The principle of solving OCPs online is commonly called *Model Predictive Control (MPC)*. Similarly, *Moving Horizon Estimation (MHE)* is an online approach to solve estimation problems.

Let us now take a more detailed look at the closed-loop setting of our interest. The analysis is based in Figure 1.1, where for brevity we avoid constraints. The task of the MHE is to *reconcile the past* – reconstruct the state trajectory (blue dots)

---

[2] The illustration is inspired by [1].

based on noisy sensor data (green squares) and past control actions (dashed red lines) on a finite estimation horizon in the past. Once the current state estimate is obtained, the blue dot at the current time, the MPC is triggered. Output of the MPC algorithm is a sequence of optimal control actions (red full line) such that the forecast (blue triangles) eventually fulfills the design requirements.

In an ideal situation, if we take the calculated control actions and apply them in open loop, the control sequence results in system evolution as predicted by the controller. A l*egitimate question to ask in this situation is:* why to employ MHE and MPC techniques at all? The answer is: there is no need for those techniques in the ideal situation. We compute optimal control actions only once – offline – and later apply them in open-loop. However, the reality is everything but ideal. First, an accurate model of a system is in general unavailable. Second, the sensor data is usually corrupted with noise and of limited accuracy. Third, the system is usually subjected to disturbances. Those are the reasons we need state estimator to reconcile the past measurements.

Quality of the state estimates is directly affected by model quality, quality of the sensor data, and naturally depends on estimator design. In effect, this means that the system is *not* going to be at the state predicted by the MPC after we apply the first step from the sequence. Thus, we apply the first optimal control action and at the next time step we repeat all calculations: estimate the state and recompute the optimal control actions on the prediction horizon. One of the bright sides of this approach is that with an appropriate control design one can expect that the system is going to be in a vicinity of the predicted state after the first control action is applied.

The key feature to remember and the one that motivates the usage of model based predictive control and estimation techniques such as MPC and MHE is the ability for a direct specification of objectives and constraints in the elegant mathematical framework of dynamic optimization. This approach cannot only improve the closed-loop performance but potentially speed up control design. In particular, heuristics other control approaches exercise can be avoided. However, those appealing properties come at the expense of potentially computationally expensive numerical calculations.

The dynamic systems we are particularly interested in are the ones with nonlinear dynamics, possibly running under nonlinear constraints. Numerical solvers for optimal control of nonlinear dynamic systems face a set of challenges. In addition to the typical need for solving linearized (sub-)problems, exactly the same as in linear MPC and MHE, treatment of nonlinear dynamics and constraints demands for additional – often expensive – online computations. For instance, the model simulation and evaluation of nonlinear functions is performed online.

Online solution to control and estimation problems not only has to be numerically correct, but also has to be performed *on time* – within the control period. This is the *real-time feasibility* requirement. Numerical calculations for MPC and MHE take a finite amount of time that is in general non-negligible in comparison to the control period. This time period is called *the computational delay*, and consists of the execution time of an MHE algorithm and the execution time of an MPC algorithm. A number of factors influence the execution times of the numerical solvers for MHE and MPC, e.g. model complexity and horizon length. Moreover, the solution to the underlying optimization problems is an iterative process in general, meaning that the execution times are varying. A direct consequence is that in a fair number of circumstances the exact optimal solution cannot be found in real-time, and the ultimate aim is to find a feasible sub-optimal solution, i.e. the partly optimized response that satisfies the constraints.

Besides the fact we require implementations to be real-time feasible, there is also the strong incentive to minimize the computational delays. Remember, the MPC approach calculates the (sub-)optimal control actions based on predictions of system evolution. Long execution times can significantly delay the moment the control action is applied to the system. This issue leads to incorrect predictions and decreased closed-loop performance [2, 3]. For highly complex nonlinear systems, incorrect predictions can even result in complete failure of the controller.

Traditionally, the control and estimation approaches based on dynamic optimization were almost exclusively applied to systems with slow dynamics: e.g. chemical reactors and refineries. Here, we also include our water basin example from the beginning. By slow dynamics we mean that the control periods are larger than one second, and typically in the order of minutes or hours. Recent advances in algorithms *and* computational hardware extended the applicability of MPC and MHE to electro-mechanical, *mechatronics*, applications: robotics, power electronics, control of electrical drives, and aerospace. These systems with fast dynamics require short control periods, much less than one second and typically in the milli- and microsecond range. Here, we list the two most important insights that led to algorithmic advancements in the last two decades and enabled MPC and MHE to be utilized for mechatronics applications.

- The underlying (sub)-problems are highly structured. Exploiting the structure is of paramount importance and yields reduced complexity algorithms [4, 5].
- In the online context, sequences of similar optimization problems are being solved. Therefore, using the solution of the current problem as initial guess for the next one can accelerate the solution search and at the same time reduce the computational delays [6].

Next to improvements in algorithm design, modern computer architectures allowed for great speed-ups of the most time consuming parts of the optimization algorithms, i.e. linear algebra routines.

Production process limitations that implied clock speed stagnation of modern processors – central processing units (CPUs) – forced manufacturers to explore different levels of parallelized processing. On a CPU level, manufacturers implement nowadays multiple computational *cores* in a single CPU. For performance boost of a single core, vendors implement nowadays various instructions for vectorized processing. In dynamic optimization, the routines for matrix-matrix and matrix-vector multiplications as well as solving solving systems of linear equations typically take most of the execution time. These operations are exactly the ones to profit the most from the efficient implementations on modern processors. Furthermore, certain optimization algorithms have inherent properties for parallelization. In a closed-loop setting multi-core CPU configurations allow big portions of the MHE and MPC algorithms to be executed in parallel.

An additional requirement that often appears in mechatronics applications is to embed the computational hardware together with the equipment. Thus, it may be necessary to choose computational hardware with limited resources, e.g. with less superior architecture, lower clock cycle, and/or reduced memory. Reasons for such decisions include (but are not limited to) smaller price tag, power efficiency, increased reliability and size of the embedded system. In the area of optimal control, algorithms are commonly labeled as *embedded* to indicate the intention to either be used for fast mechatronics applications or implemented on computationally limited hardware. Within the scope of this thesis, we use the term embedded to emphasize our intentions to apply MHE and MPC algorithms to mechatronic systems.

One of the approaches found to be efficient for software implementations intended to be used for optimal control of nonlinear mechatronics systems is automatic code generation. The basic idea of code generation is to generate the code tailored to a specific MPC or MHE formulation. One of the few publicly available tools is the ACADO Code Generation Tool [7], that has been developed at the KU Leuven since 2010. Already the first implementation showed promising results and an order of magnitude faster execution times compared to standard solvers.

However, the first implementation offered a limited set of features and no support for MHE formulations. Within this thesis we aim to extend the feature set of the ACADO Code Generation Tool such that it can be applied to a much wider range of applications. The improved implementations and more advanced features should minimize the gap between academia and industry. Eventually, the improved software will demonstrate that the nonlinear MPC and the MHE techniques are ready to be used for complex mechatronics systems.

To further motivate the use of the nonlinear MPC and MHE, we intend to apply the developed software on two challenging real-world applications. The first one is the laboratory scale overhead crane, and the second one is an airborne wind energy system. Both applications will prove the potential and computational effectiveness of the implemented methods.

1.2 Contributions and Overview

Chapter 1	Fundamentals of model based control and estimation are introduced, and the motivation for the research is given.
Chapter 2	In this chapter we introduce the necessary mathematical foundations for fast nonlinear Model Predictive Control (MPC) and Moving Horizon Estimation (MHE). A survey of the currently available techniques is presented. The method of choice, the well-established Real-time Iteration (RTI) [8] scheme is introduced. Building blocks for fast MPC and MHE are identified and discussed.

The chapter is based on the work presented in [9, 10, 11]. In [11], MV contributed with a prototype implementation of a solver for nonlinear MHE. Moreover, MV provided support leading towards successful closed-loop simulations.

Chapter 3 One of the building blocks in nonlinear MPC and MHE is a fast solution of the underlying quadratic program (QP). The chapter contributes with benchmarks presented in § 3.5. We tested NMPC solvers employing four efficient QP solvers on two challenging benchmarks. The first benchmark is related to control of a scalable chain of masses connected by springs. In the second benchmark, task of the solvers is to stabilize the unstable double and triple inverted pendulums. Three methods for solution of QPs arising in optimal control and estimation are tested: active-set methods, interior point methods and the recently developed dual Newton strategy. The active-set method requires pre-processing of the original QP, condensing, to make the approach computationally efficient. We review and analyze three different condensing approaches in details from two perspectives: solutions of the QPs from MPC and MHE. In addition to the benchmarks, two procedures for reduced complexity condensing originally developed for MPC are extended for the MHE.

The results for the chain of masses benchmark confirm the effectiveness of condensing based solvers for short to medium horizon lengths and high ratios of the number of the states to the number of the controls. Sparse solvers showed to be effective for long horizons and the state-to-control ratios. In one of the extreme test cases, we achieved execution times of a sparse NMPC solver lower than 1 ms for a dynamic system with 9 states, 3 controls, and a prediction horizon of 50 steps. For the largest considered dynamic system with 33 states and 3 control, the corresponding MPC problem formulation with 50 steps – comprising 1800 variables – is possible to solve within 50 ms. For the pendulum benchmark, where long prediction horizons were required, it was shown that a structure exploiting NMPC solver can successfully solve a test case with a prediction horizon of 150 steps in just less than 4 ms.

The chapter is partly based on the work presented in [9, 10, 12, 13]. In [12], MV implemented an interface for the qpDUNES QP solver to the ACADO CGT, that eventually led to a new NLP solver for NMPC. In [13], MV contributed with an implementation of the $\mathcal{O}(N^2)$ *condensing routine for NMPC and corresponding benchmarks.*

Chapter 4 Contributed features to the ACADO Code Generation Tool (CGT) are summarized in this chapter. The tool is a module within the open-source software package ACADO Toolkit – a software package for dynamic optimization and control. The tool allows the user to specify nonlinear MHE or MPC formulations using a convenient syntax either in C++, or using MATLAB or Python interfaces. Afterwards, the fully customized solver is exported in the form of optimized C-code. Such a solver can be compiled and deployed to a real-time platform for real-world applications. As the exported code optimization led also to dramatic decrease of code compilation time, the tool found its way to be used for rapid prototyping. The most distinguishable contributed features include:

- full support for multiple-shooting discretization and extensions for parallelization,
- extensions for support of general nonlinear path and point constraints,
- efficient condensing routines,
- support for MHE formulations,
- interfaces for three additional structure exploiting QP solvers.

The chapter is partly based on the work presented in [9, 10, 11]. The ACADO CGT has been jointly developed with Hans Joachim Ferreau, Boris Houska and Rien Quirynen and with generous help from Joel Andersson, Alexander Domahidi, Janick Frasch, and Gianluca Frison.

Chapter 5 This chapter demonstrates an application of nonlinear MPC and MHE algorithms to a mechatronics system with fast dynamics. We use the auto-generated solvers from ACADO CGT that implement the real-time iteration scheme for both the controller and the estimator to control a laboratory scale overhead crane. The experimental setup consists of a cart moving in one dimension and a varying length pendulum attached to it.

Experimental results confirm fast execution times and fast convergence of the RTI scheme. We present the closed-loop performance while employing different scenarios using two generations of the code-generation software. Both scenarios show average execution times close to one millisecond. Using the more advanced software implementations, developed within this thesis, astonishing speed-ups are obtained in the second scenario. In particular, the average execution time of the MPC from the first scenario is nearly equal to the time both the MHE and the MPC need in the second scenario.

This chapter is based on the work presented in two publications, namely [9] and [14]. In [14], MV contributed with formulations for the NMPC and the MHE as well as with extensions to the ACADO CGT. Besides the detailed comparisons shown in the chapter, the following practical contributions by MV made the laboratory experiments possible:

- *modeling, offline simulations, software for real-time control,*
- *hardware and software integration and experiments.*

The experimental work has been done jointly with Frederik Debrouwere, Wannes van Loock, Rien Quirynen, and Keivan Zavari.

Chapter 6 The aim of the second application is to validate the computational performance of the auto-generated MHE and MPC solvers on an experimental setup for rotational start-up of an airborne wind energy system. The system model describes complex nonlinear dynamics comprising 27 differential states, 1 algebraic state and 4 controls. A long prediction horizon and a short control period of 40 ms were necessary to achieve satisfactory performance of the NMPC. The same MPC formulation is solved online by two solvers employing different QP solvers. The first one is employing the structure exploiting interior point QP solver, and the second one utilizes an efficient condensing technique and an active-set QP solver.

The results show reasonably well control performance and exceptional computational performance of the solvers. The interior point QP solver manages to solve the MPC formulation with more than 1500 optimization variables in just less than 5 ms. Moreover, the total feedback time for this particular application is always less than 10 ms.

The introduction of this chapter is partly based on the work presented in [15]. In [15], MV contributed with extensions to the ACADO CGT for fast NMPC and MHE solvers as well as with the hardware and software integration. The results presented in this chapter are submitted in an extended form as the article [16] to a journal. Besides the detailed comparisons shown in the chapter, the following practical contributions by MV made the laboratory experiments possible:

- *development of software for data acquisition,*
- *development of telemetry software,*
- *development of a real-time simulator,*
- *finalization and testing of an MHE and an NMPC,*
- *hardware and software integration and*
- *performing closed-loop experiments.*

The work in the KU Leuven kite laboratory has been done jointly with Hammad Ahmad, Kurt Geebelen, Joris Gillis, Sébastien Gros, Greg Horn, Andrew Wagner, and Mario Zanon.

Chapter 7 The overall conclusions are drawn and directions for follow-up research activities are given.

2

Fast Nonlinear Model Based Predictive Control and Estimation

Model predictive control (MPC) has been originally designed and used for control of large-scale processes, typically in the chemical and petroleum industry. The slow dynamics of those systems allowed long control intervals measured in tens of seconds or even hours, leaving enough time to compute a solution to the underlying optimization problem. During the last decades, the concept of MPC proved to be powerful and has received widespread acceptance in both academia and industry [17, 18, 19, 20, 21, 22].

Progress in the area of optimization algorithms and computational hardware in the last two decades have extended the applicability of numerical optimization on embedded platforms. These developments made MPC suitable for control of fast dynamic systems with time constants in the micro- and millisecond range. As convex quadratic programming solvers became increasingly faster [23, 24, 25], especially *linear* MPC became applicable for fast dynamic systems such as mechatronic devices [26, 27]. In contrast, *nonlinear* MPC (NMPC) – that allows one to apply MPC to nonlinear dynamic systems [28] – has been mainly applied to systems exhibiting slower dynamics so far. This is mainly due to the fact that nonlinear MPC requires more computation power than linear formulations, but also ensuring convergence of iterative methods for solving non-convex optimization problems is more challenging. Overviews of existing algorithms for fast nonlinear MPC algorithms can be found in [6, 29, 30].

Feedback control strategies such as model predictive control (MPC) are typically designed under the assumption that all current process states and parameters are known. However, in most real applications it is either impossible to measure all these quantities directly or at least undesired for economic reasons. The task of recovering the full knowledge of all current process states and possibly some parameters is referred to as state and parameter estimation, respectively.

A *nearly dual problem* [6] to MPC is the Moving Horizon Estimation (MHE) approach [31, 32]. Building on the popularity of MPC, the MHE approach also gained popularity, see e.g. [33]. In comparison to the mostly used techniques for estimation – Kalman filters – this approach is deterministic in the sense that is does not have to assume Gaussian distributions of state and measurement errors. Though, this insight is valuable when tuning the weighting matrices of the least-squares objective function. Moreover, MHE offers an elegant inclusion of constraints and handling of delayed and multi-rate measurements. Other approaches to state and parameter estimation can cope with constraints and delayed/multi-rate measurements as well, but typically with more involved algorithmic techniques. For a general overview of the topic we refer to [19] and for general overview of efficient methods for fast MHE to [6].

In this chapter we summarize the necessary theoretical foundations needed for the application of MHE and MPC to fast mechatronic systems. Starting with § 2.1 we introduce the continuous in time Optimal Control Problem (OCP) formulation relevant within the scope of the thesis. Afterwards, numerical methods for discretization of OCPs are briefly reviewed in § 2.1.1, followed by the presentation of the method of choice, the direct multiple shooting. Discretization of an OCP yields a nonlinear program (NLP). The solution methods for NLPs are outlined in § 2.1.2. The concept of nonlinear MPC is introduced in § 2.2 together with the well established real-time iteration scheme. In § 2.3 we introduce the formulation of our choice and address some of the common issues. The real-time iteration scheme for MHE is presented in § 2.3.2. In § 2.4 we explain the closed-loop setting with MHE and MPC using the RTI scheme. We give details on how the control scheme can be implemented on modern computer architectures. The chapter is concluded with a summary of key building blocks that are necessary for efficient implementations of MHE and MPC.

2.1 Preliminaries

Within this thesis we consider continuous-time dynamic models described by initial condition

$$x(t_0) = x_0, \tag{2.1}$$

an index-1 *differential-algebraic equation* (DAE) [34]

$$0 = f_\mathrm{a}(\dot{x}(t), x(t), z(t), u(t)), \quad \forall t \in \mathcal{T} := [t_0, \infty), \tag{2.2a}$$

$$0 = f_\mathrm{d}(x(t), z(t), u(t)), \quad \forall t \in \mathcal{T}, \tag{2.2b}$$

and an *output function*

$$y(t) = h_y(x(t), u(t)). \tag{2.3}$$

Here, the initial value is $x_0 \in \mathbb{R}^{n_x}$, the differential states are $x : \mathcal{T} \rightarrow \mathbb{R}^{n_x}$, the algebraic states are $z : \mathcal{T} \rightarrow \mathbb{R}^{n_z}$, the controls are $u : \mathcal{T} \rightarrow \mathbb{R}^{n_u}$, and the outputs are $y : \mathcal{T} \rightarrow \mathbb{R}^{n_y}$. As the index-1 DAE is in question, it is assumed that the matrix $\partial g / \partial z$ is invertible. Additionally, it is assumed that the matrix $\partial f / \partial \dot{x}$ is invertible, so that the DAE is of semi-explicit type [8]. For notation convenience, we will refer to the DAE (2.2) in a compact form:

$$0 = f(\dot{x}(t), x(t), z(t), u(t)), \quad \forall t \in \mathcal{T} \tag{2.4}$$

and when necessary explicitly refer to (2.2). In absence of algebraic states, the DAE boils down to an *ordinary differential equation* (ODE)

$$0 = f(\dot{x}(t), x(t), u(t)), \quad \forall t \in \mathcal{T}. \tag{2.5}$$

Possible system parameters $p : \mathcal{T} \rightarrow \mathbb{R}^{n_p}$ can be additionally embedded into (2.2a) as $0 = \dot{p}(t)$, $\forall t \in \mathcal{T}$ with $p(t_0) = p_0$. On the other hand, inclusion of a disturbance model can be achieved by extending the system model with additional disturbance inputs and the corresponding dynamics.

Our intention is to minimize the cost functional of *Bolza* type

$$\int_{t_0}^{t_f} L(x(t), u(t)) \, dt + E(x(t_f)) \tag{2.6}$$

where $L(.)$ is called the *Lagrange term* and $E(.)$ is called the *Mayer term*. The prediction horizon length is defined as $T_c = t_f - t_0$. The cost functional is minimized such that the general *path constraints*

$$\underline{r}(t) \leq r(x(t), u(t)) \leq \overline{r}(t), \quad \forall t \in [t_0, t_f] \tag{2.7}$$

and the *point constraint*

$$\underline{r}_f \leq r_f(x(t_f)) \leq \overline{r}_f \tag{2.8}$$

are satisfied along the trajectory of the system described by the model (2.4).

The particular continuous-time OCP formulation we are interested in reads as follows:

$$\min_{x,\, z,\, u} \quad \frac{1}{2} \int_{t_0}^{t_f} \|h(x(t), u(t)) - \tilde{y}\|^2_{W(t)} \, dt + \|h_f(x(t_f)) - \tilde{y}_f\|^2_{W_f} \tag{2.9a}$$

$$\text{s.t.} \quad x(t_0) = \hat{x}_0, \tag{2.9b}$$

$$0 = f(\dot{x}(t), x(t), z(t), u(t)), \tag{2.9c}$$

$$\underline{x}(t) \leq x(t) \leq \overline{x}(t), \tag{2.9d}$$

$$\underline{u}(t) \leq u(t) \leq \overline{u}(t), \tag{2.9e}$$

$$\underline{r}(t) \leq r(x(t), u(t)) \leq \overline{r}(t), \quad \forall t \in [t_0, t_f], \tag{2.9f}$$

$$\underline{r}_f \leq r_f(x(t_f)) \leq \overline{r}_f. \tag{2.9g}$$

Therein, the Lagrange and the Mayer term are of the *least-squares* type. The *output* functions are defined as h and h_f and $\tilde{y}$ and $\tilde{y}_f$ are called the *references*. This type of the objective is commonly called the *tracking* objective. The positive definite matrices $W(t)$ and W_f are referred to as the *weighting matrices*. Within the *initial constraint* (2.9b) $\hat{x}_0$ denotes the *current state feedback* and (2.9c) is the system dynamics described by the DAE (2.4). From now on, we split the general path constraints (2.7) into two groups: 1) bounds on the states (2.9d) and the controls (2.9e) with appropriate lower and upper bounds and 2) nonlinear path constraints (2.9f). The point constraint, sometimes called the *terminal* constraint, is defined in (2.9g). For the treatment of more general OCP formulations including more general objective formulations and the more complete treatment of DAEs, we refer to e.g. [35, 36, 37].

2.1.1 Methods for Discretization of Optimal Control Problems

Treatment of continuous time OCP formulations on digital computers and application on real systems requires formulation *discretization*. Regarding the discretization process, there exist two big families of methods for solving the continuous OCPs: the *indirect* and the *direct* methods.

The first group of methods, the *indirect* methods (also known as *optimize-then-discretize*), build up on ideas coming from the *dynamic programming* [38] and *Pontryagin's maximum principle* [39]. The first-order necessary conditions for an infinite dimensional OCP yield a boundary value problem (BVP) that needs to be solved. After the solution of the BVP is obtained, the trajectory of optimal controls is discretized. While attractive from a theoretical point of view, the methods are less favorable for practical considerations mainly because of the significant efforts needed to analytically formulate the BVPs and because the BVPs are nontrivial to solve for higher state dimensions. Moreover, constraints are difficult to handle with this group of methods.

The *direct methods*, sometimes referred to as *discretize-then-optimize* methods, first discretize the infinite dimensional OCP into a finite dimensional *nonlinear program* (NLP). The NLP is afterwards solved numerically by an iterative algorithm. Within this group of methods, two approaches can be distinguished: *sequential* and *simultaneous*. The representative of the sequential approach is the *direct single shooting* (DSS) method, where the simulation of the dynamic system (2.9c) on the finite horizon has to be done sequentially. The numerical simulation of the system's model is done using a numerical integrator. While the DSS proposes to integrate the system dynamics from beginning of the horizon t_0 until the end t_f, the *direct multiple shooting* (DMS) approach proposes to split the integration process and introduce more degrees of freedom. While computationally slightly more expensive than DSS, the DMS typically shows a much faster convergence for highly non-linear systems. Moreover, the DMS is a preferred method for solving highly nonlinear and unstable systems where DSS approach occasionally fails. A third approach of the direct methods is called *direct collocation* [40, 41]. Instead of using the numerical integrators for simulation of the system dynamics, this approach proposes to embed this step directly into the NLP. The approach proved to be efficient for solving large-scale optimization problem typically arising in process control.

The shooting methods yield an NLP which has a specific block structure that can be efficiently handled by numerical solvers. Furthermore, the DMS can be efficiently and trivially parallelized. On the other hand, the direct collocation approach typically results in a sparse large-scale NLP that can be efficiently parallelized and solved with sparse linear algebra methods.

To this date, OCPs designed for control of small-scale systems with of up to a couple of dozen of states and few controls are typically solved with shooting methods. The main reason for this is that the structure of the NLPs can be efficiently exploited on the block level, where existing and well optimized dense linear algebra routines are used. Application of the direct collocation approach to small-scale systems is, to our knowledge, an unexplored area. The reason for this seems to be nonexistence of efficient linear algebra routines. For the reasons outlined above, our method of choice for discretization of the continuous time OCP (2.9) is the DMS method.

Direct Multiple Shooting

The discretization process of an infinite dimensional OCP (2.9) starts with a choice of N_c intervals that partition the prediction horizon into sub-intervals $[t_k, t_k + 1]$ defined by $N_\mathrm{c} + 1$ nodes

$$t_0 < t_1 < \ldots < t_{N_\mathrm{c}}, \tag{2.10}$$

where $t_{N_\mathrm{c}} = t_f$. Consequently, the lengths of the intervals are denoted as $\delta_k = t_{k+1} - t_k$, $k = 0, \ldots, N_\mathrm{c} - 1$ and the following equality always holds: $\delta_0 + \ldots + \delta_{N_\mathrm{c}-1} = T_\mathrm{c}$. The discretization intervals are sometimes also called the *shooting intervals* and the nodes t_k are sometimes referred to as the *shooting nodes*. Next to the time partitioning, controls are parametrized. The typical choice is to use the piece-wise constant control parametrization, although more complex parametrizations are possible. This parametrization choice results in N_c control variables u_k, $k = 0, \ldots, N_\mathrm{c} - 1$.

In the context of Bock's direct multiple shooting [42], the state trajectory is computed independently on each sub-interval δ_k. For that purpose, additional variables s_k^x and s_k^z are introduced to serve as initial values:

$$0 = f(\dot{x}_k(t), x_k(t), z_k(t), u_k), \quad x_k(t_k) = s_k^x,\ z_k(t_k) = s_k^z,\ t \in [t_k, t_{k+1}]. \tag{2.11}$$

Furthermore, the *matching constraints* are added to the NLP to ensure continuity of the optimal state trajectory on the whole prediction horizon:

$$s_{k+1}^x = F_k^{\text{int}}(s_k^x, s_k^z, u_k), \quad k = 0, \ldots, N_\text{c} - 1, \tag{2.12}$$

where F_k^{int} represents the solution of the initial value problem (IVP) (2.11) at time $t = t_{k+1}$.

The objective (2.9a) and the constraints (2.9d) - (2.9g) are discretized at the same grid as the differential and algebraic states and controls. The NLP that comes as a result of the multiple shooting discretization of the OCP (2.9) reads:

$$\min_{s_k^x, u} \quad \frac{1}{2} \sum_{k=0}^{N_\text{c}} \|h(s_k^x, u_k) - \tilde{y}_k\|_{W_k}^2 + \|h_{N_\text{c}}(s_{N_\text{c}}^x) - \tilde{y}_{N_\text{c}}\|_{W_{N_\text{c}}}^2 \tag{2.13a}$$

$$\text{s.t.} \quad s_0 = \hat{x}_0 \tag{2.13b}$$

$$s_k^x = F_k^{\text{int}}(s_k^x, s_k^z, u_k) \tag{2.13c}$$

$$\underline{x}_k \leq s_k^x \leq \overline{x}_k \tag{2.13d}$$

$$\underline{u}_k \leq u_k \leq \overline{u}_k \tag{2.13e}$$

$$\underline{r}_k \leq r(s_k^x, u_k) \leq \overline{r}_k \tag{2.13f}$$

$$\underline{r}_{N_\text{c}} \leq r_{N_\text{c}}(s_{N_\text{c}}^x) \leq \overline{r}_{N_\text{c}}. \tag{2.13g}$$

Within the NLP formulation (2.13) the choice is made to evaluate least-squares terms point-wise instead of integrating them on the each interval. This choice is often made to reduce computational burden. It is further motivated by an accurate approximation of the integral with the sum and a fine discretizaton of the prediction horizon, provided the objective function is sufficiently smooth. In practice, even for coarse discretization this approach can provide sufficiently accurate results. A more detailed discussion about efficient integration schemes can be found in e.g. [43]. The second choice is not to explicitly use the additional algebraic variables as optimization variables in the NLP but to make them internal to the integrator. In other words, the numerical integration algorithm is responsible to update s_k^z such that the consistency condition (2.2b) is satisfied at the end of each shooting interval, see [44]. For notational convenience we rename the OCP quantities h_f, W_f, $\underline{r}_f$, r_f, $\overline{r}_f$ to NLP quantities h_{N_c}, W_{N_c}, $\underline{r}_{N_c}$, r_{N_c}, $\overline{r}_{N_c}$.

Soft constraints The NLP constraints (2.13c) – (2.13g) are typically regarded to as *hard constraints*. In other words, they impose *strict* limitations on constrained quantities. The constraints on control inputs are usually reflect real physical limitations, e.g. a limitation of the available current a low-level servo amplifier can provide to an electrical motor. Thus, it makes sense to enforce them strictly. On the other hand, state, path and point constraints typically define design requirements that may be relaxed. As pointed out in [5], it is sometimes desirable to relax those hard constraints to avoid feasibility issues. From practical perspective, it is indeed undesirable having an algorithm declaring infeasibility in online context. For instance, the hard state constraints (2.13d) may be replaced by *soft constraints*

$$\underline{x}_k - \varepsilon_k \leq s_k^x \leq \overline{x}_k + \varepsilon_k, \quad \varepsilon_k \geq 0, \tag{2.14}$$

introducing the extra *slack* variables ε_k in the NLP and appropriately penalizing them in the objective using a combination of ℓ_1- and ℓ_2-norms [5]. Within the scope of this thesis, we treat all constraints as hard, but point out that the soft constraints can be handled with minor modifications to the NLP formulation (2.13). For more details on the topics we refer to e.g. [5].

2.1.2 Nonlinear Programming

Grouping the optimization variables of the NLP (2.13) in a vector

$$X := \left[(s_0^x)', (u_0)', (s_1^x)', \ldots, (s_{N_c}^x)'\right]', \tag{2.15}$$

the following general NLP formulation is obtained:

$$\min_{X} \quad F(X) \tag{2.16a}$$

$$\text{s.t.} \quad G(X) = 0, \tag{2.16b}$$

$$H(X) \leq 0, \tag{2.16c}$$

where $F(X)$ is the objective function, and $G(X)$ and $H(X)$ denote summarized equality and inequality constraints, respectively. Therein, we assume that the functions $F(X), G(X)$ and $H(X)$ are at least twice differentiable.

The optimality conditions for any solution of the NLP are stated in the famous *Karush-Kuhn-Tucker (KKT) conditions of optimality* [45, 46]. In particular, any local solution X^* has to satisfy the following set of conditions:

$$\nabla_X \mathcal{L}(X^*, \lambda^*, \mu^*) = 0, \tag{2.17a}$$

$$G(X^*) = 0, \tag{2.17b}$$

$$H(X^*) \leq 0, \tag{2.17c}$$

$$\mu^* \geq 0, \tag{2.17d}$$

$$(\mu^*)' H(X^*) = 0, \tag{2.17e}$$

where λ^*, μ^* are optimal multipliers. Moreover, the Lagrangian function used in (2.17) is defined as

$$\mathcal{L}(X, \lambda, \mu) := F(X) + G(X)'\lambda + H(X)'\mu. \tag{2.18}$$

In the language of nonlinear optimization, the condition (2.17a) is commonly called the *dual feasibility*, conditions (2.17b) and (2.17c) the *primal feasibility* and the *complementary conditions* are defined in (2.17d) and (2.17e).

There exist two big families of methods for solving nonlinear programs: the sequential quadratic programming (SQP) and the interior point methods (IPM). The key difference between them is how the inequality constraints in the KKT conditions (2.17) are treated. We only give a brief overview and refer for more details to the excellent books [47, 40].

Sequential Quadratic Programming

The SQP approaches propose to solve the NLP iteratively by successive (re)linearizations of the nonlinear functions F, G and H. As a result, at iterate k the following quadratic program is obtained:

$$\min_{\Delta X} \quad \frac{1}{2}\Delta X' A_k \Delta X + \nabla F(X^k)\Delta X \tag{2.19a}$$

$$\text{s.t.} \quad G(X^k) + \nabla G(X^k)'\Delta X = 0 \tag{2.19b}$$

$$H(X^k) + \nabla H(X^k)'\Delta X \leq 0, \tag{2.19c}$$

where ΔX is the *search direction*, X^k is the current linearization point, and A_k is an approximation of the Hessian of the Lagrangian function $\nabla^2_X \mathcal{L}(X^k, \lambda^k, \mu^k)$ at iterate k. Once the QP solved, the next iterate is obtained as $X^{k+1} = X^k + \alpha \Delta X^*$, where the *step length* α is chosen by a suitable *globalization* strategy – see e.g. [47].

If the matrix A_k is exactly the Hessian of the Lagrangian the *exact Hessian SQP* is obtained. The method is locally qudratically convergent and the global solution can be found if the Hessian of Lagrangian is positive semi-definite. Here, however, we are interested in the *Gauss-Newton* Hessian approximation, since the NLP objective (2.13) of our discretized OCP (2.9) is of the least-squares type. In particular, the objective is can be put in the form

$$F(X) = \frac{1}{2}\|R(X)\|_2^2$$

where $R(X)$ is the vector of residuals. If the norm of the residuals is small, the Gauss-Newton approximation to the Hessian of the Lagrangian

$$A_k = \nabla R(X^k)\nabla R(X^k)' \tag{2.20}$$

yields surprisingly good results. This SQP method is commonly referred to as the *generalized (constrained) Gauss-Newton* method. In general, the method exhibits only linear convergence rate, but in many cases shows good contracting properties. In comparison to the Exact Hessian method, the (generalized) Gauss-Newton method does not require computations of second-order derivatives. This fact is naturally attractive for the online optimization. In context of optimal control, this approach has been first used by Bock, see [48].

Monitoring the iteration progress is usually done by assessing outputs of a particular globalization strategy employed and the value of the objective function. In addition, there exists a cheap tool for monitoring the progress of an SQP method, using only the knowledge of the current linearization point X^k and the current multipliers λ^k and μ^k. The measure is called the *KKT tolerance* and is calculated as:

$$\text{KKTTOL} := |\nabla F(X^k)'\Delta X| + \sum_{i=0}^{n_G} |\lambda_i^k G_i(X^k)| + \sum_{i=0}^{n_H} |\mu_i^k H_i(X^k)| \tag{2.21}$$

This KKT tolerance was invented and used by Powell in some of his earliest SQP implementations [43]. The measure has the same unit as the objective function and summarizes possible progress of the objective function and constraint violations. As pointed out as an example in [43], *"KKTTOL $= 10^{-6}$ means that the objective value is likely to be correct to about six digits"*. In absence of a globalization strategy, this measure can serve as a reliable performance indicator.

Interior Point Methods

The idea of interior-point methods is to solve the inequality constrained NLP (2.16) as a sequence of equality constrained NLPs. The inequality constraints are typically eliminated by introducing the *slack variables* τ, such that the following *barrier* problem is obtained

$$\begin{aligned} \min_X \quad & F(X) - \tau \sum_{j=0}^{n_H} \log(-H_j(X)) \\ \text{s.t.} \quad & G(X) = 0. \end{aligned} \tag{2.22}$$

With a suitable initial guess inside the interior defined by the inequality constraints, the sequence of the barrier problems is solved. The corresponding KKT systems are solved by a Newton's methods and decreasing values of τ. The aforementioned approach is the so-called primal barrier method. For more details about interior point methods, we refer to [49, 50].

2.2 Fast Nonlinear Model Predictive Control

The main idea of Model Predictive Control (MPC) is to calculate the control as the solution of the NLP, such as the one defined in (2.13). Only the first control piece from the solution sequence is applied for the time duration of the control period. This approach for calculating controls is exercised at regular time intervals, defined by the control period, see Chapter 1. At each time instant T_i, an OCP is solved on the prediction horizon $[T_i, T_i + T_\mathrm{c}]$ – i.e. $t_0 = T_i, t_f = T_i + T_\mathrm{c}$. Now we can formalize the conventional MPC scheme in the following few steps:

1. Formulate the NLP using the data available at time T_i.
2. Solve the NLP.
3. Send the first control piece from the optimized sequence to the system.
4. Wait until the next time instant T_{i+1}.
5. Return to step 1.

In a typical real-time setting, solving an NLP until convergence criteria are met usually is impossible and is limited by the execution time of the numerical solver and the length of control period. Even if this is possible, the execution time of the algorithm might introduce substantial computational delays that possibly deteriorate the control performance or in extreme cases lead to complete failures of the controller. Moreover, given that sensors and actuator typically have limited resolution, the question arises whether a highly accurate solution is needed at all. Those two observations motivated development of the algorithms that yield sub-optimal solutions in a short time.

From another perspective, efficient algorithms must exploit that the MPC approach solves a sequence of NLPs that are similar. The solution of the NLP at time T_i should be used to obtain an initial guess for the solution of the NLP at the next time instant T_{i+1} with the aim to accelerate convergence and/or improve the quality of the next sub-optimal solution.

Finally, some algorithms aim to split the calculations with the primary aim to reduce the delivery time of the control action to the system. The secondary aim is naturally to decrease the overall execution time of the algorithm.

Several approaches have been proposed for nonlinear MPC to address the issues of long execution times and long feedback delays. Among them are the continuation/GMRES method by [51], the advanced step NMPC controller by [52] and the real-time iteration scheme by [35, 8]. For a detailed survey of algorithms for fast nonlinear MPC we refer to [6].

2.2.1 Real-time Iterations for Nonlinear MPC

The real-time iteration (RTI) algorithm has been originally developed in [35]. The basic strategy is to discretize the optimal control problem (2.9) with a multiple shooting discretization [42] using numerical integration and a piece-wise constant control parameterization. This leads to a structured nonlinear programming problem that can be solved with a sequential quadratic programming (SQP) method. As the objective consists of a least squares tracking term, it is reasonable to employ a Gauss-Newton method to approximate the Hessian matrices [48]. Thus, the algorithm requires only first-order sensitivities of the state trajectory with respect to the inputs.

The main idea of the real-time iterations is to use the control and state variables of the previous optimization run, possibly after a shift, as new linearization point, and perform only one SQP step per sampling time. Consequently, the method produces locally sub-optimal solutions. The ultimate aim of the RTI scheme is 1) to reduce the feedback delay and 2) make short control periods possible. In other words, the optimal control problem (2.9) is only approximately solved in each control period. We can integrate the dynamic system and generate corresponding sensitivities without knowing the actual value of the system state, because the state estimate $\hat{x}_0$ enters the optimization problem only via the affine constraint (2.13b). This part is called the *preparation phase*. As soon as an estimate of the system state is available we only need to solve a single QP. Afterwards, the QP solution corresponding to the control inputs of the first time-interval can be immediately sent to the real process. This phase is called the *feedback phase* and is typically much shorter than the overall SQP iteration. As the initial value enters the problem linearly, it can be shown to deliver a generalized tangential predictor to perturbations, and nominal convergence of the resulting NMPC loop can be proven [53].

For the details of this approach, we refer to [35]. Moreover, a mathematical foundation of the nonlinear real-time iteration scheme as well as stability results can be found in [54, 53, 52, 6].

Since the objective (2.13a) is of least squares form the NLP (2.13) can be solved utilizing the generalized Gauss-Newton method. Using the notation from § 2.1.2, after the QP (2.19) is solved the NLP variables are updated using the full Newton step

$$X^{+} = \bar{X} + \Delta X^{*}, \tag{2.23}$$

where ΔX^{*} denotes the solution of the underlying QP and $\bar{X}$ denotes possibly shifted NLP solution from the previous solver run. More about shifting techniques for the RTI scheme can be found in [55, 56].

Let us look now into the details of the QP resulting from linearization of the NLP (2.13). Using the Gauss-Newton Hessian approximation, linearization of the NLP (2.13) yields the following QP:

$$\begin{aligned}
\min_{\Delta u,\,\Delta s}\quad & \frac{1}{2}\sum_{k=0}^{N_\mathrm{c}-1} \begin{bmatrix}\Delta s_k\\ \Delta u_k\end{bmatrix}' \begin{bmatrix}Q_k & S_k\\ S_k' & R_k\end{bmatrix} \begin{bmatrix}\Delta s_k\\ \Delta u_k\end{bmatrix} + \begin{bmatrix}\Delta s_k\\ \Delta u_k\end{bmatrix}' \begin{bmatrix}g_k^s\\ g_k^u\end{bmatrix} \\
& + \frac{1}{2}\Delta s_{N_\mathrm{c}}' Q_{N_\mathrm{c}} \Delta s_{N_\mathrm{c}} + \Delta s_{N_\mathrm{c}}' g_{N_\mathrm{c}}^s && \text{(2.24a)}\\
\text{s.t.}\quad & \Delta s_0 = \hat{x}_0 - s_0, && \text{(2.24b)}\\
& \Delta s_{k+1} = a_k + A_k \Delta s_k + B_k \Delta u_k,\ k = 0, \dots N_\mathrm{c} - 1, && \text{(2.24c)}\\
& \underline{u}_k - u_k \ \leq\ \Delta u_k \ \leq\ \overline{u}_k - u_k, \quad k = 0, \dots N_\mathrm{c} - 1, && \text{(2.24d)}\\
& \underline{x}_k - s_k \ \leq\ \Delta s_k \ \leq\ \overline{x}_k - s_k, \quad k = 1, \dots N_\mathrm{c}, && \text{(2.24e)}\\
& \underline{m}_k \ \leq\ M_k \Delta s_k + N_k \Delta u_k \ \leq\ \overline{m}_k, \quad k = 0, \dots N_\mathrm{c} - 1, && \text{(2.24f)}\\
& \underline{m}_{N_\mathrm{c}} \ \leq\ M_{N_\mathrm{c}} \Delta s_{N_\mathrm{c}} \ \leq\ \overline{m}_{N_\mathrm{c}}. && \text{(2.24g)}
\end{aligned}$$

Therein, the objective and the linear terms are defined using

$$Q_k = (h_k^x)' W_k h_k^x,\ g_k^s = (h_k^x)' W_k (h(s_k, u_k) - \tilde{y}_k), \tag{2.25}$$

$$Q_{N_\mathrm{c}} = (h_{N_\mathrm{c}}^x)' W_{N_\mathrm{c}} h_{N_\mathrm{c}}^x,\ g_{N_\mathrm{c}}^s = (h_{N_\mathrm{c}}^x)' W_{N_\mathrm{c}} (h(s_{N_\mathrm{c}}) - \tilde{y}_{N_\mathrm{c}}), \tag{2.26}$$

with $h_k^x = \nabla_{s_k} h(s_k, u_k)$, $h_{N_\mathrm{c}}^x = \nabla_{s_{N_\mathrm{c}}} h(s_{N_\mathrm{c}})$, and

$$R_k = (h_k^u)' W_k h_k^u,\ S_k = (h_k^x)' W_k h_k^u,\ g_k^u = (h_k^u)' W_k (h(s_k, u_k) - \tilde{y}_k), \tag{2.27}$$

with $h_k^u = \nabla_{u_k} h(s_k, u_k)$. The linearized matching conditions from the multiple shooting discretization read

$$a_k = F_k^{\mathrm{int}}(s_k, s_k^z, u_k) - s_{k+1},\ A_k = \nabla_{s_k} F_k^{\mathrm{int}}(s_k, s_k^z, u_k),\ B_k = \nabla_{u_k} F_k^{\mathrm{int}}(s_k, s_k^z, u_k), \tag{2.28}$$

where A_k and B_k are called the *sensitivities* of solution to the IVP (2.11). Moreover, the Jacobians of the nonlinear path and point constraints are defined as

$$M_k = \nabla_{s_k} r(s_k, u_k),\ N_k = \nabla_{u_k} r(s_k, u_k),\ M_{N_\mathrm{c}} = \nabla_{s_{N_\mathrm{c}}} r_{N_\mathrm{c}}(s_{N_\mathrm{c}}). \tag{2.29}$$

For notational convenience, we drop the superscript x for differential variables s_k^x. The bounds on the path and point constraints in (2.24f) and (2.24g), respectively, are defined as:

$$\begin{aligned} &\underline{m}_k = \underline{r}_k - r(s_k, u_k), \quad \overline{m}_k = \overline{r}_k - r(s_k, u_k), \quad k = 0, \dots, N_\mathrm{c} - 1, \\ &\underline{m}_{N_\mathrm{c}} = \underline{r}_{N_\mathrm{c}} - r_{N_\mathrm{c}}(s_{N_\mathrm{c}}), \quad \overline{m}_{N_\mathrm{c}} = \overline{r}_{N_\mathrm{c}} - r_{N_\mathrm{c}}(s_{N_\mathrm{c}}). \end{aligned} \tag{2.30}$$

Analyzing (2.24), (2.24a) in particular, it can be easily concluded that all quantities except the linear term g_0^s can be pre-computed in the preparation phase. As soon as the current state estimate $\hat{x}_0$ becomes available, the *initial value embedding* step is executed – the estimate of the current state is embedded into the QP (2.24). The initial value embedding and the QP solution are the key steps of the feedback phase. Once the QP is solved, the control vector on the first shooting interval is updated, $u_0^+ = u_0 + \Delta u_0^*$, and immediately sent to the process.

A *vanilla* solution of the QP is naturally of complexity cubic in number of shooting intervals. Fortunately, the QPs coming as the result of discretization of the optimal control problems have a special structure that, when efficiently exploited, can be reduced to linear complexity in number of shooting intervals.

One popular approach to solve the underlying QP is to apply the *condensing* procedure to reduce the sparse block-structured QP (2.24) to a typically much smaller but dense QP. The condensing procedure is based on the observation that by knowing the first state variable Δs_0 and the control sequence $\Delta u_0, \dots, \Delta u_{N_\mathrm{c}-1}$, the affine map (2.24c) – the state trajectory – can be removed

Algorithm 2.1 Real-time iterations for the nonlinear MPC

```
Initialization: Initialize $s_k, k = 0, \dots, N_c$ and $s^z_k, u_k, k = 0, \dots, N_c - 1$.
Repeat online:
    1 Preparation step
        1.1 Possibly shift the old solution.
        1.2 Solve IVPs (2.11) and generate sensitivities $A_k$ and $B_k$.
        1.3 Evaluate objective.
        1.4 Optional: Condense large block-structured QP (2.24) into a smaller but dense QP (2.31).
        1.5 Wait for a new state feedback $\hat{x}_0$.
    2 Feedback step
        2.1 Use the state feedback to compute the linear term in the (possibly condensed) QP.
        2.2 Optional: Use the state feedback to compute the bounds on affine constraints in the condensed QP.
        2.3 Solve either the sparse QP (2.24) or the condensed QP (2.31).
        2.4 Send the first control, $u^+_0$, to the process.
        2.5 Optional: Recover the optimized state trajectory from the condensed QP solution.
```

from the QP yielding the parametric QP

$$\min_{\xi} \xi' H^{\mathrm{c}} \xi + g^{\mathrm{c}}(\hat{x}_0)' \xi \quad \text{s.t.} \quad \underline{\xi} \le \xi \le \overline{\xi},\ \underline{\xi}_{\mathrm{con}}(\hat{x}_0) \le A_{\mathrm{c}} \xi \le \overline{\xi}_{\mathrm{con}}(\hat{x}_0) \tag{2.31}$$

with the parameter $\hat{x}_0$ and optimization variables $\xi = [\Delta u_0', \dots, \Delta u_{N_c - 1}]'$. In addition to computing the linear term g^s_0 in the feedback step prior to solution of the QP, one also needs to compute the linearized affine constraints $\underline{\xi}_{\mathrm{con}}$ and $\overline{\xi}_{\mathrm{con}}$. The condensed QP can be efficiently solved with a dense linear algebra QP solver. After the solution of the QP is obtained, the variables $\Delta s_k, k = 1, \dots, N_c$ are updated using the affine map (2.24c). This step is commonly called the *expansion* step. We summarize the RTI scheme for nonlinear MPC in Algorithm 2.1.

2.3 Fast Nonlinear Moving Horizon Estimation

A typical choice for linear systems is to use the classical Kalman filter [57, 58] that turned out to be a powerful state estimation algorithm. The Kalman filter is an exact maximum likelihood estimator when the estimation problem is unconstrained and the error distribution of the estimated states is Gaussian.

There exists a great variety of state estimation methods for nonlinear systems. Some of them are extensions of linear state estimators, such as the different variants of the extended Kalman filter (EKF) that linearize the nonlinear system at the current estimate and then apply the usual linear Kalman filter recursion [58, 59]. For efficient update of Kalman-like update routines we refer to [60, 61]. Unscented Kalman filters replace this simple linearization by a more accurate approximation of the multidimensional probability density of the state vector, which is also assumed to be Gaussian [62]. In contrast, particle filter methods allow to approximate arbitrary probability densities that are integrated by means of Monte Carlo sampling [63]. Particle filters need to cover the relevant region of the state space and thus suffer from the *curse of dimensionality*. Excellent surveys on nonlinear state estimation can be found in e.g. [64, 65].

Most of the above approaches with the exception of particle filters make explicit assumptions on the error distribution of the estimated states. Thus, they may loose their theoretical justification if either the actual distribution is different or if estimated quantities are subject to (physical) constraints. This is in contrast to estimating the states by simply fitting them to the measurements in a least-squares fashion. This least-squares estimation goes back to Gauss and is a *deterministic* approach in the sense that it does not rely on any specific error distribution – though this insight is valuable when tuning the weighting matrices of the least-squares objective function.

In order to use all available information when estimating the process states, it is desirable to use all past measurements within the estimation algorithm. However, the computational burden of such full-information estimators might quickly become intractable as the number of measurements grows. Thus, most estimators are based on recursive schemes that only use the most recent measurement explicitly, while information gathered through all earlier measurements is implicitly incorporated within updating rules. An important exception is the optimization-based least-squares estimator. It allows one to deal with the growing amount of measurement data by only considering a *window* comprising a fixed number of past measurements, and possibly incorporating information of previous measurements into some of the least-squares weighting matrices. This variant of least-squares estimation is usually referred to as moving horizon estimation (MHE), see for example [66]. In contrast to Kalman filters and particle filters, handling of missing, delayed or multi-rate measurements is straightforward with the MHE. The question whether MHE is the most suited estimation algorithm for a specific application is beyond the scope of this thesis.

2.3.1 Problem Formulation

As aforementioned, a moving horizon estimator does not consider all past measurements explicitly, but only minimizes the misfit of model prediction and measurements on a fixed estimation horizon of length T_e partitioned into N_e intervals. The intervals are defined by $N_\mathrm{e} + 1$ nodes:

$$t_0 < t_1 < \ldots < t_{N_\mathrm{e}}, \tag{2.32}$$

where $t_0 = T_i - T_\mathrm{e}, t_{N_\mathrm{e}} = T_i$; T_i denotes the current time instant. This can be formulated in form of a constrained, nonlinear least-squares optimization problem. The considered MHE problem formulation in the form of an NLP that is the result of the multiple shooting discretization reads as follows:

$$\begin{aligned}
\min_{s^x,\, w} \quad & \frac{1}{2} \left\{ \|s_0^x - x_\mathrm{ac}\|_{Q_\mathrm{ac}}^2 + \sum_{k=0}^{N_\mathrm{e}-1} \|h(s_k^x, u_k) - \tilde{y}_k\|_{W_k}^2 + \|w_k\|_{Q_w}^2 \right. \\
& \left. \qquad + \|h_{N_\mathrm{e}}(s_{N_\mathrm{e}}^x) - \tilde{y}_{N_\mathrm{e}}\|_{W_{N_\mathrm{e}}}^2 \right\} && \text{(2.33a)} \\
\text{s.t.} \quad & s_{k+1}^x = F_k^\mathrm{int}(s_k^x, s_k^z, u_k, w_k) && \text{(2.33b)} \\
& \underline{r}_k \leq r(s_k^x, u_k, w_k) \leq \overline{r}_k, \quad k = 0, \ldots, N_\mathrm{e} - 1 && \text{(2.33c)} \\
& \underline{r}_{N_\mathrm{e}} \leq r_{N_\mathrm{e}}(s_{N_\mathrm{e}}^x) \leq \overline{r}_{N_\mathrm{e}}. && \text{(2.33d)}
\end{aligned}$$

Therein, $\tilde{y}_k$ are the measurements and the functions h, h_{N_e} are the output functions commonly referred as to the *measurement functions*. Typically those functions include sensor models. In addition to the models of the real system outputs, the measurement functions can also include the *pseudo-measurement* output functions that can help to define a numerically well posed problem. We can notice that in comparison to the MPC formulation the output functions, the IVP solution, and the constraint functions depend on one extra input argument, the *disturbance vector* $w \in \mathbb{R}^{nW}$. In this formulation we do not penalize the inputs variables u_k, although that can be done. The idea to penalize the misfit between the variables u_k and the past inputs can be motivated by the fact that usually the controls computed by e.g. MPC are approximately equal to the ones applied to the real system. The reasons behind this are the finite accuracy and precision of the actuators and/or the fact the connection between the computing hardware and the actuator is realized by analog lines which naturally collect electromagnetic noise. In the formulation at hand, we simply use the past control inputs for simulation purposes.

The optional first term in the least-squares objective is called the *arrival cost* [33] and is used with the purpose to summarize all information prior to the beginning of the estimation horizon. The *a priori* estimate is denoted by x_{ac} and the deviation from the variable s_0^x is penalized by the positive definite matrix Q_{ac} that is the inverse of the *a priori* covariance matrix. As it was the case with the MPC formulation, we define the weighting matrices W_k, Q_w to be symmetric positive definite. For brevity, we summarize all possible bounds and constraints on optimization variables in (2.33c) and (2.33d). In contrast to MPC where the constraints are imposed to reflect both physical limitations and design requirements, in estimation the constraints are used for safety, to ensure the physical limitations and/or validity of models are satisfied.

Weighting matrices Within formulation (2.33), it mostly depends on the choice of the weighting matrices $Q_{\text{ac}}, W_k, Q_w, W_{N_{\text{e}}}$ which estimates are considered optimal. In order to choose them properly, it is common practice to rely on the assumption that the measured quantities exhibit Gaussian distributions. Let us assume that the initial value s_0^x is normally distributed random variable with covariance matrix Σ_0^x and mean values x_0^{est}. Let also the measured outputs $\tilde{y}_k$ and unknown inputs w_k be Gaussian with mean value y_k and 0 as well as covariance matrices Σ_k^y and Σ_k^w, respectively. Then, it is well-known that (2.33) delivers maximum-likelihood estimates for the true trajectories of the current window if (i) Q_{ac} is chosen as $(\Sigma_0^x)^{-1}$, (ii) W_k and Q_w are chosen as $(\Sigma_k^y)^{-1}$ and $(\Sigma_k^w)^{-1}$, respectively, and (iii) no constraints are present. This result has been extended in [32] to the case where constraints on the estimated quantities are present.

Robust formulations The least-squares objective formulation has one important disadvantage. Namely, the presence of outliers, i.e. bad measurements, can significantly deteriorate the estimators performance as the misfits are penalized quadratically. The issue can be simply mitigated by using external logic and declare the measurement missing. However, there exist alternative methods, i.e. objective formulations that can efficiently treat outliers. One popular approach to mitigate the issue is to penalize the misfits with the *Hubber norm*, where small misfits are penalized with ℓ_2 norm and the large ones with ℓ_1 norm [67]. The norm can be efficient embedded into the Gauss-Newton framework using the algorithmic trick described in [67]. For more sophisticated approaches, such as the *M-estimators* we refer to [68, 69].

Missing and delayed measurements One of the key advantages of the MHE approach is that one can handle delayed and missing measurements in an elegant way. The missing measurements are simply handled by setting the appropriate element in the weighting matrix to 0. In statistical sense, this means that one says there infinitely small confidence in the measurement. By suitably choosing the discretization grid, the delayed measurements are simply put on the correct place in the data buffer that is given to the MHE.

Multi-rate sensor fusion Often the system outputs are monitored by the sensors that output data at different rates. For example, in aerospace applications accelerometer and gyroscope data rates are in order of 1 kHz while the data coming from a GPS is outputted at much lower rates, typically less than 50 Hz. It is desirable to fuse all the data to get the most accurate state estimates. This in turn requires more sophisticated MHE formulations involving multiple discretization grids. Consequently, the sensitivities have to be outputted on all those grid points. The *continuous-output MHE* was presented in [70], for which the efficient sensitivity generation schemes were previously studied in [44]. For large-scale applications, we refer to [69, 71] and references therein.

Arrival cost approximation The arrival cost term in the objective function (2.33) summarizes all information gathered through measurements before the beginning of the estimation horizon. The *a priori* estimate x_{ac} is typically taken from the solution of the MHE problem at the previous estimation instant. The arrival cost matrix Q_{ac} can be chosen in different ways: a constant zero matrix has been proposed in [72]. Alternatively, [32] proposed a so-called smoothed EKF-update based on sensitivity information obtained while solving the previous MHE problem, which is also what our MHE algorithm uses. It has been shown that the classical EKF is equivalent to MHE using smoothed EKF-updates and a horizon of length one [32]. A similar approach that was found to be suitable for real-time applications is presented in [73]. Approximation of the of the *full information estimator* has been a topic of extensive research during the past decades. Different approaches have been proposed based different approaches to approximate the arrival cost using either EKF, UKF or particle filters. For an extensive survey on the topic we refer to [19].

2.3.2 Real-time iterations for Nonlinear MHE

Building on the same ideas for use in nonlinear MPC, the RTI scheme has been recently adapted for solving nonlinear MHE problems [74, 73]. The direct multiple shooting technique is used for discretization of the continuous-in-time counterpart to the NLP (2.33), and the generalized Gauss-Newton method is applied for the NLP solution. Proof of convergence of MHE based on the RTI scheme is presented in [75].

The NLP (2.33) possesses a structure similar to the one arising in MPC. Comparing the MHE NLP with the MPC one (2.13) it can be easily concluded that the only structural difference is that in the MHE formulation the initial condition does not exist, as the initial state is free. Stating the obvious, the first three terms in the MHE objective can be trivially put in the form of the first term of the MPC objective (2.13).

In the RTI for MHE all expensive calculations involving sensitivity generation and possibly the condensing part involving the expensive computation of the condensed Hessian, can be done without knowledge of the current measurement. As soon as the measurement is available, it is embedded in the underlying QP. The current state estimate is sent to the controller as soon as the QP solver finishes execution.

The QP resulting from the Gauss-Newton linearization of the NLP has almost the same structure as the one in the MPC (2.24) with the exception of absence of the initial condition (2.24b). The special block-structure of the QP can naturally be exploited, yielding a reduced complexity solution in the number of the shooting intervals. Alternatively, one can opt to condense the sparse QP and solve a condensed one, similar to (2.31). The differences in the condensed

Algorithm 2.2 Real-time iterations for the nonlinear MHE

```
Initialization: Initialize $s_k, k = 0, \dots, N_e$ and $s_k^z, u_k, k = 0, \dots, N_e - 1$. Optional initialize the arrival-cost update routine.
Repeat online:
    1 Preparation step
        1.1 Optional: Update $x_{ac}, Q_{ac}$ using an arrival-cost update routine.
        1.2 Possibly shift the old solution.
        1.3 Solve IVPs (2.33b) and generate sensitivities $A_k$ and $B_k$.
        1.4 Evaluate objective.
        1.5 Optional: Condense block-structured QP into a smaller but dense QP.
        1.6 Wait for a new measurement $\tilde{y}_{N_e}$.
    2 Estimation/Feedback step
        2.1 Use the new measurement to compute the linear term in the (possibly condensed) QP.
        2.2 Solve either the sparse QP or the condensed QP.
        2.3 Optional: Recover the optimized state trajectory from the condensed QP solution.
        2.4 Send the current state estimate control, $\hat{x}_0 := s_{N_e}^+$, to the controller.
```

QP are that only the linear term depends now parametrically on the current measurement and that the vector of optimization variables contains the initial state s_0, as it is unconstrained. The latter one makes the condensing approach less favorable for practical applications where the ratio $(n_w N_e)/n_x$ is small.

The main steps of the nonlinear RTI scheme for MHE are summarized in Algorithm 2.2. In comparison to the RTI scheme for MPC, see Algorithm 2.1, we can see that in the MHE there is the optional arrival cost update that happens before the possible shift of the old solution. Furthermore, if a condensing based solution is aimed, one needs to perform the expansion step to calculate the current state estimate $s_{N_e}^+$.

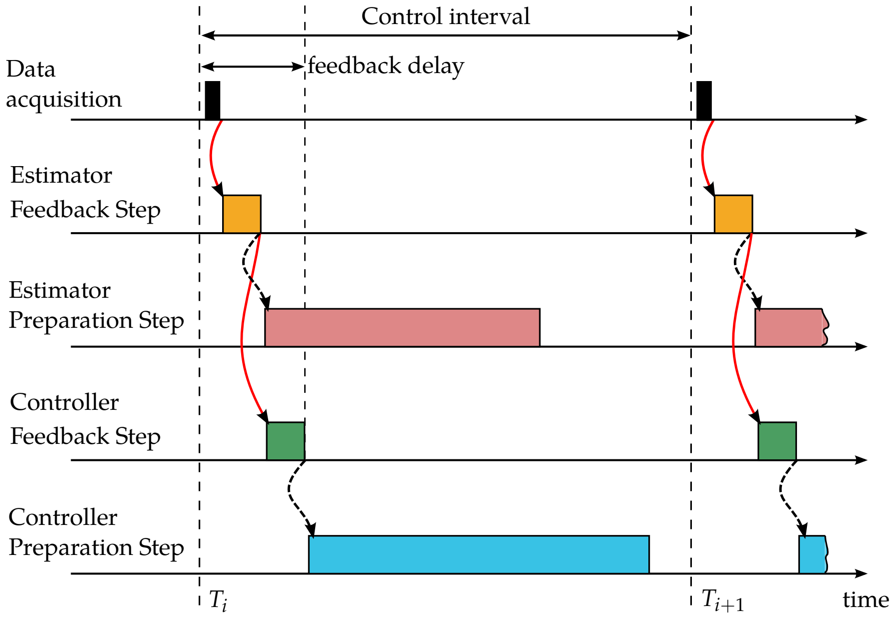


Figure 2.1: Division into preparation and feedback step.

2.4 The Big Picture: Real-time Iterations in Closed Loop with Nonlinear MPC and MHE

In a real-application setting, the control period typically starts with gathering of data from the sensors, the *data acquisition* (DAQ) task, see Figure 2.1.

Using the RTI scheme for solving the MHE NLP, as soon as the DAQ step is finished, the current measurement vector $\tilde{y}_{N_\mathrm{e}}$ is embedded into the underlying QP and the QP is solved. This step is called *measurement embedding* within the feedback step of the RTI scheme. The product of the MHE feedback step is the current state estimate $\hat{x}_0 = s^+_{N_\mathrm{e}}$.

MHE feedback step completion triggers the MPC feedback step. The current state estimate $\hat{x}_0$ gets embedded into the underlying QP. The solution of the QP yields the optimized control sequence from which the first control u^*_0 is taken and sent to the system. In this control architecture, the feedback delay lasts from the moment the DAQ step is started until the optimized control action is sent to the process. The delay is primarily defined by the time spent in solving the two QPs, but it also includes the time needed to embed the measurement and the current state estimate to corresponding QPs. Moreover, it can possibly include the time spent in condensing the QPs and expanding the solution of the condensed QP.

After each feedback step, the corresponding preparation step is triggered. In each preparation step necessary sensitivity information is generated and possibly the big portion of the condensing procedure is performed. Often, a preparation step takes much more than a corresponding feedback step. As illustrated in Figure 2.1, the preparation steps can be executed fully in parallel – as opposed to the feedback steps that need to be executed in the specific order. In the simplest setting, the one illustrated here, one needs at least a processor with two cores. We say at least to emphasize that in a real-world application next to the control and estimation tasks, the computational hardware has to perform a set of additional tasks, e.g. communication with external hardware, logging, and telemetry.

The preparation and the feedback steps can be further parallelized in the context of the RTI scheme. The feedback time can be reduced by parallelizing the solution of the underlying QPs. QP algorithms that can exploit sparse structure of the QPs arising in optimal control can be efficiently parallelized, see e.g. [76, 56]. The preparation step can be efficiently parallelized as the direct multiple shooting has the inherent property to be parallelized in the *shared memory* fashion – one integration routine is applied to multiple data.

2.5 Building Blocks for Fast Nonlinear MPC and MHE

The focus of this thesis are applications of the well-established RTI scheme. In a single iteration of the RTI scheme, the NLP is linearized with direct multiple shooting technique resulting in the block structured QP. The linearization involves evaluation of nonlinear functions, solving the initial value problems, as well as generation of sensitivity information. The QP can be solved either in sparse form employing an efficient structure exploiting QP solver, or pre-processed using the condensing technique and solved afterwards with a dense linear algebra QP solver.

Evaluation of nonlinear functions and algorithmic differentiation The nonlinear models we are interested in come from the first principles and are supplied in symbolic form. The linearization involves evaluation of nonlinear functions such as the objective functions, nonlinear constraints and right-hand sides of the model equations. In addition, the computation of the Jacobians is required. Instead of evaluating the Jacobians numerically by means of expensive and usually inaccurate finite differences [47], the *algorithmic differentiation* approach [77] is employed to calculate the derivative information exactly to machine precision. We refer to [78] for details on efficient implementations for automatic differentiation.

Numerical integration Typically the most computationally intensive numerical task within the direct multiple shooting linearization is the solution of the IVP (2.11), see e.g. [79, 34]. The solution of the IVP is commonly referred to as *integration*, and to the corresponding algorithm as *integrator*. In general, the integration process is an iterative process, where the integration algorithm solves the IVP to desired accuracy. To achieve the desired accuracy, the algorithms typically partition the shooting interval into arbitrary many integrator steps (intervals). In a real-time setting we are considering, the one where short control periods are required, solving the IVPs to high accuracy is usually impossible and typically unnecessary. Consequently, it is desirable to choose and fix the number of integration steps during the design phase of the controller and/or the estimator. For non-stiff models in form of explicit ODEs, $\dot{x} = f(x, u)$, explicit Runge-Kutta integrators typically produce results of sufficient quality. For stiff models and the models described by implicit ODEs or DAEs implicit integrators are necessary to solve the IVPs reliably. Moreover, for stiff explicit ODEs the implicit integrators often produce as good results as the explicit integrators but with much less integration steps per shooting interval. The implicit integration routines are iterative procedures and in real-time applications it is usually required to limit or fix the number of iterations to ensure deterministic execution. For an extensive discussion on solutions of IVPs and generation of corresponding sensitivity information for milli- and micro-second applications we refer to [44, 80].

Solution of the underlying quadratic program The crucial step to reduce the feedback delay within the feedback phase of the RTI scheme is to employ efficient structure exploiting QP solver. This is the topic of the next chapter.

3

Tailored Quadratic Programming Solvers for Short and Long Horizons

One of the key components for the fast solution of nonlinear MPC and MHE is a fast solver for the underlying linearized problem. In context of the real-time iteration (RTI) scheme based on SQP described in Chapter 2 we are interested in a fast solution of the underlying quadratic program (QP). The time spent solving the QP of the linearized NLP from MHE formulation almost entirely defines the time needed to deliver the state estimate to the controller. Likewise, roughly all time needed to deliver the control action to the process, from the moment state estimate is available, is time spent solving the QP with an MPC solver. In a closed-loop setting with MHE and MPC the *total feedback time*, the time period defined from the moment all measurements are available to the moment MPC produces the control action, is almost entirely spent in solving the sequence of the two QPs.

As we saw in Chapter 2, the QP coming from linearization of the NLP corresponding to an MHE formulation is of similar structure as the one coming from an MPC formulation. The only difference, structure wise, is that the initial state is free. The QP structure is preserved with addition of the arrival cost to MHE formulation. Therefore, in the text to come we will often refer to

OCPs/NLPs with fixed and free initial states, referring implicitly to MPC and MHE formulations, respectively.

Due to a specific structure of the QP, it is of paramount importance to exploit that structure. Exploiting the structure not only leads to fast and efficient algorithms, but also opens the doors for fast software implementations. In particular, the QP data can be organized in specific data structures such that modern CPUs can run the code much faster than if the data was packed in a generic way.

The QP data consists, in general, of structured sparse matrices whose dimensions are defined by dynamic model dimensions, horizon length and discretization method applied to the original OCP. Within the scope of this thesis we are interested in shooting discretization techniques. For ease of presentation, we repeat the QP formulation (2.24) arising after linearization of the MPC related NLP (2.13):

$$\begin{aligned}
\min_{\Delta u,\ \Delta s} \quad & \frac{1}{2}\sum_{k=0}^{N-1} \begin{bmatrix}\Delta s_k\\ \Delta u_k\end{bmatrix}' \underbrace{\begin{bmatrix}Q_k & S_k\\ S_k' & R_k\end{bmatrix}}_{H_k} \begin{bmatrix}\Delta s_k\\ \Delta u_k\end{bmatrix} + \begin{bmatrix}\Delta s_k\\ \Delta u_k\end{bmatrix}' \begin{bmatrix}g_k^s\\ g_k^u\end{bmatrix} \\
& + \frac{1}{2}\Delta s_N' Q_N \Delta s_N + \Delta s_N' g_N^s && \text{(3.1a)}\\
\text{s.t.} \quad & \Delta s_0 = \hat{x}_0 - s_0, && \text{(3.1b)}\\
& \Delta s_{k+1} = a_k + A_k \Delta s_k + B_k \Delta u_k,\ k = 0, \ldots N-1, && \text{(3.1c)}\\
& \underline{u}_k - u_k \ \le\ \Delta u_k \ \le\ \overline{u}_k - u_k, \quad k = 0, \ldots N-1, && \text{(3.1d)}\\
& \underline{x}_k - s_k \ \le\ \Delta s_k \ \le\ \overline{x}_k - s_k, \quad k = 1, \ldots N, && \text{(3.1e)}\\
& \underline{m}_k \ \le\ M_k \Delta s_k + N_k \Delta u_k \ \le\ \overline{m}_k, \quad k = 0, \ldots N-1, && \text{(3.1f)}\\
& \underline{m}_N \ \le\ M_N \Delta s_N \ \le\ \overline{m}_N. && \text{(3.1g)}
\end{aligned}$$

Therein, we assume that the symmetric matrices Q_k and R_k satisfy: $Q_k \succeq 0$, $R_k \succ 0$. Furthermore, we assume that the matrix of active constraints formed from (3.1b) - (3.1f) has a full row rank. For the strictly convex QP under consideration, the full row rank of the active constraints matrix[1] ensures a

---

[1] This condition is called *linear independence constraint qualification* (LICQ).

unique primal-dual solution; see e.g. [47]. The current linearization point is defined with $u_0, \ldots, u_{N-1}$ and $s_0, \ldots, s_N$. A similar QP is obtained after linearization of the MHE related NLP. The only structural difference is that the initial condition (3.1b) is nonexistent.

At this moment we do not make any special assumptions about matrices in (3.1). Later in the text, special cases that lead to non-negligible computational savings will be mentioned. For example, it is common that blocks Q_k and R_k are diagonal and blocks S_k are zero matrices. For brevity, in the rest of the chapter we use N as the number of control intervals, as opposed to using the number of prediction N_c or estimation N_e intervals directly.

In essence there are two ways to solve the QP (3.1). One way is to keep the sparse structure of the problem and apply an efficient structure exploiting algorithm, e.g. [5, 81, 82]. As it is going to be shown later, sparse solvers possess linear complexity in number of control intervals N. However, all sparse solvers suffer from cubic complexity in number of states and controls. Alternatively, one can employ a procedure called *condensing* [42] and reduce number of optimization variables at the expense of cubic complexity in number of control intervals. Afterwards, the condensed QP is passed to a dense linear algebra QP algorithm, e.g. [24]. As the solution of the dense QP is of cubic complexity in the number of intervals N, the reduction of variables using the condensing procedures is of utmost importance. For moderate horizon lengths and high enough ratios of n_x / n_u the combination of the condensing procedure and the dense solution of the condensed QPs is shown to be computationally efficient.

In this chapter we review and compare three different approaches that proved to be efficient in context of embedded MPC and MHE. For each of them we present dominant computational cost factors and the situations when they might be favorable. In § 3.1 active-set algorithms are presented. In § 3.2 we review and extend three different methods for condensing of the sparse QP. For each condensing method we study the two cases, specifically related to QPs coming from MHE and MPC formulations. Structure exploiting QP solvers based on interior point methods are described in § 3.3 and the recently developed dual Newton strategy is the topic of § 3.4. Four different software implementations of the nonlinear MPC controllers are tested on two challenging benchmarks. The benchmark setups and the results are presented in § 3.5. Finally, conclusions are drawn in § 3.6.

3.1 Active-set Quadratic Programming Solvers

In the context of this chapter, we are interested in solving a strictly convex QP [47] written in a general form

$$\min_{y} \quad \frac{1}{2}y'Hy + g'y \tag{3.2a}$$

$$\text{s.t.} \quad Gy \geq b, \tag{3.2b}$$

with the vector of the optimization variables $y \in \mathbb{R}^{n_y}$ and the symmetric positive definite Hessian matrix $H \succ 0$. In comparison with (3.9), the simple bounds and the affine constraints are lumped in (3.2b) with the constraint matrix $G \in \mathbb{R}^{n_\mathrm{c} \times n_y}$. Next, we define $G_{\mathbb{A}}$ as a matrix consisting of rows of the matrix G where the indices of the rows are indicated by a *working set* $\mathbb{A} \subseteq \{1, 2, \ldots, n_\mathrm{c}\}$. Accordingly, we define the part of the constraint vector as $b_{\mathbb{A}}$. In solution y^*, the working set is a linearly independent subset of the *active set* $\mathbb{A} = \mathbb{A}(y^*)$ – the set of active constraints. Moreover, the working set in the solution spans the same subspace as the active set. If the active set is known, the solution y^* of the strictly convex QP (3.2) is unique and is equal to the solution of the equality constrained QP. In general, when the active set is unknown *a priori*, an active-set solver involves an iterative process. Within that iterative process, the solver typically changes the working set by one constraint at the time to correctly estimate the active set.

The active-set methods can be roughly classified in three categories: primal, dual and parametric.

Primal active-set methods (see e.g. [47]) yield iterations that are always primal feasible, i.e. constraints (3.2b) are always satisfied. In the online context where a solver might need to be prematurely stopped – such that the real-time constraints are satisfied – the solver still provides a feasible solution. A disadvantage, on the other hand, of the primal methods is that it requires a feasible initial guess. If not provided, the so called *Phase I* procedure has to be performed prior to solving the actual QP [83]. The solution time of the *Phase I* procedure can take as much as solution time of the actual QP.

In contrast to the primal methods, the dual active-set methods do not require feasible initial guess. The dual feasibility is maintained, and the primal feasibility is satisfied only at the last iterate when the solution is found. From the implementation point of view, this approach is equivalent to solving the dual QP to (3.2) with a primal active set solver [83, 84].

The parametric active-set methods aim to exploit that e.g. in MPC a sequence of similar QPs is solved in a sequence. One such method designed for fast MPC applications is the *online active set strategy* [24]. The method maintains both primal and dual feasibility on a homotopy path from a solution of an old QP to the new one. Advantages of the method are that the method does not require the *Phase I* procedure and that the iterative process can be safely interrupted.

The active-set methods possess good *warm-starting* properties. In particular, having an accurate initial estimate of the active set, the active-set methods show typically low numbers of iterations. In a general case, the active-set methods need a lot of iterations to converge. Fortunately, each iteration of an active-set method is relatively cheap. For more detailed discussion about complexity factors of this family of QP solvers we refer to an excellent survey [85]. In the following we are going to discuss the most dominant speed factors of the active-set solvers in context of nonlinear MPC and MHE.

For brevity, in the rest of the section we base the presentation on primal active-set methods. For the extension to other approaches see e.g. [84]. On the lower level, the solution of a QP boils down to the solution of an equality constrained QP. At iterate k, the next iterate is computed as $y_{k+1} = y_k + \tau_k \Delta y_k$, where τ_k is the step length. The step *direction* Δy_k can be seen as a solution of a convex QP

$$\min_{\Delta y_k} \quad \frac{1}{2}\Delta y_k' H \Delta y_k + \Delta y_k' (H y_k + g) \tag{3.3a}$$

$$\text{s.t.} \quad G_{\mathbb{A}} \Delta y_k = 0, \tag{3.3b}$$

where $\mathbb{A}$ is the current working set and $G_{\mathbb{A}}$ has a full row rank.

Among a number of methods to solve the equality constrained QPs [47], we restrict the presentation to the null space approach and briefly compare it to the range space approach.

Defining a *null space basis matrix* Z whose columns form the null space of the matrix $G_{\mathbb{A}}$, i.e. $G_{\mathbb{A}} Z = 0$, every feasible point can be written as

$$\Delta y_k = Z \Delta y_{Z,k}, \ \Delta y_{Z,k} \in \mathbb{R}^{n_y - n_{\mathbb{A}}}. \tag{3.4}$$

Substituting (3.4) into (3.3), the following unconstrained convex QP is obtained

$$\min_{\Delta y_{Z,k}} \frac{1}{2}\Delta y'_{Z,k} \underbrace{\left(Z'HZ\right)}_{H_{\mathrm{R}}} \Delta y_{Z,k} + \Delta y'_{Z,k} \left(Z'(Hy_k + g)\right). \tag{3.5}$$

Now, the solution of this QP is straight-forward. Typically, the Cholesky decomposition is applied to the *reduced Hessian* H_{R} and the solution is calculated with forward and backward substitutions. Accordingly, the reduced Hessian is required to be positive definite. An appealing feature of the null space approach is that the conditioning of the reduced Hessian H_{R} is not worse than the conditioning of the original Hessian H.

The most expensive parts of the approach are calculation of the null space basis matrix which is typically done with a variant of the QR decomposition and factorization of the reduced Hessian. In addition, prior to factorization one also needs to form the reduced Hessian. Building of the reduced Hessian and its factorization are $\mathcal{O}(n_y^3)$ operations. It must be also taken into account that throughout the iterative process inside the solver, the dimensions of matrices $G_{\mathbb{A}}$, Z and H_{R} change. The more the active constraints, the smaller the dimension of the reduced Hessian is. Instead of factorizing the reduced Hessian and calculating the matrix Z at each iteration from scratch, efficient implementations rely on the fast $\mathcal{O}(n_y^2)$ routines for update of the Cholesky and QR factorizations [86]. This means that initially the expensive Cholesky and QR factorizations indeed have to be used, but later on the fast update routines are applied [87]. Initial factorization can be accelerated using the multiplier information from the previous QP if a sequence of similar QPs is solved. This fact can be exploited in the area of dynamic optimization.

In a nutshell, range space methods project the Hessian onto the *range space* of the active constraints [47, 84]. The most expensive part is the inversion of the projected Hessian $G'_{\mathbb{A}}H^{-1}G_{\mathbb{A}}$ which also involves explicit inversion of the original Hessian H. The approach is attractive when the number of active constraints is low and when H is easily invertible. A disadvantage of the method is the that it is sensitive to conditioning of the active constraint matrix and the Hessian. Efficient update procedures for range space methods exist and can be found e.g. in [88].

The MPC user might be interested in linear algebra routines for the efficient exploitation of the sparse QP structure arising therein [89]. Such update routines are developed in [87]. However, to our knowledge, no fast implementations of those routines exist. Instead, we opt to use in our benchmarks and applications an open-source implementation of the online active-set method qpOASES [90, 84] that employs null-space approach and dense linear algebra at lower levels assuming no particular structure of the QP matrices H and G. An alternative fast implementation is a dual active-set method solver QPC [91].

As briefly stated in the beginning of the chapter, the block-structured QP coming from discretization of an OCP is typically *condensed* before fed to a dense linear algebra active-set QP solver. The condensing procedures produce the dense version of the same QP, reducing the dimensions of the QP at the same time. The QP solver internally does the factorization of the reduced Hessian. However, even when the superior $\mathcal{O}(n_y^2)$ routines are used (see § 3.2.2), the overall complexity of the NMPC (or MHE) solver is still cubic in number of control intervals because of the $\mathcal{O}(n_y^3)$ factorization of the reduced Hessian in the QP solver.

3.2 Condensing Procedures

The goal of the condensing procedure [42] is to reduce the number of optimization variables in the QP (3.1). The system of linearized dynamic equations (3.1c) can be regarded as an affine time-varying dynamic system with steps Δs_k. Depending on the particular formulation of an OCP we can distinguish between two condensing approaches.

Full condensing In the NMPC, the initial state Δs_0 is fixed and assuming the variables $\Delta u_0, \dots, \Delta u_{N-1}$ are known, the variables $\Delta s_1, \dots, \Delta s_N$ can be obtained by a forward simulation. Therefore, we can define a vector of independent variables Δw_{ind} and a vector of dependent variables Δw_{dep}:

$$\Delta w_{\text{ind}} = [\Delta u_0', \dots, \Delta u_{N-1}']', \tag{3.6}$$

$$\Delta w_{\text{dep}} = [\Delta s_1', \dots, \Delta s_N']'. \tag{3.7}$$

Note that here Δs_0 gets eliminated through the condensing procedure as well.

Partial condensing A QP that comes as a result of discretization of an MHE problem, Δs_0 is considered to be independent variable as well. The vector of

Table 3.1: Number of optimization variables for the sparse and the condensed QP.

	Sparse QP	Condensed QP
MPC	$N(n_u + n_x)$	Nn_u
MHE	$N(n_u + n_x) + n_x$	$Nn_u + n_x$

independent variables now reads:

$$\Delta w_{\text{ind}} = [\Delta s_0', \Delta u_0', \dots, \Delta u_{N-1}']'. \tag{3.8}$$

The vector of the dependent variables remains the same as in (3.7). This approach can be applied to NMPC formulations as well, at the expense of more costly procedure and memory consumption.

Applying the condensing procedure to the original sparse QP (3.1), an equivalent, condensed, QP is obtained as:

$$\min_{\Delta w_{\text{ind}}} \quad \frac{1}{2}\Delta w_{\text{ind}}' H_{\text{c}} \Delta w_{\text{ind}} + \Delta w_{\text{ind}}' g_{\text{c}} \tag{3.9a}$$

$$\text{s.t.} \quad \Delta \underline{w}_{\text{b}} \;\leq\; \Delta w_{\text{ind}} \;\leq\; \Delta \overline{w}_{\text{b}}, \tag{3.9b}$$

$$\Delta \underline{w}_{\text{c}} \;\leq\; A_{\text{c}} \Delta w_{\text{ind}} \;\leq\; \Delta \overline{w}_{\text{c}}. \tag{3.9c}$$

In the full condensing, the condensed Hessian is denoted as H_{c} and the condensed linear term as g_{c}. The partial condensing equivalent counterparts will be referred to as H^{cp} and g^{cp}. These and other items in (3.9) are dependent on the sparse QP problem definition and the applied condensing algorithm, thus will be properly defined in the following sub-sections.

The number of optimization variables for the sparse (3.1) and for the dense QP (3.9) formulation is summarized in Table 3.1.

It can be noted here that the structure of the original sparse QP is almost lost during the condensing procedure. In particular, the condensed Hessian H^{c} is dense now. The typical choice at this point is to employ a general dense linear algebra QP solver to solve the QP (3.9). An efficient active-set algorithm can be favored in this situation, mainly because of superior warm-starting features in comparison to interior-point methods.

In MPC, the state estimate $\hat{x}_0$ enters linearly the sparse QP (3.1) via the initial constraint (3.1b). This characteristic enables certain relocation of computations. In particular, only the linear term g^{c} and the constraints $\Delta\underline{w}_{\mathrm{c}}$, $\Delta\overline{w}_{\mathrm{c}}$ depend on the state feedback $\hat{x}_0$. Effectively this means that those three quantities have to be calculated in the feedback step of the RTI scheme. The most costly part, building of the condensed Hessian, can be moved to the preparation phase. In the MHE, the linear term g^{cp} is the only one that needs to be calculated in the feedback phase, prior to triggering the QP solver.

The *classical condensing* [42] is of $\mathcal{O}(N^3)$ complexity and can be optimized using the block structure information from the sparse QP [43]. A comparison with off-the-shelve sparse solvers is presented in [92]. In comparison to a *vanilla* implementation, the optimized one can achieve order of magnitude speed-ups. The first indication of a reduced complexity condensing algorithm has been reported in [93]. Based on this indication, the reduced complexity $\mathcal{O}(N^2)$ implementation of the full condensing routine was developed in [94]. The same procedure was later found and presented independently in [78]. The extension for the partial condensing is developed and will be presented in § 3.2.2. In [95] the authors propose a procedure for factorization of the fully condensed Hessian H^{c} in $\mathcal{O}(N^2)$ time. Moreover, the paper proposes an efficient solution to the KKT system based on the reduced complexity, $\mathcal{O}(N^2)$, factorization. In § 3.2.3 we extend the ideas presented in [95] to the partial condensing.

Within the context of this thesis we are interested in cases where the number of the states n_x is typically much higher than number of inputs n_u. The extreme case is the one with $n_x = n_u$ coming from an MHE formulation with the full process noise in the system dynamics. For the cases where $n_x < n_u$, the *complementary condensing scheme* is developed in [87].

3.2.1 Classical Condensing

Full Condensing

The affine map (3.1c) to be removed can be defined as follows:

$$\Delta w_{\text{dep}} = d + C\Delta s_0 + E\Delta w_{\text{ind}}. \tag{3.10}$$

Here, matrices $C \in \mathbb{R}^{(N\cdot n_x)\times n_x}$ and $E \in \mathbb{R}^{(N\cdot n_x)\times(N\cdot n_u)}$ have a special block structure:

$$C = \begin{bmatrix} C_0 \\ C_1 \\ \cdots \\ C_{N-1} \end{bmatrix}, \quad E = \begin{bmatrix} E_{0,0} & & & \\ E_{0,1} & E_{1,1} & & \\ \vdots & \vdots & \ddots & \\ E_{0,N-1} & & \cdots & E_{N-1,N-1} \end{bmatrix}, \tag{3.11}$$

where the corresponding blocks are computed as:

$$\begin{aligned} &C_k = A_k \cdot \ldots \cdot A_0, && k = 0,\ldots,N-1, \\ &E_{k,m} = A_k E_{k-1,m}, && k = 1,\ldots,N-1,\ m = 0,\ldots,k-1, \\ &E_{k,k} = B_k, && k = 0,\ldots,N-1. \end{aligned} \tag{3.12}$$

Vector d can be computed by a forward simulation of the affine dynamic system (3.9) with $\Delta s_0 = 0$ and $\Delta w_{\text{ind}} = 0$. This yields a recursion:

$$d_0 = a_0, \quad d_k = a_k + A_k d_{k-1}, \quad k = 1,\ldots,N-1. \tag{3.13}$$

The Hessian and the linear term of the condensed QP are obtained as:

$$H^{\text{c}} = \overline{R} + E'\overline{Q}E + E'\overline{S} + \overline{S}'E, \tag{3.14a}$$

$$g^{\text{c}} = g^u + E'(\overline{Q}(d + C\Delta s_0) + g^s) + \overline{S}'(d + C\Delta s_0) + \begin{bmatrix} S_0' \\ 0_{(N-1)\cdot n_u \times n_x} \end{bmatrix} \Delta s_0 \tag{3.14b}$$

where $\overline{Q} \in \mathbb{R}^{(N\cdot n_x)\times(N\cdot n_x)}$ and $\overline{R} \in \mathbb{R}^{(N\cdot n_u)\times(N\cdot n_u)}$ are block diagonal matrices:

$$\overline{Q} = \text{diag}(Q_1,\ldots,Q_N), \quad \overline{R} = \text{diag}(R_0,\ldots,R_{N-1}) \tag{3.15}$$

and

$$\overline{S} = \begin{bmatrix} 0_{(N-1)\cdot n_x \times n_u} & \operatorname{diag}(S_1, \ldots, S_{N-1}) \\ 0_{n_x \times n_u} & 0_{n_x \times (N-1)\cdot n_u} \end{bmatrix}. \tag{3.16}$$

The items in (3.9b) are defined as:

$$\Delta \underline{w}_{\mathrm{b}} = \underline{w}_{\mathrm{ind}} - w_{\mathrm{ind}}, \quad \Delta \overline{w}_{\mathrm{b}} = \overline{w}_{\mathrm{ind}} - w_{\mathrm{ind}}, \tag{3.17}$$

$$\underline{w}_{\mathrm{ind}} = [\underline{u}_0', \ldots, \underline{u}_{N-1}']', \quad \overline{w}_{\mathrm{ind}} = [\overline{u}_0', \ldots, \overline{u}_{N-1}']', \quad w_{\mathrm{ind}} = [u_0', \ldots, u_{N-1}']'. \tag{3.18}$$

One way to define quantities in (3.9c) is as follows:

$$\underbrace{\begin{bmatrix} \underline{w}_{\mathrm{dep}} - \tilde{w}_{\mathrm{dep}} \\ \underline{m} - \tilde{m} \end{bmatrix}}_{\Delta \underline{w}_{\mathrm{c}}} \leq \underbrace{\begin{bmatrix} A_{\mathrm{c},0} \\ A_{\mathrm{c},1} \end{bmatrix}}_{A_{\mathrm{c}}} \Delta w_{\mathrm{ind}} \leq \underbrace{\begin{bmatrix} \overline{w}_{\mathrm{dep}} - \tilde{w}_{\mathrm{dep}} \\ \overline{m} - \tilde{m} \end{bmatrix}}_{\Delta \overline{w}_{\mathrm{c}}} \tag{3.19}$$

with

$$A_{\mathrm{c},0} = E, \tag{3.20a}$$

$$\tilde{w}_{\mathrm{dep}} = w_{\mathrm{dep}} + d + C\Delta s_0, \tag{3.20b}$$

$$\underline{w}_{\mathrm{dep}} = [\underline{x}_1', \ldots, \underline{x}_N']', \quad \overline{w}_{\mathrm{dep}} = [\overline{x}_1', \ldots, \overline{x}_N']', \quad w_{\mathrm{dep}} = [x_1', \ldots, x_N']'. \tag{3.20c}$$

The aforementioned quantities define bounds on the state variables which are transformed to the affine constraints in the condensed QP. Linearized path constraints from the sparse QP, (3.1f) and (3.1g), are transformed into condensed affine constraints using the following relations:

$$\underline{m} = [\underline{m}_0', \ldots, \underline{m}_N']', \quad \overline{m} = [\overline{m}_0', \ldots, \overline{m}_N']', \tag{3.21}$$

$$\tilde{m} = \begin{bmatrix} M_0 \Delta s_0 \\ 0_{\dim_{\tilde{m}1} \times 1} \end{bmatrix} + \overline{M}(d + C\Delta s_0), \quad A_{\mathrm{c},1} = \overline{M}E + \overline{N}, \tag{3.22}$$

where

$$\overline{M} = \begin{bmatrix} 0_{\dim_{m_1} \times N \cdot n_x} \\ \operatorname{diag}(M_1, \ldots, M_N) \end{bmatrix}, \quad \overline{N} = \begin{bmatrix} \operatorname{diag}(N_0, \ldots, N_{N-1}) \\ 0_{\dim_{m_N} \times N \cdot n_u} \end{bmatrix}, \tag{3.23}$$

$$\dim_{\tilde{m}^1} = \dim_{m_1} + \ldots + \dim_{m_N} . \tag{3.24}$$

An efficient implementation should exploit that only lower triangular blocks of the matrix E need to be computed and stored in the memory. Apart from possible memory savings, there are positive implications on the memory access performance of the algorithms that use matrix E, e.g. most notably the calculation of the condensed Hessian. In principle there are two ways to calculate and store blocks of the matrix E. The first one is column-by-column ordering, where block-columns are concatenated forming together an eventually slim block-matrix of dimension $N(N+1)n_x/2 \times n_u$. Here, we address a non-zero block $E_{i,j}$ as

$$E(i,j) = E.\operatorname{vec}\left(\frac{1}{2}j(2N - j + 1) + i\right). \tag{3.25}$$

An alternative is to use block-wise row-by-row ordering where a non-zero block $E_{i,j}$ is referred as:

$$E(i,j) = E.\operatorname{vec}\left(\frac{i(i+1)}{2} + j\right). \tag{3.26}$$

In order not to clutter the presentation of the algorithms in the text that follows, we will simply refer to the aforementioned storage options when needed. Even though the amount of work to build up the matrix E is the same, see Table 3.2, there are some subtle differences about how the calculations are actually done. As defined in Algorithm 3.1, the first storage option for E forms one block-column of the matrix E in each pass of the outer loop, but at the expense of (re)reading the already read (used) blocks A_k; e.g. in the worst case the block A_{N-1} is going to be read $N-1$ times. On the contrary, the row-by-row scheme reads each block A_k exactly once, see Algorithm 3.2, meaning better temporal locality of the algorithm. For long horizons and medium values of number of states n_x this might be beneficial, as long memory jumps are avoided. The choice which of the two implementations to use is usually dictated by algorithms that are the bottleneck of the procedure. In the condensing procedure, that is the

Algorithm 3.1 Building of the matrices E and C (3.12).

```
for $i = 0, \dots, N-1$ do                          ▷ Build $E$
    $E_{i,i} \leftarrow B_i$
    for $j = i+1, \dots, N-1$ do
        $E_{j,i} \leftarrow A_j E_{j-1,i}$
    end for
end for
$C_0 \leftarrow A_0$                                 ▷ Build $C$
for $i = 1, \dots, N-1$ do
    $C_i \leftarrow A_i C_{i-1}$
end for
```

Algorithm 3.2 An alternative implementation to build the matrix E.

```
$E_{0,0} \leftarrow B_0$
for $i = 1, \dots, N-1$ do
    for $j = 0, \dots, i-1$ do
        $E_{i,j} \leftarrow A_i E_{i-1,j}$
    end for
    $E_{i,i} \leftarrow B_i$
end for
```

algorithm for building of the condensed Hessian. Finally, building of the matrix C is trivial and an efficient implementation might even consider to overwrite blocks A_k with the blocks C_k if they are not needed later.

A procedure to build up the condensed Hessian H^c is given in Algorithm 3.3. This implementation exploits the symmetry of H^c and builds up only the upper triangular blocks. Lower triangular blocks can be obtained later at almost no extra cost by copying appropriate blocks. Before the calculations are actually done, it is wise to pre-compute the product $\overline{Q}E$ and avoid redundant matrix-matrix multiplications. Exploiting symmetry reduced the number of block multiplications from N^3 to $N(N+1)(N+2)/6$, i.e. number of block

Table 3.2: Computational cost of Algorithm 3.1.

Quantity	Cost [FLOPs]
E	$(N-1)Nn_x^2 n_u + Nn_x n_u$
C	$(N-1)n_x^3 + n_x^2$

Algorithm 3.3 Full condensing procedure for building the condensed Hessian H^{c} (3.14a).

```
for $i = 0, \dots, N-1$ do                                          ▷ $W \leftarrow \overline{Q}E$
    for $j = i, \dots, N-1$ do
        $W_{j,i} \leftarrow Q_{j+1} E_{j,i}$
    end for
end for
for row $= 0, \dots, N-1$ do                                        ▷ Build $H^{\mathrm{c}}$
    $H^{\mathrm{c}}_{\mathrm{row,row}} \leftarrow R_{\mathrm{row}}$
    for col $=$ row $+ 1, \dots, N-1$ do
        $H^{\mathrm{c}}_{\mathrm{row,col}} \leftarrow E'_{\mathrm{col,row}} S_{\mathrm{col}}$
    end for
    for col $=$ row$, \dots, N-1$ do
        for blk $=$ col$, \dots, N-1$ do
            $H^{\mathrm{c}}_{\mathrm{row,col}} \leftarrow H^{\mathrm{c}}_{\mathrm{row,col}} + E'_{\mathrm{blk,row}} W_{\mathrm{blk,col}}$
        end for
    end for
end for
```

Table 3.3: Computational cost of Algorithm 3.3.

Quantity	Cost [FLOPs]
$W \leftarrow \overline{Q}E$	$N(N+1)n_x^2 n_u$
H^{c}	$\left(\frac{N^3}{3} + 2N^2 + N\right) n_x n_u^2 + N n_u^2$

multiplications is reduced by 70% or more for $N > 7$; this computational cost reduction was observed first in [43]. The cost analysis of the algorithm is given in Table 3.3. The computation of H^{c} is of $\mathcal{O}(N^3)$ complexity, but free of calculations of cubic complexity in the number of the states n_x. The cost of forming the blocks of the matrix W can be further reduced if blocks Q_k are diagonal, which is often the case in NMPC. The blocks of H^{c} are formed row-wise (outermost loop) and blocks of matrices E and W are accessed linearly column-wise (innermost loop). Therefore, this implementation should profit from using the column-wise storage for matrices E and W.

An algorithm for calculation of the linear term g^{c} is presented in Algorithm 3.4. Computations are intentionally organized so that the algorithms facilitates matrix-vector products as much as possible. As a consequence, two extra vectors

Algorithm 3.4 Full condensing procedure for building of the linear term g^{c} (3.14b).

```
$w^1 \leftarrow d$                                   ▷ $w^1 \leftarrow d + C\Delta s_0$
for $i = 0, \dots, N-1$ do
    $w_i^1 \leftarrow w_i^1 + C_i \Delta s_0$
end for
$w^2 \leftarrow g^s$                                 ▷ $w^2 \leftarrow g^s + \overline{Q} w^1$
for $i = 0, \dots, N-1$ do
    $w_i^2 \leftarrow w_i^2 + Q_{i+1} w_i^1$
end for
$g^{\mathrm{c}} \leftarrow g^u$                      ▷ Build $g^{\mathrm{c}}$ (3.14b)
$g_0^{\mathrm{c}} \leftarrow g_0^{\mathrm{c}} + S_0' \Delta s_0$
for $i = 0, \dots, N-1$ do
    for $j = i, \dots, N-1$ do
        $g_i^{\mathrm{c}} \leftarrow g_i^{\mathrm{c}} + E_{j,i}' w_j^2$
    end for
end for
for $i = 1, \dots, N-1$ do
    $g_i^{\mathrm{c}} \leftarrow g_i^{\mathrm{c}} + S_i w_{i-1}^1$
end for
```

Table 3.4: Computational cost of Algorithm 3.4.

Quantity	Cost [FLOPs]
$w^1 \leftarrow d + C\Delta s_0$	$Nn_x + 2Nn_x^2$
$w^2 \leftarrow g^s + \overline{Q} w^1$	$Nn_x + 2Nn_x^2$
g^{c}	$(N^2 + 3N) n_x n_u + N n_u$

are allocated w^1 and w^2 to hold intermediate results. The amount of work is still $\mathcal{O}(N^2)$, see Algorithm 3.4, but block operations are of lower complexity than for forming H^{c}. The cost of Algorithm 3.4 is presented in Table 3.4.

Algorithm 3.5 Full condensing procedure for the linearized path constraints.

```
$\tilde{m}_0 \leftarrow M_0 \Delta s_0$                                ▷ Build $\tilde{m}$
for $i = 1, \dots, N$ do
    $\tilde{m}_i \leftarrow M_i w^1_{i-1}$                  ▷ Reuse $w^1$ from the Algorithm 3.4
end for
for $i = 0, \dots, N-1$ do                              ▷ Build $A^{c,1}$
    $A^{c,1}_{i,i} \leftarrow N_i$
    for $k = i+1, \dots, N$ do
        $A^{c,1}_{k,i} \leftarrow M_k E_{k,i}$
    end for
end for
```

Table 3.5: Computational cost of Algorithm 3.5.

Quantity	Cost [FLOPs]
$\tilde{m}$	$n_x \sum_{i=0}^{N} \dim_{m_i}$
$A^{C,1}$	$n_u \sum_{i=0}^{N-1} \dim_{m_i} + n_x n_u \sum_{i=1}^{N} \sum_{j=i}^{N} \dim_{m_j}$

Transformation of the pure state constraints (3.20) to affine constraints in the condensed QP is trivial and can reuse some intermediate products from Algorithm 3.4. Condensing of the linear path constraints is presented in Algorithm 3.5 and its cost is summarized in Table 3.5. The amount of work to form the block $A^{C,1}$ is again quadratic in number of intervals, with the notion that the higher the index i on the horizon, the more work is required for the transformation. This observation is useful in cases the NLP has some of the path constraints undefined.

Partial Condensing

In partial condensing, we assume Δs_0 is also an optimization variable in the vector of independent variables Δw_{ind} (3.9). The affine map (3.1c) now reads:

$$\Delta w_{\text{dep}} = d + \underbrace{\begin{bmatrix} C & E \end{bmatrix}}_{E_{\text{p}}} \Delta w_{\text{ind}}. \tag{3.27}$$

Consequently, the expressions involved in forming the condensed Hessian H^{cp} and the condensed linear term g^{cp} are slightly modified:

$$H^{\text{cp}} = \begin{bmatrix} Q_0 & S_0 & & \\ S_0' & R_0 & & \\ & & \ddots & \\ & & & R_{N-1} \end{bmatrix} + E_{\text{p}}'\overline{Q}E_{\text{p}} + E_{\text{p}}'\overline{S}_{\text{p}} + \overline{S}_{\text{p}}'E_{\text{p}} \tag{3.28}$$

$$g^{\text{cp}} = \begin{bmatrix} g_0^s \\ g^u \end{bmatrix} + (E_{\text{p}}'\overline{Q} + \overline{S}_{\text{p}}')d + E_{\text{p}}'g^s, \quad \text{with} \tag{3.29}$$

$$\overline{S}_{\text{p}} = \begin{bmatrix} 0_{N \cdot n_x \times n_x} & \overline{S} \end{bmatrix}. \tag{3.30}$$

The bounds in the condensed QP are defined as

$$\underline{w}_{\text{ind}} = [\underline{s}_0', \underline{u}_0', \ldots, \underline{u}_{N-1}']', \quad \overline{w}_{\text{ind}} = [\overline{s}_0', \overline{u}_0', \ldots, \overline{u}_{N-1}']', \tag{3.31}$$

together with the current linearization point $w_{\text{ind}} = [s_0', u_0', \ldots, u_{N-1}']'$. The state bounds and the path constraints from the sparse QP are transformed into affine constraints using the relations:

$$A_{\text{c},0} = E_{\text{p}}, \quad \tilde{w}_{\text{dep}} = w_{\text{dep}} + d \quad \text{and} \tag{3.32}$$

$$A_{\text{c},1} = \overline{M}E_{\text{p}} + \left[\begin{array}{c|c} M_0 & \multirow{2}{*}{$\overline{N}$} \\ 0_{\dim_{\tilde{m}1} \times n_x} & \end{array}\right], \quad \tilde{m} = \overline{M}d. \tag{3.33}$$

The condensed Hessian H^{cp} can be partitioned in the following blocks:

$$H^{\text{cp}} = \begin{bmatrix} H_{0,0} & H_{0,1} \\ H_{0,1}' & H_{1,1} \end{bmatrix}, \tag{3.34}$$

Algorithm 3.6 Calculation of the condensed Hessian in the partial condensing procedure.

```
$H_{0,0} \leftarrow Q_0$                                    ▷ Calculate $H_{0,0}$
for $i = 0, \ldots, N-1$ do
    $H_{0,0} \leftarrow H_{0,0} + C_i' Q_{i+1} C_i$
end for
$H_{0,1}(0) \leftarrow S_0$                                 ▷ Calculate $H_{0,1}$
for $i = 1, \ldots, N-1$ do
    $H_{0,1}(i) \leftarrow C_{i-1}' S_i$
end for
Call the first part of Algorithm 3.3 to calculate $W$
for $i = 0, \ldots, N-1$ do
    for $j = i, \ldots, N-1$ do
        $H_{0,1}(i) \leftarrow H_{0,1}(i) + C_j' W_{j,i}$
    end for
end for
Call the second part of Algorithm 3.3 to calculate $H_{1,1}$
```

where

$$H_{0,0} = Q_0 + C'\overline{Q}C, \tag{3.35}$$

$$H_{0,1} = \begin{bmatrix} S_0 & 0_{n_x \times (N-1)n_u} \end{bmatrix} + C'\overline{Q}E + C'S, \tag{3.36}$$

$$H_{1,1} = H^{\text{c}}. \tag{3.37}$$

Now it is clear that the partially condensed Hessian H^{cp} can reuse a large portion of the computations from the full condensing procedure. The cost analysis in Table 3.6 suggests $\mathcal{O}(N)$ complexity for forming $H_{0,0}$ and $\mathcal{O}(N^2)$ complexity for forming block $H_{0,1}$. However, the nice property of the full condensing procedure to be free of operations cubic in n_x is lost. This is evident when forming the block $H_{0,0}$. Note that certain code optimizations can be performed to reduce work for symmetric products $C_i' Q_{i+1} C_i'$, see e.g. [96].

Using the same ideas as for the Hessian H^{cp}, we can efficiently partition the linear term g^{cp}:

$$g^{\text{cp}} = \begin{bmatrix} g_0^s + C'(\overline{Q}d + g^s) \\ g^u + E'(\overline{Q}d + g^s) + S'd \end{bmatrix}, \tag{3.38}$$

Table 3.6: Computational cost of Algorithm 3.6.

Quantity	Cost [FLOPs]
$H_{0,0}$	$4Nn_x^3 + n_x^2$
$H_{0,1}$	$(N^2 + 2N - 1)n_x^2 n_u$
$H_{1,1}$	See Table 3.3

Algorithm 3.7 Partial condensing procedure for building of the linear term g^{cp}.

```
$w \leftarrow g^s$                                        ▷ $w \leftarrow g^s + \overline{Q}d$
for $i = 0, \dots, N-1$ do
    $w_i \leftarrow w_i + Q_{i+1}d_i$
end for
$g^{\text{cp}} \leftarrow ((g_0^s)', (g^u)')'$                          ▷ Build $g^{\text{cp}}$
for $i = 0, \dots, N-1$ do
    $g_0^{\text{cp}} \leftarrow g_0^{\text{cp}} + C_i' w_i$
end for
for $i = 0, \dots, N-1$ do
    for $j = i, \dots, N-1$ do
        $g_{j+1}^{\text{cp}} \leftarrow g_{j+1}^{\text{cp}} + E_{j,i}' w_j$
    end for
end for
for $i = 1, \dots, N-1$ do
    $g_i^{\text{cp}} \leftarrow g_i^{\text{cp}} + S_i' d_{i-1}$
end for
```

that leads to Algorithm 3.7. At first glance, those computations indeed look a bit simpler than in the full condensing. Compared to the full condensing, the Algorithm 3.7 is using a bit less temporary memory than Algorithm 3.4. The complexity of forming g^{cp} is however the same, $\mathcal{O}(N^2)$. The cost of Algorithm 3.7 is summarized in Table 3.7.

Transformation of state bounds to appropriate affine constraints is trivial

Table 3.7: Computational cost of Algorithm 3.7.

Quantity	Cost [FLOPs]
$w \leftarrow g^s + \overline{Q}d$	$2Nn_x^2$
g^{cp}	$(N^2 + 3N)n_x n_u + 2Nn_x^2 + Nn_u + n_x$

Algorithm 3.8 Partial condensing of the linearized path constraints.

```
$\tilde{m}_0 \leftarrow 0$                                              ▷ Build $\tilde{m}$
for $i = 1, \dots, N$ do
    $\tilde{m}_i \leftarrow M_i d_{i-1}$
end for
$A_{C,1}(0,0) \leftarrow M_0$, $A_{C,1}(0,1) \leftarrow N_0$            ▷ Build $A_{C,1}$
for $i = 1, \dots, N-1$ do
    $A_{C,1}(i,0) \leftarrow M_i C_{i-1}$
    for $j = 1, \dots, i$ do
        $A_{C,1}(i,j) \leftarrow M_i E_{i-1,j-1}$
    end for
    $A_{C,1}(i,i+1) \leftarrow N_i$
end for
$A_{C,1}(N,N) \leftarrow M_N E_{N-1,N-1}$
```

Table 3.8: Computational cost of Algorithm 3.8.

Quantity	Cost [FLOPs]
$\tilde{m}$	Same as in Table 3.5
$A_{C,1}$	$\dim_{m_0}(n_x + n_u) + (n_x^2 + n_u)\sum_{i=1}^{N-1}\dim_{m_i}$ $+ n_x n_u (\sum_{i=1}^{N-1}\sum_{j=1}^{i}\dim_{m_i} + \dim_{m_N})$

again. An algorithm for condensing of the linearized path constraints is derived in Algorithm 3.8 and corresponding computational cost is summarized in Table 3.8. The same as in the full condensing, the complexity of this part of the procedure is $\mathcal{O}(N^2)$.

3.2.2 $\mathcal{O}(N^2)$ Condensing

Full Condensing

In the following, we mainly present results originally published in [78]. Moreover, we comment on computational demands of the $\mathcal{O}(N^2)$ condensing procedure and its abilities for parallelization.

The affine map (3.1c) can also be defined in a bit different form:

$$\underbrace{\begin{bmatrix} I & & & \\ -A_1 & I & & \\ & \ddots & \ddots & \\ & & -A_{N-1} & I \end{bmatrix}}_{\overline{A}} \Delta w_{\text{dep}} = \underbrace{\begin{bmatrix} a_0 + A_0 \Delta s_0 \\ a_1 \\ \vdots \\ a_{N-1} \end{bmatrix}}_{\overline{a}} + \\ + \underbrace{\begin{bmatrix} B_0 & & & \\ & B_1 & & \\ & & \ddots & \\ & & & B_{N-1} \end{bmatrix}}_{\overline{B}} \Delta w_{\text{ind}}, \quad (3.39)$$

yielding more compact representation $\Delta w_{\text{dep}} = \overline{d} + E \Delta w_{\text{ind}}$ with:

$$E = \overline{A}^{-1} \overline{B} \quad \text{and} \quad \overline{d} = \overline{A}^{-1} \overline{a}. \quad (3.40)$$

It can be easily shown [78] that the matrix E is equivalent to the one defined in (3.12). However, here it can be observed that the matrix E is formed in $\mathcal{O}(N^2)$ time by exploiting the special structure of $\overline{A}$ and $\overline{B}$. In particular, the product $\overline{A}^{-1}\overline{B}$ can be formed easily using block-wise forward substitutions, where each substitution of $\overline{A}$ is of order $\mathcal{O}(N)$. Given the special structure of $\overline{A}$ not a single matrix inverse has to be performed.

The expression for the condensed Hessian H^{c} becomes:

$$\begin{aligned} H^{\mathrm{c}} &= \overline{R} + E'\overline{Q}E \qquad\qquad + \overline{S}'E \ + \ E'\overline{S} \\ &= \overline{R} + \overline{B}'\underbrace{\overline{A}'^{-1}(\overline{Q}E)}_{P} + \overline{S}'E \ + \ E'\overline{S}. \end{aligned} \tag{3.41}$$

Given that E is lower block-triangular and $\overline{Q}$ is block-diagonal, the product $\overline{Q}E$ retains the same block triangular structure as before. The product P can be formed by applying block-wise backward substitutions – for each of N block-columns of E the block-wise backward substitution of order $\mathcal{O}(N)$ has to be done. This is the crucial step to observe and it is indeed the one that reduces the complexity of the condensing procedure to $\mathcal{O}(N^2)$. Furthermore, exploiting the lower block triangular structure of E and that only upper (or lower) block triangle of H^{c} has to be calculated, the amount of calculations can be further reduced. Multiplication of $\overline{B}$ with P is trivial since $\overline{B}$ is block diagonal and preserves the computational complexity of the procedure.

Similarly, the aforementioned observations enable computation of the linear term g^{c} in linear time in N:

$$\begin{aligned} g^{\mathrm{c}} &= g^u + E'(\overline{Q}\overline{d} + g^s) \qquad\quad + \overline{S}'\overline{d} \\ &= g^u + \overline{B}'\underbrace{\overline{A}'^{-1}(\overline{Q}\overline{d} + g^s)}_{p} + \overline{S}'\overline{d} \end{aligned} \tag{3.42}$$

utilizing block-wise substitutions for $\overline{A}'$ to form the vector p.

The state bounds computation gets even simpler, with $\tilde{w}_{\mathrm{dep}} = w_{\mathrm{dep}} + \overline{d}$, cf. (3.20). An extension for condensing of the path constraints can be formulated as follows:

$$A_{\mathrm{c},1} = \overline{M}E + \overline{N} \quad \text{and} \quad \tilde{m} = \begin{bmatrix} M_0 \Delta s_0 \\ 0_{\dim_{\tilde{m}1} \times 1} \end{bmatrix} + \overline{M}\overline{d}. \tag{3.43}$$

Calculation of $\tilde{m}$ is trivial and building of $A_{\mathrm{c},1}$ can be done reusing a part of the Algorithm 3.5. The complexity of this part of the procedure in principle cannot be reduced and is still $\mathcal{O}(N^2)$.

For completeness and ease of the presentation we present algorithms for

Algorithm 3.9 $\mathcal{O}(N^2)$ full condensing of the condensed Hessian H^c.

```
for $i = 0, \ldots, N-1$ do
    $W \leftarrow Q_N E_{N-1,i}$
    for $k = N-1, \ldots, i+1$ do
        $H_{i,k} \leftarrow E'_{k-1,i} S_k + W' B_k$
        $W \leftarrow Q_k E_{k-1,i} + A'_k W$
    end for
    $H^c_{i,i} \leftarrow B'_i W + R_i$
end for
```

forming H^c and g^c in Algorithm 3.9 and Algorithm 3.10, respectively [78]. The complexities of both implementations are given in Table 3.9. As in the classical condensing, Algorithm 3.9 presents computation of the upper triangular blocks of H^c. The first to be observed is that forming H^c does not require caching of the matrix products $\overline{Q}E$ as in Algorithm 3.3. Instead, a small work matrix $W \in \mathbb{R}^{n_x \times n_u}$ is needed. This leads to better temporal locality of the algorithm – the same work vector is reused (and updated) many times. Next, it can be observed that the i-th pass of the outer loop calculates the i-th row of H^c using only the column i of the matrix E. In other words, each row of the condensed Hessian H^c can be computed individually. Obviously, this enables one to parallelize the computations. Nevertheless, since the work performed for each row is unequal, the amount of work lowers as the index i grows, special care needs to be taken to keep all processors busy for approximately the same amount of time. For example, for an OCP with $N = 8$ intervals and a processor with two cores, core 1 can process the first three rows and core 2 can process rows four till eight. The core 1 processes 21 blocks and core 2 processes the remaining 15 blocks. Since the computations are totally decoupled (block row-wise), the core 2 can proceed with e.g. processing of g^c. In this simple analysis, we assumed memory copies needed to form H^c take negligible time. Further implementation optimizations in this context are possible as well. The algorithms 3.1 and 3.9 can be interleaved [78]. This way, the i-th column of the matrix E is calculated in the forward sweep and used in the backward sweep to create row i of H^c. Consequently, spatial locality of the implementation can be improved.

A reduced complexity $\mathcal{O}(N)$ algorithm for the computation of the linear term g^c is given in Algorithm 3.10. Compared to Algorithm 3.4 only a small work vector $w \in \mathbb{R}^{n_x}$ has to be allocated compared to the two working vectors of size Nn_x.

Algorithm 3.10 $\mathcal{O}(N)$ full condensing of the condensed linear term g^{c}.

```
$g^{\mathrm{c}} \leftarrow g^{u}$
$w \leftarrow Q_N d_{N-1} + g^{s}_{N-1}$
for $k = N-1, \dots, 1$ do
    $g^{\mathrm{c}}_k \leftarrow g^{\mathrm{c}}_k + B_k' w + S_k' d_{k-1}$
    $w \leftarrow Q_k d_{k-1} + g^{s}_{k-1} + A_k' w$
end for
$g^{\mathrm{c}}_0 \leftarrow g^{\mathrm{c}}_0 + B_0 w + S_0 \Delta s_0$
```

Table 3.9: Computational costs of Algorithms 3.9 and 3.10.

Quantity	Cost [FLOPs]
H^{c}	$2N^2(n_x n_u^2 + n_x^2 n_u)$
g^{c}	$N(2n_x^2 + 2n_x n_u + n_u)$

Partial Condensing

Building on the same ideas as in the full condensing, we write the affine map for the linearized system dynamics:

$$\underbrace{\begin{bmatrix} I & & & \\ -A_1 & I & & \\ & \ddots & \ddots & \\ & & -A_{N-1} & I \end{bmatrix}}_{\overline{A}} \Delta w_{\mathrm{dep}} = \underbrace{\begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_{N-1} \end{bmatrix}}_{\overline{a}_{\mathrm{p}}} + \\ + \underbrace{\begin{bmatrix} A_0 & B_0 & & & \\ & & B_1 & & \\ & & & \ddots & \\ & & & & B_{N-1} \end{bmatrix}}_{\overline{B}_{\mathrm{p}}} \Delta w_{\mathrm{ind}} \tag{3.44}$$

Algorithm 3.11 Partial condensing of the condensed Hessian H^{cp}.

```
$V \leftarrow Q_N C_{N-1}$
for $i = N-1, \ldots, 1$ do
    $H_{0,1}(i) \leftarrow V' B_i + C'_{i-1} S_i$
    $V \leftarrow Q_i C_{i-1} + A'_i V$
end for
$H_{0,1}(0) \leftarrow S_0 + V' B_0$
$H_{0,0} \leftarrow Q_0 + A'_0 V$
Call Algorithm 3.9 to calculate $H_{1,1}$
```

the yields the following relations

$$E_{\mathrm{p}} = \overline{A}^{-1}\overline{B}_{\mathrm{p}} = \left[\begin{array}{c|c} \overline{A}^{-1} \underbrace{\begin{bmatrix} A_0 \\ 0_{(N-1)\cdot n_x \times n_x} \end{bmatrix}}_{\overline{A}_0} & \overline{A}^{-1}\overline{B} \end{array}\right] = \begin{bmatrix} C & E \end{bmatrix} \quad \text{and} \tag{3.45}$$

$$d = \overline{A}^{-1}\overline{a}_{\mathrm{p}}. \tag{3.46}$$

Partitioning the condensed Hessian H^{cp}

$$C'\overline{Q}C = \overline{A}'_0\overline{A}'^{-1}\overline{Q}C \quad \text{and} \quad C'\overline{Q}E = \overline{A}'_0\overline{A}'^{-1}\overline{Q}E \tag{3.47}$$

leads to the efficient – $\mathcal{O}(N)$ – computation of the blocks $H_{0,0}$ and $H_{0,1}$ as presented in the Algorithm 3.11. The overall complexity for computation of H^{cp} is still $\mathcal{O}(N^2)$, however. The linear term g^{cp} is computed in linear in N time, cf. Algorithm 3.12. The linearized path constraints can be computed using the Algorithm 3.8.

Algorithm 3.11 is not free of $\mathcal{O}(n_x^3)$ computations, like in the classical condensing. Note that computation of the blocks $H_{0,0}$ and $H_{0,1}$ can be done separately, i.e. in parallel with computation of $H_{1,1}$. Moreover, in the classical condensing the blocks $H_{0,0}$ and $H_{1,1}$ are $\mathcal{O}(N^2)$ and $\mathcal{O}(N)$ computations, respectfully, while in this procedure $H_{0,0}$ comes at $\mathcal{O}(n_x^3)$ cost reusing results from $\mathcal{O}(N)$ computation of the block $H_{0,1}$. The number of FLOPs (floating point operations) for the algorithms to form H^{cp} and g^{cp} is summarized in Table 3.10.

Algorithm 3.12 Partial condensing of the condensed linear term g^{cp}.

```
$g^{\text{cp}} \leftarrow \begin{bmatrix} g_0^s \\ g^u \end{bmatrix}$
$w \leftarrow g_N^s + Q_N d_{N-1}$
$g_N^{\text{cp}} \leftarrow g_{N-1}^u + B_{N-1}' w$
for $k = N-1, \dots, 1$ do
    $g_{k+1}^{\text{cp}} \leftarrow g_{k+1}^{\text{cp}} + S_k' d_{k-1}$
    $w \leftarrow g_k^s + Q_k d_{k-1} + A_k' w$
    $g_k^{\text{cp}} \leftarrow g_k^{\text{cp}} + B_{k-1}' w$
end for
$g_0^{\text{cp}} \leftarrow g_0^{\text{cp}} + A_0' w$
```

Table 3.10: Computational costs of Algorithms 3.11 and 3.12.

Quantity		Cost [FLOPs]
H^{cp}		
	$H_{0,0}$ and $H_{0,1}$	$4N(n_x^3 + n_x^2 n_u) - 2n_x^2 n_u$
	$H_{1,1}$	See Table 3.9
g^{cp}		$N(6n_x n_u + 2n_x^2 + n_u) + 2n_x^2 - 4n_x n_u + n_x$

3.2.3 $\mathcal{O}(N^2)$ Factorization of the Condensed Hessian

Full Condensing

Let us consider the permuted vector of independent and dependent variables:

$$\Delta\hat{w}_{\text{ind}} = [\Delta u_{N-1}', \dots, \Delta u_0']' \quad \text{and} \quad \Delta\hat{w}_{\text{dep}} = [\Delta s_N', \dots, \Delta s_1']. \tag{3.48}$$

Next, permuted transition matrices $\hat{E}$ and $\hat{C}$ can be defined as:

$$\hat{E}_{i,j} = E_{N-1-i,N-1-j} \quad \text{and} \quad \hat{C}_i = C_{N-1-i}. \tag{3.49}$$

The permuted condensed Hessian has the same form as in (3.14a) with the permuted quantities: $\hat{H}^{\text{c}} = \hat{\bar{R}} + \hat{E}'\hat{\bar{Q}}\hat{E} + \hat{E}'\hat{\bar{S}} + \hat{\bar{S}}'\hat{E}$. A recursive algorithm to compute the factorization of the condensed Hessian $\hat{H}^{\text{cp}} = (\hat{U}^{\text{c}})'\hat{U}^{\text{c}}$ in $\mathcal{O}(N^2)$ time is based on the following observations. First, the transition matrix $\hat{E}$ can be defined in a recursive fashion. If we refer to non-zero blocks in the row i of the upper-triangular matrix $\hat{E}$ as $\hat{E}_i$, then

$$\hat{E}_i = \begin{bmatrix} \hat{B}_i & \hat{A}_i \hat{E}_{i+1} \end{bmatrix}, \tag{3.50}$$

with $\hat{E}_{N-1} = B_0$. Second, the blocks of the condensed Hessian $\hat{H}^{\mathrm{c}}$ can be written in a more compact form. The first two rows of the permuted Hessian read:

$$\hat{H}^{\mathrm{c}} = \begin{bmatrix} \hat{R}_0 + \hat{B}_0' \hat{F}_0 & (\hat{S}_0 + \hat{F}_0' \hat{A}_0)\hat{E}_1 & \\ * & \hat{R}_1 + \hat{B}_1' \hat{F}_1 & (\hat{S}_1 + \hat{F}_1' \hat{A}_1)\hat{E}_2 \\ * & * & \vdots \end{bmatrix}, \tag{3.51}$$

with

$$\hat{F}_0 = Q_N \hat{B}_0, \tag{3.52}$$

$$\hat{F}_1 = Q_{N-1} \hat{B}_1 + \hat{A}_0' Q_N \hat{A}_0 \hat{B}_1. \tag{3.53}$$

The recursion starts from the upper left block of $\hat{H}^{\mathrm{c}}$, namely $\hat{H}^{\mathrm{c}}(0,0)$ can be factored as:

$$D_0 = \mathrm{chol}(\hat{R}_0 + \hat{B}_0'(Q_N \hat{B}_0)),$$

where chol(.) denotes the operator that returns the upper-triangular Cholesky factor of the input matrix. Next, the first row of the factorized condensed Hessian becomes:

$$\hat{U}^{\mathrm{c}} = \begin{bmatrix} D_0 & L_0 \hat{E}_1 \\ & \vdots \end{bmatrix} \tag{3.54}$$

with $L_0 = D_0^{-1}(\hat{S}_0 + (Q_N \hat{B}_0)' \hat{A}_0)$. The next step updates the remaining block of $\hat{H}^{\mathrm{c}}$ to be factorized:

$$\begin{aligned} \hat{H}^{\mathrm{c}*}(1:N-1, 1:N-1) &= \hat{H}^{\mathrm{c}}(1:N-1, 1:N-1) - \hat{E}_1' L_0' L_0 \hat{E}_1 \\ &= \begin{bmatrix} \hat{R}_1 + \hat{B}_1' \hat{F}_1^* & (\hat{S}_1 + (\hat{F}_1^*)' \hat{A}_1)\hat{E}_2 \\ * & \vdots \end{bmatrix}. \end{aligned} \tag{3.55}$$

After some minor rearrangements it can be shown that the updated block $\hat{F}_1^*$ can be expressed as:

$$\hat{F}_1^* = Q_{N-1}^* \hat{B}_1 + \hat{A}_0' Q_N \hat{A}_0 \hat{B}_1, \tag{3.56}$$

with the updated weight $Q^*_{N-1} = Q_{N-1} - L'_0 L_0$. We can see now that the structure of the updated block is the same as the one of the original matrix $\hat{H}^{\mathrm{c}}$, thus the same procedure can be applied. Finally, given the special structure of $\hat{E}$ intermediate results for calculation of temporary block F_k can be done efficiently, see Algorithm 3.13. The explained procedure is repeated N times, until the last block $\hat{H}^{\mathrm{c}}(N-1, N-1)$ is factorized.

In the original paper [95] that proposed this strategy it was observed that for the solution of the unconstrained problem (3.1a) – (3.1c), the explicit calculation of the whole factorized Hessian $\hat{U}^{\mathrm{c}}$ is not needed at all. Instead, only blocks D_k and L_k are required to calculate the solution of the corresponding KKT system. In this case, lines 9 to 12 and line 19 of Algorithm 3.13 can be omitted.

Algorithm 3.13 An algorithm for $\mathcal{O}(N^2)$ factorization of the condensed Hessian, **full** condensing case.

```
 1: $T \leftarrow Q_N$
 2: for col $= 0, \dots, N-1$ do                                  ▷ Initialize temporary matrix $F$
 3:     $F_{\mathrm{col}} \leftarrow T\hat{E}_{0,\mathrm{col}}$
 4: end for
 5: for blk $= N-1, \dots, 1$ do                                  ▷ Build $\hat{U}^{\mathrm{c}}$
 6:     row $= N - 1 -$ blk
 7:     $D_{\mathrm{row}} \leftarrow \mathrm{chol}(R_{\mathrm{blk}} + B'_{\mathrm{blk}} F_{\mathrm{row}})$
 8:     $L_{\mathrm{row}} \leftarrow (D'_{\mathrm{row}})^{-1}(S'_{\mathrm{blk}} + F'_{\mathrm{row}} A_{\mathrm{blk}})$
 9:     $\hat{U}^{\mathrm{c}}_{\mathrm{row,row}} \leftarrow D_{\mathrm{row}}$
10:     for col $=$ row $+ 1, \dots, N-1$ do
11:         $\hat{U}^{\mathrm{c}}_{\mathrm{row,col}} \leftarrow L_{\mathrm{row}} \hat{E}_{\mathrm{row+1,col}}$
12:     end for
13:     $T \leftarrow Q_{\mathrm{blk}} - L'_{\mathrm{row}} L_{\mathrm{row}}$                  ▷ Update $Q_{\mathrm{blk}}$
14:     for col $=$ row $+ 1, \dots, N-1$ do                          ▷ Update $F$
15:         $F_{\mathrm{col}} \leftarrow T\hat{E}_{\mathrm{row+1,col}} + A'_{\mathrm{blk}} F_{\mathrm{col}}$
16:     end for
17: end for
18: $D_{N-1} \leftarrow \mathrm{chol}(R_0 + B'_0 F_{N-1})$
19: $\hat{U}^{\mathrm{c}}_{N-1,N-1} \leftarrow D_{N-1}$
```

Partial Condensing

Extension of the $\mathcal{O}(N^2)$ procedure for building of a factor of the condensed Hessian when the initial state is free is straightforward. Compared to the previous procedure one extra block column (the last one) of $\hat{H}^{\text{cp}}$ has to be processed to form the factor $\hat{U}^{\text{cp}}$, see line 14 of Algorithm 3.14. This procedure requires one more work matrix G, which is used to propagate the intermediate results along the horizon and it is used at the end of the procedure to calculate the last two blocks, $\hat{U}^{\text{cp}}_{N-1,N}$ and $\hat{U}^{\text{cp}}_{N,N}$. This propagation, together with building of matrix C, introduces extra $\mathcal{O}(n_x^3)$ calculations.

Algorithm 3.14 An algorithm for $\mathcal{O}(N^2)$ factorization of the condensed Hessian, **partial** condensing case.

```
 1: $T = Q_N$
 2: for col $= 0, \dots, N-1$ do                          ▷ Initialize temporary matrix $F$
 3:     $F_{\text{col}} \leftarrow T\hat{E}_{0,\text{col}}$
 4: end for
 5: $G \leftarrow T\hat{C}_0$                              ▷ Initialize temporary matrix $G$
 6: for blk $= N-1, \dots, 1$ do                          ▷ Build $\hat{U}^{\text{cp}}$
 7:     row $= N - 1 - $ blk
 8:     $D_{\text{row}} \leftarrow \text{chol}(R_{\text{blk}} + B'_{\text{blk}}F_{\text{row}})$
 9:     $L_{\text{row}} \leftarrow (D'_{\text{row}})^{-1}(S'_{\text{blk}} + F'_{\text{row}}A_{\text{blk}})$
10:     $\hat{U}^{\text{cp}}_{\text{row,row}} \leftarrow D_{\text{row}}$
11:     for col $=$ row $+ 1, \dots, N-1$ do
12:         $\hat{U}^{\text{cp}}_{\text{row,col}} \leftarrow L_{\text{row}}\hat{E}_{\text{row}+1,\text{col}}$
13:     end for
14:     $\hat{U}^{\text{cp}}_{\text{row},N} \leftarrow L_{\text{row}}\hat{C}_{\text{col}+1}$
15:     $T \leftarrow Q_{\text{blk}} - L'_{\text{row}}L_{\text{row}}$            ▷ Update $Q_{\text{blk}}$
16:     for col $=$ row $+ 1, \dots, N-1$ do               ▷ Update $F$
17:         $F_{\text{col}} \leftarrow T\hat{E}_{\text{row}+1,\text{col}} + A'_{\text{blk}}F_{\text{col}}$
18:     end for
19:     $G \leftarrow T\hat{C}_{\text{row}+1} + A'_{\text{blk}}G$              ▷ Update $G$
20: end for
21: $D_{N-1} \leftarrow \text{chol}(R_0 + B'_0F_{N-1})$
22: $\hat{U}^{\text{cp}}_{N-1,N-1} \leftarrow D_{N-1}$
23: $L_{N-1} \leftarrow (D'_{N-1})^{-1}(S'_0 + B'_0G)$
24: $\hat{U}^{\text{cp}}_{N-1,N} \leftarrow L_{N-1}$
25: $D_N \leftarrow \text{chol}(Q_0 + A'_0G)$
26: $\hat{U}^{\text{cp}}_{N,N} \leftarrow D_N$
```

Table 3.11: Computational costs of Algorithms 3.13 and 3.14.

Quantity	Cost [FLOPs]
$\hat{U}^{\mathrm{c}}$	$N^2(2n_x^2 n_u + n_x n_u^2) + N\left(\frac{n_u^3}{3} + 2n_x^2 n_u + 2n_x n_u^2 + n_u^2\right)$ $-(2n_x^2 n_u + 3n_x n_u^2 - n_x^2 + n_u^2)$
$\hat{U}^{\mathrm{cp}}$	$\mathrm{flops}(\hat{U}^{\mathrm{c}}) + 2N(2n_x^3 + n_x^2 n_u) + \frac{8}{3}n_x^3 - 2n_x^2 n_u + n_x n_u^2$

Computational Cost Analysis

Besides the complexity reduction, the $\mathcal{O}(N^2)$ routines for the factorization of the reduced Hessian have one more benefit: the condensed Hessian can be factorized before the call to a QP solver code. This is in contrast to the standard full condensing approach where the condensed Hessian is factorized inside the QP solver. However, using the standard approach the factorization of the condensed Hessian is free of any computations depending on n_x, while the reduced complexity factorization routines introduced $\mathcal{O}(n_x^2)$ computations. On the other hand, the partial condensing still involves $\mathcal{O}(n_x^3)$ computations present in both the classical and enhanced condensing procedures. The computational costs of both approaches are summarized in Table 3.11[2].

Let us now briefly discuss the factorization techniques in conjunction with null space active-set QP solvers. In NMPC, the cost of the initial factorization of the condensed Hessian in the cold-started QP solver is $\mathcal{O}((Nn_u)^3)$. If the solver is warm-started, one also has to take into account cost to form the reduced Hessian that might not be negligible if not many constraints were active in the previous QP. Let us remind the reader, again, that in the nonlinear MPC the factorization typically has to be performed at each time instant. A first guess that leads to complexity reduction of factorization of the reduced Hessian is to provide the solver with the Hessian and corresponding factorization. At first sight this is reasonable since the newly developed routines reduce the complexity from cubic to square in number of intervals N. Our preliminary analysis revealed that this approach is competitive in the full condensing to the standard approach (supplying only the condensed Hessian to QP solver) only for ratios of n_x/n_u less than 2 and independent from N.

---

[2] The $\mathrm{flops}(.)$ operator returns computational cost of an algorithm in number of FLOPs.

The partial condensing involves $\mathcal{O}(n_x^3)$ calculations and the complexity of factorization of the condensed Hessian (and the QP solution) inside the QP solver is $\mathcal{O}((Nn_u + n_x)^3)$. As we saw in the text above, the $\mathcal{O}(N^2)$ procedure for factorization of the partially condensed Hessian suffers the same issue. Another preliminary analysis revealed that supplying the QP solver with both the Hessian and its factorization is always faster than providing the solver solely with the Hessian. This is indeed expected because of the complexity reduction in number of intervals N. Moreover, both approaches depend on $\mathcal{O}(n_x^3)$ operations – in the full condensing internal factorization of the condensed Hessian was not dependent on number of states n_x. However, the biggest speed-ups were observed for the low ratios n_x/n_u. Following this observation, the new approach for factorization of the condensed Hessian might be beneficial when the user wants to use a standard variant of MHE with process noise, where number of states is equal to the number of inputs.

3.3 Structure Exploiting Interior Point Method Solvers

The second way of solving (3.1) is to keep and optimize all variables: Δs and Δu, i.e. avoid the aforementioned condensing procedure. This leaves far more variables in the problem. However, compared to the condensed, dense QP (3.9), the problem described in (3.1) is highly structured. This structure can be exploited to yield a computational complexity of $O\left(N(n_x + n_u)^3\right)$, i.e. linear in number of intervals N, but cubic in number of states and controls. On the other side, in the nonlinear MPC condensing based approach is free of $\mathcal{O}(n_x^3)$ computations at the cost of increased complexity in number of intervals – $\mathcal{O}(N^3)$ – because of the factorization of reduced Hessian. In the MHE, the partial condensing suffers from $\mathcal{O}(n_x^3)$ computations as well, unfortunately. Interior point methods can significantly reduce the execution time of the QP solution, and consequently execution time of an RTI, for long prediction horizons compared to the condensing based approaches. The exploration of this banded structure in the solution of nonlinear control problems was first proposed in [4] and for linear MPC in [5].

As for the active-set solvers, the primal barrier interior point methods might require an expensive *Phase I* procedure to obtain a feasible initial guess. One of the successful fast implementations for linear MPC based on primal barrier method is presented in [97]. On the other hand, the primal-dual interior methods (IPMs) do not require expensive Phase I procedure and showed to be successful in practice [49]. A structure exploiting QP solver for linear MPC based on Mehrotra's [98] primal-dual IPM is presented in [5]. Following the ideas from that paper, more optimized implementations for embedded MPC-like QPs are derived [99, 100].

The sparse QP (3.1) can be cast in the following general QP form:

$$\min_{w} \quad \frac{1}{2} w' \mathcal{H} w + g' w, \tag{3.57a}$$

$$\text{s.t.} \quad \mathcal{F} w = b_{\mathrm{e}}, \tag{3.57b}$$

$$\mathcal{C} w \leq b_{\mathrm{i}}, \tag{3.57c}$$

with stacked *stage* variables $w = [w_0', w_1', \ldots, w_N']'$. The stage variables are defined as $w_k = [\Delta s_k', \Delta u_k']'$, $k = 0, \ldots, N-1$ and $w_N = \Delta s_N$. In the primal-dual IPMs, the inequality constraints (3.57c) are relaxed with slack variables t:

$$\mathcal{C} w + t = b_{\mathrm{i}}, \; t \geq 0,$$

where t is the vector of slack variables $t = [t_0', t_1', \ldots, t_N']'$. The KKT conditions for the relaxed QP can be put in the compact form

$$\begin{bmatrix} \mathcal{H} w + \mathcal{F}' \lambda + \mathcal{C}' \mu + g \\ -\mathcal{F} w + b_{\mathrm{e}} \\ -\mathcal{C} w - t + b_{\mathrm{i}} \\ \langle \mu, t \rangle \end{bmatrix} = 0, \quad \text{with} \quad \mu \geq 0, \; t \geq 0. \tag{3.58}$$

Therein, the vectors of the Lagrangian multipliers are denoted as $\lambda = [\lambda_0', \lambda_1', \ldots, \lambda_{N-1}']'$, $\mu = [\mu_0', \mu_1', \ldots, \mu_N']'$ and .

The KKT system (3.58) is typically solved with a Newton-like algorithm. The algorithm generates a sequence of iterates where the search direction is computed as

$$\begin{bmatrix} \mathcal{H} & \mathcal{F}' & \mathcal{C}' & \\ -\mathcal{F} & & & \\ -\mathcal{C} & & & -I \\ & & D_t & D_\mu \end{bmatrix} \begin{bmatrix} \Delta w \\ \Delta \lambda \\ \Delta \mu \\ \Delta t \end{bmatrix} = \underbrace{\begin{bmatrix} r_H \\ r_F \\ r_C \\ r_t \end{bmatrix}}_{r}, \tag{3.59}$$

where $D_\mu = \operatorname{diag}(\mu)$ and $D_t = \operatorname{diag}(t)$. The vector of residuals r is specific for a particular IPM implementation [49, 47]. Elimination of Δt followed by elimination of $\Delta \mu$ yields a reduced system

$$\begin{bmatrix} \mathcal{H} + \mathcal{C}' D_\mu D_t^{-1} \mathcal{C} & \mathcal{F}' \\ -\mathcal{F} & \end{bmatrix} \begin{bmatrix} \Delta w \\ \Delta \lambda \end{bmatrix} = \begin{bmatrix} r_H - \mathcal{C}' D_t^{-1} (D_\mu r_c + r_t) \\ r_F \end{bmatrix}. \tag{3.60}$$

Given the block diagonal structure of $\mathcal{H}$ and the special block structure of the matrix $\mathcal{C}$, the upper left block in the coefficient matrix in (3.60) retains the same block structure as the matrix $\mathcal{H}$. Another way around, the reduced system (3.60) has exactly the same structure as an unconstrained QP, e.g. the sparse QP (3.1) without the inequality constraints. In [5] it is shown that with suitable grouping of Δw_k and $\Delta \lambda_k$, e.g. $\xi = [\Delta \lambda_0', \Delta w_0', \Delta \lambda_1', \Delta \ldots, \Delta w_N']'$, the reduced system can be cast in a symmetric block tri-diagonal form where the coefficient matrix reads

$$\begin{bmatrix} & \mathbb{I} & & & \\ I & \tilde{Q}_0 & \tilde{S}_0 & -A_0' & \\ & * & \tilde{R}_0 & -B_0' & \\ & * & * & \ddots & I \\ & & & I & \tilde{Q}_N \end{bmatrix}. \tag{3.61}$$

Such a system can be efficiently solved by a discrete Riccati-like recursion. Efficient implementations for factorization of the block tri-diagonal matrices exhibits $\mathcal{O}(N(n_x + n_u)^3)$ complexity. We refer for implementation details in the MPC context to [5], and to [33] for implementation details related to an MHE solver. Let us observe briefly that the sub-problem (3.60) is always of the same structure and dimensions for one particular sparse QP (3.1) during the whole iteration process. An equality constrained QP (3.3) that is a sub-problem in the active-set methods is always changing the structure as constraints and bounds are (re)moved from the working set. For a software implementation it means that for a particular inequality constrained QP, only one subroutine needs to be coded. This makes the reduced system particularly interesting for extreme software optimizations for embedded (linear) MPC, as explored in [96, 100, 101, 102].

An alternative approach to solution of (3.58) further reduces (3.60) applying the Schur complement for the zero block in the coefficient matrix [97, 99]. The approach yields the further reduced system with a block tri-diagonal coefficient matrix. The linear system can be solved with Riccati-like recursion at the same complexity – $\mathcal{O}(N(n_x + n_u)^3)$. On the linear algebra level, this approach allows for better structure exploitation and code optimizations than original implementation [5]. In particular, in [99] is studied how implementation can be optimized based on the sparsity of the weighting matrices and type of constraints involved in an OCP.

Unlike the active-set QP solvers, the interior-point methods show limited capabilities for warm-starting from which the MPC applications might profit. Warm-staring strategies for fast MPC exist and for more details on this topic we refer to [103, 104]. In practice, IPMs in general need much less iterations than active-set solvers and the number of iterations is independent of the number of active constraints. However, one IPM iteration of a structure exploiting QP solver for MPC is in general more costly than one iteration of a dense active-set solver employed to solve the condensed QP. An advantage that is attractive from purely theoretical point of view is the polynomial complexity in number of iterations of the IPMs. In contrast, the theoretical worst-case complexity for the active-set methods is exponential.

3.4 The Dual Newton Strategy

The advantage of this so-called *dual Newton strategy*, introduced in [82] and extended in [76, 105], is that it combines structure exploitation capabilities of interior point methods with the warm-starting capabilities of active set methods; in particular it comes with only a linear runtime complexity in the horizon length. Note that in contrast to classical active-set methods this approach permits several active-set changes in each Newton-type iteration. Based on original ideas from [106] and [107] the stage coupling constraints connecting the MPC problem over the prediction horizon are dualized and the resulting QP is solved in a two level approach, using a non-smooth Newton method in the multipliers of the stage coupling constraints on the higher level, and a primal active-set method in the decoupled parametric QPs of each stage on the lower level. Due to the structural equivalence, moving horizon estimation (MHE) can be cast into the same framework and can be solved efficiently using the RTI scheme. In the following, we subsume both problem classes, MHE and MPC, under the term MPC for clarity of the presentation.

Within the RTI scheme we repeatedly need to solve the following subproblem, that can be interpreted as a linear MPC problem, see Chapter 2. Here, we group the optimization variables, state increments Δs_k and control increments Δu_k, in stage variables $z_k = [\Delta s_k', \Delta u_k']'$, for $k = 0, \dots N-1$, and $z_N = [\Delta s_N', 0]'$ for the last stage; cf. (3.1). In the context of the RTI scheme, we are consequently interested in repeatedly solving the following QP in an efficient manner:

$$\min_{z} \quad \sum_{k=0}^{N} \left(\frac{1}{2} z_k' H_k z_k + g_k' z_k \right) \tag{3.62a}$$

$$\text{s.t.} \quad E_{k+1} z_{k+1} = C_k z_k + c_k, \quad \text{for all } k = 0, \dots, N-1, \tag{3.62b}$$

$$\underline{d}_k \leq D_k z_k \leq \overline{d}_k, \quad \text{for all } k = 0, \dots, N. \tag{3.62c}$$

Here, we assume that all first order stage coupling terms C_k have full row rank and that the term E_k have special structure $E = [1_{n_x \times n_x}, 0]$, where 0-matrix has appropriate dimensions. Moreover, we assume full row rank of the stage constraints' matrices D_k.

The main idea of the dual Newton strategy is to decouple the QP stages by dualizing constraints (3.62b). Introducing $\lambda = [\lambda_1', \lambda_2', \dots, \lambda_N']' \in \mathbb{R}^{Nn_x}$ we can express (3.62a) and (3.62b) by the partial Lagrangian function:

$$\begin{aligned} \mathcal{L}(z, \lambda) &= \sum_{k=0}^{N} \left(\frac{1}{2} z_k' H_k z_k + g_k' z_k + \begin{bmatrix} \lambda_k \\ \lambda_{k+1} \end{bmatrix}' \begin{bmatrix} -E_k \\ C_k \end{bmatrix} z_k + \lambda_{k+1}' c_k \right) \\ &= \sum_{k=0}^{N} L_k(z_k, \lambda_k, \lambda_{k+1}), \end{aligned} \tag{3.63}$$

where we define zero matrices $E_0 = C_N = 0_{n_x \times n_x}$ and redundant multipliers $\lambda_0 = \lambda_{N+1} = 0_{n_x \times 1}$ only for notational convenience.

By elementary Lagrangian duality theory the primal QP (3.62) is equivalent to

$$\begin{aligned} \max_{\lambda} \quad \min_{z} \quad & \sum_{k=0}^{N} L_k(z_k, \lambda_k, \lambda_{k+1}) \\ \text{s.t.} \quad & \underline{d}_k \leq D_k z_k \leq \overline{d}_k, \quad \text{for all } k = 0, \ldots, N. \end{aligned} \tag{3.64}$$

Observing that (3.64) is separable in stage variables z_k, the problem (3.62) can be written in as

$$\max_{\lambda} f^*(\lambda) = \max_{\lambda} \sum_{k=0}^{N} f_k^*(\lambda), \tag{3.65}$$

where

$$\begin{aligned} f_k^*(\lambda) \quad = \quad \min_{z_k} \quad & \frac{1}{2} z_k' H_k z_k + \underbrace{\left(g_k' + \begin{bmatrix} \lambda_k \\ \lambda_{k+1} \end{bmatrix}' \begin{bmatrix} -E_k \\ C_k \end{bmatrix} \right)'}_{p_k(\lambda)} z_k + \lambda_{k+1}' c_k \\ \text{s.t.} \quad & \underline{d}_k \leq D_k z_k \leq \overline{d}_k \end{aligned} \quad . \tag{3.66}$$

Under the assumption that a feasible solution for (3.62) exists, it was shown in [82] and [76] that $f^*(\lambda)$ exists and further is a concave, piecewise quadratic, and once continuously differentiable function. The unconstrained piecewise-quadratic program (3.65) is solved by employing a non-smooth Newton method, as originally proposed in [106].

Due to its temporal coupling, problem (3.65) possesses a specific structure that can be exploited for an efficient solution. In particular, here we are interested in an efficient solution for step direction $\Delta\lambda$

$$-\frac{\partial^2 f^*}{\partial \lambda^2}(\lambda_i) \Delta\lambda = \frac{\partial f^*}{\partial \lambda}(\lambda_i) \tag{3.67}$$

that is typically the computational bottleneck; λ_i denotes the current iterate. The dual gradient is easily seen to only depend on two neighboring stages in each block λ_k, and the dual Hessian possesses a block tri-diagonal structure as only neighboring multipliers λ_k, λ_{k+1} can have a joint contribution to f^*:

$$\frac{\partial f^*}{\partial \lambda}(\lambda) = \begin{bmatrix} \frac{\partial f_0^*}{\partial \lambda_1} + \frac{\partial f_1^*}{\partial \lambda_1} \\ \frac{\partial f_1^*}{\partial \lambda_2} + \frac{\partial f_2^*}{\partial \lambda_2} \\ \vdots \\ \frac{\partial f_{N-1}^*}{\partial \lambda_N} + \frac{\partial f_N^*}{\partial \lambda_N} \end{bmatrix}(\lambda), \quad \frac{\partial^2 f^*}{\partial \lambda^2}(\lambda) = \begin{bmatrix} \frac{\partial^2 f^*}{\partial \lambda_1^2} & \frac{\partial^2 f^*}{\partial \lambda_1 \lambda_2} & & \\ * & \frac{\partial^2 f^*}{\partial \lambda_2^2} & \ddots & \\ & \ddots & \ddots & \frac{\partial^2 f^*}{\partial \lambda_{N-1} \lambda_N} \\ & & * & \frac{\partial^2 f^*}{\partial \lambda_N^2} \end{bmatrix}(\lambda). \tag{3.68}$$

Knowing this special structure, the solution of the (possibly regularized) Newton system requires $\mathcal{O}(N n_x^3)$ FLOPs [76]. Furthermore, in [76] the authors propose a method for parallel factorization of the dual Hessian.

Each stage problem (3.66) has a fixed second order term H_k and a parametric first-order term $p_k(\lambda)$. An efficient method to solve such parametric problems repeatedly for changing parameter values λ is the so-called online active-set strategy [24]. For algorithmic details on explicit derivative computation we refer to [76]. By using this approach, the computational complexity to solve (3.66) is cubic in number of stage variables.

In the special case when H_k is a diagonal matrix and D_k is an identity matrix, i.e. only bounds on states and controls of the MPC problem exist, the optimal solution z_k^* can conveniently be computed by component-wise *clipping* of the unconstrained solution, as presented in [82]. In this special case, the computation effort to solve (3.66) is negligible and can be easily parallelized [82].

3.5 Numerical Simulations

In this section we perform a series of tests to bechmark the NMPC solvers available in the ACADO Code Generation Tool suite. In particular, we test the following four types of controllers.

qpoases_cn2	The solvers are based on the $\mathcal{O}(N^2)$ condensing routine and embedded version of qpOASES [90] QP solver is used to solve the condensed QP.
forces	Those solvers employ the structure exploiting IPM QP solver FORCES [108]. For each NMPC problem, a tailored QP solver is auto-generated.
qpdunes	The underlying QP is solved by the structure exploiting QP solver qpDUNES [109] that implements the dual Newton strategy.
hpmpc	The HPMPC [110] based NMPC solvers utilize a structure exploiting IPM.

The two benchmarks, a chain of masses connected by springs and an inverted pendulums, are formulated in such a way that all four types of controllers can handle them. Certain solvers in the ACADO CGT suite can handle more complex NMPC formulations, however, they are not covered.

All simulations are performed on a desktop computer equipped with one 3.4 GHz Intel Core i7-3770 CPU, running the 64-bit version of Ubuntu Linux 14.04. All generated code generated by ACADO CGT and FORCES is compiled with the Clang 3.4 compiler, using optimization flag `-O3`. The HPMPC and qpDUNES libraries are compiled with the same compiler. In addition, HPMPC library is compiled with an additional compiler flag `-mavx`, such that the full capabilities of the library are exploited by the target CPU. Execution times are measured with the Linux function `clock_gettime()` that yields a resolution in the nanosecond range. All tests are ran in a single thread mode.

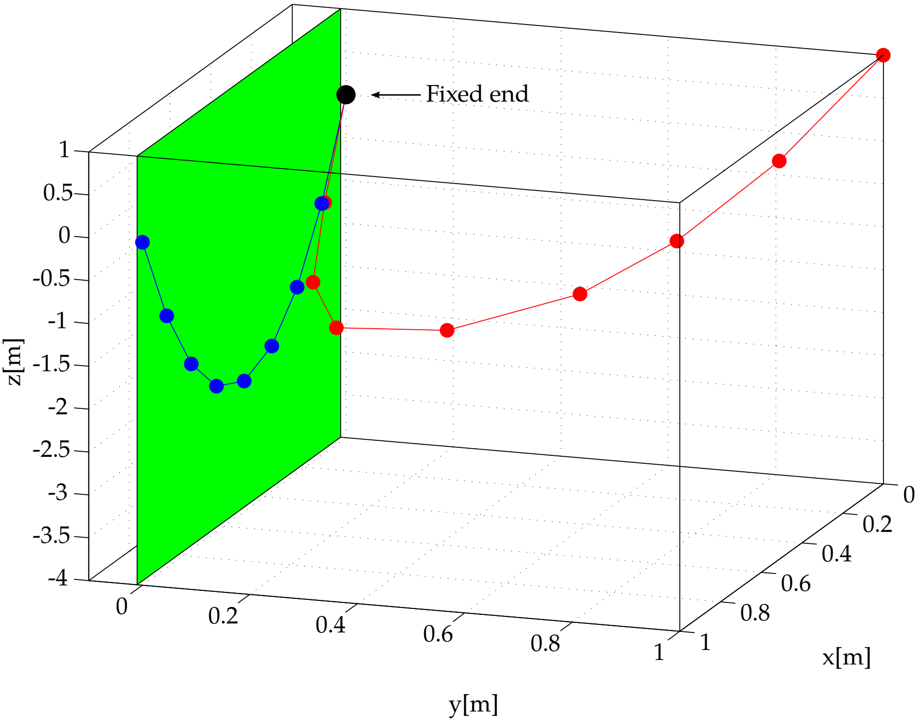


Figure 3.1: A chain of masses connected with springs. The blue chain denotes equilibrium state, the red chain illustrates disturbed state. The green patch represents a wall and black dot the fixed point of the chain.

3.5.1 Chain of Masses Connected by Springs

The first benchmark problem is a chain of masses connected with springs [111], illustrated in Figure 3.1. The dynamics of this system is nonlinear due to nonlinear spring forces. Furthermore, this model is easily scalable, meaning that the problem complexity can be increased by adding additional masses to the chain. The goal of each controller is to move back the chain from its disturbed to its equilibrium state, respecting upper and lower bounds on both states and input variables.

Simulation Model

We consider a chain with M balls of equal mass m that are connected with springs. All springs have equal rest length L and spring constant D. At both ends additional identical springs are attached. One end of the chain is attached to a fixed point, while the velocity of the other end $u = (u_x, u_y, u_z)$ can be controlled. The center of each ball is represented by a 3-dimensional coordinate $p_i = (p_{x,i}, p_{y,i}, p_{z,i})$. For the chain comprising M balls, an ODE that describes model dynamics has $n_x = (2M + 1) \cdot 3$ states and $n_u = 3$ controls. In the rest of the paper it is assumed that all states can be measured. For all details specific to the model we refer to [111].

Simulation Scenario

The continuous ODE model is parametrized by the multiple shooting technique using intervals of $T_s = 200$ ms. For reliable integration the model and sensitivity generation we chose implicit Gauss-Legendre integrator of order four. One integration step per shooting interval was used for a chain with up to three masses and the two steps per shooting interval were used for chains with four and five masses. This setting enabled us to run successfully all solvers for nearly all test problems. All simulations are noise-free and the controllers are supplied with the perfect state feedback, as predicted by a particular controller. The formulation of the NMPC is the same as in the original paper [111], with the two exceptions. First, we impose simple input bounds

$$-1 \text{ m/s} \leq u_j \leq 1 \text{ m/s}, \quad j \in \{x, y, z\},$$

and a set of tight state bounds imposed by a wall next to the equilibrium plane (see Figure 3.1):

$$-0.01 \text{ m} \leq p_{y,i}, \quad i = 0, \ldots, M - 1.$$

Compared to the simulation settings in [111], the wall is placed at a closer distance to the equilibrium plane. This makes the problem even harder to solve, and this was a design decision made to increase the number of active constraints during the simulations. Note that for condensing based NMPC state bounds are transformed to affine inequalities, in the form (3.9c). Relevant controller QP dimensions are summarized in Table 3.12. Therein, the number of variables, bounds and constraints in the corresponding QPs are denoted with

Table 3.12: Relevant dimensions for the chain of masses benchmark.

			Sparse QP			Condensed QP		
M	n_x	n_u	n_v	n_b	n_c	n_v	n_b	n_c
1	9	3	$12N$	$8N$	0	$3N$	$6N$	$2N$
2	15		$18N$	$9N$				$3N$
3	21		$24N$	$10N$				$4N$
4	27		$30N$	$11N$				$5N$
5	33		$36N$	$12N$				$6N$

n_v, n_b, n_c, respectively. The second difference is the addition of the terminal penalty in the objective that is equal to the part of the stage cost related to the penalization of the state vector. Our experience showed that terminal penalty with the weighting matrix coming from the solution from the discrete Riccati equation was unnecessary to successfully solve the problem.

Simulations are performed for five different variants of the chain mass problem, namely for $M = 1, \ldots, 5$. For each dynamic system four types of NMPC controllers, as explained above. Each of those four types of the controllers is tested for six different horizon lengths N: $5, 10, 20, 30, 40$, and 50. In total each problem is solved by 24 different controllers or, in other words, each of the four types of the controllers (based on the QP solver in use) is tested on 30 different problems. Longer horizon lengths, based on our experience, were unnecessary to successfully solve the benchmark problems – to bring the chain to the equilibrium state.

3.5.2 Double and Triple Pendulums

The second type of benchmark problems is based on the work presented in [112, 113]. Namely, the papers deal with swing-up and stabilization of the double- and triple-pendulum on a cart – both simulations and experimental validations. The authors developed high fidelity models of the corresponding experimental setups. We use the two models and do swing-ups with different NMPC controllers. The models for both pendulums are highly stiff and represent nonlinear and unstable dynamics. Thus, fast sampling times are required, much faster than the time needed for a swing-up. Consequently, long prediction horizons needed to be used. We test each of the four types of the

Table 3.13: Relevant dimensions for the pendulum benchmark.

			Sparse QP			Condensed QP		
Pendulum	n_x	n_u	n_{v}	n_{b}	n_{c}	n_{v}	n_{b}	n_{c}
Double	6	1	$7N$	$6N$	0	N	$2N$	$4N$
Triple	8		$9N$					

NMPC controllers on both pendulum examples for different horizon lengths $N = 50, 60, \dots, 150$.

The cart position is denoted by p and the velocity with v. The cart is actuated by a strong motor with highly dynamic lower level controller. In the original publication is chosen, under certain assumptions, that the input to the system is the cart acceleration a. The state vector of the double pendulum reads

$$x = [p, v, \theta_1, \omega_1, \theta_2, \omega_2]', \tag{3.69}$$

where θ_i and ω_i are the angle and the angular velocity of segment i. Similarly, the state vector for the triple pendulum is defined as

$$x = [p, v, \theta_1, \omega_1, \theta_2, \omega_2, \theta_3 \omega_3]'. \tag{3.70}$$

For the double pendulum, in the least-squares objective for the NMPC we define the output functions as:

$$\begin{aligned} h(x, u) &= [p, v, a, \cos\theta_1, \sin\theta_1, \omega_1, \cos\theta_2, \sin\theta_2, \omega_2]', \\ h_N(x) &= [p, v, \cos\theta_1, \sin\theta_1, \omega_1, \cos\theta_2, \sin\theta_2, \omega_2]'. \end{aligned} \tag{3.71}$$

Extension for the triple pendulum follows accordingly. As we are uninterested whether a segment makes turns during the swing-up, we choose the output functions with sines and cosines of the segment angles. Finite length of the rail on which the cart moves and the limited torque the motor can provide are translated in input and state bounds in NMPC formulations. The dimensions relevant for this benchmark are summarized in Table 3.13.

In the original papers short sampling times were used, $T_{\mathrm{s}} \leq 2\,\mathrm{ms}$. Our simulations revealed that by using the implicit Runge-Kutta integrators of order 4 with 2 steps per multiple shooting interval with sampling time $T_{\mathrm{s}} = 20\,\mathrm{ms}$ was sufficient. Similar to the chain of masses benchmark, all simulations executed within the pendulum benchmark are noise-free.

3.5.3 Performance Profiles

One common way to compare different optimization solvers is by the *performance profiles* [114]. Let us consider a benchmark with a set of problems $\mathcal{P}$ and a set of solvers $\mathcal{S}$. The time solver $s \in \mathcal{S}$ needs to the solve problem $p \in \mathcal{P}$ is denoted by $t_{p,s}$. The *performance ratio* is a quantity that compares the time solver s needs to solve problem p compared to the fastest solver:

$$r_{p,s} = \frac{t_{p,s}}{\min t_{p,s} : s \in \mathcal{S}}. \tag{3.72}$$

If the solver s cannot solve the problem p, the ratio takes an arbitrary value $r_m \geq r_{p,s}, \forall p \in \mathcal{P}, s \in \mathcal{S}$. Using the performance ratio one can make a plot of the performance profile for solver s. Such a profile is a plot of the cumulative distribution of the performance ratio:

$$\rho_s(\tau) = \frac{1}{\mathrm{card}(\mathcal{P})} \mathrm{card}\left(p \in \mathcal{P} : r_{p,s} \leq \tau\right). \tag{3.73}$$

The value $\rho_s(1)$ reflects the probability that that solver is going to perform better than any other solvers. Increasing τ, we get the percentage of problems the solver s will solve within the time relative to the time needed by the best solver. In a general benchmark, solvers that exhibit high values of $\rho_s(\tau)$ should be favored.

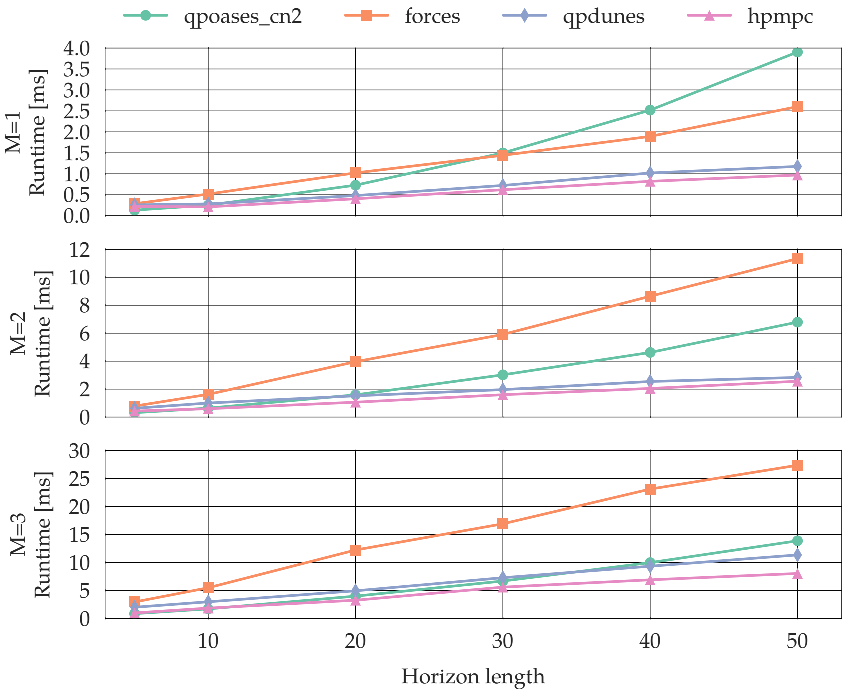


Figure 3.2: Maximum execution times of four different NMPC solvers for a chain of masses with up to three masses in the chain.

3.5.4 Results

Simulation results for the chain of masses benchmark for up to three masses are presented in Figure 3.2. All plots show the maximum run-times for all types of controllers. Simulation results show that in an extreme case for a test problem with 9 states, 3 control inputs ($M = 1$) and 50 prediction intervals the execution time of the HPMPC based solver by factor of 4 lower than the condensing based approach.

The condensing based approach is competitive for short horizon length in comparison with FORCES and qpDUNES based solvers. This is in concordance with the observations commonly found in the literature that the condensing based approach is faster than the sparse solvers for shorter horizon lengths. The *break-even* point moves higher on the scale for longer horizon lengths, mainly

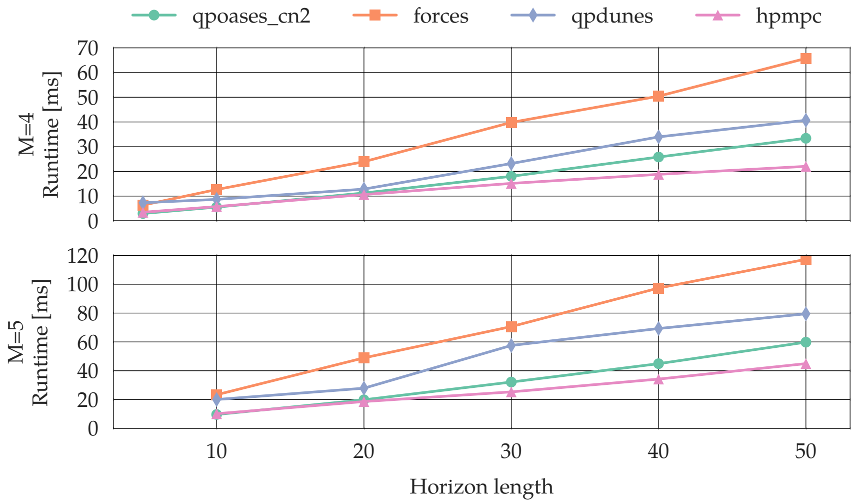


Figure 3.3: Maximum execution times of four different NMPC solvers for a chain of masses with four and five masses in the chain.

because the sparse structure exploiting solvers have $\mathcal{O}(n_x^3)$ complexity.

Although the FORCES generated code for the QP solver is well optimized, the loops that call lower level linear algebra routines are fully unrolled. Consequently, this makes the compiled code slower. In comparison with [10], here we use the superior $\mathcal{O}(N^2)$ condensing on the one hand and an updated version of qpOASES on the other hand. Recent modifications to qpOASES solver led to speed-ups of up to 40 % for building and factorizing of the reduced Hessian.

The results obtained for the solver based on qpDUNES are similar to the results presented in [105], where it was observed that the sparse solver outperforms the condensing one for yet short horizon lengths when the ratio n_x/n_u is small.

The HPMPC based solver seems to always produce outstanding execution times. This is mainly because of the fact the code is well optimized for the architecture where the code is executed. The only case when the condensing solver is actually faster is for $M = 1$ and $N = 5$ – here the condensed QP has only 15 variables, compared to 60 variables in the sparse QP.

The maximum execution times for the chain of masses example for four and fives masses in the chain are shown in Figure 3.3. For such a high ratio n_x over n_u we can observe that even highly optimized solvers based on HPMPC QP solver cannot significantly outperform the condensing based solvers for short horizon lengths. However, for longer horizons, HPMPC solver shows again superior performance. Although this is unlikely from pure algorithmic point of view, it is a consequence of a well optimized implementation. None of the solvers was able to solve the chain of masses with $M = 5$ and short horizon $N = 5$. In all cases infeasibility was detected[3]. The cause of the failures is related to the insufficiently long horizon and/or lack of a stabilizing terminal cost and accompanying terminal constraint; for more information about stabilizing MPC schemes we refer to e.g. [115, 116].

For a fixed number number of prediction intervals N, the condensing based NMPC solver has Nn_u optimization variables, i.e. independent of the number of the states n_x. However, an increase of the number of states n_x leads to an increased number of affine constraints (3.9c) – MN. This directly makes an NMPC problem harder to solve, i.e. possibly requires more working set recalculations when using an active set based QP solver, resulting in longer runtimes, as it can be observed in Figure 3.2 and Figure 3.3.

The number of iterations for this benchmark is summarized in Figure 3.4. The condensing based solver with qpOASES typically show higher number of iterations, which is expected. On the other hand, the maximum number of iterations qpDUNES solver needs is typically low. It only becomes higher than for HPMPC based solver for some of the more challenging problems with $M = 4, 5$. The difference in number of iterations between the two IPM solvers, FORCES and HPMPC, is due to the fact they implement different line search and initialization procedures.

---

[3] The infeasibility was confirmed by the qpOASES QP solvers. For the other QP solvers we observed stagnation of the iteration process, eventually hitting the high iteration limits.

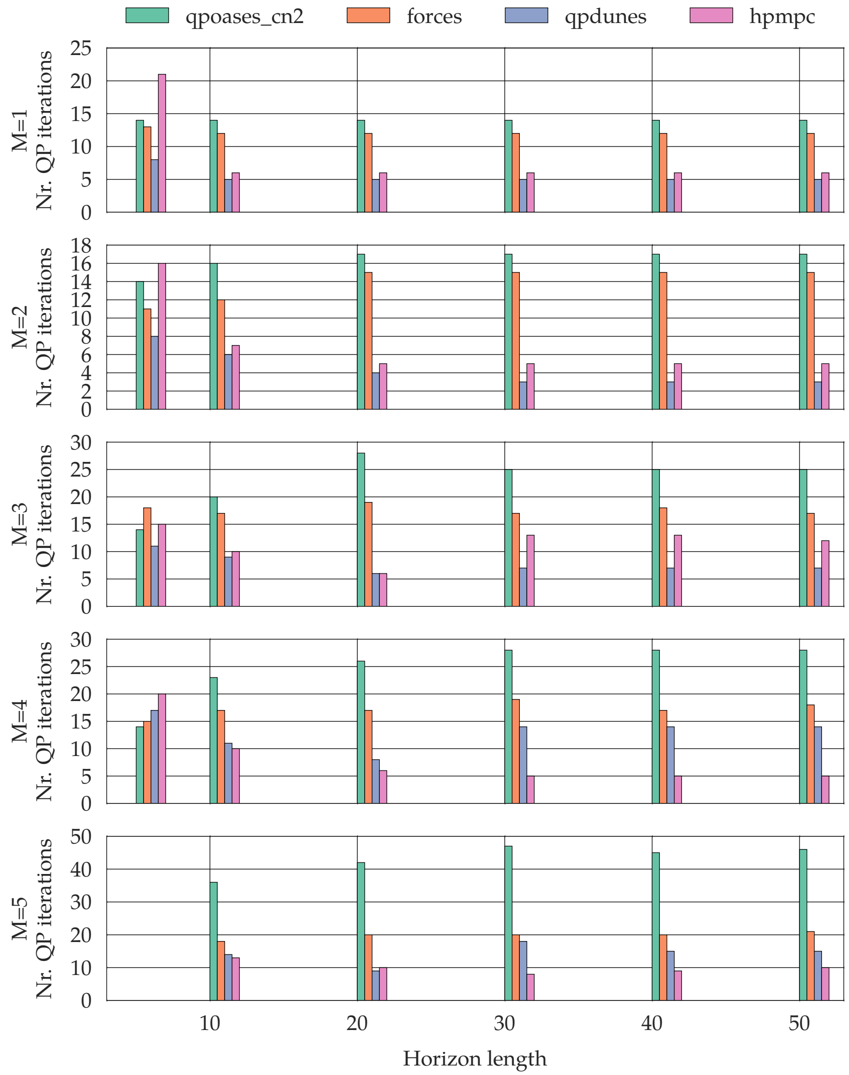


Figure 3.4: Maximum number of QP iterations for different NMPC solvers in the chain of masses benchmark.

The maximum feedback times for the benchmark are shown in Figure 3.5. For the low ratio n_x/n_u and the short horizon, all solvers seem to produce fast feedback. Most notably for the $M = 1$ and $N = 5$. The exception is the condensing based solver that outperforms all others. This is expected, as stated above, because the condensed QP dimensions are low. Prolonging the horizon makes condensing less competitive, and the HPMPC and the qpDUNES based solvers dominate. The FORCES based solver outperforms the condensing based solver only for $M = 1$, $N \geq 32$. The plots suggest that for the low state/control ratios and long horizons the qpDUNES and the HPMPC based solver should be preferred. With the increase of number of masses, i.e. increase of the number of the states, condensing becomes more and more competitive. In particular, $M = 4, 5$ the condensing based solver can compete with the HPMPC up to 18 and 22 prediction intervals, respectively. The plots indicate that the qpDUNES based solver will eventually overpower condensing based approach for $N > 50$ and $M > 4$. Moreover, the FORCES based solver is expected as well to eventually provide faster feedback than the condensing based approach – because of the dominant $\mathcal{O}(N^3)$ complexity of the condensing approach.

Comparing the total execution times in Figure 3.2 and Figure 3.3 to the feedback times in Figure 3.5 it can be concluded that condensing based approaches are more competitive from the viewpoint of providing the fast feedback than overall execution times. This is the consequence of relocation of the condensed Hessian computation to the preparation step of the RTI.

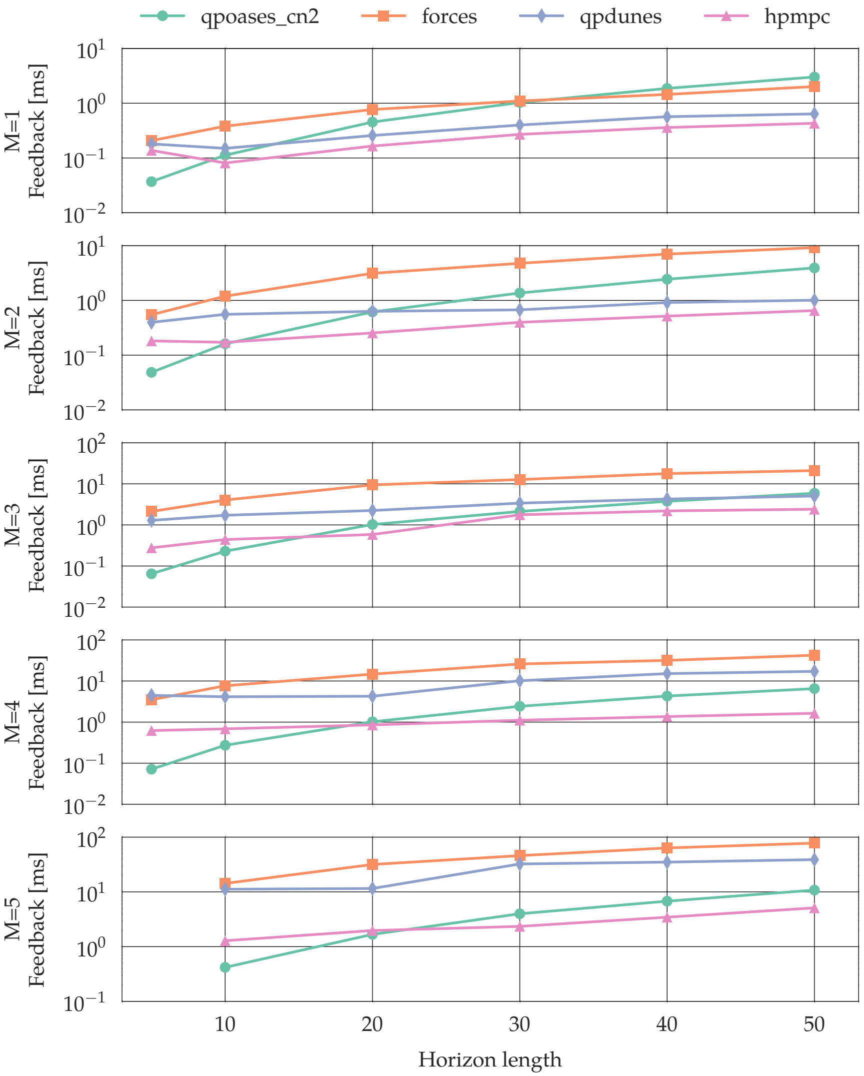


Figure 3.5: Maximum feedback times for the chain of masses benchmark.

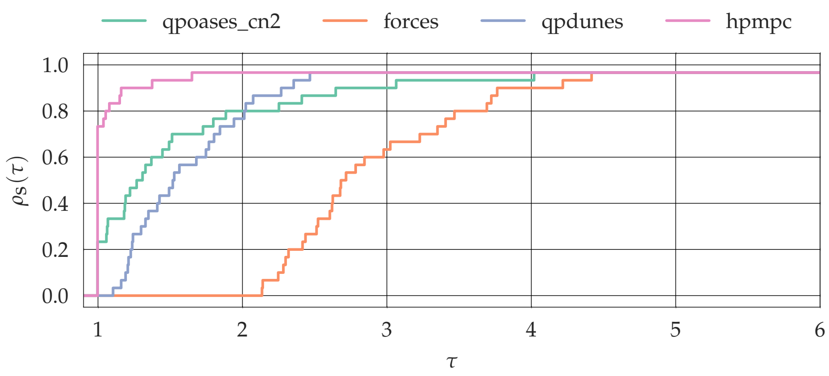


Figure 3.6: Performance profiles for the chain of masses benchmark.

The performance profiles, see § 3.5.3, for the chain of masses benchmark are presented in Figure 3.6. As expected, the HPMPC based NMPC solver shows the highest probability to win over all other; this can be observed for $\tau = 1$. The plot also suggests that the condensing based solver should be faster than qpDUNES based solver up to the factor of 2 of the best solver. Neither of the solver reaches $\rho_s = 1$, as none of the solvers was capable to solve the problem with $M = 5, N = 5$.

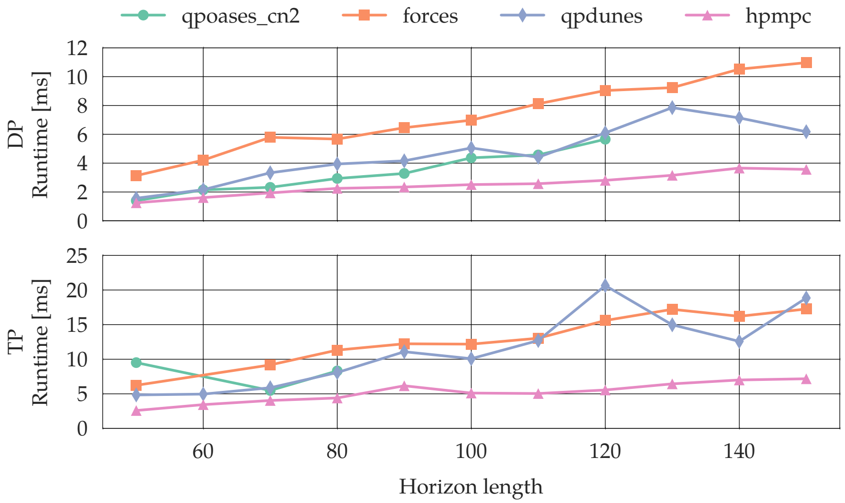


Figure 3.7: Maximum execution times of different NMPC controllers for the double (DP) and the triple (TP) pendulums.

The maximum runtimes of the NMPC solvers applied for swing-up of the double and triple pendulums are presented in Figure 3.7.

For the double pendulum, we can observe that except qpOASES based solver, all other solvers manage to solve all problems successfully. The condensing based solver fails because of the bad conditioning of the problem – consequently the failure of the Cholesky factorization of the condensed Hessian. The FORCES solvers shows the slowest performance because of the large number of function calls they have to perform – in the generated solvers all routines for calling linear algebra routines are unrolled. The qpDUNES solvers show similar performance to the condensing ones. Finally, the HPMPC again shows that it is the one with the most optimized code. This can be concluded based on small fluctuations of execution times compared to the number of iterations.

We observe that the condensing based solver fails to solve most of the problems for the triple pendulum – in fact, it manages to solve only three. For $N = 60$ infeasibility was detected[4]. The cause of the failure for $N \geq 90$ was the bad conditioning of the reduced Hessian. When it manages to solve the problem, a large number of iterations was observed. Almost all structure exploiting solvers

---

[4] The infeasibility is the consequence of the shifting initialization for the RTI scheme.

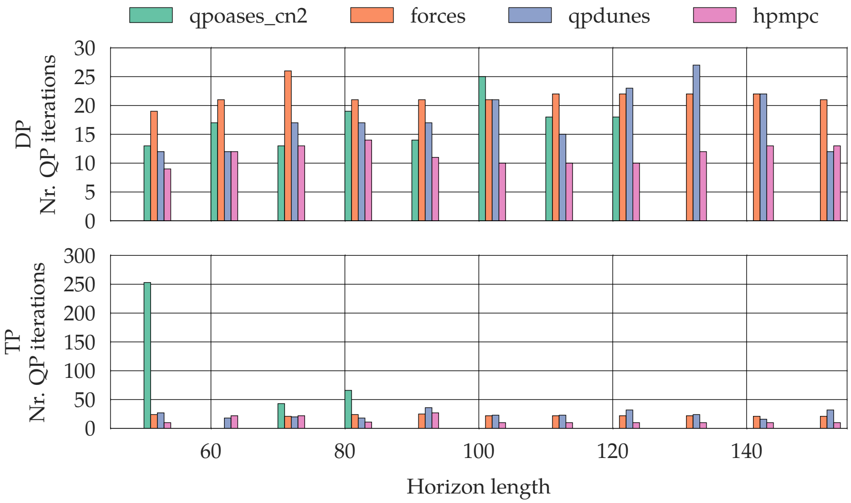


Figure 3.8: Maximum number of iterations for different NMPC solvers in the pendulum benchmark.

manage to successfully solve all the problems. The exception is the FORCES based solver which fails to solve the problem for $N = 60$[5]. As for the previous benchmark, it seems that the size of the code FORCES generated solvers heavily influences the execution times. The fastest performance in this benchmark was observed for the HPMPC based solvers. Cache efficiency of the solvers is confirmed with small fluctuations of the execution times compared to observed number of iterations Figure 3.8.

The maximum number of iterations for the double pendulum for all solvers except the HPMPC is similar, but the HPMPC shows a bit less iterations. To our surprise, the number of iterations of qpOASES QP solver was similar to the other solvers – typically the active-set solvers need more iterations than IPM or dual Netwon strategy based QP solvers. The condensing based solvers for the triple pendulum need more iterations than the other. Comparing the number of iterations qpDUNES based solver needs to solve the problems with the corresponding execution times in Figure 3.7, a conclusion can be made that the cache efficiency of the solver might be improved.

Let us take a closer look at the conditioning problem observed for condensing

---

[5] The solver reached the maximum number of iterations, set to 10000.

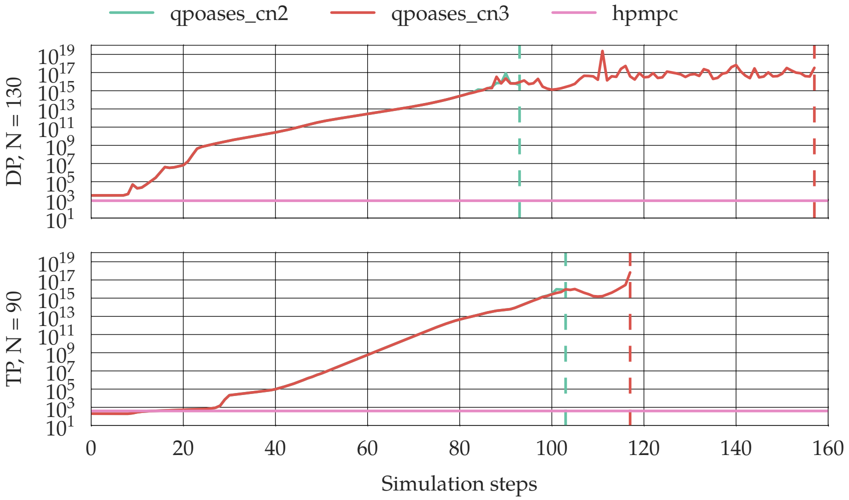


Figure 3.9: Conditioning within the condensing based solvers in the pendulum benchmark. For clarity, the moments of failure are denoted with dashed lines.

based NMPC solvers. The task of the controller is to bring the pendulums to the upright position, that is, to the *unstable* equilibrium. Low friction of the mechanical structure makes the system, thus the model, very sensitive to perturbations in the vicinity of the unstable equilibrium. Consequently, it is reasonable to expect badly conditioned sensitivity matrices A_k and B_k in that area. Loosely speaking, the condensing approach proposes the removal of the state trajectory from the sparse QP, using the affine map (3.10). In the affine map, matrices E and C are the ones to propagate the system dynamics. Conditioning of the propagation matrices C and E highly depends on conditioning of the sensitivity matrices and the horizon length N. This is due to row-wise propagation of sensitivities by multiplication. Consequently, the conditioning of the condensed Hessian is directly affected by the conditioning of the propagation matrix E[6]. To illustrate the issue on the concrete problems, we present in Figure 3.9 the results from two test cases. Therein, we also included simulation results for the NMPC solver with the classical condensing; the solver is denoted with *qpoases_cn3*. The first test is for the double pendulum (DP) and the controller with $N = 130$. The second test case is for the triple pendulum (TP) and the controller with $N = 90$. As the controller brings the system closer to the

---

[6] see § 3.2 for details about building of the condensed Hessian.

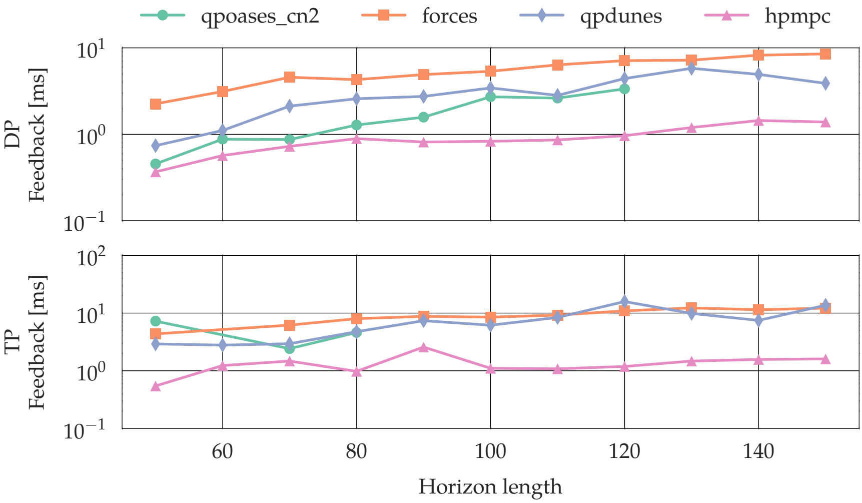


Figure 3.10: Maximum feedback times for the pendulum benchmark.

equilibrium, the conditioning of the condensed Hessian gets worse for the both solvers: qpoases_cn2 and qpoases_cn3 [7]. The low, constant, condition number for the sparse QP is obtained in a simulation with the HPMPC based solver. The both condensing controllers fail due to ill-conditioning of the reduced Hessian inside the QP solver[8]. While both solvers produce high condition numbers, the classical condensing based solver survives for a little longer and eventually fails. The results presented in Figure 3.9 give strong indications that the condensing approach should be avoided in the situations involving unstable dynamics and long horizons. Furthermore, the results demonstrate that for the considered test cases the conditioning of the condensed Hessian is not compromised by employing the $\mathcal{O}(N^2)$ condensing routine.

The maximum feedback times obtained in the pendulum benchmark are presented in Figure 3.10. It can be distinguished that the qpOASES solver, for the double pendulum, always provides faster feedback than qpDUNES, despite the fact that the overall execution time is sometimes larger. We can also observe that a number of solvers can provide fast feedback in less than 1 millisecond. For the

---

[7] Condition numbers were estimated using the routine `dgesvd` from LAPACK [117]. The routine computes singular value decomposition (SVD) for a given matrix, and the condition number is computed as the ratio between the maximum and the minimum singular value.

[8] The both implementations employ the same implementation of the qpOASES QP solver.

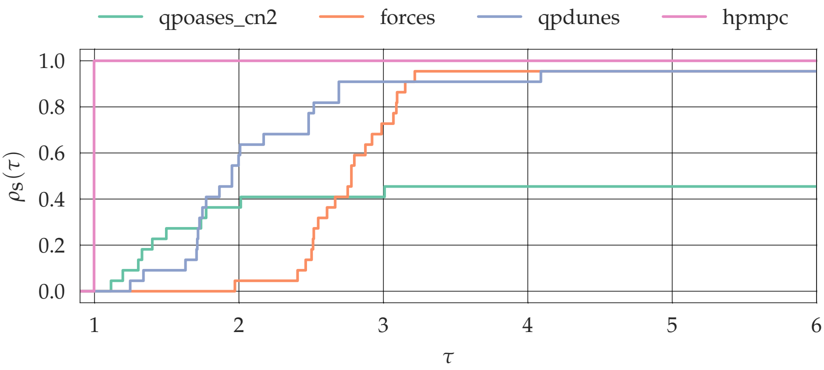


Figure 3.11: Performance profiles for different NMPC solvers in the pendulum benchmark.

double pendulum, the HPMPC solver provides feedback in microseconds for $N \leq 120$ and two times for the triple pendulum. The qpOASES and qpDUNES based solvers can provide feedback in the microsecond range as well, but for shorter horizons.

The joint performance profiles for the pendulum benchmark are summarized in Figure 3.11. The plots suggest that after the HPMPC based solver the qpOASES one might be most effective for shorter horizons and that qpDUNES might be more effective for longer ones. The FORCES based solvers might be more efficient than qpDUNES for long horizons – cf. Figure 3.7.

Table 3.14: Percentages of the RTI scheme spent in the feedback phase.

NMPC Solver	Benchmark			
	Chain of masses		Pendulums	
	average	maximum	average	maximum
qpoases_cn2	12	77	19	76
forces	62	81	63	79
qpdunes	20	70	18	77
hpmpc	15	62	17	42

For practical applications it might be beneficial to know the typical percentage of the RTI spent in the feedback phase. For this purpose, we summarized all information from the two benchmarks in Table 3.14. The average percentages are calculated from the average runtimes, and the maximum percentages are calculated from the maximum runtimes. In average, the feedback phase is much shorter than the preparation phase, as commonly stated in the literature. For a well optimized implementation, typically up to 20 % is spent solving the QP and possibly condensing the sparse QP. The maximum runtimes should be evaluated as case-specific – depending on the nature of the problem, limit on the number of iterations and so on. The user should be aware that the maximum runtimes can be significantly longer than the average. Reasons for high maximum percentages include optimized implementations of the fixed-step integrators and the condensing procedures.

3.6 Conclusions

This chapter gave an overview of QP solvers tailored for possibly nonlinear MPC and MHE applications that were found to be useful and successful in the literature but also in the scope of this thesis. In essence, two approaches were reviewed. The sparse one, where efficiency is gained by exploiting in depth the structure of the sparse QP coming from MPC and MHE formulations. The second approach based on condensing proves to be efficient if the sparse QP structure is exploited to reduce the original QP to a dense one with typically much smaller dimensions. Three condensing approaches have been studied in detail, for the both cases: QPs coming from MPC and MHE formulations. Within § 3.2, the $\mathcal{O}(N^2)$ approaches have been extended for partial condensing.

We compared four different NMPC solvers on two benchmarks in § 3.5. In particular, we compared NMPC solvers that employ the $\mathcal{O}(N^2)$ condensing and the qpOASES QP solver with NMPC solvers that employ three different structure exploiting QP solvers. Out of those three, two are based on the primal-dual interior point methods. One of them, FORCES, is an auto-generated QP solver. The other one, HPMPC, is the library consisting of routines specifically optimized for solving the *kernel* of the interior-point methods: the discrete Riccati equation. The fourth type of the NMPC solvers is based on the newly developed dual decomposition strategy that should utilize advantages of both worlds: low iteration count of interior-point methods and capability to be warm-started like the active-set approaches.

Short horizons For the very short horizons, e.g. $N \leq 5$ and the low state/-control ratio, the condensing should be favored because the QP dimensions are small. In this case, the condensing based solvers provide the fastest feedback as well as the total times. Keeping the horizon short, but increasing the state/control ratio, the condensing approach continues to provide the best results. With the increase of the ratio, the condensing approach becomes even more competitive while increasing of the horizon length. It should be further observed that relocation of the computations in the condensing based approach makes condensing based solver even more competitive from the fast feedback point of view. Next to the condensing approach, qpDUNES and HPMPC based solvers show to be competitive at short horizon lengths for the low state/control ratio.

Long horizons With the increase of the horizon length, the condensing and the QP solution become expensive. Eventually, the $\mathcal{O}(N^3)$ complexity of the solution makes them the slowest for long enough horizons. Furthermore, our benchmarks revealed that for the highly nonlinear, possibly unstable, systems the condensing procedure leads to bad conditioning. Although appealing, and reasonable, the code-generation approach FORCES employs seems not to be the only nor the best path to achieve fast execution times of an interior-point QP solvers. The slow performance of the solvers is suspected to be a consequence of the huge amount of the generated code for longer horizons. According to our benchmarks, qpDUNES and the HPMPC structure exploiting QP solver should be favored for longer horizons. The performance of the qpDUNES based solvers is promising, especially for the lower state/control ratios. Low iteration counts and low execution times per iteration renders HPMPC the solver with the best overall performance. The implementation demonstrates that IPMs for MPC/MHE applications can be well optimized, even for larger state/control ratios. While the feedback time with FORCES solvers is always longer than the preparation step, qpDUNES and HPMPC solvers maintain the average ratio between the feedback and the preparation phases low.

4

The ACADO Code Generation Tool

During the design phase of an MPC controller or an MHE estimator a control engineer tries out different formulations. Different objectives are tested to achieve satisfactory performance in simulations. Constraints are added to meet certain design requirements and account for physical limitations. Model equations might need to be adjusted to fit within the set of features of a particular software package. Once satisfied with a preliminary design, one needs to think about *how to adjust the model and/or formulations such that the controller/estimator can be executed in real-time at desired sampling rate using available software?* This step requires in-depth knowledge of a set of features of the software package of choice. Knowing software advantages and limitations the designer usually adjusts e.g. horizon lengths, sometimes model equations, or number of constraints. Moreover, in this step the designer makes choices such as the QP solver, integration routine and number of integration steps depending on the system dynamics. Eventually control engineer becomes satisfied and chooses to try out the controller in a real-time simulator and later in experiments. At this point all relevant dimensions to the controller/estimator formulations are known. This particular fact strongly motivates code generation of a customized solver for a particular formulation. In such a solver fixed problem dimensions can help compilers to better optimize the code leading to fast execution times. Another important benefit is that by fixing the problem dimensions dynamic memory allocation is avoided, which is the preferred way of memory management in real-world applications.

The ACADO Code Generation Tool (CGT) [7, 118, 119] is the software tool designed primarily for the purpose to generate customized solvers and integrators with the aim to be used in fast applications with milli- and micro-second sampling times. The tool is a module within the open-source software package ACADO Toolkit [120] – a toolkit for automatic control and dynamic optimization. The tool was originally inspired by the software package `CVXGEN` [121] that allows the user to generate customized interior-point solvers for small-scale LP and QP problems as arising in linear MPC problems. A somewhat similar software package `AutoGenU` [122] generates source code implementing the continuation/GMRES method.

The main idea is to tailor the auto-generated code to the specific problem structure and optimize for fast execution based on a symbolic problem formulation. The number of required operations in the evaluation of the nonlinear right-hand side of the differential equation, objective and constraints as well as in the associated derivatives is optimized for the particular formulation. The derivatives are symbolically simplified employing automatic differentiation tools and taking into account zero-entries in the Jacobian. Similarly, the tool exports tailored fixed-step Runge-Kutta methods for integration of the model equations and generation of corresponding sensitivity information. The underlying structured QP is solved either directly, or after the condensing step.

Next to the primary role, recent improvements to both the generator and the generated code made possible to use the tool for rapid prototyping. Namely, as code generation and compilation became faster, it became convenient to use the tool in the design phase. This valuable use case became attractive due to interfaces of the ACADO Toolkit as well the generated code to MATLAB & Simulink as well as to Python language using the software package rawesome [123].

Features of the toolkit are summarized in § 4.1. Interfaced QP solvers are outlined in § 4.2. Structure of an exported QP solver is described in § 4.3. Several known real-world applications of the ACADO CGT solvers and/or integrators are listed in § 4.4.

4.1 Features

The ACADO Code Generation tool exports efficient self-contained C-code that implements the Gauss-Newton RTI scheme for both the nonlinear MHE and the nonlinear MPC. The user interface allows one to specify nonlinear dynamic model equations as well as objective and constraint functions. The tool supports continuous-in-time explicit ODEs, as well as implicit ODEs and DAEs. The OCP formulation is discretized using the direct multiple shooting technique[1]. The solution of the underlying QP can be done using one of the four available QP solvers. The QP can be condensed using either the standard $\mathcal{O}(N^3)$ or the superior $\mathcal{O}(N^2)$ condensing techniques. Afterwards, the condensed QP is solved with the embedded version of the qpOASES solver [90]. Alternatively, the user can opt for structure exploiting QP solvers: FORCES [108] and HPMPC [110] implement interior point methods while the qpDUNES [109] implements the dual Newton strategy; see Chapter 3. Before a solver gets exported, problem structure and dimensions are exploited together with sparsity patterns to remove all unnecessary computations and remove any need for dynamic memory allocations.

The ACADO CGT provides *the complete solution* for doing fast nonlinear MPC and MHE. The automatic differentiation can be done either using the implementation within ACADO or an external one. The recent improvements allow the user to supply external code for symbolic evaluation and link against the generated code, providing more flexibility. The suite of fixed step numerical integrators enables the formulation and handling of models of various complexity in size and stiffness. The ACADO CGT itself does not offer auto-generation of a QP solver, but the choice has been made to interface current state-of-the-art solvers.

In the rest of the section we briefly survey the implemented features; for more details about the implemented features and limitations we refer to the ACADO manual [124].

Nonlinear MPC solvers The original implementation of the tool allowed for restrictive MPC formulations involving penalization of the full state and control vector and bounds on the states and the controls. Moreover, the only way to discretize the OCP was by means of the single shooting approach. Already at

---

[1] The direct single shooting is supported only for $\mathcal{O}(N^3)$ condensing based solvers.

that stage the tool showed promising performance in simulations [7] and in the first experimental validation [9]. Addition of the multiple shooting, nonlinear residual functions in the least-squares objectives and nonlinear constraints made problem specifications possible for more complex applications. Initial applicability of the tool was limited by inefficiency of both the generator and the generated code for longer horizons. This was a direct consequence of the vanilla implementation of the condensing routine. Later extensions led to improved standard condensing and the $\mathcal{O}(N^2)$ condensing routines. Interfaces to structure exploiting QP solvers resulted in reduced computational burden for long horizon formulations. One of the latest improvements extends the scope of the tool to general objective formulations and Exact Hessian RTI scheme [125].

Nonlinear MHE solvers With the aim to solve complex estimation problems with complex dynamics possibly involving (nonlinear) constraints or to efficiently handle the cases when other approaches fail, the tool has been extended to support MHE formulations [11]. This extension showed to be simple, given the similarities between MPC and MHE formulations, see Chapter 2. Since the original MHE implementation, the tool has been extended to support the general Gauss-Newton type formulation. As the arrival cost is sometimes necessary to handle complex use cases, the prototype implementation of the smoothed arrival cost update [73] has been implemented; the complex cases include e.g. parameter estimation and avoidance of long horizons needed when short sampling times are required.

Integrators One of the key components for the efficient MPC and MHE solution is the integrator. The ACADO CGT implements a variety of efficient numerical integration schemes[2]. Using variational differential equations explicit ODEs and corresponding sensitivities can be handled efficiently with explicit Runge-Kutta integrators. Support for implicit integrators has been added [80] to prevent prohibitively long execution times while integrating stiff systems and to tackle more challenging nonlinear dynamic systems represented by DAEs. In addition, linear input and output sub-systems can be exploited leading to great reductions in execution time [126]. Continuous-output integrators meant to be used for multi-sensor fusion are developed within [44] and later verified in a prototype MHE implementation [70]. The implemented integrators can be used in MHE/MPC solvers but also as the stand-alone components. For example,

---

[2] The numerical integrators within the ACADO CGT have been developed by Rien Quirynen.

this feature was used to develop a real-time simulator in Chapter 6 to simulate an implicit DAE with 27 states, 4 controls and 8 outputs at the sampling rate of 1 kHz.

4.2 Interfaces to QP solvers

The exported solver code can use one of the four interfaced QP solvers[3]. In the following we list the interfaced solvers.

qpOASES [90] The QP solver qpOASES implements the efficient online active-set strategy [24]. In particular, the version that is interfaced to exported solver is the special, embedded, variant of the solver that uses the static memory allocation. This solver is used always in conjunction with one of the implemented condensing techniques. The qpOASES solver is provided in the form of self-contained ANSI C++ code, thus creates slightly more dependencies. The solver is freely available under the LGPL license. The solver provides solutions in time proportional to the cube of optimization variables. However, in conjunction with the efficient condensing techniques, the solver shows competitive performance for short to medium horizons and large ratio n_x / n_u.

FORCES [108] The structure exploiting QP solver FORCES [99] implements the Mehrotra's primal-dual IPM. Structure exploitation leads to linear complexity in number of the shooting intervals. The tool supports QP formulations, as well as linear programs (LPs) and QPs with quadratic constraints (QCQPs). Problem formulation is specified in either a MATLAB or Python script. Afterwards, the data is sent to a web-server that generates the custom code for the specific formulation. The generated code can profit from knowledge about the sparsity of the stage Hessian blocks, see § 3.3. In particular, if the stage Hessians are diagonal, which is a common in MPC formulations, factorization of the reduced Hessian becomes trivial. One of the disadvantages of the exported code is that all for-loops that call the low-level linear algebra routines are unrolled. This is the design decision that seemed to be necessary to support general multi-stage problem formulations, i.e. more general multi-stage formulations that arise in optimal control. The FORCES is released as

---

[3] Initially the CVXGEN [127] QP solver was interfaced as well, but the support for this QP solver was dropped because of the low interest.

academically free service, and the generated code is released under the GPL license.

qpDUNES [109] This solver implements the dual Newton strategy [76, 105] that is specifically designed for block structured QPs arising in MPC and MHE. The advantage of the dual Newton strategy is that it combines structure exploitation capabilities of interior point methods with the warm-starting capabilities of active set methods. Consequently, the complexity of the solution is linear in the horizon length. The solver is provided in form of C99 C-code under the industry friendly LGPL license. The solver uses dynamic memory allocation, but all the memory is pre-allocated before the solver is used.

HPMPC [110] This is yet another IPM QP solver that implements the Mehrotra's predictor corrector method. The solver, at the version that is used in this thesis, provides an efficient implementation for QP arising in MPC with box constraints on states and controls. The implementation, at the lower level, implements the discrete Riccati recursion as described in [128]. The key feature that differentiates this solver from the rest is that a kernel for Riccati recursion is well optimized for modern CPU architectures. Special memory layouts are used to reduce both CPU cache misses. Moreover, the tool implements customized linear algebra routines for matrices and vectors of small dimensions optimized for vector instructions. For more details we refer to [96, 100]. The library is provided in form of C99 compatible C-code, released under the LGPL license.

4.3 Structure of the Exported Solver

The structure of an exported solver for MPC or MHE is illustrated in Figure 4.1. The solver itself contains routines for fast evaluation of nonlinear functions (objective and constraints) and generation of necessary derivative information. The integrator, that can be also used standalone, implements an efficient integration Runge-Kutta scheme. Data coming as a result of direct multiple shooting discretization can be directly passed to one of the three interfaced structure exploiting QP solvers: FORCES, qpDUNES or HPMPC. Alternatively, the DMS data is passed to a condensing routine. The condensed QP data is then forwarded to the qpOASES QP solver.

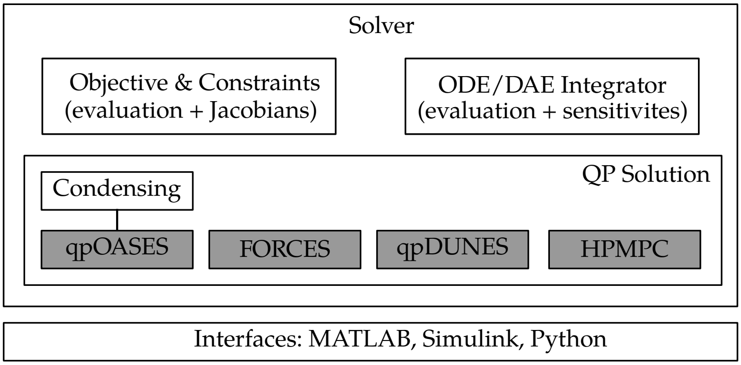

Solver
Objective & Constraints (evaluation + Jacobians)
ODE/DAE Integrator (evaluation + sensitivites)
QP Solution
Condensing
qpOASES
FORCES
qpDUNES
HPMPC
Interfaces: MATLAB, Simulink, Python

Figure 4.1: Structure of an exported MHE/MPC solver. The shaded blocks indicate the external software packages.

In addition to the generated code implementing the tailored MPC or MHE solver, the tool can also export wrappers for MATLAB, Simulink, or Python. For usage in MATLAB scripts, the `mex` wrapper is provided. The Simulink wrapper is provided so that the solver can be directly used in a Simulink diagram. The wrapper is created with specific aim to enable one to use the solver in real-world applications. The software package rawesome [123] provides a convenient interface to the Python language. In [129], the MPC and MHE solvers were successfully interfaced to the LabView environment.

4.4 Real-world Applications

In the following we list several known applications of the ACADO Code Generation Tool. In particular, we list *real-world* applications, where the generated code was used in online context in real experiments.

- To our knowledge, the first real-world application of ACADO CGT generated NMPC solver is the one applied to the overhead crane at KU Leuven [9]. The follow-up experiments [14] reported improvements in execution times due to usage of more advanced implementations. More details can be found in Chapter 5.
- The first application of a generated solver that was executed on embedded hardware is the one reported in [130]. Therein, auto-generated MHE solver

with 6 intervals and arrival cost approximation from [73] was used for state estimation of an induction motor. The model equations comprise 5 states and 2 controls. The achieved sampling frequency was 1.5 kHz with the solver running on a 1 GHz low power Texas Instruments DSP. The maximum recorded execution time was less than 270 µs, i.e. less than a half of the sampling period.

- The first closed-loop experiments for rotational start-up of an airborne wind energy system using the auto-generated MHE and MPC solvers are reported in [15]. Execution times less than 5 ms were reported for MHE and MPC consisting of 10 intervals each, using nonlinear ODE models with 22 states and 2 controls. The follow-up study is presented in Chapter 6.

- In [129, 131] generated MHE and MPC solvers are used for state and parameter estimation, as well as for control of an agricultural tractor utilizing an adaptive nonlinear kinematic model. The model comprises 4 states, 2 controls and 2 parameters. Using 15 intervals for both the MHE and the MPC, feedback execution times in the range 0.6-1.6 ms were achieved on a modern CPU. The total time execution time of the both solvers was reported to be less than 5 ms.

- An industrial robotic arm is successfully controlled by a nonlinear MPC controller generated by the ACADO CGT at 1 kHz [132]. Therein, a path-following NMPC scheme was experimentally validated on the KUKA lightweight robot IV.

- An approach for online solution for optimization of reference trajectories is reported in [133]. The proposed control strategy is validated on a reaction wheel pendulum. The corresponding model consists of 4 states and 1 control input. The NMPC control period was chosen to be 70 ms and the number of control intervals $N_c = 29$. On an embedded platform, execution time of 29 ms was observed.

- Research presented in [134] reports the application of the auto-generated NMPC solvers for control of autonomous cars. Using a spatial reformulation of the time-optimal objective, it was possible to solve the optimization problem in the constrained Gauss-Newton framework. The authors report execution times less than 10 ms for NMPC formulations consisting of 4 states, 2 inputs and up to 50 intervals.

5

Real-time Control of an Overhead Crane

This chapter demonstrates the application of nonlinear MPC and MHE algorithms to a mechatronics system with fast dynamics. We use the auto-generated solvers, presented in Chapter 4, that implement the real-time iteration scheme (see Chapter 2) for both the controller and the estimator to control a laboratory scale overhead crane. The experimental setup consists of a cart moving in one dimension and a varying length pendulum attached to it. In [135, 136], similar crane setups have been controlled by means of nonlinear MPC and linear time-optimal MPC with constant cable length. In contrast, line length variations which greatly increase the nonlinearities of the system are controlled directly by the nonlinear MPC. Experiments show that the worst case execution times are much faster than the necessary sampling time of 10 milliseconds. A similar NMPC based approach implemented on a PLC and applied to an overhead crane is reported in [137].

We present results from the two sets of experiments, demonstrating capabilities of two software generations of the ACADO Code Generation Tool (see Chapter 4). The first set of experiments, presented in [9], utilizes a standard MPC formulation and a simple estimator. Although real-time feasible and demonstrating promising performance, the lack of an estimator and the use of a simple objective formulation showed to be insufficient for achieving a higher

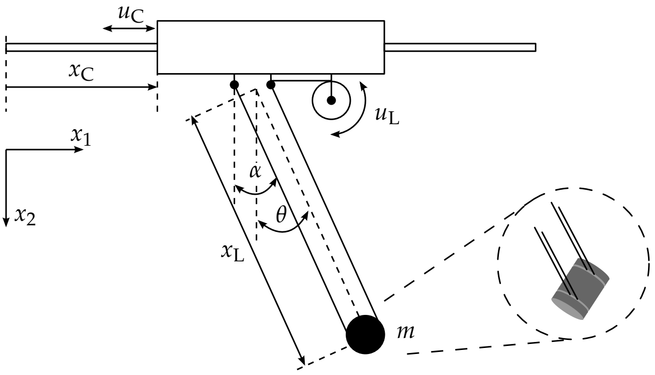


Figure 5.1: Schematic of the experimental overhead crane.

tracking performance. Therefore, with improved software, using more advanced formulations and a proper estimator, we show how the control performance can be improved in the second set of experiments [14]. Moreover, we present speed improvements of the auto-generated solvers over the previous generation used in the first set of experiments.

The overhead crane setup and its dynamic model are presented in detail in § 5.1. The control architectures used in the experiments are detailed in § 5.2. The resulting closed-loop performance while performing point-to-point motions or rejecting external disturbances is discussed in § 5.3. Finally, conclusions are drawn in § 5.4.

5.1 Experimental setup and Dynamic Model

This section briefly describes the experimental setup as well as the dynamic model which is used in the nonlinear model predictive controller and later in the moving horizon estimator.

5.1.1 Experimental Setup

A schematic description of the considered overhead crane is given in Figure 5.1. The cart is actuated by a Servotube 1108 linear motor from Copley Controls. The motor has an integrated incremental encoder which measures the position of the cart x_{C} with a resolution of 5 µm. Its maximum stroke is 0.6 m. The pendulum consists of a cylindrical load with mass $m = 1.3\,\mathrm{kg}$ hanging on two parallel cables. One end of each cable is mounted to a fixed point on the cart, Figure 5.1, while the other two ends are connected to a winch mechanism. The winch mechanism consists of a pulley and a coupled DC motor (A-max 32 from Maxon Motors) with a gearbox reduction ratio 18. An incremental encoder with a resolution of 500 pulses per revolution is attached to the winch motor yielding a resolution of the cable length measurement x_{L} of 2.15 µm. The maximum cable length is 0.95 m, while the minimum cable length is bounded to 0.5 m for safety reasons. The angular deflection α of the left cable is measured with rotary incremental encoder BFH 1P.05A40000-B2-5 from Baumer Electric with a resolution of 40000 pulses per revolution. The relation between the angular deflection of the pendulum θ and of the left cable α is detailed in § 5.1.2. The horizontal and the vertical position of the load are denoted with x_1 and x_2, respectively.

The inputs to this system are the voltages u_{C} and u_{L}, which represent set-points for the associated velocity controllers. These voltages are internally limited to ±10 V. The control software is implemented using the OROCOS Toolchain [138] and runs on a PC with an Intel Xeon 2.53 GHz quad core processor, 12 GB RAM memory, and a preemptive Linux kernel as operating system.

5.1.2 Dynamic Model

The equation of motion of the variable cable length overhead crane is given by [139]:

$$\ddot{x}_{\mathrm{C}} \cos(\theta) = -x_{\mathrm{L}}\ddot{\theta} - 2\dot{x}_{\mathrm{L}}\dot{\theta} - g\sin(\theta), \tag{5.1}$$

where g denotes the gravitational constant. The damping of the pendulum is neglected.

The cart dynamics $G_{\mathrm{C}}(s)$, from the set-point to the dedicated velocity controller u_{C} to the cart position x_{C}, is modeled as the second order system. The winch mechanism dynamics $G_{\mathrm{L}}(s)$ from the input of the winch velocity controller u_{L} to the cable length x_{L} is modeled as a second order system as well. Thus the

Table 5.1: Estimate of the model parameters.

Parameter	Description	Value
A_C	gain of $G_\mathrm{C}(s)$	0.0474 m/s/V
A_L	gain of $G_\mathrm{L}(s)$	0.0341 m/s/V
τ_C	time constant of $G_\mathrm{C}(s)$	0.0128 s
τ_L	time constant of $G_\mathrm{L}(s)$	0.0247 s
g	gravitational constant	9.81 m/s^2

transfer functions are given by

$$G_\mathrm{C}(s) = \frac{A_\mathrm{C}}{s(\tau_\mathrm{C} s + 1)} \text{ and } G_\mathrm{L}(s) = \frac{A_\mathrm{L}}{s(\tau_\mathrm{L} s + 1)}. \tag{5.2}$$

Non-parametric estimates of the frequency response functions (FRFs) of these systems were obtained from random phase multisine excitations [140]. The parameters A_C, A_L, τ_C and τ_L were subsequently identified by a weighted nonlinear least squares frequency domain identification method [140]. The numerical values for the parameters of the model are given in Table 5.1.

The closed-loop transfer function $G_\mathrm{C}(s)$ can be simplified to the pure integrator $G_\mathrm{C}(s) \approx A_\mathrm{C}/s$ due to the fast dynamics of the velocity controller compared to the overall system dynamics. Moreover, the higher order model requires smaller discretization steps if an explicit integrator is used. Consequently, this scenario leads to a much longer integration time of the model.

By combining (5.1) and (5.2) and introducing voltage rate variables u_CR and u_LR, which allow us to account for actuator torque constraints indirectly, we find the following nonlinear ordinary differential equation:

$$\begin{aligned}
&\dot{x}_\mathrm{C} = v_\mathrm{C}, \quad \dot{v}_\mathrm{C} = -\frac{1}{\tau_\mathrm{C}} v_\mathrm{C} + \frac{A_\mathrm{C}}{\tau_\mathrm{C}} u_\mathrm{C}, \\
&\dot{x}_\mathrm{L} = v_\mathrm{L}, \quad \dot{v}_\mathrm{L} = -\frac{1}{\tau_\mathrm{L}} v_\mathrm{L} + \frac{A_\mathrm{L}}{\tau_\mathrm{L}} u_\mathrm{L}, \\
&\dot{\theta} = \omega, \quad \dot{\omega} = -\frac{1}{x_\mathrm{L}} \left(\dot{v}_\mathrm{C} \cos(\theta) + g \sin(\theta) + 2 v_\mathrm{L} \omega \right), \\
&\dot{u}_\mathrm{C} = u_\mathrm{CR}, \\
&\dot{u}_\mathrm{L} = u_\mathrm{LR}.
\end{aligned} \tag{5.3}$$

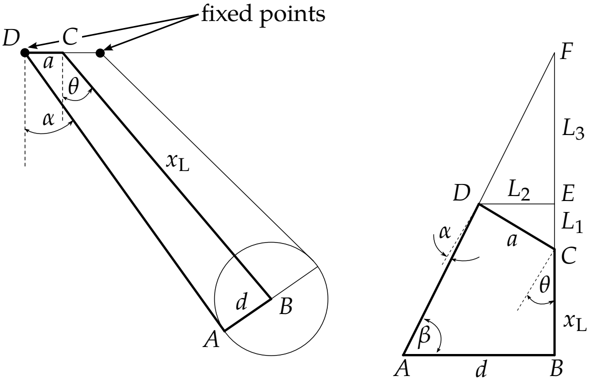


Figure 5.2: Relation between measured angle α and pendulum angle deflection θ.

These equation will in the following be summarized as $\dot{x} = f(x, u)$ where

$$x := [x_\mathrm{C}, v_\mathrm{C}, x_\mathrm{L}, v_\mathrm{L}, \theta, \omega, u_\mathrm{C}, u_\mathrm{L}]' \quad \text{and} \quad u := [u_\mathrm{CR}, u_\mathrm{LR}]' \tag{5.4}$$

denote the states and the controls, respectively. The sampling frequency is chosen to $T_\mathrm{s} = 100\,\mathrm{Hz}$, such that good control performance can be achieved.

The outputs that the equipment provides can be summarized in a vector

$$y := [x_\mathrm{C}, x_\mathrm{L}, \alpha]'. \tag{5.5}$$

The two states are measured directly: the position of the cart x_C and the cable length x_L. The third measurement $\alpha = f_\alpha(\theta, x_\mathrm{L})$ is a nonlinear function of the swinging angle θ and the cable length x_L, see Figure 5.1.

Measurement of the angular deflection A detailed situation is depicted in Figure 5.2. Noting that triangles ABF and DEF are similar, the following equation can be obtained:

$$\frac{L_3}{L_2} = \frac{L_3 + L_1 + x_\mathrm{L}}{d} = \tan(\beta), \tag{5.6}$$

with:

$$\begin{aligned} &\beta = \pi/2 + \alpha - \theta,\ L_1 = a\sin(\theta),\ L_2 = a\cos(\theta), \\ &a = 9.9\ \text{mm},\ d = 14.5\ \text{mm}. \end{aligned} \tag{5.7}$$

If needed, solving (5.6) gives the angle deflection θ in function of α. In practice, a second order polynomial approximation produces sufficient results.

5.2 Control Architectures

The two sets of experiments employed different control architectures, mainly governed by availability of the software for optimal control and estimation. In the text to come, we refer to the two control architectures as *Scenario 1* and *Scenario 2*. The first scenario describes the control architecture used in [9] and the second scenario the one used in [14].

5.2.1 Scenario 1

Nonlinear MPC formulation

At each sampling time the NLP:

$$\min_{\substack{x_0,\dots,x_{N_c} \\ u_0,\dots,u_{N_c-1}}} \quad \sum_{k=0}^{N_c} \|x_k - x_{\text{ref}}\|_Q^2 + \sum_{k=0}^{N_c-1} \|u_k - u_{\text{ref}}\|_R^2 \tag{5.8}$$

$$\text{s.t.} \quad x_0 = \hat{x}_0 \tag{5.9}$$

$$x_{k+1} = F^{\text{int}}(x_k, u_k),\ \text{for}\ k = 0, \dots, N_c - 1, \tag{5.10}$$

$$\underline{x} \leq x_k \leq \overline{x},\ \text{for}\ k = 0, \dots, N_c, \tag{5.11}$$

$$\underline{u} \leq u_k \leq \overline{u},\ \text{for}\ k = 0, \dots, N_c - 1, \tag{5.12}$$

is solved with the following bounds on controls and states

$$\begin{aligned} -10\,\mathrm{V} &\le u_{\mathrm{C}} \le 10\,\mathrm{V}, \\ -10\,\mathrm{V} &\le u_{\mathrm{L}} \le 10\,\mathrm{V}, \\ -100\,\mathrm{V/s} &\le u_{\mathrm{CR}} \le 100\,\mathrm{V/s}, \\ -100\,\mathrm{V/s} &\le u_{\mathrm{LR}} \le 100\,\mathrm{V/s}. \end{aligned} \tag{5.13}$$

The system inputs u_{C} and u_{L} are constrained by internal limitations of the corresponding velocity controller. Constraints for the control rates u_{CR} and u_{LR} are tuned such that saturation in the cart and winch actuators are avoided. The state and control vectors x_k and u_k and the system description are defined as in (5.4) and (5.3), respectively. The control horizon is chosen to be $T_{\mathrm{c}} = 1\,\mathrm{s}$, which was found to be suitable by empirical testing. The discretized system dynamics are represented by the function F^{int}. A continuous-in-time formulation counterpart to (5.8) is discretized using the single shooting technique on an equidistant grid with $N_{\mathrm{c}} = 10$ intervals.

The MPC formulation (5.8) incorporates references for full state and control vectors and we keep the references constant over the control horizon. Furthermore, only the references for the cart position and cable length are changed online, while all other references are always set to zero:

$$x_{\mathrm{ref}} := [x_{\mathrm{C,ref}}, 0, x_{\mathrm{L,ref}}, 0, 0, 0, 0, 0]' \quad \text{and} \quad u_{\mathrm{ref}} := [0, 0]'.$$

Moreover, we choose weighting matrices Q and R defined as:

$$\begin{aligned} Q &:= \mathrm{diag}\Big(56\,\mathrm{m}^{-2}, 6\,\mathrm{s}^2/\mathrm{m}^2, 115\,\mathrm{m}^{-2}, 0.01\,\mathrm{s}^2/\mathrm{m}^2, \\ &\qquad 10, 10\,\mathrm{s}^2, 10^{-8}\,\mathrm{V}^{-2}, 10^{-8}\,\mathrm{V}^{-2}\Big), \\ R &:= \mathrm{diag}\Big(10^{-5}\,\mathrm{s}^2/\mathrm{V}^2, 10^{-5}\,\mathrm{s}^2/\mathrm{V}^2\Big). \end{aligned}$$

Since our intention is to perform accurate point-to-point motions, the weights corresponding to the tracking errors of the cart position and the cable length are chosen to be large in comparison to the weights which penalize the tracking error of the velocities. Similarly, the weights for the control cost are low. The aim of this tuning is to enable fast but well damped closed-loop behavior.

Software implementation The solver that solves online the NLP (5.8) is generated by the ACADO CGT. The first generation of ACADO generated NMPC solvers that is used in the first validation was limited to the NLP formulation given in (5.8). In addition, the user was constrained to use only the single shooting technique to discretize the OCP. The solver implemented the $\mathcal{O}(N_\mathrm{c}^3)$ condensing technique and qpOASES QP solver was employed to solve the underlying QP.

Simple estimator

Note that the nonlinear MPC controller itself requires estimates for all the eight states as an input. However, only the positions are directly measured by the encoders. Consequently, we have to estimate the corresponding velocities $v_\mathrm{C}, v_\mathrm{L}$ and ω. Limited by the available software, we opt for finite differences approximations. For the cart velocity we use $v_\mathrm{C}[k] = (x_\mathrm{C}[k] - x_\mathrm{C}[k-1])/T_\mathrm{s}$, $T_\mathrm{s} = 10\,\mathrm{ms}$. To reduce effects of high-frequency noise, we filter all velocities using a first order low-pass digital filter with a cut-off frequency of $f_\mathrm{c} = 10\,\mathrm{Hz}$. This choice is justified by the slow pendulum dynamics and the assumption that the NMPC does not trigger fast modes of the cart mechanism.

5.2.2 Scenario 2

Nonlinear MPC formulation

In this scenario we used the following discretized OCP formulation for the NMPC:

$$\min_{\substack{x_0,\dots,x_{N_\mathrm{c}}\\ u_0,\dots,u_{N_\mathrm{c}-1}}} \quad \sum_{k=0}^{N_\mathrm{c}-1} ||h_\mathrm{r}(x_k,u_k)-\tilde{r}_k||_R^2 + |||h_\mathrm{r,N}(x_{N_\mathrm{c}})-\tilde{r}_N||_{R_N}^2 \tag{5.14a}$$

$$\text{s.t.} \quad x_0 = \hat{x}_0 \tag{5.14b}$$

$$x_{k+1} = F^\mathrm{int}(x_k,u_k),\ \text{for } k=0,\dots,N-1, \tag{5.14c}$$

$$x_k^\mathrm{lo} \le x_k \le x_k^\mathrm{up},\ \text{for } k=0,\dots,N_\mathrm{c}, \tag{5.14d}$$

$$u_k^\mathrm{lo} \le u_k \le u_k^\mathrm{up},\ \text{for } k=0,\dots,N_\mathrm{c}-1, \tag{5.14e}$$

where discretized system dynamics are represented by the function F. The current state estimate provided by an estimator is denoted $\hat{x}_0 \in \mathbb{R}^{n_x}$. The controlled system output is captured with reference functions in (5.14a): $h_\mathrm{r} \in \mathbb{R}^{n_r}$ and $h_\mathrm{r,N} \in \mathbb{R}^{n_{r,N}}$, and the corresponding weighting matrices are $R \in \mathbb{R}^{n_r \times n_r}$ and $R_N \in \mathbb{R}^{n_{r,N} \times n_{r,N}}$. Variables $\tilde{r}_k \in \mathbb{R}^{n_r}$ and $\tilde{r}_N \in \mathbb{R}^{n_{r,N}}$ denote time-varying references. As in the previous scenario, we use the same box constraints defined in (5.13). Finally, the number of control intervals is denoted as N_c. The control horizon is chosen to be 1 s with $N_\mathrm{c} = 10$ shooting intervals, which was found to be suitable by empirical testing.

Output model for MPC In the first scenario, the load position and swinging is controlled indirectly by controlling the cart position x_C, cable length x_L and angle θ. For more effective control of the load position while using the objective in (5.8), one needs to introduce off-diagonal weights in matrix Q. Tuning of those extra terms might be involved and counter intuitive. In order to avoid those problems and come up with an easy to tune objective, we aim here to control the position and swinging of the load in the x_1-x_2 plane directly. For this purpose, the selected reference output function are the load position x_1

and x_2, the swinging velocity ω and the control slew-rates:

$$\begin{aligned} h_r(x,u) &= [x_\mathrm{C} + x_\mathrm{L}\sin(\theta), x_\mathrm{L}\cos(\theta), \omega, u_\mathrm{CR}, u_\mathrm{LR}]', \\ h_{N,r}(x,u) &= [x_\mathrm{C} + x_\mathrm{L}\sin(\theta), x_\mathrm{L}\cos(\theta), \omega]'. \end{aligned} \tag{5.15}$$

The MPC formulation incorporates a reference vector $\tilde{r}$, which is a nonlinear combination of the state vector x. Point to point motions are executed by giving step reference changes, hence the references are constant over the whole control horizon. We consider the following reference for $\tilde{r}$ and u:

$$\tilde{r}_k = [x_{1,\mathrm{ref}}, x_{2,\mathrm{ref}}, 0, 0, 0]'. \tag{5.16}$$

Here $x_{1,\mathrm{ref}}, x_{2,\mathrm{ref}}$ are the desired load x_1 and x_2 position, while the desired angular velocity is zero to suppress swinging of the load. The control reference is set to zero to minimize the total control effort. The empirically found suitable weight reads:

$$R = \mathrm{diag}(100\ \mathrm{m}^{-2}, 100\ \mathrm{m}^{-2}, 1\ \mathrm{s}^2, 10^{-5}\ \mathrm{s}^2/\mathrm{V}^2, 10^{-5}\ \mathrm{s}^2/\mathrm{V}^2) \tag{5.17}$$

Since we want accurate point to point motions, the weight on the position x_1 and x_2 is large with respect to the weight on the control inputs.

Software implementation The NMPC is auto-generated by the ACADO CGT. We use $\mathcal{O}(N_\mathrm{c}^3)$ based condensing and the dense linear algebra QP solver qpOASES. The OCP is discretized with the direct multiple shooting technique. In comparison with the first scenario, we use the implicit Runge-Kutta integrator of order four for integration of the system dynamics. This way we avoid a large number of integrator steps needed if explicit integrators are used. Let us note here that the system dynamics in the first scenario had to be simplified (the dynamics of the cart) because of availability of only the explicit integrator in the ACADO CGT suite. In addition to more advanced integrators, we use the more advanced $\mathcal{O}(N_\mathrm{c}^3)$ structure exploiting condensing procedure, detailed in § 3.2.1.

Nonlinear MHE formulation

The moving horizon estimation (MHE) problem solved at each time step reads:

$$\min_{\substack{x_0,\dots,x_{N_\mathrm{e}}\\ u_0,\dots,u_{N_\mathrm{e}-1}}} \quad \sum_{k=0}^{N_\mathrm{e}-1} ||h_y(x_k,u_k)-\tilde{y}_k||_S^2 + |||h_{y,N_\mathrm{e}}(x_{N_\mathrm{e}})-\tilde{y}_{N_\mathrm{e}}||_{S_{N_\mathrm{e}}}^2 \tag{5.18a}$$

$$\text{s.t.} \qquad x_{k+1} = F^{\mathrm{int}}(x_k,u_k), \text{ for } k=0,\dots,N_\mathrm{e}-1, \tag{5.18b}$$

where the number of estimation intervals is N_e. Measurement functions are denoted with $h_y \in \mathbb{R}^{n_y}$ and $h_{y,N_\mathrm{e}} \in \mathbb{R}^{n_{y,N_\mathrm{e}}}$ and the corresponding weighting matrices are $S \in \mathbb{R}^{n_y \times n_y}$ and $S_{N_\mathrm{e}} \in \mathbb{R}^{n_{y,N_\mathrm{e}} \times n_{y,N_\mathrm{e}}}$.

We choose to have the same dynamic model for both the controller (5.14) and the estimator (5.18). The main motivation for this approach is that we want the estimator to capture the same dynamics we want to control using NMPC.

Measurement model for MHE The measured states of the system are $y = [x_\mathrm{C}, x_\mathrm{L}, \alpha, u_\mathrm{C}, u_\mathrm{L}, u_\mathrm{CR}, u_\mathrm{LR}]'$, cf. Figure 5.2. Strictly speaking, u_C, $u_\mathrm{L}, u_\mathrm{CR}, u_\mathrm{LR}$ are *pseudo-measurements*. We cannot measure them, but we can penalize the deviation from the references calculated by the NMPC. The output functions h_y and h_{y,N_e} read:

$$\begin{aligned} h_y(x,u) &= [x_\mathrm{C}, x_\mathrm{L}, \alpha, u_\mathrm{C}, u_\mathrm{L}, u_\mathrm{CR}, u_\mathrm{LR}]', \\ h_{y,N_\mathrm{e}}(x) &= [x_\mathrm{C}, x_\mathrm{L}, \alpha, u_\mathrm{C}, u_\mathrm{L}]'. \end{aligned} \tag{5.19}$$

Optimizing the controls allows $u_\mathrm{C}, u_\mathrm{L}$ one to account for disturbances and non-modeled dynamics. This approach is motivated by the fact that the control hardware and the actuators are connected via analog lines. Those analog lines collect noise and disturb the control signal that is sent to the system.

The weights on the diagonal of the weighting matrix

$$\begin{aligned} S = \mathrm{diag}(&16.5\ \mathrm{m}^{-2}, 25.1\ \mathrm{m}^{-2}, 119.4, 1.2\ \mathrm{V}^{-2}, 0.4\ \mathrm{V}^{-2}, \\ &\qquad\qquad\qquad\qquad 0.01\ \mathrm{s}^2/\mathrm{V}^2, 0.01\ \mathrm{s}^2/\mathrm{V}^2) \end{aligned}$$

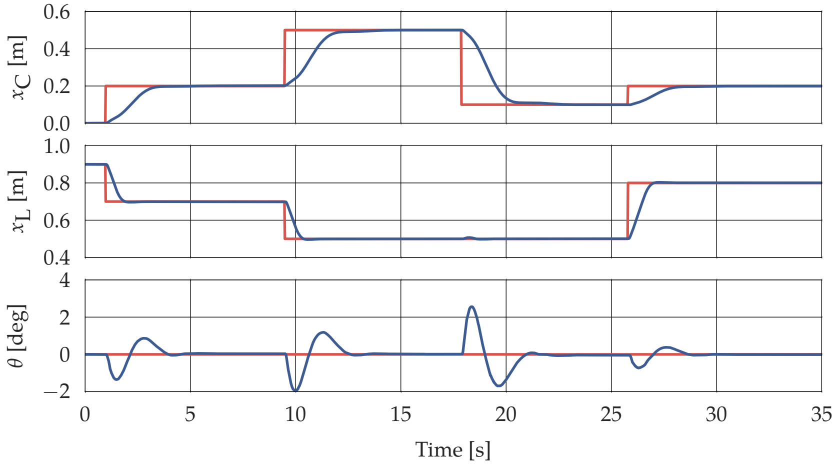
are obtained experimentally and each weight denotes the inverse of the covariance of the corresponding measurement. The estimation horizon is chosen to be 0.2 s with $N_\mathrm{e} = 20$ shooting intervals, which was found to be suitable by empirical testing.

Software implementation The MHE solver is auto-generated by the ACADO CGT and implements the $\mathcal{O}(N_\mathrm{e}^3)$ condensing routine outlined in § 3.2.1. The continuous-in-time counterpart to (5.18) is discretized by the direct multiple shooting technique.

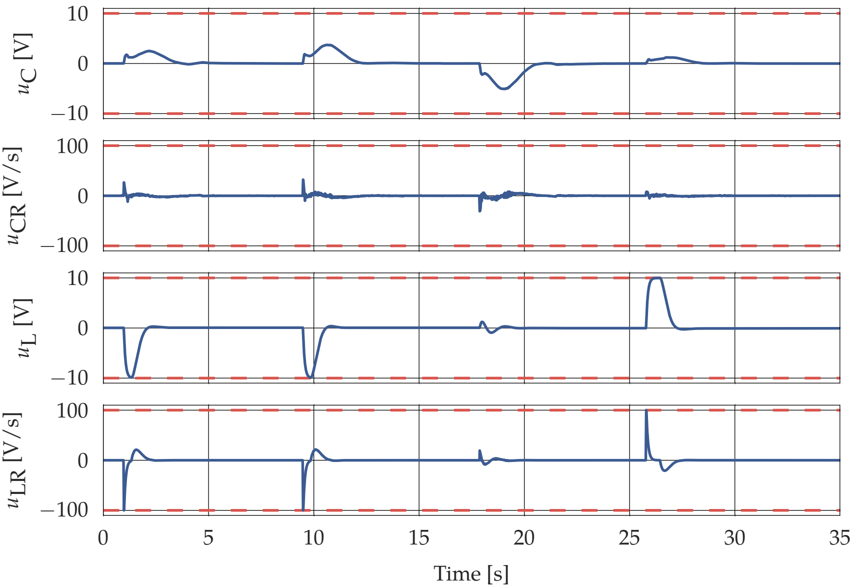
5.3 Experimental Results

5.3.1 Scenario 1

A first experiment illustrates the controller performance for point-to-point motions, see Figure 5.3. Multiple reference changes are applied to the controller (Figure 5.3a). The steady state error on x_C is less than 2 mm and less than 1 mm on x_L. The control inputs and corresponding control rates are shown in Figure 5.3b. Note that the winch control and its slew-rate, u_L and u_LR, hit their limits because the corresponding weights in the Q and R matrix are set aggressively. Weights in the Q and R matrices which correspond to motion of the cart and to pendulum swinging are set to conservative values resulting in slow yet accurate motion. Let us note that the third setpoint change, a change in x_C only, results in a control input for the winching motor as well.



(a) Cart position x_C, cable length x_L.



(b) Controls u_C, u_L and control rates u_{CR}, u_{LR}.

Figure 5.3: Point-to-point motions in the vertical plane: states and controls; solid blue lines: measurements, solid red lines: references, dashed red lines: constraints.

Table 5.2: Execution times of the auto-generated NMPC.

	Feedback phase [ms]	Preparation phase [ms]	Overall [ms]
Maximum	0.19	0.90	1.08
Average	0.04	0.86	0.93

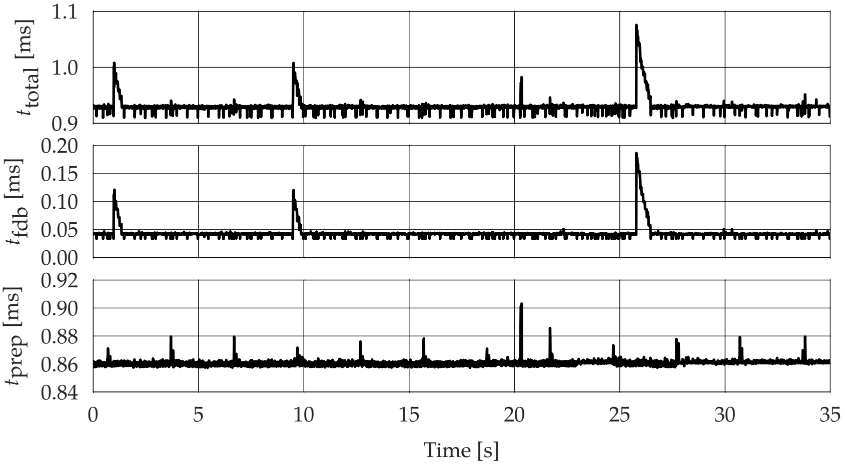
Figure 5.4a validates the low execution time of the auto-generated C-code. The average execution time is less than 1 ms. The peak execution time measures less than 1.1 ms. In the worst case, the time spent in the feedback phase is 17 % of the overall execution time, see Table 5.2. The peak times in the feedback phase are directly proportional to the number of active-set changes for solving the QP. The ratio between the average time spent in the feedback and preparation phase is almost equal to results obtained in synthetic tests in [7]. Execution times are measured with OROCOS timer services. Those services internally use the Linux function `clock_gettime()`, which provides a resolution in the nanosecond range.

Performing only one SQP iteration at each sampling time results in sub-optimal control. Therefore it is beneficial to know how well the controller performs in terms of optimality. In the software implementation used in this scenario, the controller performance is measured using the absolute value of the first term from the Tailor expansion of the objective function[1]. The measurements presented in Figure 5.4b show a good convergence rate of the proposed MPC controller. Jumps in the optimality measure occur at big reference changes since the solution obtained from one SQP iteration is only a rough approximation to the optimal solution in these situations. However, due to the good contraction properties of the Gauss-Newton real-time iteration scheme, the optimality measure decreases quickly in subsequent sampling instants. Note that the measure would be identical to zero for a linear system because in each RTI a QP is solved exactly; thus, the non-zero values of the measure are an indication of the nonlinearity of the optimal control problem.

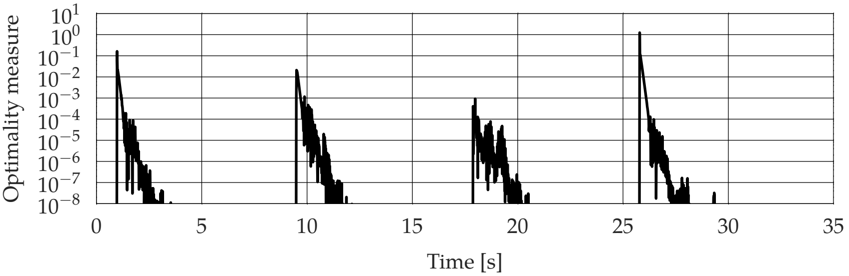
In a second experiment an external disturbance acts on the load by pushing it from its equilibrium. As seen in Figure 5.5a, the controller successfully rejects the disturbance. Note that the cable length does not change despite a non-

---

[1] This measure is the first term of the KKT tolerance, see § 2.1.2, monitoring only the progress of the objective function. This was one of the limitations of the first implementation of the ACADO CGT.



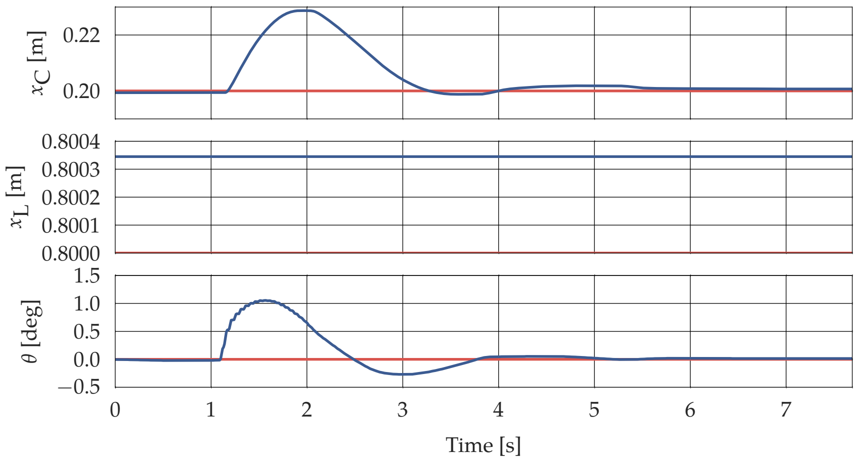
(a) $t_{\text{total}} = t_{\text{fdb}} + t_{\text{prep}}$: overall NMPC execution time, t_{fdb}: duration of the feedback phase, t_{prep}: duration of the preparation phase.



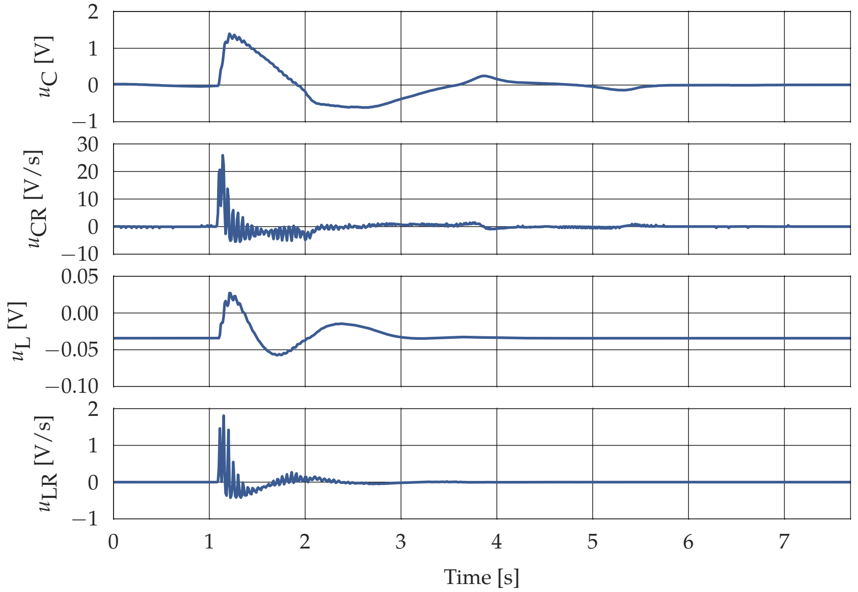
(b) KKT tolerance.

Figure 5.4: Point-to-point motions in the first scenario: performance of the nonlinear MPC.

zero control signal (Figure 5.5b), due to a low voltage signal which cannot overpower the static friction in the winch mechanism. Furthermore, since there is no integral action on the states x_{C} and x_{L} in the model (5.3), the NMPC is outputting small non-zero values for the voltages u_{C} and u_{L} in the steady state.



(a) Cart position x_C, cable length x_L, angle deflection θ.



(b) Controls u_C, u_L and control rates u_{CR}, u_{LR}.

Figure 5.5: Rejection of an external disturbance force applied to the hanging mass in the first scenario: states and controls.

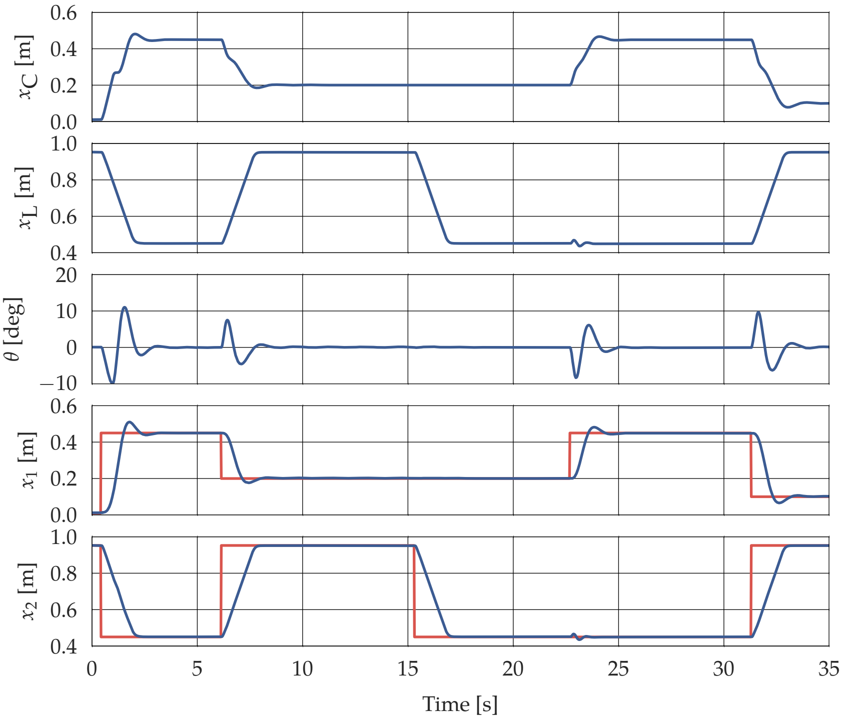


Figure 5.6: Point-to-point motions in the second scenario: cart position x_C, cable length x_L, angle deflection θ and position of the load x_1, x_2.

5.3.2 Scenario 2

In the following we present the results illustrating the closed-loop performance in the second set of experiments. In particular, we present results related to the point-to-point motions of the load, originally published in [14]. The second set of experiments also included disturbance rejection and servo-tracking experiments, however, the interested reader is referred to [14] for more information.

Multiple step references for the x_1 and x_2 position of the load are applied to the controller. The response of the controller is shown in Figure 5.6. The steady state errors for the x_1 and x_2 are less than 1 mm and 0.5 mm, respectively. The settling time for large reference jumps is less than 3.5 s. It can be seen that the

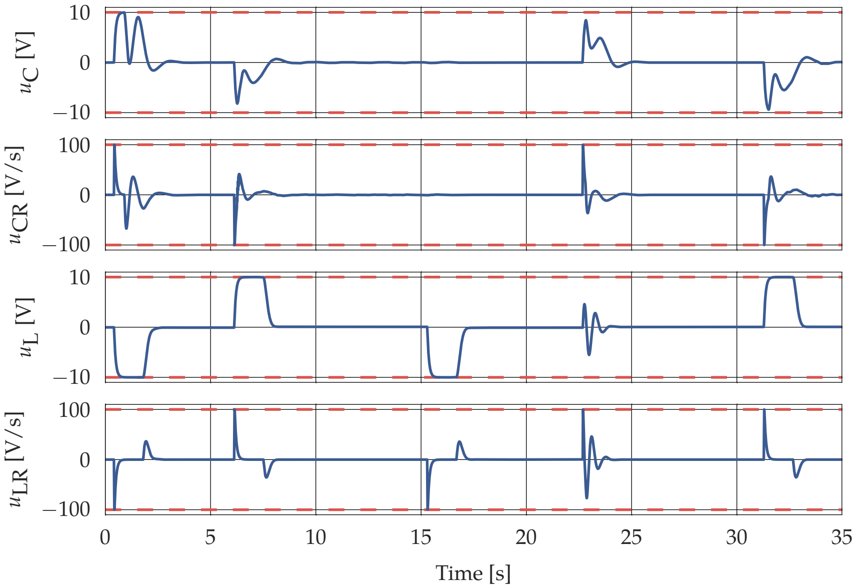


Figure 5.7: Point-to-point motions in the second scenario: controls: u_C, u_L and control rates u_{CR}, u_{LR}.

angle is still oscillating around zero with small amplitude (about 0.04 degrees). This small residual oscillation on θ is, however, not due to swinging in the $x_1 - x_2$ plane but due to the yawing of the load around the vertical axis. This yawing motion is induced by the motion of the crane and is a consequence of the front and back of the load hanging imperfectly horizontal. This yaw results in front and back cable motions and hence results in oscillating θ measurements. Since the set-up cannot control the yawing of the load, this residual cannot be controlled to zero. The control reaction to this residual is negligible.

The control inputs and control rates are given in Figure 5.7. Here one can see that the cart control u_C, control rate u_{CR}, cable control u_L and control rate u_{LR} hit their limits. This aggressive performance is due to the small weights in the R matrix on the control effort, and large weights on the tracking error $(h_r - \tilde{r})$. The large weights on the tracking error will drive the controller toward the system limits to minimize the set point error as fast as possible.

The aggressive settings of the controller are made possible by using the alternative OCP formulation, the proper estimator and the proper numerical

Table 5.3: Execution times for the auto-generated MHE and NMPC algorithms.

	Feedback phase [ms]	Preparation phase [ms]	Overall [ms]
Maximum runtimes			
MHE	0.15	0.73	0.88
NMPC	0.46	0.41	0.74
Total	0.54	1.03	1.45
Average runtimes			
MHE	0.08	0.62	0.70
NMPC	0.04	0.26	0.29
Total	0.12	0.88	1.00

integrator. In the first set of experiments, the aggressive tuning was impossible mainly because of the lack of the estimator. We approximated the velocities with finite differences. Furthermore, we filtered the velocities to make them smoother, but as a consequence we delayed those signals. With this in mind, more aggressive tuning was unattainable. In addition, more dynamic closed-loop responses were unachievable because the dynamics of the cart was simplified in lack of the proper integrators.

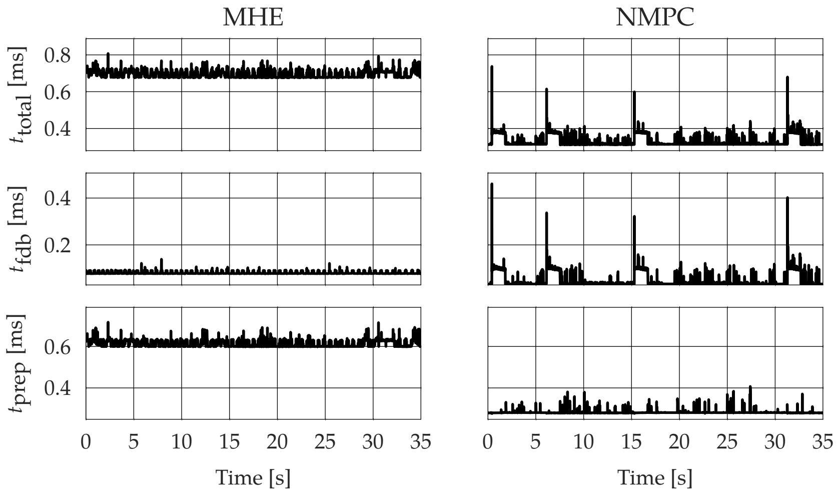
Finally, the real intention is to control the position of the load, and position of the cart during load movements is of minor importance. Using the alternative objective formulation we achieved direct control of the load position. Indirectly, use of the objective formulation in (5.14) allowed for easier and more intuitive tuning.

The maximum execution time of the estimator is 0.88 ms, see Table 5.3. The execution is expected to be constant, given that we solve an equality constrained optimization problem (5.18) and that the solver solves the problem in a single iteration. Deviations in the execution times, i.e. between the average and the maximum, are a consequence of the solver running in a multi-threaded environment.

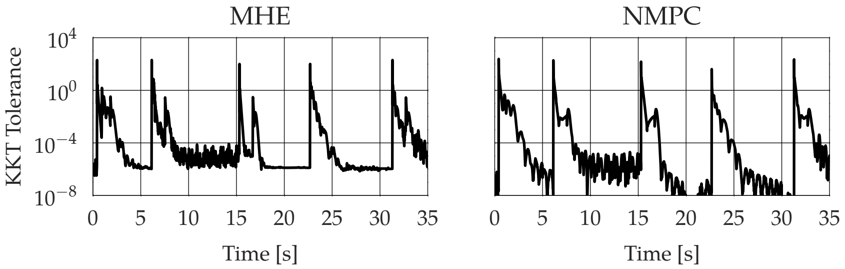
On the contrary, it is unexpected from the NMPC solver to have constant execution times since it solves online an inequality constrained OCP (5.14). The maximum execution time equal to 0.74 ms.

Here, we can conclude that both the solvers complete in 1.45 ms worst case, far more faster than the sampling time $T_{\mathrm{s}} = 10\,\mathrm{ms}$! In comparison to the previous set of experiments, cf. Table 5.2, the average total execution time is slightly higher than the average time that was needed just to solve the OCP (5.8) in the NMPC; cf. Figure 5.4 and Table 5.3. The *total feedback time*, that is, the sum of feedback times of the estimator and the controller is 0.54 ms. This is higher than in the previous set of experiments and a direct consequence of more aggressive settings of the controller. More efficient condensing procedure led to significantly lower preparation time of the NMPC than in Table 5.2 – 0.41 ms versus 0.90 ms.

For completeness, we present the execution time profiles as well as the KKT tolerances in Figure 5.8. Despite sometimes high values of the KKT tolerances, no problems were detected in the operation of the controller, nor the estimator. The high values of the tolerances are consequences of: 1) large reference jumps and 2) scaling issues in the corresponding formulations of optimization problems.



(a) Execution times for MHE and the NMPC.



(b) KKT tolerances.

Figure 5.8: Point-to-point motions in the second scenario: computational performance of the MHE and the NMPC.

5.4 Conclusions

In this chapter we have experimentally validated nonlinear MPC and MHE strategies using automatic code generation applied to a fast mechatronics system. Both control strategies showed satisfactory tracking accuracy and steady state error in positioning of the load. Slightly better performance is achieved in the second set of experiments. This was possible by using a more detailed model, the proper state estimator in conjunction with a more advanced formulation of the objective in the NMPC.

The both scenarios are real-time feasible, exhibiting execution times far below the sampling time. However, using the more advanced software implementations in the second control scenario resulted in astonishing speed-ups. In particular, the average execution time of the controller in the first scenario is nearly equal to the time both the estimator and the controller need in the second scenario. The maximum time in the second scenario is higher than in the first one, predominantly because of the aggressive settings of the controller resulting in larger maximum feedback times of the NMPC. Both approaches confirm the fast contraction rate of the RTI scheme [35].

6

Real-time Control of an Airborne Wind Energy System

The concept of Airborne Wind Energy (AWE), introduced by Loyd [141], proposes energy harvesting using a kite in crosswind flight – i.e. in the direction perpendicular to the air flow. The fundamental difference in comparison with the standard wind turbines is that the kites can fly at much higher altitudes where the wind speeds are much highers than at the altitudes reachable by the classical wind turbines. Stating the obvious, the tethered flight voids the need for tons of concrete, steel and composites needed to build the conventional wind turbines.

An illustration of an AWE concept is depicted in Figure 6.1. There, the kite is in fact an airplane tethered to the grounded generator. The airplane flies along a cyclic *lying eight* trajectory, most of the time pulling the cable, see e.g. [142]. Consequently, the generator is producing electricity. Only for about 30 % of the cycle, according to [143], a small amount of energy needs to be invested to retract the cable. Other approaches exist, using different kind of airfoils and possibly on-board generators. For a detailed overview on the topic we refer to [144].

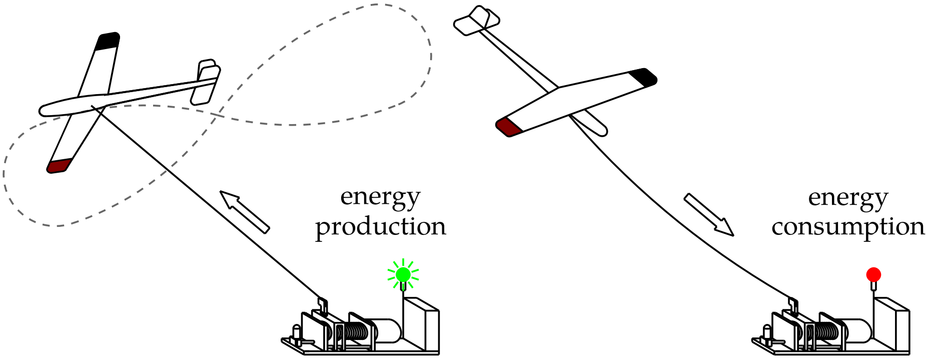


Figure 6.1: An airborne wind energy concept[1].

Economic viability of the operation requires automation of the following phases:

1. start-up phase,
2. energy production phase and
3. the landing phase.

Here, we are interested in the start-up phase, the one that deals with bringing the airplane from a parked position on ground to high altitudes. The approach we are interested in is the *rotational start-up* [146]. This approach proposes to gain altitude by rotation. The airplane is attached to the rotating carousel and starting at a short tether length, the tether is released to gain the altitude. In this setting, the generator is mounted on the carousel, see [147].

Tethered flight is a highly nonlinear, unstable and constrained system subjected to often strong disturbances. Those characteristics motivate usage of the MPC and the MHE for control and estimation. Within the scope of the thesis, we are interested mainly to test computational performance of the well-established RTI scheme for fast MHE and NMPC from Chapter 2 and employ fast algorithms from Chapter 3 for the underlying QPs. In particular, we aim to validate the applicability of the auto-generated solvers – from Chapter 4 – on complex MHE and MPC formulations involving available models that describe the complex nonlinear dynamics of one particular AWE system. In essence, this chapter presents the results from the second round of experiments at the KU Leuven

---

[1] The illustration is adapted from [145].

kite lab. The first set of experiments has been previously published in [15]. The question whether the MHE and the NMPC are the best techniques for rotational start-up of an AWE system is out of the scope of this thesis.

The details on hardware and software are presented in § 6.1. We give a detailed description of the hardware setup, as well as of the software used for real-time control. The control architecture is the central topic of § 6.2. Our aim is to employ MHE as estimator and MPC as controller in closed-loop experiments. The intention is to fly at a constant tether length and a constant speed while changing the roll angle and consequently height of the airplane. The experimental results are presented in § 6.3. The same MPC formulation is tested with two QP solvers. The first one is employing the structure exploiting IPM QP solver HPMPC, and the second one utilizes the efficient $\mathcal{O}(N^2)$ condensing technique and the active-set QP solver qpOASES. The conclusions are drawn in § 6.4.

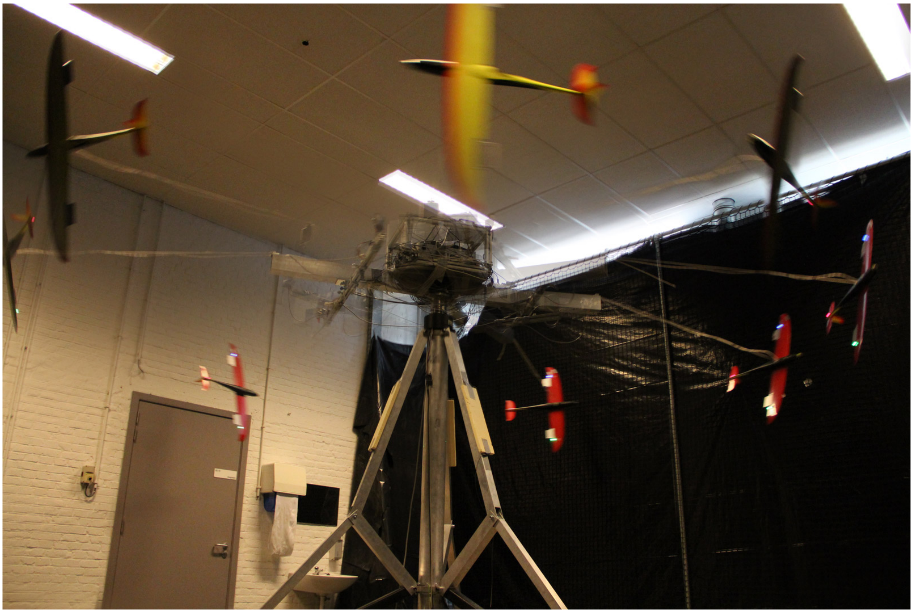
6.1 Experimental Setup

The experimental setup at the KU Leuven has been built with the aim to gain experience with sensors, actuators and real-time software on the one hand and modeling, estimation and control techniques on the other. Those research activities are expected to lead to eventual start-up of an airborne wind energy system. In-depth description of the initial hardware setup is provided in [148], and some of the later modifications are presented in [149, 15]. The latest hardware modifications are going to be summarized in this section.

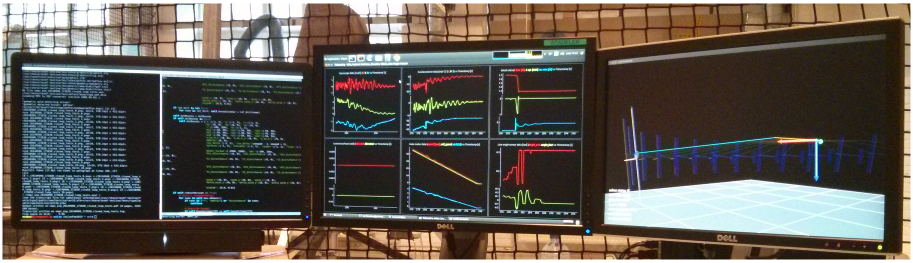
The setup is depicted in Figure 6.2. One can already identify three main entities: the spinning carousel, the plane and the ground station. The carousel consists of a tripod like structure with an arm mounted on its top. The arm itself can rotate around the carousel's central axis. The main control PC, most of the DAQ hardware and the winching mechanism are located at the arm. The tether is winded up on the winch, and one end of the tether is connected to the airplane. The winching mechanism enables us to fly on longer or shorter tether lengths during the experiments. The plane has three control surfaces for actuation – two ailerons in differential configuration and an elevator. This effectively means that one can control roll and pitch angle of the aircraft. The carousel and the attached plane sit in *a cage* enclosed by thick Plexiglas sheets and safety nets. The base of the cage measures approximately 8×8 square meters, effectively allowing for tether lengths of up to 2.1 m. Finally, there is a ground station outside the cage operated by an engineer. It consists of a PC that allows the operator to have total control over the experimental setup. The ground station PC is connected to the main control PC on top of the carousel via a wireless connection. Typically the operator controls the control PC via a terminal (left screen in Figure 6.2b). The software running on the ground station PC provides live feed of all vital information: sensor and actuator data (middle screen in Figure 6.2b). Additionally, there is 3D visualizer for easier monitoring of motions of the airplane during the experiments (right screen in Figure 6.2b).

6.1.1 Hardware

A schematic representation of the hardware is sketched in Figure 6.3. The ground station PC (*Operator's PC*) communicates with the *Main controller PC* via a WiFi connection. The operator can send commands to the control PC through



(a) A composite image of the carousel setup in motion



(b) A screenshot from the ground station PC

Figure 6.2: The experimental setup.

a terminal and the telemetry data is streamed from the control PC to the ground station PC for visualization.

The arm rotation is actuated by the *Main motor* – a DC motor connected to the shaft. Similarly, the winching mechanism is actuated by the *Winch motor*. Horizontal and vertical angles of the tether relative to the arm's tip can be measured by the *Line angle sensor*. The *3D vision system* consisting of the two cameras on a boom mounted on the arm of the carousel *is unused* in

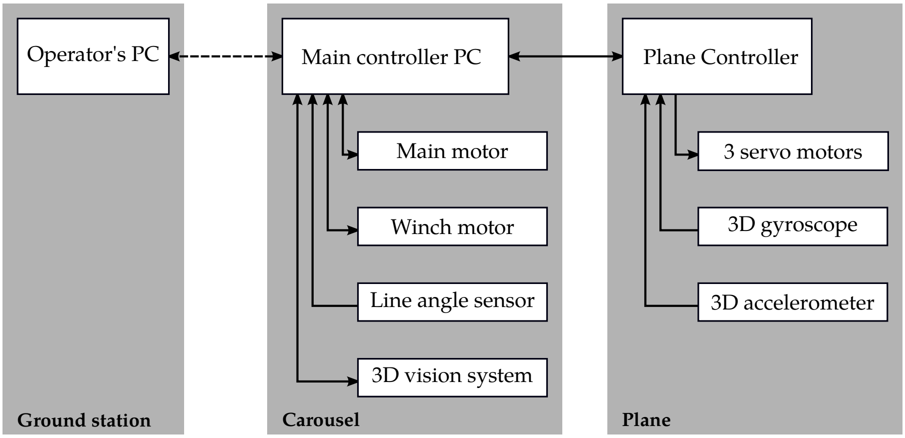


Figure 6.3: A schematic representation of the experimental setup.

the experiments to be presented in this chapter; interested reader is referred to [150, 15] for more information on this subsystem.

The main controller PC and the *Plane controller*, an interface board inside the plane, communicate via the (tethered) Ethernet connection using the TCP/IP protocol. The interface board sends references, received from the main controller PC, to the servo motors (servos). In addition, sensor data from gyroscopes and accelerometers is streamed down to the main controller PC.

Carousel

The carousel mechanics is designed for high stiffness, such that forces and torques induced by the airplane have minimal influence on the structure of the carousel [148]. The height of the setup has been chosen to be 2.5 m, allowing some ground clearance for the airplane. The rotation of the arm is actuated by a 300 W DC motor, which is coupled to the main shaft through a gearbox. A rotational encoder is mounted on the motor, measuring the angle of the motor's shaft, see Figure 6.4. The main motor has a dedicated power electronics device which has an integrated current controller. The current reference, current measurement and encoder readings are all fed to an E/BOX device [151]. The E/BOX device is a DAQ system with a number of analog and digital inputs and outputs as well as quadrature encoder inputs and PWM (pulse width modulation) outputs. The communication with the device is done via the fast

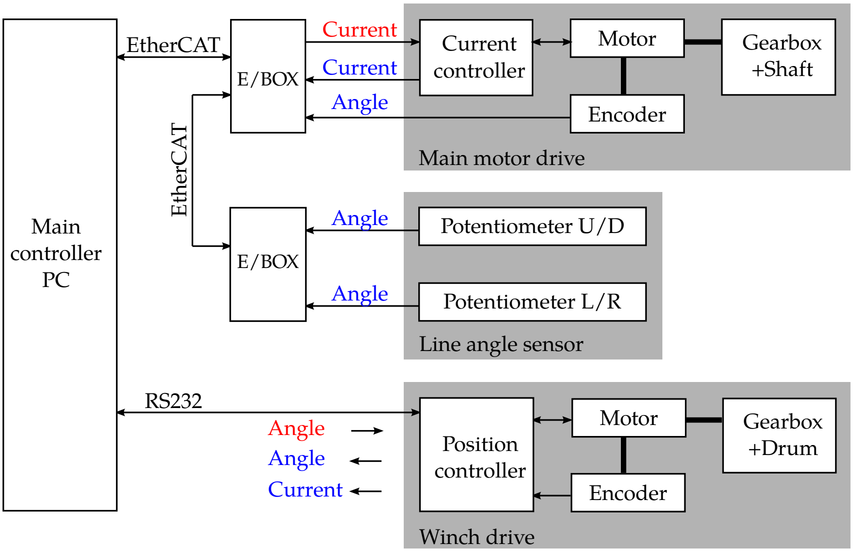


Figure 6.4: A schematic representation of the carousel control hardware. Measurements are denoted in blue and references in red. Thick lines refer to the mechanical connections.

EtherCAT protocol that utilizes Ethernet infrastructure on hardware layer. As depicted in Figure 6.4, the devices are daisy-chained.

The line angle sensor is mounted on the tip of the arm and comprises of a joystick mechanism and a carbon rod. One end of the carbon rod is attached to the joystick mechanism and the other one slides on the tether. Two potentiometers mounted on the joystick mechanism capture horizontal and vertical movements of the tether relative to the arm tip. The analog signals from the potentiometers are fed into another E/BOX device after appropriate signal conditioning.

The winching mechanism is powered with a 400 W brushless DC (BLDC) motor and the dedicated servo amplifier. The motor and the drum are coupled via a gearbox and an elastic belt [149]. The angle of the motor shaft is measured by a rotary encoder. The servo amplifier is connected to the main controller PC with the standard RS232 connection and the communication is done using a protocol defined by the manufacturer.

The main controller PC is an industrial PC equipped with a 2.3 GHz Intel Core i7-3610QE CPU, 8 GB of RAM, a high speed 512 GB solid state drive (SSD) and

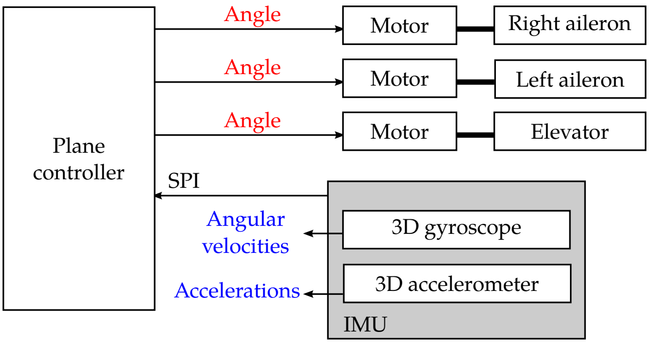


Figure 6.5: A schematic representation of control hardware inside the plane. Measurements are denoted in blue and references in red. Thick lines refer to the mechanical connections.

enough peripheral interfaces to connect all external hardware.

Airplane

The airplane used for the experiments is an Ariane P5 which is typically used for radio controlled (RC) model racing. Those airplanes are supposed to take sharp turns at high velocities. Therefore, they are built from carbon fiber that gives them high strength and stiffness. The aircraft is equipped with sensors, actuators, an interface board (*Plane controller* in Figure 6.5) and corresponding electronics.

The sensor package consists of an inertial measurement unit (IMU) which is connected to the interface board's micro-controller (MCU) via the fast serial peripheral interface (SPI). Communication protocol is defined by the IMU manufacturer. The IMU has a 3D gyroscope, a 3D accelerometer and a 3D magnetometer[2]. Besides the sensors, the IMU possesses dedicated signal conditioning circuits.

The aircraft has three control surfaces which are actuated independently with dedicated RC servo motors. The RC servo motors used for RC applications typically have integrated motor, gearbox, potentiometer and a control circuit. The control circuit accepts angular reference signals that are defined by the pulse width of the PWM signal that is supplied to the circuit. Obvious disadvantage

---

[2] We do not use the magnetometer in the experiments.

of using such servos is that they do not provide the feedback signal. The three reference signals are sent from the MCU on the interface board.

The interface board comprises of an ARM Cortex-M3 MCU, dedicated ports and accompanying electronics. In particular, the heart of the board is the Texas Instruments LM3S9B92 MCU clocked at 80 MHz. As briefly mentioned before, the MCU has three simple tasks:

1. gather data from the IMU,
2. send reference signals to the three servos and
3. communicate with the main controller PC via Ethernet.

The communication between the interface board and the main PC is bidirectional – the MCU receives the references which are sent to the servos and the sensor data is streamed down to the main controller PC.

6.1.2 Software

The software used within the experimental setup can be roughly divided in two groups. In the first group we have the necessary software for online tasks. Here we briefly refer to operating systems (OSs), real-time middleware (a software layer between OS kernel and user application), micro-controller firmware, safety, telemetry, logging, data acquisition, and finally real-time control tasks. The second software group can be regarded as a framework for modeling, control and estimation. This software group is for offline tasks.

The ground station PC is running a 64-bit version of Ubuntu 14.04 using the standard Linux kernel. This machine executes only noncritical tasks thus installation of a real-time kernel was unnecessary. The main function of this machine is to allow operator to control the main control PC. Besides the main function, the ground station PC also has another important role – to visualize telemetry data as fast as possible during the experiments; cf. Figure 6.2b.

The main controller PC is running a 64-bit version of Ubuntu 12.04 using a patched real-time preemptive kernel. The RT-PREEMPT patch [152] allows for advanced prioritization of user tasks (processes and threads) with the aim to minimize latency and jitter. The middleware used for this application is the

OROCOS toolchain [153] – an open-source software framework for control of robotic systems.

OROCOS is a component based framework that allows one to write real-time software components. Typically, a real-time application consists of a number of real-time tasks that need to be executed concurrently and synchronized in some prescribed way. A component in OROCOS terminology is a basic unit of functionality that executes real-time tasks in a single thread. Communication between the components is done via ports – a component can have a number of input and output ports. The toolchain itself ensures thread-safety and lock-free data exchange between the components. A component also has a set of operations that can be called by the operator or an another component. While the development of components is done in C++, the deployment is more conveniently done in a scripting language. At deployment level, the user typically does the components' configuration, e.g. defines a period of execution or sets component specific properties. Moreover, using deployment scripts the user defines connections between the components, defines priorities and CPU affinities. The deployment scripts also allow the user to start components. During runtime, the operator can execute the deployment scripts and interact with components (e.g. start, stop, and configure them) via a terminal application without breaking real-time data flow nor other real-time requirements. Furthermore, the terminal application has functionality for introspection of data on the components' ports.

While most of the software for online tasks runs on the main controller PC, there is also a firmware running on the MCU inside the airplane. The firmware code contains a set of interrupt service routines (ISRs) that execute the tasks described in § 6.1.1. There is an ISR periodically triggered by the IMU that gathers data from the sensors. A couple of other ISRs are necessary for running a TCP/IP server for communication with the main controller PC.

The real-time OROCOS components running on the main controller PC can be roughly organized as follows:

1. abstraction layer for sensors and actuators,
2. safety component,
3. controller and estimator components,
4. logging and
5. telemetry.

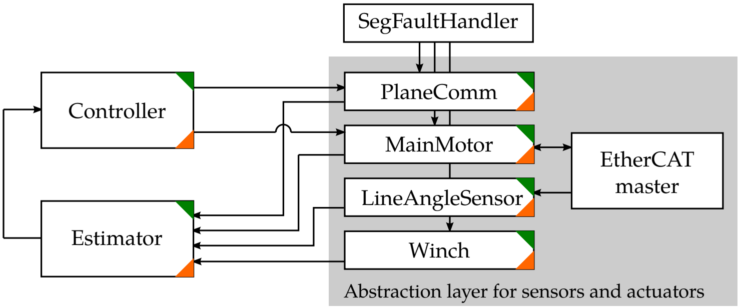


Figure 6.6: Simplified software organization.

A simplified organization of the real-time components is depicted in Figure 6.6. The abstraction layer for sensors and actuators encapsulates all components that directly and/or indirectly penetrate with hardware.

PlaneComm All communication with the airplane is done within this component. Communication is done in client-server fashion, where the component is the client. At each run of the component, the code sends references for the angles of the control surfaces to the motors. In case references for the angular velocities are received, the component internally calculates the reference angles. Furthermore, the component publishes sensor data on the corresponding output port.

EtherCAT master The EtherCAT master acts as a bus master and directly communicates with the two E/BOXs, cf. Figure 6.4.

MainMotor Communicates with the E/BOX that is connected to the main motor power amplifier and the encoder. The component can accept torque or torque slew-rate reference on an input port and publishes processed sensor data on the output port. In addition, this component can execute a speed PI controller which is used in the manual mode to warm-up the carousel after which the predictive controller takes over.

LineAngleSensor The horizontal and the vertical tether angles are published on the output port of the component upon data arrival from corresponding E/BOX.

Winch The component communicates with the dedicated servo amplifier. The component accepts tether length reference, and on an output port publishes actual tether length, tether speed (estimated by the servo amplifier) and the motor current.

The component deployment used in experiments always includes a safety component, *SegFaultHandler*[3]. The component's aim is to capture faulty OS events and perform a safe shutdown procedure: reset the airplanes control surfaces to neutral position, stop the carousel rotation at low deceleration and retract the cable to a default length. This way the equipment stays undamaged.

In the automatic mode of operation, the *Controller* and the *Estimator* components are employed. The estimator executes a sensor fusion algorithm using the information available on input ports (see Figure 6.6) and outputs the current state estimate on the output port. The controller component uses the state estimate to calculate control action that is applied to the system. The control algorithm outputs references for the airplanes' control surfaces and torque reference for the main motor. To this date, the automatic controller cannot control the tether length for the reasons that are going to be explained in § 6.2.3. Each of those two components is executed on a dedicated core, isolated from other tasks. This allows for a fully parallelized implementation of the real-time iteration (RTI) scheme, see § 2.4.

Components tagged with the green triangles, see Figure 6.6, have dedicated logging components. The logging components are automatically turned on at the beginning of each experiment and record data on output ports until the end of experiment. Logging components have low priority, thus buffered connections are required such that the data samples are not missed. A set of telemetry components gather the data from dedicated components (marked with orange triangles, see Figure 6.6), serializes it using the Google Protobuf software library and publishes the serialized data on a port using the ZeroMQ sockets. Those components are set to low priority as well, thus usage the non-buffered connections prefers publishing of the newest samples. Software executed on the ground station PC subscribes to appropriate data ports, deserializes data, and visualizes the data on screens (see Figure 6.2) with some negligible delay.

Synchronization of OROCOS components depicted in Figure 6.6 is summarized in Table 6.1. A bit more detailed analysis reveals that all hardware related

---

[3] The name comes after the *segmentation fault* OS signal which occurs upon a bad memory access.

Table 6.1: Synchronization of the real-time software components.

Component	Type	Period [ms]	Trigger
SegFaultHandler	Event driven	/	OS events
PlaneComm	Periodic	2	A timer
EtherCAT master	Periodic	1	A timer
MainMotor	Event driven	/	EtherCAT master
LineAngleSensor	Event driven	/	EtherCAT master
Winch	Periodic	20	A timer
Estimator	Periodic	40	A timer
Controller	Event driven	/	Estimator

components are indeed periodic, providing data on corresponding output ports at some regular rate. The estimator component is triggered by a timer at the rate that will be justified in § 6.2. The controller naturally gets triggered after the state estimate becomes available, running at the same pace as the estimator does.

Real-time Software Deployment

Mapping of the software components to computational cores inside the CPU is illustrated in Figure 6.7. The main controller PC is equipped with a CPU with 4 physical cores, each of which can execute simultaneously 2 instruction streams. This makes 8 logical cores in total. Operating system tasks are allowed to run on physical cores 0 and 1, i.e. logical cores 0, 4, 1, and 5. Core 0 is dedicated to execution of non-real-time critical tasks: logging and telemetry; cf. Figure 6.6. Those components, 12 in total, are scheduled using a non-real-time Linux scheduler with the lowest priority[4]. All other tasks are scheduled using a real-time scheduler. Real-time tasks such as the SegFaultHandler component and the components for communication with the sensors and the actuators are executed on core 1. All components run with high priority, except the Winch component. This decision is made based on the slower execution period and longer execution times compared to the other components executing on logical core 5. Cores 2 and 3 are completely isolated, and the user is responsible to map tasks to them. We use those two cores for execution of the most expensive computational tasks in our deployment: the controller and the estimator.

---

[4] In Linux, priority level 0 is the lowest priority, and priority level 99 is the highest priority.

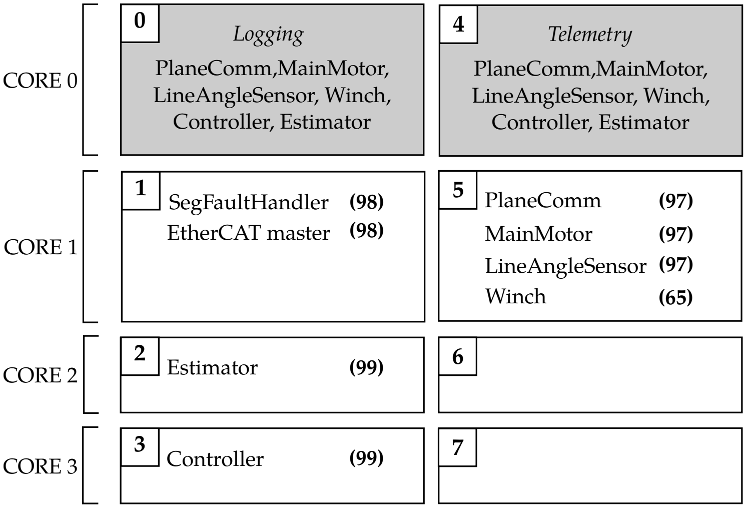


Figure 6.7: Simplified deployment of the real-time software components. Non-real-time components are shaded. Task priorities are marked in parentheses.

Mapping of the components to the cores, in our experience, showed to be one of the most sensitive practical tasks. Inappropriate mappings sometimes resulted in increased, random, jitter[5]. Moreover, unexpected and sometimes random increases in execution times were observed.

Data Acquisition System

The data acquisition system (DAQ) comprises a set of real-time software components and dedicated hardware for signal conditioning and acquisition, cf. § 6.1.1. Examining Figure 6.6 we can identify the following DAQ components: *PlaneComm*, *MainMotor*, *LineAngleSensor* and *Winch*.

The measurements gathered from the experimental setup are summarized in Table 6.2. Making the assumption the connection between the motor, gearbox, and the shaft is rigid, the angle of the carousel can be easily calculated. In other words, the carousel angle can be measured based on encoder measurements and gearbox ratio (a fixed parameter). The assumption holds if appropriate actions are taken to avoid the resonant modes of the carousel, cf. § 6.2.3. The torque of the carousel can be measured directly based on the current measurements and

---

[5] A deviation from the desired periodicity of a task.

Table 6.2: Measurement information.

Source component	Measurement	Period [ms]	(Frequency) ([Hz])	Delay [ms]
PlaneComm	Angular velocities	2	500	1
	Accelerations	2	500	1
Main motor	Carousel angle	1	1000	0
	Carousel torque	1	1000	0
LineAngleSensor	Tether angles	1	1000	0
Winch	Tether length	20	50	10
	Tether speed	20	50	10
	Winch motor current	20	50	10

the electrical motor constant. This statement holds true if the motor is operated within its nominal regime – in this application this is always true mainly because of the limitations of the dedicated power amplifier. Under similar assumptions cable length can be directly measured based on encoder measurements and the fixed parameters: gearbox and belt transmission ratios and drum diameter.

All measurements' sample rates are within the manufacturer specified ranges and/or subject to software limitations. For example, the IMU inside the airplane provides data at 800 Hz clock rate but the communication link implementation between the MCU and main controller PC limits the reliable sample rate to 500 Hz. The sample delays are mainly due to communication delays; e.g. the 10 ms delay of the tether measurement is introduced by the slow RS232 connection the winch servo amplifier provides.

Real-time Simulator

Testing newly developed control algorithms and real-time components is a tedious process usually requiring debugging activities. Exercising such activities on the real hardware, in experiments, can be potentially dangerous for both the operator and equipment. Instead, the usual practice is to perform 1) offline simulations, 2) software-in-the-loop (SIL) and/or 3) hardware-in-the-loop (HIL) simulations. While the offline simulations are typically exercised during the control design phase, the SIL and HIL simulations are typically used to test hardware and software integration before deployment to the real hardware. SIL simulations involve a software simulator running on the same machine as e.g. estimator and controller, running in real-time, to simulate accurately the hardware – model of the system, sensors, actuators, communication delays etc. A simulator in HIL simulations is typically executed on a separate piece of computational hardware.

Utilizing the flexible component based architecture, the DAQ components can be easily replaced by a simulator component. We can perform the SIL simulations with the main aim to test control and estimation algorithms in the real-time environment. The DAQ real-time components' testing is done separately. In our case, the simulator component executes the same simulation model used for design of the estimator and the controller at 1 kHz and outputs the measurements as prescribed in Table 6.2.

A Software Framework for Modeling, Control and Estimation

The offline tasks include system modeling, control design, code generation of controllers and estimators, calculation of initial guesses and references for the controllers. A Python tool called rawesome [123] is created with the idea to provide easier integration of the existing tools for modeling, nonlinear optimization and optimal control. In particular, the tool is mainly the front-end for ACADO CGT and CasADi. The tool provides intuitive API (application programming interface) for symbolic modeling and specification of OCPs as well as general NLPs using the CasADi framework for automatic differentiation and optimal control [154]. Once the OCPs are specified, either for MHE or MPC, ACADO CGT is called to generate the optimized code. The tool compiles the generated code and the compiled code can be used via a convenient wrapper in offline simulations to test the control design. On the other hand, the compiled code can be directly used within the real-time components. Furthermore, the offline control design process involves a set of Python scripts for offline calculations of initial guesses for the OCP solvers as well as the steady state references that are embedded into the real-time control component – those topics are going to be covered in § 6.2.

6.2 Control Architecture

Recent improvements and extensions to the ACADO CGT enabled testing of NMPC controllers employing different QP solvers. Those improvements allowed for shorter sampling times with the aim to react faster to disturbances acting on the airplane. Moreover, improvements of the control software enabled us to test and use different formulations for the MHE and the MPC on longer horizons.

Hardware and software improvements resulted in a redesigned data-acquisition (DAQ) system, i.e. inclusion of tether length, tether (line) angles and main motor current measurements in the state estimation algorithm. Tether length and angles measurements replaced the 3D camera system used for gathering absolute measurements, simplifying the sensor fusion process and testing of the estimator. Such estimator is expected to eventually work outdoors. Direct control of the torque on the carousel's main axis with the NMPC is made possible with the main motor current measurement.

In this section we present the simulation model used for experimental purposes as well as design of the estimator and the controller. The dynamics of the carousel coupled with 6-DOF (degrees-of-freedom) dynamics of the airplane are described by a DAE developed in [155, 156]. The estimation and control strategies we use here for the experiments are based on the work presented in [157, 158, 159]. Those strategies are extended and modified taking further into account hardware and software limitations, as well as the experience gained in the laboratory.

Ultimately we aim to employ the moving horizon estimator together with the nonlinear model predictive controller in the closed-loop experiments. We want to demonstrate capabilities of the experimental setup and the available software for fast NMPC and MHE while performing simple maneuvers with the airplane. With the current hardware, our desire is to fly at a constant tether length and a constant rotational speed while changing the roll angle and consequently height of the airplane. In comparison with [15] we use the same control strategy but with a more detailed model and updated formulations for the controller and the estimator. Let us stress once again that a discussion whether the NMPC and the MHE are the best strategies for rotational start-up of an AWE system is out of the scope of this thesis.

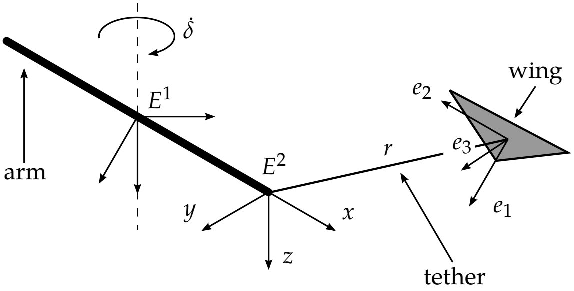


Figure 6.8: The schematic representation of the carousel.

6.2.1 Modeling

The dynamics of the carousel are coupled with the 6-DOF (degrees-of-freedom) dynamics of the wing, and both are jointly described by a DAE model first proposed in [155, 156].

The model layout is illustrated in Figure 6.8. A right-handed inertial reference frame E^1 is attached to the axis of the carousel at the level of the arm. Another reference frame E^2 is attached to the tip of the arm at the attachment point of the tether, and rotates with the arm around its z-axis, which points down. The position of the center of mass (CM) of the wing is expressed in Cartesian coordinates $p = [x, y, z]'$ in the reference frame E^2. A third reference frame e is attached at the CM of the wing, and is aligned with its main axis. The orientation of the wing relative to the reference frame E^2 is captured via the direction cosine matrix (DCM) $R \in \mathbb{R}^{3\times3}$ which describes the rotation that aligns reference frame e to E^2. The rows of $R = [e_1', e_2', e_3']'$ provide three vectors describing the main axes of the wing in the reference frame E^2 (see Figure 6.8).

The fact that the airplane is tethered is reflected in an algebraic constraint:

$$c = \frac{1}{2}\left((p + R'p_{\mathrm{T}})'(p + R'p_{\mathrm{T}}) - r^2\right) = 0, \tag{6.1}$$

where p_{T} denoted the tether attachment point position at the aircraft expressed in the body frame and r is the tether length.

Modeling of system dynamics can be done in the framework of Lagrangian mechanics and yields an index-3 DAE [156]. Using the index reduction techniques an index-1 DAE is obtained, put in the form

$$\begin{aligned} 0 &= f(X, \dot{X}, Z, U) \\ 0 &= C(X) \end{aligned} \tag{6.2}$$

where $C(X)$ represents the consistency conditions that must be enforced at some time instant of the system's trajectory. Vectors of the differential states, the algebraic states and the inputs read:

$$X = [p', \dot{p}', e_1', e_2', e_3', \omega', r, \dot{r}, \ddot{r}, \mathrm{c}_\delta, \mathrm{s}_\delta, \dot{\delta}, u_\mathrm{t}, u_\mathrm{a}, u_\mathrm{e}]' \in \mathbb{R}^{27}, \tag{6.3}$$

$$Z = [\mu] \in \mathbb{R}, \tag{6.4}$$

$$U = [\dddot{r}, u_\mathrm{st}, u_\mathrm{sa}, u_\mathrm{se}]' \in \mathbb{R}^4. \tag{6.5}$$

Here, the angular velocity of the aircraft expressed in the body frame is denoted with ω. Use of the carousel rotation angle δ directly in (6.5) introduces an unbounded variable that can easily lead to numerical problems in the integrator. To avoid this problem, we opt to use cosine c_δ and sine s_δ of the carousel rotation angle as the states in the model. In contrast to the simulation studies [157, 158, 155], we prefer to control torque u_t on the carousel rotation axis instead of the acceleration $\ddot{\delta}$. The carousel rotational dynamics from u_t to $\dot{\delta}$ is modeled as a first order LTI system, taking into account carousel's inertia and friction. Angles of the control surfaces for actuation of ailerons and the elevator are denoted with u_a and u_e, respectively. Let us remind the reader that u_t, u_a, u_e and r are the references sent to the low-level controllers, as explained in § 6.1.1 and § 6.1.2.

The multiplier μ is associated with the consistency conditions in (6.2):

$$C = [c,\ \dot{c},\ e_1'e_1 - 1,\ e_2'e_2 - 1,\ e_3'e_3 - 1,\ e_1'e_2,\ e_1'e_3,\ e_2'e_3,\ \mathrm{c}_\delta^2 + \mathrm{s}_\delta^2 - 1]'. \tag{6.6}$$

The first condition is the algebraic constraint of the DAE (6.1) and the second one comes as a result of the index reduction. The remaining seven conditions must be enforced to preserve the properties of the over-parametrized rotations.

The vector of inputs counts four elements: the third order derivative of the tether length and the slew-rates of the real control actions. The advantage of this is that we are smoothing the control actions and able to limit the corresponding derivatives. On the other hand, this approach introduces four extra states in the system dynamics model.

Outputs The experimental setup is equipped with a number of sensor as described in § 6.1.2. The DAQ software provides the measurements summarized in Table 6.2. In this context, the outputs of the system can be summarized as:

$$h = [a'_{\mathrm{IMU}}, \omega'_{\mathrm{IMU}}, \alpha_{\mathrm{LAS}}, \beta_{\mathrm{LAS}}, r, u_{\mathrm{t}}, \mathrm{c}_\delta, \mathrm{s}_\delta]'. \tag{6.7}$$

In particular, some of the states can be directly measured: $r, u_{\mathrm{t}}, \mathrm{c}_\delta, \mathrm{s}_\delta$, under the assumptions made in § 6.1.2. The accelerometer and the gyroscope measurements output functions read:

$$a_{\mathrm{IMU}} = R_{\mathrm{IMU}} R' \left(\ddot{p} - \dot{\delta}^2 \begin{bmatrix} x + r_{\mathrm{A}} \\ y \\ 0 \end{bmatrix} + 2\dot{\delta} \begin{bmatrix} -\dot{y} \\ \dot{x} \\ 0 \end{bmatrix} + \ddot{\delta} \begin{bmatrix} -y \\ x + r_{\mathrm{A}} \\ 0 \end{bmatrix} - \begin{bmatrix} 0 \\ 0 \\ g \end{bmatrix} \right), \tag{6.8}$$

$$\omega_{\mathrm{IMU}} = R_{\mathrm{IMU}} \omega, \tag{6.9}$$

where R_{IMU} is the orientation of the IMU w.r.t. the body frame. The line angle sensor provides measurements of the horizontal and vertical tether angles:

$$\alpha_{\mathrm{LAS}} = \arctan \frac{y + z_{\mathrm{T}} e_{32}}{x + z_{\mathrm{T}} e_{31}}, \quad \beta_{\mathrm{LAS}} = \arctan \frac{z + z_{\mathrm{T}} e_{33}}{x + z_{\mathrm{T}} e_{31}}, \tag{6.10}$$

where $e_3 = [e_{31}, e_{32}, e_{33}]$.

Uncertainties The uncertainties enter the aerodynamics part of the model only, as proposed in [158]. In [158], authors propose introduction of uncertainties as translational and rotational components of the wind field modeled as the first order Markov-chains. We use a similar approach, the one introduced in [159], where the uncertainties are modeled as scaled virtual forces w^{fa} and scaled virtual torques w^{ta} acting on the aircraft. The uncertainties are modeled as integrating white noise. The actual virtual forces and torques introduce six extra states:

$$w_X = [w^{\mathrm{ta}}_x, w^{\mathrm{ta}}_y, w^{\mathrm{ta}}_z, w^{\mathrm{fa}}_x, w^{\mathrm{fa}}_y, w^{\mathrm{fa}}_z]', \tag{6.11}$$

with the corresponding inputs:

$$w_U = [\dot{w}^{\mathrm{ta}}_x, \dot{w}^{\mathrm{ta}}_y, \dot{w}^{\mathrm{ta}}_z, \dot{w}^{\mathrm{fa}}_x, \dot{w}^{\mathrm{fa}}_y, \dot{w}^{\mathrm{fa}}_z]'. \tag{6.12}$$

Model discretization In the dynamic optimization framework that is used for control and estimation, the dynamics is discretized using the direct multiple shooting. The implicit DAE (6.2) is discretized using the implicit Runge-Kutta integrator of order two, with two steps per shooting interval. The aforementioned simulation studies suggested that the sampling period of $T_\mathrm{s} = 100\,\mathrm{ms}$ is appropriate. However, during the first round of experiments [15], it was learned that the proposed sampling period is too long. Main reasons for this claim come from the facts that some of the model parameters are approximate and that strong disturbances are acting on the system. Additionally, let us emphasize that the model is developed for open-air scenarios, however we aim to perform experiments in a confined and small space. For those reasons it is chosen to decrease the sampling time to $T_\mathrm{s} = 40\,\mathrm{ms}$ with the ultimate idea to allow the controller to react faster to the disturbances. Furthermore, the sampling times of the estimator and the controller are chosen to be exactly the same, $T_\mathrm{s} = T_\mathrm{est} = T_\mathrm{ctrl}$.

Steady State Calculation

Numerical algorithms that are going to be used for integration of the DAE model (6.2) expect consistent initial guess, the one that satisfies the *invariant-in-time* constraint $C(X) = 0$. In the context of rotation start-up of an AWE system, it is natural to think about an initial guess representing the steady state, calculated for arbitrary tether length r_ss and carousel speed $\dot{\delta}_\mathrm{ss}$. Flying in a steady state, one can properly initialize the estimator and the controller without worrying too much about safety issues.

A naive approach to find a steady state enforces both nonlinear equations in (6.2) in NLP. However, both equations describe the same dynamics and the linear independence constraint qualifications (LICQ) issue arises. In practice, an iterative solver employed for steady state calculation typically needs higher number of iterations to successfully solve the problem. To avoid this issue, we use the approach based on projection of the constraint-Jacobian $J = (\nabla C)'$ suggested in [160] and based on ideas from [161]. The proposed strategy yields a condition

$$0 = \dot{X} - f(X, U) - J^{\dagger} C \tag{6.13}$$

in which $J^\dagger$ is the Moore-Penrose pseudo-inverse of constraint-Jacobian J. As $\dot{X}$ and Z enter linearly in the DAE it is possible to obtain explicit expressions

$$\begin{bmatrix} \dot{X} \\ Z \end{bmatrix} = \begin{bmatrix} f(X,U) \\ g(X,U) \end{bmatrix},$$

as explained in [160]. Accordingly, the explicit expression for $\dot{X}$ is embedded into (6.13).

Finally, we come to the *steady state calculation* procedure. The steady state characterized by the tether length r_{ss} and the carousel speed $\dot{\delta}_{\mathrm{ss}}$ one can get by solving the following NLP:

$$\min_{X,\dot{X}} \quad u_{\mathrm{a}}^2 + u_{\mathrm{e}}^2 \tag{6.14a}$$

$$\text{s.t.} \quad \text{projected system dynamics (6.13)}, \tag{6.14b}$$

$$r = r_{\mathrm{ss}},\ \dot{\delta} = \dot{\delta}_{\mathrm{ss}}, \tag{6.14c}$$

$$\underline{l}_{\mathrm{ss}} \leq c_{\mathrm{ss}}(X,\dot{X}) \leq \overline{u}_{\mathrm{ss}}, \tag{6.14d}$$

$$U = 0. \tag{6.14e}$$

where $\underline{l}_{\mathrm{ss}}$ and $\overline{u}_{\mathrm{ss}}$ denote the operating constraints.

The constraint (6.13) can be formed in a symbolic form and we use the symbolic framework and advanced automatic differentiation features of CasADi [154] software tool for this purpose. The NLP is solved by employing the IPOPT solver.

6.2.2 Estimator

The estimator that we aim to employ is a moving horizon estimator that at each time instant solves a nonlinear least-squares equality constrained problem. The NLP formulation reads:

$$\min_{X_{\text{aug}}, w_U} \quad \frac{1}{2}\rho \left(\sum_{k=0}^{N_\text{e}} \left\| \begin{bmatrix} h(X_{\text{aug},k}) - \tilde{y}_k \\ h^\text{p}(X_{\text{aug},k}) - \tilde{y}_k^\text{p} \end{bmatrix} \right\|_{W_X}^2 + \sum_{k=0}^{N_\text{e}-1} \|w_{U,k}\|_{W_U}^2 \right), \tag{6.15a}$$

$$\text{s.t.} \quad X_{\text{aug},k+1} = F_\text{int}(X_{\text{aug},k}, Z_k, U_k, w_{U,k}),\ k = 0, \dots N_\text{e} - 1, \tag{6.15b}$$

$$C(X_{\text{aug},k}, Z_k)|_{k=N_\text{e}} = 0. \tag{6.15c}$$

The vector of augmented differential variables is denoted with $X_{\text{aug},k} = [X'_k, w'_{X,k}]'$, the vector of inputs with U_k, the vector of algebraic variables $Z_k = [\mu]$ as in (6.5). The output of the numerical integrator is F_int. A continuous-in-time formulation counterpart is discretized on an equally spaced time grid with $T_\text{est} = 40\,\text{ms}$.

Besides penalization of misfits between the real measurements $\tilde{y}_k$ and the output function h, there is also penalization of the *pseudo*-outputs defined with the following output function:

$$h^\text{p} = [\dot{r}, \ddot{r}, u_\text{a}, u_\text{e}, w'_X]'. \tag{6.16}$$

Since the aim is to perform the experiments at the constant tether length and we know that the speed of the tether is low in the manual mode of operation (i.e. when the controller is off), we can incorporate that knowledge by penalizing $\dot{r}$ and $\ddot{r}$. Next, even though we cannot gather measurements of the control surfaces, we penalize the misfit between the references sent to the actuators and corresponding optimization variables such that we get the smooth state estimates and well conditioned estimator. For the same reasons the virtual torques and forces are penalized. *Summa summarum*, the vector of pseudo measurements at the time instant i is formed as

$$\tilde{y}^{\text{p},\,[\text{i}]} = [0, 0, u_\text{a}^{\text{ref},\,[\text{i-1}]}, u_\text{e}^{\text{ref},\,[\text{i-1}]}, 0_{1\times 6}]'. \tag{6.17}$$

The diagonal elements of weighting matrices W_X and W_U represent inverse of variance of particular measurements. The corresponding standard deviations are defined after consulting specifications from sensor manufacturers and incorporating some practical knowledge. The standard deviations are summarized in Table 6.3, and the scaling factor in the objective is chosen to be $\rho = 0.001$.

The horizon length is chosen to be $N_\text{e} = 15$ samples, i.e. 0.6 seconds, with the

Table 6.3: Standard deviations for measurements and disturbances.

	Standard deviation	Unit
Measurements		
a_{IMU}	1.633	$\mathrm{m/s^2}$
ω_{IMU}	0.174	rad/s
$\alpha_{\mathrm{LAS}}, \beta_{\mathrm{LAS}}$	0.035	rad
r	0.01	m
$\dot{r}$	0.1	m/s
$\ddot{r}$	0.1	$\mathrm{m/s^2}$
u_{t}	0.02	Nm
$\mathrm{c}_\delta, \mathrm{s}_\delta$	0.009	-
$u_{\mathrm{a}}, u_{\mathrm{e}}$	0.006	rad
w^{ta}	0.1	$\mathrm{Nms^2/kg}$
w^{fa}	0.1	Ns/kg
Disturbances		
$\dot{w}^{\mathrm{ta}}$	1.0	Nms/kg
$\dot{w}^{\mathrm{fa}}$	1.0	N/kg

Table 6.4: Dimensions relevant to the MHE formulation (6.15).

				Sparse QP			Condensed QP		
n_x	n_z	n_u	N_{e}	n_{v}	n_{b}	n_{c}	n_{v}	n_{b}	n_{c}
33	1	6	15	618	0	9	123	0	9

chosen sampling time of $T_{\mathrm{est}} = 40\,\mathrm{ms}$. This choice of horizon length is found to provide reliable estimates even in the absence of the arrival cost. In addition, this length of the horizon is in accordance with the findings presented in [159].

The consistency condition (6.15c) is enforced at the last node, $k = N_{\mathrm{e}}$, such that the consistent state feedback is provided to the controller. Using the consistent feedback signal, the invariants are preserved in the controller by numerical integration and enforcement of the multiple shooting matching constraints. The MHE formulation (6.15) relevant dimensions are summarized in Table 6.4. Therein, n_x denotes the dimension of the augmented state vector X_{aug} and n_u denotes the dimension of the disturbance vector w_U.

A consistent initial guess provided to the MHE solver is calculated offline, using the technique explained in § 6.2.1. All shooting nodes are initialized with some steady state value and it is assumed that the operator turns on the estimator in the vicinity of the initial guess. Afterwards, before the solver is turned on, the cosine c_δ and s_δ are adjusted to approximately match the real measurements. Once turned on, the solver outputs the estimates when $k \geq N_\mathrm{e}$. Before this condition is met, a data buffer is filled in with the measurements.

Pre-processing of the Measurements

The measurements provided by the DAQ software come at multiple rates, cf. Table 6.2. Although MHE can operate on multi-rate data, the software tool capable of supporting such a setup is not available at the moment. Given this limitation, the formulation (6.15) is justified as follows.

As an example, consider a scenario where measurements coming from the IMU, the LAS and the main motor are synchronized to a common (high) frequency f_s in the range $100 - 500$ Hz. In such a setting, the MHE algorithm has to either embed the *a priori* information from an arrival cost calculator [33] or to employ a high number of estimation intervals. Those requirements are related to the systems dynamics, and the need for estimation horizons of at least 0.6 s for the production of reliable estimates; see [159] for more details. A formulation without an arrival cost and a high number of the intervals is avoided because of the unacceptable execution times of the currently unavailable software implementations. Arrival cost calculation is not under consideration because of the lack of a proper software implementation.

For the aforementioned reasons a hybrid approach is employed. For each (pseudo-) output with sampling rate higher than f_est, the corresponding measurement sent to the MHE at time instant T_i is approximated using the real measurements obtained on interval $(T_i - 1, T_i]$. In particualar, the estimator working at $f_\mathrm{est} = 25\,\mathrm{Hz}$ receives at each sampling instant, in average: 40 samples from the main motor and the LAS, and 20 samples from the IMU; cf. Table 6.2. Each of the outputs $\{a_\mathrm{IMU}, \omega_\mathrm{IMU}, \alpha_\mathrm{LAS}, \beta_\mathrm{LAS}, u_\mathrm{t}, \mathrm{c}_\delta, \mathrm{s}_\delta, u_\mathrm{a}, u_\mathrm{e}\}$ has a dedicated least-squares interpolator to fit the buffered data using a second order polynomial. Based on our experience, the second order polynomial fits provided satisfactory results. The measurements of tether length r coming at the slower rate are appropriately embedded on the discretization grid of the MHE.

Software Implementation

The solver used for simulations and the experiments is auto-generated by the ACADO CGT. The only solver in the ACADO CGT suite that is able to solve the formulation (6.15) is the NLP solver which relies of the $\mathcal{O}(N^3)$ condensing routine, cf. § 3.2.1. Furthermore, the underlying QP is solved by the embedded version of the qpOASES QP solver. In the context of the RTI scheme, both the preparation and the feedback phase scale with $\mathcal{O}(N^3)$. Offline simulations revealed that for feedback execution times, the time spent in the QP solver, are more sensitive to horizon length than the time spent in the preparation phase. This observation heavily influenced the choice of the horizon length. The cubic complexity in the number of the discretization intervals also influenced the selection of input variables. In the formulation (6.15) the real DAE inputs U_k are *not* the optimization variables, but the past inputs used solely for the model simulation. Inclusion of the variables U_k to the vector of optimization variables possibly leads to a real-time feasible formulation, but the offline simulations showed long execution times of the feedback step, predominated by the time spent in the QP solver.

6.2.3 Controller

As outlined in the beginning of this chapter, the goal of the closed-loop experiments is to do simple maneuvers with the airplane, flying at a constant tether length. The aircraft should repeatedly move from one steady state to the another one. The initial reference is a steady state that is calculated for a specific tether length $r_{\text{ref},0}$ and the carousel speed $\dot{\delta}_{\text{ref},0}$. Consequently, the steady state calculator gives the initial height of the airplane $z_{\text{ref},0}$. The second reference steady state is calculated for an arbitrary height of the plane $z_{\text{ref},1}$ that is above the initial one. In an oversimplified view, such a maneuver is primarily done by changing the angle of the ailerons and a consequence is a change of the roll angle of the aircraft. Our intention is not only to perform the maneuvers as fast as possible but also to show that the controller can efficiently stabilize the airplane at each steady state.

The dynamic optimization problem the NMPC controller solves at each sampling instant is defined by a following NLP:

$$\min_{X,U} \quad \frac{1}{2}\rho \left(\sum_{k=0}^{N_\mathrm{c}} \|X_k - X_{\mathrm{ref},k}\|_{W_X}^2 + \sum_{k=0}^{N_\mathrm{c}-1} \|U_k\|_{W_U}^2 \right), \tag{6.18a}$$

$$\text{s.t.} \quad X_{k+1} = F_\mathrm{int}(X_k, Z_k, U_k),\ k = 0, \dots N_\mathrm{c} - 1, \tag{6.18b}$$

$$\underline{C}_X \leq C_X(X_k) \leq \overline{C}_X,\ k = 0, \dots N_\mathrm{c}, \tag{6.18c}$$

$$\underline{C}_U \leq C_U(U_k) \leq \overline{C}_U,\ k = 0, \dots N_\mathrm{c} - 1, \tag{6.18d}$$

that comes from a continuous-in-time OCP formulation that is discretized on an equidistant grid with $T_\mathrm{ctrl} = T_\mathrm{s} = 40\,\mathrm{ms}$. The NLP (6.18) is a classic formulation with a least-squares objective, and constraints comprising discretized system dynamics (6.18b) and the box constraints on the states (6.18c) and the controls (6.18d). Feasibility issues that might occur in absence of terminal constraint and terminal penalty calculated from the discrete Riccati equation are avoided by using the long enough horizon N_c. The practical experience showed that for performing fast maneuvers short horizons, i.e. less than one second, work well but are insufficient for effective stabilization around setpoints far above the initial steady state. For those reasons it is chosen to work with $N_\mathrm{c} = 50$ intervals, equivalent to 2 seconds.

The diagonals of the diagonal weighting matrices W_X and W_U read

$$\begin{aligned} \mathrm{diag}(W_X) = [&0.0025, 0.04, 0.04, 0.4, 0.4, 40.0 \\ &10^2 \cdot 1_{1\times 3}, 10^3 \cdot 1_{1\times 3}, 10^2 \cdot 1_{1\times 3}, 0.04, 0.0004, 0.0004, \\ &10^{-12} \cdot 1_{1\times 3}, 10^2 \cdot 1_{1\times 2}, 2.533, \\ &0.002, 205.175, 73.863], \end{aligned} \tag{6.19}$$

$$\mathrm{diag}(W_U) = [10^{-12}, 0.053, 182.378, 65.656], \tag{6.20}$$

and the scaling factor is chosen to be $\rho = 0.002$. Note that in (6.20) appropriate units are chosen such that the dimensionless objective value is obtained. The choice of the weights forces the controller to primarily align the orientation of the airplane to the reference orientation. In essence, aligning the y-axis in the body frame of the aircraft with the unit vector e_2 of the reference orientation

Table 6.5: Dimensions relevant to the NMPC formulation (6.18).

				Sparse QP			Condensed QP		
n_x	n_z	n_u	N_c	n_v	n_b	n_c	n_v	n_b	n_c
27	1	4	50	1550	700	0	200	350	350

should provide accurate tracking of the roll angle. Consequently tracking of the height of the plane z should be satisfactory. Moreover, the NMPC should keep the speed as close as possible to the reference one. High penalties on the control actions should ensure a *non-nervous* behavior of the controller even in the presence of strong disturbances and non-modeled dynamics.

The box constraints on the states (6.18c) are summarized as follows:

$$-0.1\ \mathrm{m} \le z, \tag{6.21a}$$

$$-0.174\ \mathrm{rad} \le u_\mathrm{a}, u_\mathrm{e} \le 0.174\ \mathrm{rad}, \tag{6.21b}$$

$$-21.697\ \mathrm{Nm} \le u_\mathrm{t} \le 21.697\ \mathrm{Nm}, \tag{6.21c}$$

together with the box constraints on the control inputs (6.18d)

$$\dddot{r} = 0, \tag{6.22a}$$

$$-0.262\ \mathrm{rad/s} \le u_\mathrm{sa}, u_\mathrm{se} \le 0.262\ \mathrm{rad/s}, \tag{6.22b}$$

$$-43.394\ \mathrm{Nm/s} \le u_\mathrm{st} \le 43.394\ \mathrm{Nm/s}. \tag{6.22c}$$

The constraint on the height of the plane z is the safety one, preventing the controller to push the airplane close to the ceiling of the laboratory. The constraints on the control surfaces' angles u_a and u_e, equivalent to 10 deg, are chosen such that the airplane flies within the nominal flight envelope [158, 157, 123]. In this light, the tight constraints on corresponding slew rates are chosen to avoid bang-bang behavior of the controller. The dimensions relevant to the MPC formulation are summarized in Table 6.5.

The constraint on the motor torque is a hard constraint, reflecting the amount of current the current controller can supply to the motor. Together with the conservative bound on torque slew rate u_st and the high weights on u_t and u_st we ensure smooth control of the torque and speed on the carousel's rotational axis. As a consequence, the non-modeled resonant modes are avoided.

Preliminary steady-state experiments with $u_\mathrm{a} = u_\mathrm{e} = 0$ revealed that control of the tether length is limited at the speeds $|\dot{\delta}| \geq 50\,\mathrm{rpm}$. While prolongation of the tether is possible, the retraction is not. This is due to the limited power of the winch motor, incapable of overpowering strong forces in the tether. Consequently, it has been decided to perform experiments at constant tether length. Instead of removing the tether dynamics and fixing the tether length as a parameter, as it was done in [15], a simple tether model is kept in the proposed AWE model. Benefits of this decision are twofold. First, the simple tether model provides four extra degrees of freedom and the estimator can account for flexibility of the tether. Second, using the more complete model one can better assess computational demands for the MPC and the MHE that are eventually going to be employed on further AWE rotational start-up systems.

After each run of the NMPC solver, a set of references is sent to dedicated software components communicating with corresponding actuators. In particular, the optimized slew rates $u^+_{\mathrm{sa},0}$ and $u^+_{\mathrm{se},0}$ are being sent to a component in charge to calculate the reference signals for the control surfaces; see § 6.1.2. In a similar fashion, a component that communicates with the main motor receives $u^+_{\mathrm{st},0}$ and calculates the torque reference. In all cases, linear interpolation is employed inside the dedicated software components to yield approximate piece-wise linear references for the actuators.

The steady state initial guess for the solver is calculated offline using a method presented in § 6.2.1. All multiple shooting nodes are initialized with a steady state, and the cosine and the sine of the carousel rotation angle, c_δ and s_δ, are properly adjusted prior to activation of the controller.

Reference Generation and Sequencing

Instead of giving a sharp reference transition to the controller, the transition is smoothed. This approach is favored over an optimal trajectory mainly for safety reasons. Starting from the initial steady state characterized by $r_{\mathrm{ref},0}$ and $\dot{\delta}_{\mathrm{ref},0}$, we generate a sequence of steady states, such that the height changes smoothly

$$z_{\mathrm{ss}}(k) = z_{\mathrm{ref},0} + \frac{\Delta z}{1 + \exp(-l(k))} \tag{6.23}$$

from $z_{\text{ref},0}$ to $z_{\text{ref},0} + \Delta z$ using an input sequence $l(k) \in [-50, 50]$ of arbitrary length. In this sequence, the tether length and the carousel speed are always set to the same values. The sequence of NLPs of type (6.14) is solved offline employing the IPOPT [162] solver. Each NLP (6.14) is modified for the constraint $z = z_{\text{ss}}(k)$.

Taking into account the hardware capabilities, it has been decided to perform the experiments at the tether length of $r_{\text{ref},0} = 1.7$ m, and carousel speed of -55 rpm, i.e. $\dot{\delta}_{\text{ref},0} = -5.76$ rad/s. The negative speed accounts for the chosen direction of rotation of the carousel. Given those two parameters the airplane flies, ideally, in the *initial* steady state that is $z_{\text{ref},0} = 0.183$ m below the arm level. The second steady state is chosen to be $\Delta z = 0.125$ m above the initial one such that no constraints defined in (6.21) and (6.22) are hit.

Concatenating the sequence given in (6.23) and its mirrored version a periodic sequence is obtained. The first reference given to the controller is always the initial one. Samples of the periodic sequence of steady states are placed at the end of the horizon at an arbitrary speed. In other words, the operator can choose online the pace at which the new samples from the periodic sequence are introduced to the controller.

Software Implementation

The solver for the controller is generated by the ACADO CGT. The references and the initial guess are generated offline and exported to a C header file that is compiled together with the rest of the controller code.

All four types of NMPC solvers from the ACADO CGT suite, based on the particular QP solver used to solve the underlying QP, are able to solve the NLP (6.18). However, preliminary simulations revealed that only the condensing based OCP solver and the solver using the HPMPC QP solver give solutions in real-time and provide fast enough feedback. The qpDUNES based OCP solvers initially showed promising results while running in ideal closed-loop simulations for shorter horizons. However, the much higher number of iterations was observed in simulations with longer horizons. This is suspected to be related to initialization of the dual variables in the QP solver. Consequently, the higher number of iterations led to unacceptable execution times. Testing an OCP solver based on the FORCES QP solver exposed long execution times already for a single QP iteration. Moreover, it has been decided not to use this type of solver because, in general, it does not produce fast feedback times (cf. § 3.5.4).

6.3 Experimental Results

A typical experiment starts at zero speed of the carousel in the manual mode. The operator activates the rotation of the carousel by setting a reference speed to a low-level PI controller. In our case the initial reference speed is set to be -50 rpm. Moreover, the initial, default, tether length is $r_{\text{init}} \approx 1.3$ m, and extended to the reference length of 1.7 m manually during carousel acceleration. During this *warm-up* phase, the operator typically sets some negative angle on the elevator to decrease the counter-torque the aircraft induces on the carousel rotation axis. Once the carousel rotation approximately reaches the reference speed, the estimator is turned on. The estimator is turned on at this point because the initial guess is, at the moment, calculated offline for the specific reference speed and cable length. Activating the estimator far away from the initial guess is possible, but occasionally fails.

Once the estimator is converged, typically within one second, the operator can safely turn on the automatic controller. The initial reference for the NMPC controller is set to be the steady state defined by the reference speed of -55 rpm and the same tether length of 1.7 m. The NMPC increases the speed of the carousel and stabilizes the plane around the setpoint. Once satisfied, the operator enters the command to start maneuvering – moving the plane between the two steady state setpoints.

The shutdown phase consists of slowing down of the carousel, retracting the tether to the default length and putting the control surfaces of the airplane to a neutral position. The tether retraction is done manually and the deceleration of the carousel is done by the external PI controller.

In the following we present closed-loop experimental results with the MHE and the NMPC. The results show the closed-loop performance for the two phases: 1) stabilizing around the initial setpoint (approximately first 50 seconds) and 2) maneuvering (the rest of the experiment).

The estimated position of the airplane is shown in Figure 6.9. One of the main objectives is to obtain accurate control of the height of the plane, the z-axis. We can see that this objective is on average well satisfied and that the controller can

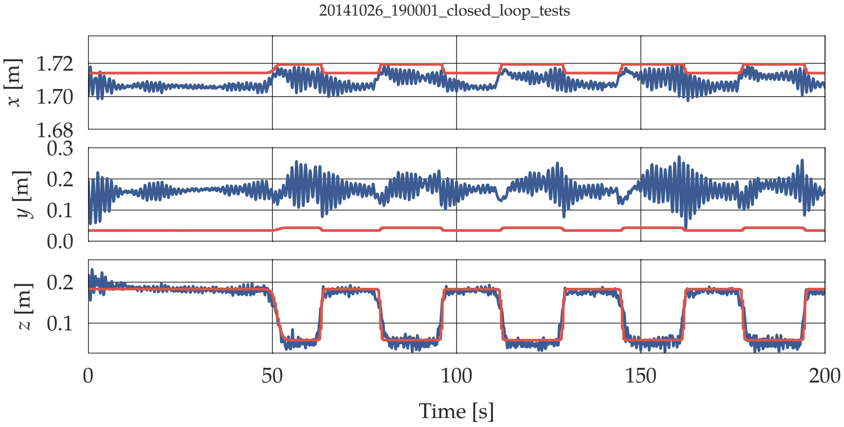


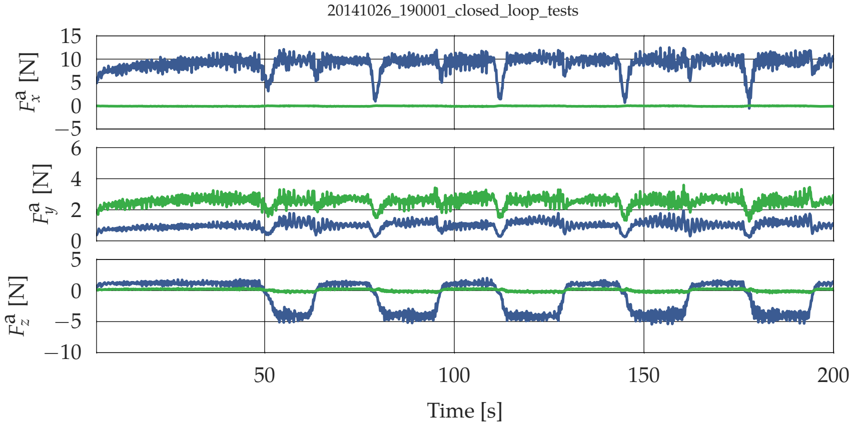
Figure 6.9: Position of the plane obtained in closed-loop experiments. Blue lines denote the estimates, and the red lines denote the references.

successfully stabilize the plane around the setpoints. The tracking of the x-axis is not the focus of these experiments, however, it is satisfactory.

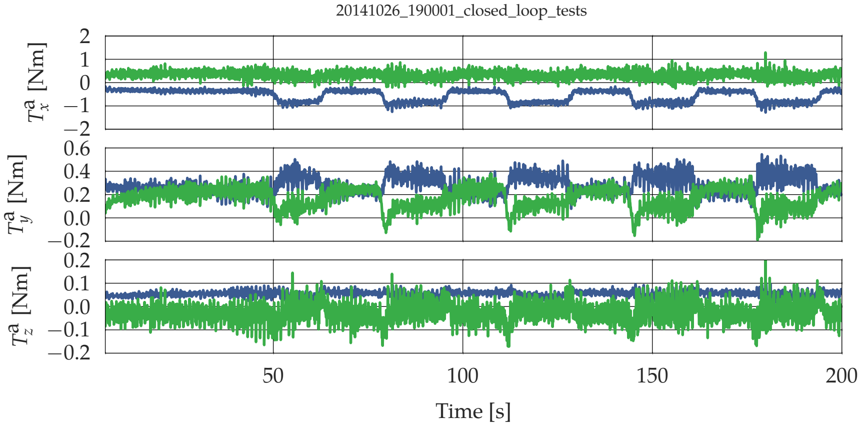
The unsatisfactory tracking of the horizontal position is a consequence of the following facts:

- The setpoints are calculated using the same simulation model that is used for control design. In other words, the disturbances are neglected while calculating the references.
- The estimates of the disturbances w^{fa} and w^{ta} are not fed into the controller.
- The controller is unable to compensate the oscillations because of the hardware limitations.

Online calculation of setpoints that incorporate knowledge about the disturbances is explored to some extent, but led to unsatisfactory results – essentially failures of the controller. Feeding the controller with the disturbances but using the *ideal* references also led to unsatisfactory results. It is believed that those attempts failed mainly because of the presence of strong, non-constant, disturbances; see Figure 6.10. In particular, external torques on x-and y-axis of



(a) Estimated aerodynamic forces acting on the plane



(b) Estimated aerodynamic torques acting on the plane

Figure 6.10: Aerodynamic forces and torques estimated in closed-loop experiments. The blue lines denote quantities estimated from model and the green ones depict virtual quantities (disturbances).

the aircraft and the strong external force in y-direction are responsible for the plane lagging far behind the arm.

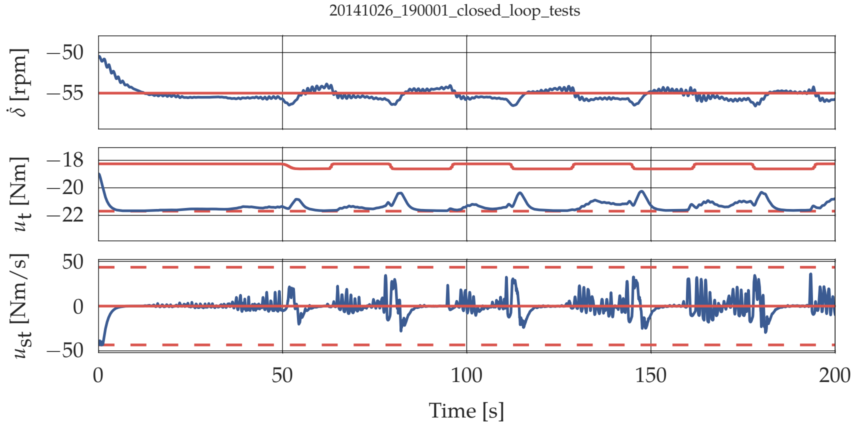


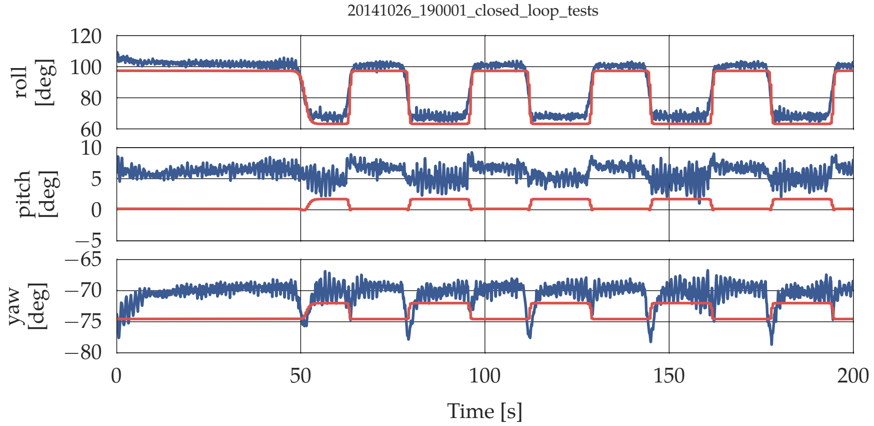
Figure 6.11: Carousel speed $\dot{\delta}$, torque u_{t} and torque slew rate u_{st} obtained in closed-loop experiments. The constraints are denoted by the dashed red lines.

It is arguable that the oscillatory motions in the horizontal direction can be compensated with the motor torque. However, this is not possible with the current hardware. Let us analyze the Figure 6.11. While maintenance of the reference speed is indeed possible, the torque is almost always at the constraint. From another point of view, even if we have had a stronger motor to drive the carousel, more dynamic torque actions possibly require proper handling of the resonant modes of the carousel – both in the model and the controller.

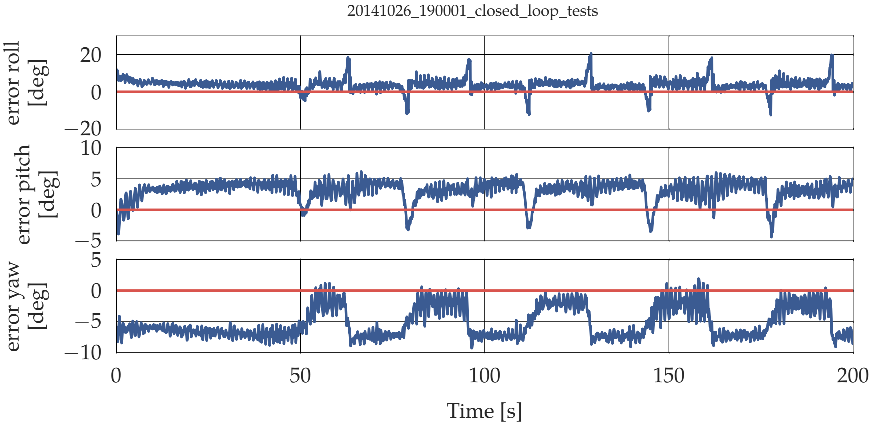
In Figure 6.11 we can see that the speed $\dot{\delta}$ can be tracked reasonably well, while the controller can compensate for the maneuvers. The torque slew rate constraint is hit only at the start of the experiment, while the carousel is accelerating to the initial setpoint.

The attitude of the airplane is depicted in Figure 6.12. The roll angle tracking is reasonably well. The small steady state offsets, on average less than six degrees (see Figure 6.12b), are the consequence of the disturbance torque acting on the x-axis.

The hardware limit on the torque is almost always active, and the NMPC struggles to maintain the speed by pitching the airplane. Finally, the controller does not have authority on the yaw angle at all in the lack of a rudder.



(a) Euler angles



(b) Error Euler angles obtained from error rotation $R'_{\mathrm{est}}R_{\mathrm{ref}}$

Figure 6.12: Euler angles obtained in closed-loop experiments.

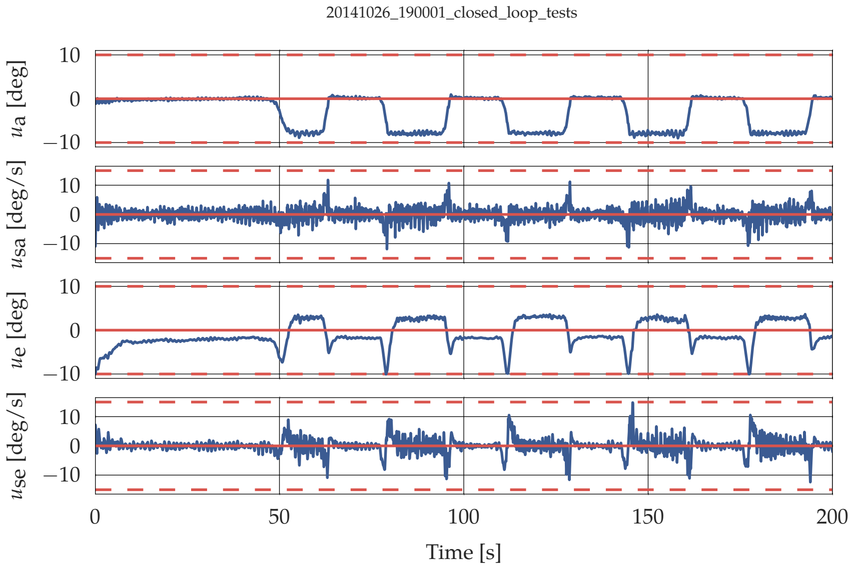


Figure 6.13: Control surfaces' angles and corresponding slew rates obtained in closed-loop experiments.

The control actions on the control surfaces of the airplane are given in Figure 6.13. We can see that all the constraints are well satisfied. Experience gained in the laboratory showed that more effective results are possible by tracking zero references for the aileron and the elevator than the relevant references given by steady state calculations. This is unsurprising given that the strong disturbances act on the plane and that the estimates of those are not fed into the controller.

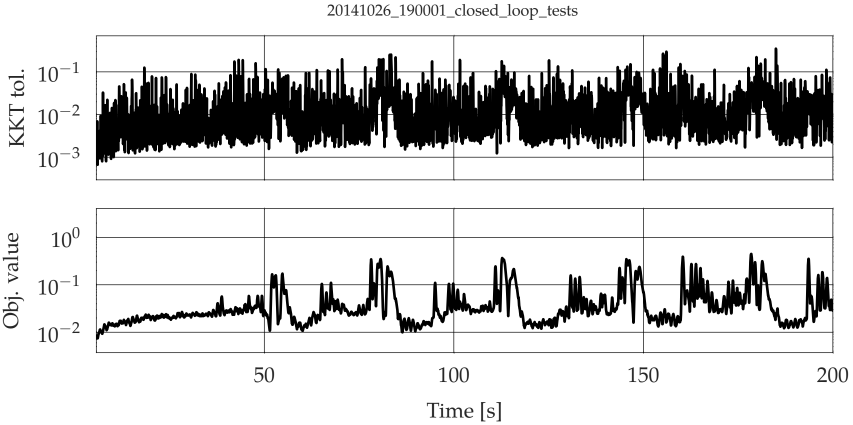


Figure 6.14: Performance indicators for the MHE obtained in closed-loop experiments.

Computational Performance

The MHE solves at each step the equality constrained nonlinear least-squares problem (6.15). The solution method employs the RTI scheme, where at the low level an equality constrained QP is solved at each time step. The QP is solved online by the online active-set solver qpOASES. Consequently, it is reasonable to expect practically constant execution time of the solver. The execution times obtained in the experiments are virtually flat, with fluctuations less than 200 µs, which we consider more than acceptable – compared to the sampling time and the execution time. The execution time of the feedback step is less than 3.5 ms on average and the preparation step is on average 8.5 ms long. In total, the execution time of the MHE solver is less than 12 ms and well below the sampling time of the estimator of $T_{\text{est}} = 40$ ms.

The computational performance of the MHE solver is illustrated in Figure 6.14. The low, non-drifting, objective values and the KKT tolerance suggest that the underlying optimization problem in the MHE is well defined and properly scaled.

Enforcement of the DAE invariants is confirmed in Figure 6.15. None of the constraints are drifting, even in the absence of stabilization of the invariants proposed in the simulation studies. However, one should take into account that

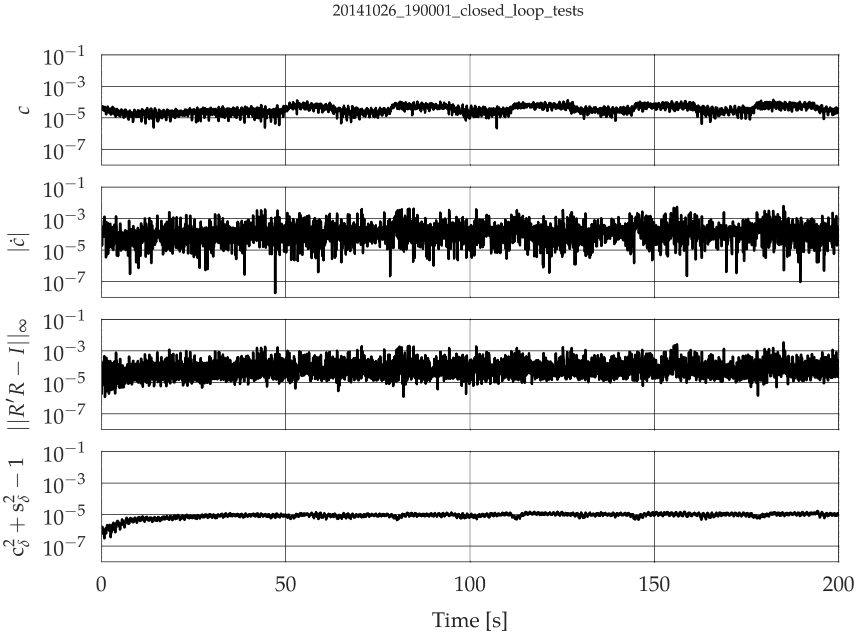


Figure 6.15: Estimated invariants defined in the equation (6.6).

prediction horizons and/or shooting intervals in the aforementioned simulation studies were longer than the ones used for the experiments presented in this chapter. In addition, we did not observe drifting invariants in other experiments performed over extended periods of time.

The task of the NMPC controller is to solve at each time instant the OCP given in the NLP form (6.18). The OCP consists of the least-squares objective and includes a set of bounds on the state and the control variables. Although the formulation can be solved by all solvers in the ACADO CGT suite, we employ only two solvers:

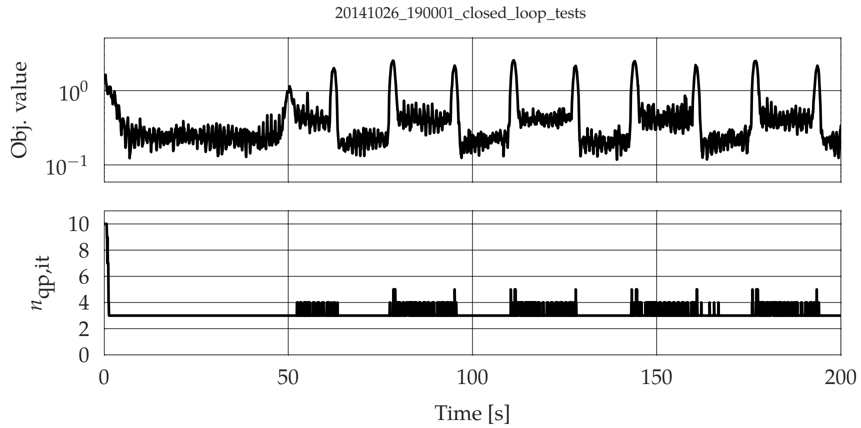
1. the solver coupled with sparsity exploiting QP solver HPMPC and
2. the solver based on the condensing procedure coupled to dense linear algebra QP solver qpOASES.

The reasons for this decision have been explained in § 6.2.3.

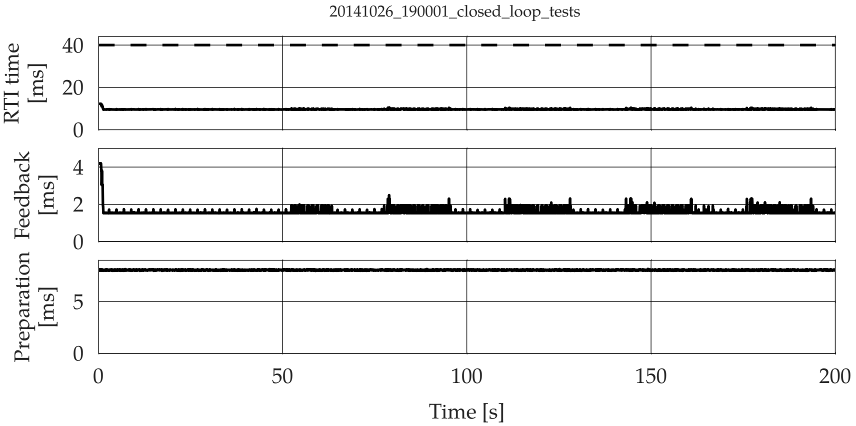
The closed-loop experimental results presented so far come from the experiments where the first NMPC solver is employed. The computational performance of the first NMPC solver is presented in Figure 6.16. The objective values suggest that the OCP is well scaled and we can also observe that on average the values are stable, i.e. not drifting[6]. The number of iteration is initially high, because we want to move the system to the reference point which is far away from the initial state of the system. Afterwards, i.e. during maneuvers, we can observe that the number of iterations is lower. This is in accordance with general observation that the number of iterations for the IPM-like solvers is general low. Next, we observe that the execution time of the RTI scheme is always below 13 ms, i.e. far below the period of execution of the controller. Within the execution time needed for the RTI step, the preparation step takes just above 8 ms. Finally, the feedback is calculated in less than 5 ms initially and in less than 3 ms afterwards. The optimization problem the solver has at hand consists of $50 \cdot 27 + 50 \cdot 4 = 1550$ variables, thus we can say the solver solution times are indeed outstanding!

---

[6] Computation of the KKT tolerance for the HPMPC based NMPC solvers was unavailable.



(a) Performance indicators for the NMPC

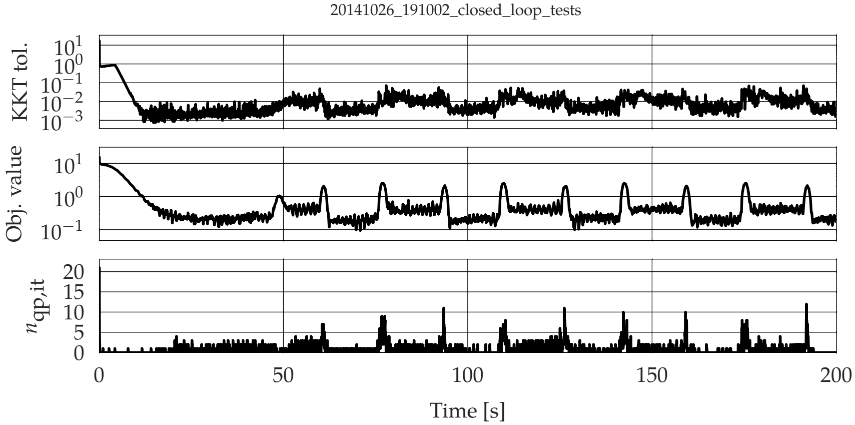


(b) Execution times for the NMPC. Dashed line in the top plot denotes the sampling time of the controller.

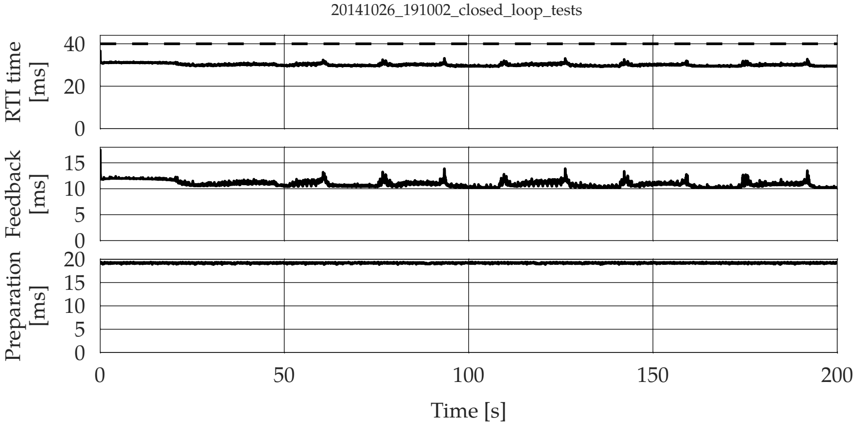
Figure 6.16: Performance indicators and execution times for the NMPC obtained in closed-loop experiments while using the HPMPC QP solver.

The computation performance of the second NMPC solver that utilizes the condensing strategy and solves the underlying QP with qpOASES solver is presented in Figure 6.17. The reader should be aware that the results come from a separate experiment, i.e. nonidentical results should be expected. The second solver solves exactly the same NMPC formulation as the first solver does. The number of iterations $n_{\text{qp, it}}$ is on average lower than with the first solver when the system is running near the steady state. However, the number of iterations during maneuvers is higher. This observation is in line with the observations commonly found in the literature that the active-set solvers typically show more iterations than interior-point solvers. Furthermore, we can observe that the number of iterations is high during the acceleration phase, where initially the system is far away from the reference.

The NMPC solver is always real-time feasible, but shows much higher execution times than the first solver which utilizes the *sparse*, i.e. non-condensing approach. Compared to the preparation time of the sparse solver, cf. Figure 6.16, we can observe that approximately 12 ms is spent in condensing. This is expected given that we have a high number of the shooting intervals. The feedback time of the solver is dominated by the time QP solver spends in factorizing the reduced Hessian, the computation proportional to $\mathcal{O}(n_u^3)$. A small amount of time is spent in condensing of the linear term that is $\mathcal{O}(N)$ operation. The *total feedback time* is higher than with the first solver, below 18 ms and always real-time feasible.

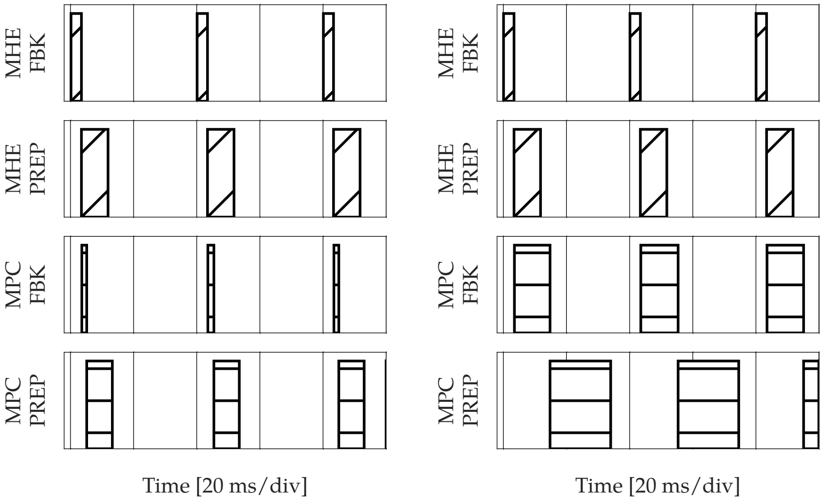


(a) Performance indicators for the NMPC



(b) Execution times for the NMPC

Figure 6.17: Performance indicators and execution times for the NMPC obtained in closed-loop experiments while using the condensing based approach and qpOASES QP solver.



(a) HPMPC based NMPC solver (b) qpOASES based NMPC solver

Figure 6.18: Samples of timing diagrams obtained in closed loop experiments.

Timing diagrams obtained in the closed loop experiments are depicted in Figure 6.18. A control period starts with the execution of the MHE feedback step (MHE FBK). After the state estimate is obtained, the MPC feedback phase is triggered (MPC FBK). The MHE preparation phase (MHE PREP) is triggered at the same time as the MPC FBK, because the MPC and the MHE are run in parallel, on separate processor cores. This reduces the overall time required to achieve the preparation phase of the overall scheme. The overall feedback time (i.e. time from measurements to control action) mostly amounts to solving the QP underlying the MHE and the MPC problems.

Low execution times of the HPMPC based NMPC solver and the MHE solver suggest that execution of both the NMPC and the MHE on a single core is in principle possible at the proposed sampling time, see Figure 6.18a. However, parallel execution is preferred in order to have a large safety margin, and allow possibly higher number of iterations inside the QP solver. In contrast, the qpOASES based solver exhibits much longer execution times. In this case, depicted in Figure 6.18b, separate execution of the MHE and the NMPC is of utmost importance to satisfy the real-time feasibility.

6.4 Conclusions

In the beginning of the chapter and in § 6.2 we defined our objectives as to perform simple maneuvers with the airplane: tracking of the height of the airplane. Tracking of the height is in turn a consequence of tracking of the reference orientation. Moreover, we required all prescribed constraints to be satisfied as we use the control techniques that by design ensure this. Finally, we wanted to demonstrate the capabilities of the developed software tools. In more details, we wanted to show 1) that the solvers provide real-time feasible solutions and 2) that the solvers can provide fast enough feedback. The NMPC and the MHE problems are efficiently solved in real-time using recent features added to the auto-generated solvers from the ACADO CGT software tool. We experimentally verified two variants of NMPC solvers, employing different structure exploiting methods for the solution of the underlying QP, in combination with an MHE solver. The results show that both variants are real-time feasible and emphasize the benefits of parallel execution of the solvers on multi-core hardware.

It is worth noting that experiments with the condensing based NMPC showed similar control performance to the first NMPC solver that employs the structure exploiting QP solver. This can observed assessing the profiles of the objective values in Figure 6.16a and Figure 6.17a. At the moment we do not have tools for comparing the control performance between NMPC solvers, and a detailed comparison remains a topic for eventual future research activities.

Throughout this chapter we characterized the uncertainties essentially as disturbances, implicitly assuming the model is accurate enough. On a broader perspective, uncertainties include non-modeled dynamics as well. The results presented in this chapter might indicate that the model parameters are insufficiently well identified and that the parameter identification experiments should be performed in addition to the existing ones [163].

The reason for doing the indoors experiments is to verify the models, control design and software as a preparation for outdoors experiments. The knowledge gained through laboratory work should be invested further in the outdoors experiments where the system should be operated in a more realistic environment. Usage of the approximate model *and* the controller that is unaware of disturbances acting on the system resulted in control performance with obvious steady state offsets. On the other hand, the presented results indicate that the controller can successfully cope with such working conditions.

It remains an open question how to successfully cope with disturbances in a realistic scenario. The simulation study [157] suggest that modeling of the disturbances by introduction of the wind field (linear components only) for rotational start-up is possible, but there is no relevant information about offset-free tracking of the references. In the power production regime, simulation studies [164, 158] demonstrate successful NMPC schemes with both linear and rotational wind components as disturbances. The former study indicates that such a scheme might even produce more energy than specified.

7

Conclusions and Outlook

This thesis contributes in two areas relevant to bring nonlinear MPC and MHE closer to industrial applications. The first area is software, where the ACADO CGT has been extended to treat efficiently more general formulations than original implementation offered, thus be usable for a much wider range of applications. In the second area are applications. We tested the developed code on two challenging mechatronics applications: an overhead crane and an airborne wind energy system.

The ACADO CGT has been extended to support wider range of applications: Next to support for MPC formulations including nonlinear output functions and constraints, the extension for MHE formulations has been developed. The MHE and the MPC formulations are efficiently solved with the well-established RTI scheme, presented in Chapter 2. One of the crucial components for fast solution of the formulations and the key one to reduce the feedback delays is the fast solution of the underlying quadratic program. Efficient structure exploiting algorithms are outlined and extended in Chapter 3. One way to go for the fast QP solution is to condense the sparse QP into a smaller but dense one that can be solved by e.g. efficient active-set method QP solver. Just by counting FLOPs, a smart implementation of the classical condensing shows speed-ups greater than 70 % already for short horizons. Going even further, the new $\mathcal{O}(N^2)$ algorithm promises extended applicability of the condensing approach. Alternative way to solve the QP is to directly solve it by an efficient structure exploiting QP solver. We interfaced three different state-of-the-art structure exploiting solvers aimed for fast embedded applications.

The benchmark results from Chapter 3 indicate that fast solutions for nonlinear MPC are possible, even for the systems of relatively high state dimensions and formulations with fairly long horizons. For short horizons, the condensing approach proves to be competitive, as all the states are removed and the expensive computations are relocated to the preparation phase. Furthermore, this approach seems to be more preferable as the ratio n_x / n_u goes higher. On the other side of the spectrum, a well known fact is confirmed: in general, the structure exploiting QP solvers are more effective for longer horizons. In particular, those solvers are more effective as the ratio n_x / n_u goes lower. The well optimized HPMPC QP solver greatly supports the fact that CPU-specific optimizations for linear algebra routines are of paramount importance to achieve low execution times.

The ACADO CGT has been successfully used, to our knowledge, in at least seven real-worlds applications; see Chapter 4. Two out of those seven applications are studied in this thesis. We studied the practical considerations and applicability of the nonlinear MPC and MHE to real-world mechatronics applications in Chapters 5 and 6.

Two different control strategies with MPC in the loop have been tested on the laboratory scale overhead crane in Chapter 5. The choices on the control strategy have been mostly based on available features in the ACADO CGT. Although both approaches showed good closed-loop performance and the execution times, slightly better performance has been observed with MHE in the loop in the second control scenario. In addition to usage of the MHE in the loop, in the second scenario we also employed MPC with better optimized condensing strategy and more efficient integrators. As a result, it was possible to execute both MHE and the NMPC in slightly higher time than to run the MPC only with the first implementation of the auto-generated MPC from ACADO CGT.

In Chapter 6 are presented results from the second set of experiments on the Leuven kite carousel. The complex system dynamics is described with a DAE model that required usage of implicit integrators. In order to efficiently cope with strong disturbances, a fairly long horizon length was necessary. We tested two nonlinear MPC solvers, and the both approaches showed to be real-time feasible. On the available multi-core machine we were able to execute the preparation steps of the RTI scheme fully in parallel. Parallelization showed to be necessary in the case when condensing based MPC solver was used. In the second case, the structure exploiting solver showed superior performance being able to solve the MPC problem with more than 1500 variables in less than 5 ms.

7.1 Directions for Future Research

The benchmarks in Chapter 3 tested the formulations with diagonal stage Hessians and box constraints on the states and the controls. This case was chosen because it is the most usual use case. Furthermore, this is the formulation that is at this moment supported by all currently interfaced QP solvers. In this light, the benchmarks should be extended to at least test all available features of the available QP solvers; e.g. FORCES and qpDUNES QP solvers support the affine constraints as well. We used all QP solvers with their default settings, set by corresponding developers. Our observations showed that all solvers provided very similar solutions. More rigorous tests can reveal the relations between the execution time and the desired accuracy of the solution. The techniques for such comparisons are developed in [165]. Finally, the benchmarks we conducted do not include any comparisons for MHE solvers. While it is reasonable to expect that structure exploiting sparse QP will overpower the condensing approaches for standard MHE formulations with full state noise, the general case with arbitrary number of disturbances remains an open question.

The tracking objectives we are using in this thesis cover a wide range of applications. Nonetheless, the range of applications can we widened with support for more general objective for which the Exact Hessian approximation would be necessary. For example, the so called economic formulations and the time-optimal formulations for MPC are the perfect candidates. Initial work presented in [125] already shows promising results.

Recently, linear MPC was successfully realized on field programmable gate arrays (FPGAs), see e.g. [166, 167]. Even more, an application of an MPC with a custom running at 1 MHz was reported in [168]. An FPGA represents configurable hardware, a chip with basic logic elements such as AND- and OR-gates , fixed/floating point arithmetic-logic units (ALUs) and memory elements. Typically the number of ALUs is much higher than in modern CPUs, allowing for efficiently parallelized implementations of numerical algorithms. What makes FPGAs really attractive for MPC, or optimal control in general, is flexibility and number of processing elements in a single chip. Instead of programming the hardware at hand, one can create a fully customized architecture for specific application. Fortunately, in linear MPC some crucial calculations can be done before deployment. For example, based on the fixed sensitivity matrices A_k and B_k number of solver iterations can be calculated offline. Based on the fixed number of iterations, the custom solver implementation can be further optimized. Nonlinear MPC brings on the table even more challenges: algorithmic differentiation and numerical integrators. Experimental nonlinear MPC implementations on FPGAs have been reported in [169, 170, 171]. All three references report successful implementations for particular applications, however a (semi-)automated solver design is still missing.

Bibliography

[1] J. B. Rawlings, "Optimal dynamic operation of chemical processes: Assessment of the last 20 years and current research opportunities." Bayer Lecture, Department of Chemical Engineering, Carnegie Mellon University, Pittsburgh, Pennsylvania, April 2010. pages 3

[2] W. Chen, D. Ballance, and J. O'Reilly, "Model Predictive Control of Nonlinear Systems: Computational Delay and Stability," *IEEE Transactions on Automatic Control*, vol. 147, no. 4, pp. 387–394, 2000. pages 5

[3] R. Findeisen and F. Allgöwer, "Computational Delay in Nonlinear Model Predictive Control." Proc. Int. Symp. Adv. Control of Chemical Processes, ADCHEM, 2003. pages 5

[4] M. Steinbach, *Fast recursive SQP methods for large-scale optimal control problems*. PhD thesis, Universität Heidelberg, IWR, 1995. pages 6, 69

[5] C. Rao, S. Wright, and J. Rawlings, "Application of Interior-Point Methods to Model Predictive Control," *Journal of Optimization Theory and Applications*, vol. 99, pp. 723–757, 1998. pages 6, 20, 41, 69, 70, 71, 72

[6] M. Diehl, H. J. Ferreau, and N. Haverbeke, *Nonlinear model predictive control*, vol. 384 of *Lecture Notes in Control and Information Sciences*, ch. Efficient Numerical Methods for Nonlinear MPC and Moving Horizon Estimation, pp. 391–417. Springer, 2009. pages 6, 14, 25

[7] B. Houska, H. Ferreau, and M. Diehl, "An Auto-Generated Real-Time Iteration Algorithm for Nonlinear MPC in the Microsecond Range," *Automatica*, vol. 47, no. 10, pp. 2279–2285, 2011. pages 7, 98, 100, 118

[8] M. Diehl, H. Bock, J. Schlöder, R. Findeisen, Z. Nagy, and F. Allgöwer, "Real-time optimization and Nonlinear Model Predictive Control of Processes governed by differential-algebraic equations," *Journal of Process Control*, vol. 12, no. 4, pp. 577–585, 2002. pages 7, 15, 25

[9] M. Vukov, W. V. Loock, B. Houska, H. Ferreau, J. Swevers, and M. Diehl, "Experimental Validation of Nonlinear MPC on an Overhead Crane using Automatic Code Generation," in *Proceedings of the 2012 American Control Conference, Montreal, Canada.*, 2012. pages 8, 9, 10, 100, 103, 105, 110

[10] M. Vukov, A. Domahidi, H. J. Ferreau, M. Morari, and M. Diehl, "Auto-generated Algorithms for Nonlinear Model Predicitive Control on Long and on Short Horizons," in *Proceedings of the 52nd Conference on Decision and Control (CDC)*, 2013. pages 8, 9, 83

[11] H. Ferreau, T. Kraus, M. Vukov, W. Saeys, and M. Diehl, "High-speed moving horizon estimation based on automatic code generation," in *Proceedings of the 51th IEEE Conference on Decision and Control (CDC 2012)*, 2012. pages 8, 9, 100

[12] J. V. Frasch, M. Vukov, H. Ferreau, and M. Diehl, "A new Quadratic Programming Strategy for Efficient Sparsity Exploitation in SQP-based Nonlinear MPC and MHE," in *Proceedings of the 19th IFAC World Congress*, 2013. pages 9

[13] J. A. E. Andersson, J. V. Frasch, M. Vukov, and M. Diehl, "A Condensing Algorithm for Nonlinear MPC with a Quadratic Runtime in Horizon Length," *Automatica*, 2013. Submitted. pages 9

[14] F. Debrouwere, M. Vukov, R. Quirynen, M. Diehl, and J. Swevers, "Experimental Validation of Combined Nonlinear Optimal Control and Estimation of an Overhead Crane," in *Proceedings of the 19th World Congress of the International Federation of Automatic Control*, 2014. pages 10, 103, 106, 110, 121

[15] K. Geebelen, M. Vukov, A. Wagner, H. Ahmad, M. Zanon, S. Gros, D. Vandepitte, J. Swevers, and M. Diehl, "An experimental test setup for advanced estimation and control of an airborne wind energy systems," in *Airborne Wind Energy* (U. Ahrens, M. Diehl, and R. Schmehl, eds.), Springer, 2013. pages 11, 104, 129, 130, 132, 143, 147, 155

[16] M. Vukov, S. Gros, G. Horn, G. Frison, K. Geebelen, J. B. Jørgensen, J. Swevers, and M. Diehl, "Real-time Nonlinear MPC and MHE for a Large-scale Mechatronic Application," 2015. (submitted to Control Engineering Practice). pages 11

[17] M. Morari and J. Lee, "Model predictive control: past, present and future," *Computers and Chemical Engineering*, vol. 23, pp. 667–682, 1999. pages 13

[18] F. Allgöwer and A. Zheng, *Nonlinear Predictive Control*, vol. 26 of *Progress in Systems Theory*. Basel Boston Berlin: Birkhäuser, 2000. pages 13

[19] J. Rawlings and D. Mayne, *Model Predictive Control: Theory and Design*. Nob Hill, 2009. pages 13, 14, 33

[20] S. Qin and T. Badgwell, "A survey of industrial model predictive control technology," *Control Engineering Practice*, vol. 11, pp. 733–764, 2003. pages 13

[21] J. Rawlings, "Optimal dynamic operation of chemical processes: Assessment of the last 20 years and current research opportunities." Talk at K.U. Leuven, Belgium, June 2009. pages 13

[22] R. Lopez-Negrete, F. J. D'Amato, L. T. Biegler, and A. Kumar, "Fast nonlinear model predictive control: Formulation and industrial process applications," *Computers & Chemical Engineering*, vol. 51, pp. 55–64, 2013. pages 13

[23] A. Bemporad, M. Morari, V. Dua, and E. Pistikopoulos, "The explicit linear quadratic regulator for constrained systems," *Automatica*, vol. 38, pp. 3–20, 2002. pages 13

[24] H. J. Ferreau, H. G. Bock, and M. Diehl, "An online active set strategy to overcome the limitations of explicit MPC," *International Journal of Robust and Nonlinear Control*, vol. 18, no. 8, pp. 816–830, 2008. pages 13, 41, 43, 75, 101

[25] J. Mattingley, Y. Wang, and S. Boyd, "Code generation for receding horizon control," in *Proceedings of the IEEE International Symposium on Computer-Aided Control System Design*, (Yokohama, Japan), 2010. pages 13

[26] A. Wills, D. Bates, A. Fleming, B. Ninness, and S. Moheimani, "Application of MPC to an active structure using sampling rates up to 25kHz," 44th IEEE Conference on Decision and Control and European Control Conference ECC'05, Seville, 2005. pages 13

[27] L. Van den Broeck, J. Swevers, and M. Diehl, "Experimental validation of Time Optimal MPC on a linear drive system," in *Proceedings of the 11th International Workshop on Advanced Motion Control*, (Nagaoka, Japan), pp. 355–360, 2010. pages 13

[28] L. Biegler and J. Rawlings, "Optimization approaches to nonlinear model predictive control," in *Proc. 4th International Conference on Chemical Process Control - CPC IV* (W. Ray and Y. Arkun, eds.), pp. 543–571, AIChE, CACHE, 1991. pages 13

[29] L. T. Biegler, "A survey on sensitivity-based nonlinear model predictive control," in *10th IFAC International Symposium on Dynamics and Control of Process Systems, Mumbai, India*, The International Federation of Automatic Control, 2013. pages 14

[30] M. Alamir, "Fast nmpc: A reality-steered paradigm: Key properties of fast nmpc algorithms," in *Control Conference (ECC), 2014 European*, pp. 2472–2477, IEEE, 2014. pages 14

[31] K. Muske, J. Rawlings, and J. Lee, "Receding Horizon Recursive State Estimation," in *American Control Conference*, (San Francisco), pp. 900–904, 1993. pages 14

[32] D. Robertson, *Development and Statistical Interpretation of Tools for Nonlinear Estimation*. PhD thesis, Auburn University, 1996. pages 14, 31, 32, 33

[33] C. V. Rao, J. B. Rawlings, and D. Q. Mayne, "Constrained state estimation for nonlinear discrete-time systems: Stability and moving horizon approximations," *IEEE Transactions on Automatic Control*, vol. 48, no. 2, pp. 246–258, 2003. pages 14, 31, 71, 151

[34] E. Hairer, S. Nørsett, and G. Wanner, *Solving Ordinary Differential Equations II – Stiff and Differential-Algebraic Problems*. Springer Series in Computational Mathematics, Berlin: Springer, 2nd ed., 1996. pages 15, 37

[35] M. Diehl, *Real-Time Optimization for Large Scale Nonlinear Processes*. PhD thesis, Universität Heidelberg, 2001. pages 17, 25, 126

[36] M. Diehl, I. Uslu, R. Findeisen, S. Schwarzkopf, F. Allgöwer, H. Bock, T. Bürner, E. Gilles, A. Kienle, J. Schlöder, and E. Stein, "Real-Time Optimization for Large Scale Processes: Nonlinear Model Predictive Control of a High Purity Distillation Column," in *Online Optimization of Large Scale Systems: State of the Art* (M. Grötschel, S. O. Krumke, and J. Rambau, eds.), pp. 363–384, Springer, 2001. download at: http://www.zib.de/dfg-echtzeit/Publikationen/Preprints/Preprint-01-16.html. pages 17

[37] D. Leineweber, I. Bauer, A. Schäfer, H. Bock, and J. Schlöder, "An Efficient Multiple Shooting Based Reduced SQP Strategy for Large-Scale Dynamic Process Optimization (Parts I and II)," *Computers and Chemical Engineering*, vol. 27, pp. 157–174, 2003. pages 17

[38] R. Bellman, *Dynamic programming*. Princeton University Press, 1957. pages 17

[39] L. Pontryagin, V. Boltyanski, R. Gamkrelidze, and E. Miscenko, *The Mathematical Theory of Optimal Processes*. Chichester: Wiley, 1962. pages 17

[40] L. T. Biegler, *Nonlinear Programming*. MOS-SIAM Series on Optimization, SIAM, 2010. pages 17, 22

[41] J. Betts, *Practical Methods for Optimal Control and Estimation Using Nonlinear Programming*. SIAM, 2nd ed., 2010. pages 17

[42] H. Bock and K. Plitt, "A multiple shooting algorithm for direct solution of optimal control problems," in *Proceedings 9th IFAC World Congress Budapest*, pp. 242–247, Pergamon Press, 1984. pages 18, 25, 41, 45, 47

[43] D. Leineweber, *Efficient reduced SQP methods for the optimization of chemical processes described by large sparse DAE models*, vol. 613 of *Fortschritt-Berichte VDI Reihe 3, Verfahrenstechnik*. Düsseldorf: VDI Verlag, 1999. pages 19, 23, 47, 52

[44] R. Quirynen, "Automatic code generation of Implicit Runge-Kutta integrators with continuous output for fast embedded optimization," Master's thesis, KU Leuven, 2012. pages 20, 32, 37, 100

[45] W. Karush, "Minima of Functions of Several Variables with Inequalities as Side Conditions," Master's thesis, Department of Mathematics, University of Chicago, 1939. pages 21

[46] H. Kuhn and A. Tucker, "Nonlinear programming," in *Proceedings of the Second Berkeley Symposium on Mathematical Statistics and Probability* (J. Neyman, ed.), (Berkeley), University of California Press, 1951. pages 21

[47] J. Nocedal and S. Wright, *Numerical Optimization*. Springer Series in Operations Research and Financial Engineering, Springer, 2 ed., 2006. pages 22, 37, 41, 42, 43, 44, 71

[48] H. Bock, "Recent advances in parameter identification techniques for ODE," in *Numerical Treatment of Inverse Problems in Differential and Integral Equations* (P. Deuflhard and E. Hairer, eds.), Boston: Birkhäuser, 1983. pages 23, 25

[49] S. Wright, *Primal-Dual Interior-Point Methods*. Philadelphia: SIAM Publications, 1997. pages 23, 70, 71

[50] A. Wächter and L. Biegler, "On the Implementation of a Primal-Dual Interior Point Filter Line Search Algorithm for Large-Scale Nonlinear Programming," *Mathematical Programming*, vol. 106, no. 1, pp. 25–57, 2006. pages 23

[51] T. Ohtsuka, "A Continuation/GMRES Method for Fast Computation of Nonlinear Receding Horizon Control," *Automatica*, vol. 40, no. 4, pp. 563–574, 2004. pages 25

[52] V. M. Zavala and L. Biegler, "The Advanced Step NMPC Controller: Optimality, Stability and Robustness," *Automatica*, vol. 45, pp. 86–93, 2009. pages 25

[53] M. Diehl, R. Findeisen, and F. Allgöwer, "A Stabilizing Real-time Implementation of Nonlinear Model Predictive Control," in *Real-Time and Online PDE-Constrained Optimization* (L. Biegler, O. Ghattas, M. Heinkenschloss, D. Keyes, and B. van Bloemen Waanders, eds.), pp. 23–52, SIAM, 2007. pages 25

[54] F. Allgöwer, T. Badgwell, J. Qin, J. Rawlings, and S. Wright, "Nonlinear Predictive Control and Moving Horizon Estimation – An Introductory Overview," in *Advances in Control, Highlights of ECC'99* (P. M. Frank, ed.), pp. 391–449, Springer, 1999. pages 25

[55] M. Diehl, *Real-Time Optimization for Large Scale Nonlinear Processes*, vol. 920 of *Fortschr.-Ber. VDI Reihe 8, Meß-, Steuerungs- und Regelungstechnik*. Düsseldorf: VDI Verlag, 2002. Download also at: http://www.ub.uni-heidelberg.de/archiv/1659/. pages 26

[56] J. V. Frasch, *Parallel Algorithms for Optimization of Dynamic Systems in Real-Time*. PhD thesis, KU Leuven and University of Magdeburg, 2014. pages 26, 36

[57] R. Kalman, "A New Approach to Linear Filtering and Prediction Problems," *Transactions of the ASME–Journal of Basic Engineering*, vol. 82, pp. 35–45, 1960. pages 29

[58] A. Gelb, *Applied Optimal Estimation*. MIT Press, 1974. pages 29

[59] V. Becerra, P. Roberts, and G. Griffiths, "Applying the extended Kalman Filter to systems described by nonlinear differential-algebraic equations," *Control Engineering Practice*, vol. 9, pp. 267–281, 2001. pages 29

[60] P. Park and T. Kailath, "New Square-Root Algorithms for Kalman Filtering," *IEEE Transactions on Automatic Control*, vol. 40, no. 5, pp. 895–899, 1995. pages 29

[61] M. Verhaegen and P. Van Dooren, "Numerical aspects of different Kalman filter implementations," *IEEE Transactions on Automatic Control*, vol. 31, no. 10, pp. 907–917, 1986. pages 29

[62] S. Julier, J. Uhlmann, and H. Durrant-Whyte, "A new method for the nonlinear transformation of means and covariances in filters and estimators," *IEEE Transactions on Automatic Control*, vol. 45, pp. 477–482, 2000. pages 29

[63] A. Doucet, N. D. Freitas, and N. Gordon, *Sequential Monte Carlo methods in practice*. Springer, 2001. pages 29

[64] F. Daum, "Nonlinear Filters: Beyond the Kalman Filter," *Aerospace and Electronic Systems Magazine, IEEE*, vol. 20, no. 8, pp. 57–69, 2005. pages 29

[65] J. Rawlings and B. Bakshi, "Particle filtering and moving horizon estimation," *Computers and Chemical Engineering*, vol. 30, pp. 1529–1541, 2006. pages 29

[66] C. Rao and J. Rawlings, "Nonlinear Moving Horizon State Estimation," in *Nonlinear Predictive Control* (F. Allgöwer and A. Zheng, eds.), (Basel Boston Berlin), pp. 45–69, Birkhäuser, 2000. pages 30

[67] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge: University Press, 2004. pages 32

[68] B. Nicholson, R. López-Negrete, and L. T. Biegler, "On-line state estimation of nonlinear dynamic systems with gross errors," *Computers & Chemical Engineering*, 2013. pages 32

[69] R. López-Negrete, *Nonlinear Programming Sensitivity Based Methods for Constrained State Estimation*. PhD thesis, Carnegie Mellon University, 2011. pages 32

[70] R. Quirynen, S. Gros, and M. Diehl, "Fast auto generated ACADO integrators and application to MHE with multi-rate measurements," in *Proceedings of the European Control Conference*, 2013. pages 32, 100

[71] R. López-Negrete and L. T. Biegler, "A moving horizon estimator for processes with multi-rate measurements: A nonlinear programming sensitivity approach," *Journal of Process Control*, vol. 22, no. 4, pp. 677–688, 2012. pages 32

[72] H. Michalska and D. Q. Mayne, "Moving horizon observers and observer-based control," *IEEE Transactions on Automatic Control*, vol. 40, no. 6, pp. 995–1006, 1995. pages 32

[73] P. Kühl, M. Diehl, T. Kraus, J. P. Schlöder, and H. G. Bock, "A real-time algorithm for moving horizon state and parameter estimation," *Computers & Chemical Engineering*, vol. 35, no. 1, pp. 71–83, 2011. pages 33, 100, 104

[74] T. Kraus, P. Kühl, L. Wirsching, H. G. Bock, and M. Diehl, "A Moving Horizon State Estimation algorithm applied to the Tennessee Eastman Benchmark Process," in *Proc. of IEEE Robotics and Automation Society conference on Multisensor Fusion and Integration for Intelligent Systems*, 2006. pages 33

[75] A. Wynn, M. Vukov, and M. Diehl, "Convergence guarantees for moving horizon estimation based on the real-time iteration scheme," 2014. pages 33

[76] J. V. Frasch, S. Sager, and M. Diehl, "A Parallel Quadratic Programming Method for Dynamic Optimization Problems," *Mathematical Programming Computations*, 2013. pages 36, 72, 74, 75, 102

[77] A. Griewank, *Evaluating Derivatives, Principles and Techniques of Algorithmic Differentiation*. No. 19 in Frontiers in Appl. Math., Philadelphia: SIAM, 2000. pages 37

[78] J. Andersson, *A General-Purpose Software Framework for Dynamic Optimization*. PhD thesis, Arenberg Doctoral School, KU Leuven, Department of Electrical Engineering (ESAT/SCD) and Optimization in Engineering Center, Kasteelpark Arenberg 10, 3001-Heverlee, Belgium, October 2013. pages 37, 47, 59, 61

[79] E. Hairer, S. Nørsett, and G. Wanner, *Solving Ordinary Differential Equations I*. Springer Series in Computational Mathematics, Berlin: Springer, 2nd ed., 1993. pages 37

[80] R. Quirynen, M. Vukov, and M. Diehl, "Auto Generation of Implicit Integrators for Embedded NMPC with Microsecond Sampling Times," in *Proceedings of the 4th IFAC Nonlinear Model Predictive Control Conference* (M. Lazar and F. Allgöwer, eds.), 2012. pages 37, 100

[81] C. Kirches, H. G. Bock, J. P. Schlöder, and S. Sager, "A factorization with update procedures for a KKT matrix arising in direct optimal control," *Mathematical Programming Computation*, vol. 3, no. 4, pp. 319–348, 2011. pages 41

[82] H. Ferreau, A. Kozma, and M. Diehl, "A parallel active-set strategy to solve sparse parametric quadratic programs arising in MPC," in *Proceedings of the 4th IFAC Nonlinear Model Predictive Control Conference, Noordwijkerhout, The Netherlands*, 2012. pages 41, 72, 74, 75

[83] R. Fletcher, *Practical Methods of Optimization*. Chichester: Wiley, 2nd ed., 1987. pages 42, 43

[84] H. Ferreau, "An Online Active Set Strategy for Fast Solution of Parametric Quadratic Programs with Applications to Predictive Engine Control," Master's thesis, University of Heidelberg, 2006. pages 43, 44, 45

[85] M. Herceg, C. N. Jones, and M. Morari, "Dominant speed factors of active set methods for fast MPC," *Optimal Control Applications and Methods*, 2014. pages 43

[86] P. Gill, G. Golub, W. Murray, and M. A. Saunders, "Methods for Modifying Matrix Factorizations," *Mathematics of Computation*, vol. 28, no. 126, pp. 505–535, 1974. pages 44

[87] C. Kirches, H. Bock, J. Schlöder, and S. Sager, "Block structured quadratic programming for the direct multiple shooting method for optimal control," *Optimization Methods and Software*, vol. 26, pp. 239–257, April 2010. pages 44, 47

[88] P. Gill, N. Gould, W. Murray, M. Saunders, and M. Wright, "A Weighted Gram-Schmidt Method for Convex Quadratic Programming," *Mathematical Programming*, vol. 30, pp. 176–195, 1984. pages 44

[89] C. Kirches, L. Wirsching, S. Sager, and H. Bock, "Efficient numerics for nonlinear model predictive control," in *Recent Advances in Optimization and its Applications in Engineering* (M. Diehl, F. F. Glineur, and E. J. W. Michiels, eds.), pp. 339–357, Springer, 2010. pages 44

[90] "`qpOASES`." `http://www.qpOASES.org`, 2007–2015. [Online; accessed 3-April-2015]. pages 45, 76, 99, 101

[91] A. Wills, "QPC homepage." http://sigpromu.org/quadprog/, 2006–2011. pages 45

[92] C. Kirches, L. Wirsching, H. Bock, and J. Schlöder, "Efficient Direct Multiple Shooting for Nonlinear Model Predictive Control on Long Horizons," *Journal of Process Control*, vol. 22, no. 3, pp. 540–550, 2012. pages 47

[93] D. Axehill and M. Morari, "An alternative use of the riccati recursion for efficient optimization," *Systems & Control Letters*, vol. 61, no. 1, pp. 37–40, 2012. pages 47

[94] G. Frison, "Numerical methods for model predictive control," Master's thesis, Department of Informatics and Mathematical Modelling, Technical University of Denmark, Kgs. Lyngby, Denmark, 2012. pages 47

[95] G. Frison and J. Jørgensen, "A Fast Condensing Method for Solution of Linear-Quadratic Control Problems," in *Proceedings of the 52nd IEEE Conference on Decision and Control*, 2013. pages 47, 66

[96] G. Frison and J. Jørgensen, "Efficient implementation of the riccati recursion for solving linear-quadratic control problems," in *IEEE Multi-conference on Systems and Control (MSC)*, pp. 1117–1122, IEEE, 2013. pages 56, 72, 102

[97] Y. Wang and S. Boyd, "Fast model predictive control using online optimization," *IEEE Transactions on Control Systems Technology*, vol. 18, no. 2, pp. 267–278, 2010. pages 70, 72

[98] S. Mehrotra, "On the Implementation of a Primal-Dual Interior Point Method," *SIAM Journal on Optimization*, vol. 2, no. 4, pp. 575–601, 1992. pages 70

[99] A. Domahidi, A. Zgraggen, M. Zeilinger, M. Morari, and C. Jones, "Efficient Interior Point Methods for Multistage Problems Arising in Receding Horizon Control," in *IEEE Conference on Decision and Control (CDC)*, (Maui, HI, USA), pp. 668 – 674, Dec. 2012. pages 70, 72, 101

[100] G. Frison, H. B. Sørensen, B. Dammann, and J. Jørgensen, "High-performance small-scale solvers for linear model predictive control," in *IEEE European Control Conference (ECC)*, pp. 128–133, IEEE, 2014. pages 70, 72, 102

[101] G. Frison, L. Sokoler, and J. Jørgensen, "A family of high-performance solvers for linear model predictive control," in *IFAC World Congress*, pp. 3074–3079, IFAC, 2014. pages 72

[102] G. Frison, D. Kufualor, L. Imsland, and J. Jørgensen, "Efficient implementation of solvers for linear model predictive control on embedded devices," in *IEEE Multi-conference on Systems and Control (MSC)*, pp. 1954–1959, IEEE, 2014. pages 72

[103] A. Shahzad, E. Kerrigan, and G. Constantinides, "A warm-start interior-point method for predictive control," tech. rep., Imperial College London, 2010. pages 72

[104] A. Shahzad and P. Goulart, "A new hot-start interior-point method for model predictive control," in *Proceedings of the IFAC World Congress*, 2011. pages 72

[105] J. V. Frasch, M. Vukov, H. Ferreau, and M. Diehl, "A new quadratic programming strategy for efficient sparsity exploitation in SQP- based nonlinear MPC and MHE," in *Proceedings of the 19th IFAC World Congress*, 2014. pages 72, 83, 102

[106] W. Li and J. Swetits, "A new algorithm for solving strictly convex quadratic programs," *SIAM Journal of Optimization*, vol. 7, no. 3, pp. 595–619, 1997. pages 72, 74

[107] Y.-H. Dai and R. Fletcher, "New algorithms for singly linearly constrained quadratic programs subject to low and upper bounds," *Mathematical Programming*, vol. 106, no. 3, pp. 403–421, 2006. pages 72

[108] "FORCES - fast optimization for real-time control on embedded systems." `http://http://forces.ethz.ch/`, 2012. pages 76, 99, 101

[109] "qpDUNES - a dual newton strategy for convex quadratic programming." `http://http://github.com/jfrasch/qpDUNES`, 2013. pages 76, 99, 102

[110] "HPMPC - library for high-performance implementation of solvers for mpc." `http://github.com/giaf/hpmpc`, 2014. pages 76, 99, 102

[111] L. Wirsching, H. G. Bock, and M. Diehl, "Fast NMPC of a chain of masses connected by springs," in *Proceedings of the IEEE International Conference on Control Applications, Munich*, pp. 591–596, 2006. pages 77, 78

[112] K. Graichen, M. Treuer, and M. Zeitz, "Swing-up of the double pendulum on a cart by feedforward and feedback control with experimental validation," *Automatica*, vol. 43, no. 1, pp. 63 – 71, 2007. pages 79

[113] T. Glueck, A. Eder, and A. Kugi, "Swing-up control of a triple pendulum on a cart with experimental validation," *Automatica*, vol. 49, no. 3, pp. 801 – 808, 2013. pages 79

[114] E. Dolan and J. Moré, "Benchmarking optimization software with performance profiles," *Mathematical Programming.*, vol. 91, pp. 201–213, 2002. pages 81

[115] D. Mayne, J. Rawlings, C. Rao, and P. Scokaert, "Constrained model predictive control: stability and optimality," *Automatica*, vol. 26, no. 6, pp. 789–814, 2000. pages 84

[116] L. Grüne, "NMPC Without Terminal Constraints," in *Proceedings of the IFAC Conference on Nonlinear Model Predictive Control 2012*, 2012. pages 84

[117] E. Anderson, Z. B. C., Bischof, S. Blackford, J. Demmel, J. Dongarra, J. D. Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen, *LAPACK Users' Guide*. Philadelphia, PA: SIAM, third ed., 1999. pages 92

[118] R. Quirynen, M. Vukov, M. Zanon, and M. Diehl, "Autogenerating Microsecond Solvers for Nonlinear MPC: a Tutorial Using ACADO Integrators," *Optimal Control Applications and Methods*, 2014. pages 98

[119] R. Quirynen, M. Vukov, and M. Diehl, *Contributions in Mathematical and Computational Sciences*, ch. Multiple Shooting in a Microsecond. Springer, 2014. Submitted. pages 98

[120] B. Houska, H. Ferreau, and M. Diehl, "ACADO Toolkit – An Open Source Framework for Automatic Control and Dynamic Optimization," *Optimal Control Applications and Methods*, vol. 32, no. 3, pp. 298–312, 2011. pages 98

[121] J. Mattingley and S. Boyd, *Convex Optimization in Signal Processing and Communications*, ch. Automatic Code Generation for Real-Time Convex Optimization. Cambridge University Press, 2009. pages 98

[122] H. Seguchi and T. Ohtsuka, "Nonlinear Receding Horizon Control of an Underactuated Hovercraft," *International Journal of Robust and Nonlinear Control*, vol. 13, no. 3–4, pp. 381–398, 2003. pages 98

[123] "RAWESOME – the airborne wind energy simulation, optimization and modeling environment." `https://github.com/ghorn/rawesome`. [Online; accessed 10-October-2013]. pages 98, 103, 142, 154

[124] H. Ferreau, B. Houska, M. Vukov, and R. Quirynen, "ACADO Code Generation User's Manual." http://www.acadotoolkit.org, 2011. pages 99

[125] R. Quirynen, B. Houska, M. Vallerio, D. Telen, F. Logist, J. Van Impe, and M. Diehl, "Symmetric Algorithmic Differentiation Based Exact Hessian SQP Method and Software for Economic MPC," in *Conference on Decision and Control*, 2014. pages 100, 175

[126] R. Quirynen, S. Gros, and M. Diehl, "Efficient NMPC for nonlinear models with linear subsystems," in *Proceedings of the 52nd IEEE Conference on Decision and Control*, 2013. pages 100

[127] J. Mattingley and S. Boyd, "CVXGEN webage." http://cvxgen.com, 2008–2011. pages 101

[128] C. Rao, J. Rawlings, and J. Lee, "Constrained linear state estimation - a moving horizon approach," *Automatica*, vol. 37, no. 2, pp. 1619–1628, 2001. pages 102

[129] T. Kraus, H. J. Ferreau, E. Kayacan, H. Ramon, J. De Baerdemaeker, M. Diehl, and W. Saeys, "Moving horizon estimation and nonlinear model predictive control for autonomous agricultural vehicles," *Computers and Electronics in Agriculture*, vol. 98, pp. 25–33, October 2013. pages 103, 104

[130] D. Frick, A. Domahidi, M. Vukov, S. Mariéthoz, M. Diehl, and M. Morari, "Moving Horizon Estimation for Induction Motors," in *IEEE International Symposium on Sensorless Controls for Electrical Drives*, (Milwaukee, WI, USA), pp. 1–6, Sept. 2012. pages 103

[131] E. Kayacan, E. Kayacan, H. Ramon, and W. Saeys, "Distributed nonlinear model predictive control of an autonomous tractor-trailer system," *Mechatronics*, vol. 24, no. 8, pp. 926 – 933, 2014. pages 104

[132] T. Faulwasser, J. Matschek, P. Zometa, and R. Findeisen, "Predictive path-following control: Concept and implementation for an industrial robot," in *Control Applications (CCA), 2013 IEEE International Conference on*, pp. 128–133, IEEE, 2013. pages 104

[133] J. Huber, C. Gruber, and M. Hofbaur, "Online trajectory optimization for nonlinear systems by the concept of a model control loop - applied to the reaction wheel pendulum," in *Control Applications (CCA), 2013 IEEE International Conference on*, pp. 935–940, Aug 2013. pages 104

[134] 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 53rd Conference on Decision and Control (CDC)*, 2014. pages 104

[135] D. Schindele and H. Aschemann, "Fast nonlinear MPC for an overhead travelling crane," in *Proceedings of the IFAC World Congress*, 2011. pages 105

[136] L. Van den Broeck, M. Diehl, and J. Swevers, "Experimental validation of Time Optimal MPC on a flexible motion system," in *Proceedings of the 2011 American Control Conference*, (San Francisco, USA), pp. 4749–4754, 2011. pages 105

[137] B. Käpernick and K. Graichen, "PLC implementation of a nonlinear model predictive controller," in *Proceedings of the 19th IFAC World Congress*, (Cape Town, South Africa), pp. 1892–1897, 2014. pages 105

[138] Orocos, "Orocos - Open Robot Control Software project.," September 2011. pages 107

[139] M. Fliess, J. Lévine, and P. Rouchon, "Flatness and defect of nonlinear systems: Introductory theory and examples," *International Journal of Control*, vol. 61, pp. 1327–1361, 1995. pages 107

[140] J. Schoukens and R. Pintelon, *Identification of Linear Systems: A Practical Guide to Accurate Modeling*. Pergamon Press, 1991. pages 108

[141] M. Loyd, "Crosswind Kite Power," *Journal of Energy*, vol. 4, pp. 106–111, July 1980. pages 127

[142] A. Ilzhoefer, B. Houska, and M. Diehl, "Nonlinear MPC of kites under varying wind conditions for a new class of large scale wind power generators," *International Journal of Robust and Nonlinear Control*, vol. 17, no. 17, pp. 1590–1599, 2007. pages 127

[143] B. Houska and M. Diehl, "Optimal Control for Power Generating Kites," in *Proc. 9th European Control Conference*, (Kos, Greece,), pp. 3560–3567, 2007. (CD-ROM). pages 127

[144] M. Diehl, "Airborne wind energy: Basic concepts and physical foundations," in *Airborne Wind Energy*, Springer, 2013. pages 127

[145] Ampyx Power, "Technology Concept." `http://www.ampyxpower.com/Overview.html`, 2015. pages 128

[146] M. Diehl and B. Houska, "Windenergienutzung mit schnell fliegenden Flugdrachen: eine Herausforderung für die Optimierung und Regelung - Wind Power via Fast Flying Kites: a Challenge for Optimization and Control," *at-automatisierungstechnik*, vol. 57, no. 10, pp. 525–533, 2009. pages 128

[147] M. Clinckemaillie, "An experimental set-up for energy generation using balanced kites," Master's thesis, KU Leuven, 2012. pages 128

[148] K. Geebelen and J. Gillis, "Modelling and control of rotational start-up phase of tethered aeroplanes for wind energy harvesting," Master's thesis, KU Leuven, June 2010. pages 130, 132

[149] D. Cosaert, K. Elst, K. Geebelen, M. Diehl, J. Swevers, and D. Vandepitte, "Design of a winch for modeling and control of the tethered flight of a model airplane," Master's thesis, KU Leuven, 2011. pages 130, 133

[150] K. Geebelen, H. Ahmad, M. Vukov, S. Gros, J. Swevers, and M. Diehl, "An experimental test set-up for launch/recovery of an Airborne Wind Energy (AWE) system," in *Proceedings of the 2012 American Control Conference*, 2012. pages 132

[151] Ebox, "E-box project page." http://cstwiki.wtb.tue.nl/index.php?title=E-box, 2013. pages 132

[152] "RT-PREEMPT Patch." `https://rt.wiki.kernel.org/index.php/RT_PREEMPT_HOWTO`. [Online; accessed 29-12-2014]. pages 135

[153] H. Bruyninckx, "Open robot control software: the OROCOS project," in *Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on*, vol. 3, pp. 2523–2528, IEEE, 2001. pages 136

[154] "CasADi." `http://casadi.org`, 2013. [Online; accessed 10-October-2013]. pages 142, 148

[155] S. Gros and M. Diehl, *Airborne Wind Energy*, ch. Modeling of Airborne Wind Energy Systems in Natural Coordinates. Springer, 2013. pages 143, 144, 145

[156] S. Gros, M. Zanon, M. Vukov, and M. Diehl, "Nonlinear MPC and MHE for Mechanical Multi-Body Systems with Application to Fast Tethered Airplanes," in *Proceedings of the 4th IFAC Nonlinear Model Predictive Control Conference, Noordwijkerhout, The Netherlands*, 2012. pages 143, 144

[157] M. Zanon, S. Gros, and M. Diehl, "Rotational Start-up of Tethered Airplanes Based on Nonlinear MPC and MHE," in *Proceedings of the European Control Conference*, 2013. pages 143, 145, 154, 171

[158] M. Zanon, S. Gros, and M. Diehl, "Model Predictive Control of Rigid-Airfoil Airborne Wind Energy Systems," in *Airborne Wind Energy* (U. Ahrens, M. Diehl, and R. Schmehl, eds.), Springer, 2013. pages 143, 145, 146, 154, 171

[159] K. Geebelen, M. Vukov, S. Gros, J. Swevers, and M. Diehl, "Comparison of moving horizon estimators with kinematic and dynamic models based on tethered flight experiments," 2015. (submitted to IEEE Transactions on Control Systems Technology). pages 143, 146, 150, 151

[160] J. Gillis, G. Horn, and M. Diehl, "Joint design of stochastically safe setpoints and controllers for nonlinear constrained systems by means of optimization," in *Proceedings of the 19th IFAC World Congress*, August 2014. pages 147, 148

[161] J. Sternberg, S. Gros, B. Houska, and M. Diehl, "Approximate Robust Optimal Control of Periodic Systems with Invariants and High-Index Differential Algebraic Systems," in *In Proceedings of the 7th IFAC Symposium on Robust Control Design*, pp. 678–683, 2012. pages 147

[162] A. Wächter and L. Biegler, "IPOPT - an Interior Point OPTimizer." https://projects.coin-or.org/Ipopt, 2009. pages 156

[163] S. Gros, H. Ahmad, K. Geebelen, and M. Diehl, "In-flight estimation of the aerodynamic roll damping and trim angle for a tethered aircraft based on multiple-shooting," in *System Identification Conference*, 2012. pages 170

[164] S. Gros, M. Zanon, and M. Diehl, "Control of Airborne Wind Energy Systems Based on Nonlinear Model Predictive Control & Moving Horizon Estimation," in *European Control Conference*, 2013. pages 171

[165] E. D. Dolan, J. J. Moré, and T. S. Munson, "Optimality measures for performance profiles," *SIAM Journal on Optimization*, vol. 16, pp. 891–909, Mar. 2006. pages 175

[166] J. Jerez, P. Goulart, S. Richter, G. Constantinides, E. Kerrigan, and M. Morari, "Embedded Predictive Control on an FPGA using the Fast Gradient Method," in *European Control Conference*, (Zurich, Switzerland), pp. 3614 – 3620, July 2013. pages 175

[167] H. Peyrl, A. Zanarini, T. Besselmann, J. Liu, and M.-A. Boéchat, "Parallel implementations of the fast gradient method for high-speed MPC," *Control Engineering Practice*, vol. 33, pp. 22–34, 2014. pages 175

[168] J. Jerez, P. Goulart, S. Richter, G. Constantinides, E. Kerrigan, and M. Morari, "Embedded Online Optimization for Model Predictive Control at Megahertz Rates," *IEEE Transactions on Automatic Control*, Apr. 2014. pages 175

[169] G. Knagge, A. Wills, A. Mills, and B. Ninness, "ASIC and FPGA Implementation Strategies for Model Predictive Control," in *European Control Conference (ECC)*, aug 2009. pages 176

[170] A. Joos and W. Fichter, "Parallel Implementation of Constrained Nonlinear Model Predictive Controller for an FPGA-based Onboard Flight Computer," in *Advances in Aerospace Guidance, Navigation and Control* (F. Holzapfel and S. Theil, eds.), pp. 273–286, Springer Berlin Heidelberg, 2011. pages 176

[171] B. Käpernick, S. Sub, E. Schubert, and K. Graichen, "A synthesis strategy for nonlinear model predictive controller on FPGA," in *Control (CONTROL), 2014 UKACC International Conference on*, pp. 662–667, July 2014. pages 176

Curriculum Vitae

Milan Vukov was born in Vršac, Serbia, in 1983. He received the M.S. degree in electrical engineering from the University of Belgrade, Serbia, in 2008. After graduation, he joined the Laboratory for Digital Control of Electrical Drives at the same faculty. His work in the laboratory was related to control of AC drives, robotics and fast real-time monitoring systems for mechatronics applications. In early 2010, Milan moved to the Robotics Laboratory at the Mihailo Pupin Institute. Since December 2010, Milan has been a PhD student at the Department of Electrical Engineering (ESAT) at KU Leuven, under the supervision of Professor Moritz Diehl. His research is focused on fast implementations of the algorithms for nonlinear model predictive control (NMPC) and moving horizon estimation (MHE), and their applications to fast mechatronics systems.

List of Publications

Journal papers

1. R. Quirynen, M. Vukov, M. Zanon, and M. Diehl, "Autogenerating Microsecond Solvers for Nonlinear MPC: a Tutorial Using ACADO Integrators," *Optimal Control Applications and Methods*, 2014.
2. A. Wynn, M. Vukov, and M. Diehl, "Convergence guarantees for moving horizon estimation based on the real-time iteration scheme," *IEEE Transactions on Automatic Control*, 2014.
3. K. Geebelen, M. Vukov, S. Gros, J. Swevers, and M. Diehl, "Comparison of moving horizon estimators with kinematic and dynamic models based on tethered flight experiments," 2015. (submitted to IEEE Transactions on Control Systems Technology).
4. M. Vukov, S. Gros, G. Horn, G. Frison, K. Geebelen, J. B. Jørgensen, J. Swevers, and M. Diehl, "Real-time Nonlinear MPC and MHE for a Large-scale Mechatronic Application," 2015. (submitted to Control Engineering Practice).

Book chapters

1. K. Geebelen, M. Vukov, A. Wagner, H. Ahmad, M. Zanon, S. Gros, D. Vandepitte, J. Swevers, and M. Diehl, "An experimental test setup for advanced estimation and control of an airborne wind energy systems," in *Airborne Wind Energy* (U. Ahrens, M. Diehl, and R. Schmehl, eds.), Springer, 2013.
2. R. Quirynen, M. Vukov, and M. Diehl, *Contributions in Mathematical and Computational Sciences*, ch. Multiple Shooting in a Microsecond. Springer, 2014. Submitted.
3. M. Zanon, J. V. Frasch, M. Vukov, S. Sager, and M. Diehl, "Model Predictive Control of Autonomous Vehicles," in *Proceedings of the Workshop on Optimization and Optimal Control of Automotive Systems*, 2014.

Conference papers

1. H. Ferreau, T. Kraus, M. Vukov, W. Saeys, and M. Diehl, "High-speed moving horizon estimation based on automatic code generation," in *Proceedings of the 51th IEEE Conference on Decision and Control (CDC 2012)*, 2012.
2. D. Frick, A. Domahidi, M. Vukov, S. Mariéthoz, M. Diehl, and M. Morari, "Moving Horizon Estimation for Induction Motors," in *IEEE International Symposium on Sensorless Controls for Electrical Drives*, (Milwaukee, WI, USA), pp. 1–6, Sept. 2012.
3. K. Geebelen, H. Ahmad, M. Vukov, S. Gros, J. Swevers, and M. Diehl, "An experimental test set-up for launch/recovery of an Airborne Wind Energy (AWE) system," in *Proceedings of the 2012 American Control Conference*, 2012.
4. S. Gros, M. Zanon, M. Vukov, and M. Diehl, "Nonlinear MPC and MHE for Mechanical Multi-Body Systems with Application to Fast Tethered Airplanes," in *Proceedings of the 4th IFAC Nonlinear Model Predictive Control Conference, Noordwijkerhout, The Netherlands*, 2012.
5. R. Quirynen, M. Vukov, and M. Diehl, "Auto Generation of Implicit Integrators for Embedded NMPC with Microsecond Sampling Times," in *Proceedings of the 4th IFAC Nonlinear Model Predictive Control Conference* (M. Lazar and F. Allgöwer, eds.), 2012.
6. M. Vukov, W. V. Loock, B. Houska, H. Ferreau, J. Swevers, and M. Diehl, "Experimental Validation of Nonlinear MPC on an Overhead Crane using Automatic Code Generation," in *The 2012 American Control Conference, Montreal, Canada.*, 2012.
7. S. Gros, M. Vukov, and M. Diehl, "A Real-time MHE and NMPC Scheme for the Control of Multi-Mega Watts Wind Turbines," in *Conference on Decision and Control*, 2013.
8. M. Vukov, A. Domahidi, H. J. Ferreau, M. Morari, and M. Diehl, "Auto-generated Algorithms for Nonlinear Model Predicitive Control on Long and on Short Horizons," in *Proceedings of the 52nd Conference on Decision and Control (CDC)*, 2013.
9. F. Debrouwere, M. Vukov, R. Quirynen, M. Diehl, and J. Swevers, "Experimental Validation of Combined Nonlinear Optimal Control and Estimation of an Overhead Crane," in *Proceedings of the 19th World Congress of the International Federation of Automatic Control*, 2014.
10. J. V. Frasch, M. Vukov, H. Ferreau, and M. Diehl, "A new quadratic programming strategy for efficient sparsity exploitation in SQP- based nonlinear MPC and MHE," in *Proceedings of the 19th IFAC World Congress*, 2014.

FACULTY OF ENGINEERING SCIENCE
DEPARTMENT OF ELECTRICAL ENGINEERING (ESAT)
STADIUS CENTER FOR DYNAMICAL SYSTEMS, SIGNAL PROCESSING AND DATA ANALYTICS
Kasteelpark Arenberg 10
B-3001 Heverlee

Persist.