FACULTY OF

SCIENCE AND ENGINEERING

Master's Thesis in applied Mathematics

Nonlinear Model Predictive Control and Estimation applied to Selective Catalytic Reduction

Author:

Oscar Aaltonen, 41173

Supervisors:

Mikael Kurula

Jari Böling

2022

Abstract

Nonlinear Model Predictive Control (NMPC) is an advanced optimization-based control method for both linear and nonlinear dynamical systems. In this thesis, a NMPC software is developed in Matlab to control a Selective Catalytic Reduction (SCR) process, which is a process to reduce nitrogen oxide emissions from diesel and gas engines using ammonia or a urea solution. The SCR model that is used in this thesis is modeled as a state space model consisting of three nonlinear ordinary differential equations. A simplified nonlinear version of this model is used in the NMPC as a prediction model. State estimation is used to estimate missing measurements from the SCR process; a Moving Horizon Estimator (MHE) is implemented in Matlab for this purpose. Since no theory is available for this kind of nonlinear output feedback MPC, the results of the control and estimation are presented through simulation. The simulations show that the SCR can be controlled with only a few measurements using MHE and NMPC. A major advantage with NMPC is that the ammonia slip can also be controlled. Some mathematical results of NMPC combined with nonlinear MHE are discussed and MHE convergence for a linear detectable plant is proved, slightly improving the corresponding results in the research literature.

Acknowledgements

This master's thesis is a part of the project Clean Propulsion Technologies, which is funded by Business Finland. The project aims to develop cleaner solutions for marine and off-road transport in collaboration with industry and research organizations. The study was conducted in Turku at the Faculty of Science and Engineering at Åbo Akademi University with Wärtsilä Finland Oy as collaborating partner.

I would like to thank all the people that have supported me during this work. At Åbo Akademi University, I would like to thank my supervisors Mikael Kurula and Jari Böling for their continuous help and support throughout the whole project. At Wärtsilä, I would like to thank Jan Torrkulla and Markus Hanhisuanto for joining our weekly meetings and especially Jan for sharing his knowledge and expertise on the SCR process.

Additionally, I would like to thank my partner Annika, and also my family and friends for their support during my studies.

Contents

1 Introduction **4**

2 Linear system theory **7**

2.1 Continuous-time systems 7

2.2 Discretization using sampling and Zero-Order Hold 8

2.3 Setpoint tracking 11

3 State estimation **19**

3.1 Moving horizon estimation 20

4 Nonlinear Model Predictive Control **27**

4.1 Historical background 27

4.2 Constraints 28

4.3 Introduction to NMPC 29

4.3.1 The NMPC algorithm 31

5 Nonlinear tracking, NMHE and NMPC **35**

5.1 Nonlinear setpoint tracking 35

5.1.1 Nonlinear setpoint tracking combined with NMPC 36

5.2 Nonlinear MHE 38

5.2.1 NMHE compared to other estimation methods 39

5.2.2 Nonlinear MHE combined with NMPC 40

5.3 NMPC, Nonlinear setpoint tracking and nonlinear MHE 41

5.4 Conclusion 41

6 Selective catalytic reduction **43**
6.1 The full SCR model 43
6.2 A simplified SCR model 45

7 Simulation results **48**
7.1 Prediction model validation 48
7.2 NMPC simulation setup 49
7.3 Validation of the NMHE 50
7.4 NMPC combined with setpoint tracking and NMHE 54
7.5 PI-control 56
7.5.1 NMPC compared to PI-control 58
7.6 Conclusion 62

8 Swedish summary **64**

A Matlab code **68**
A.1 Prediction model validation 68
A.2 PI-controller 75
A.3 Main file for the SCR control simulator 79
A.4 Nonlinear MHE 87
A.5 Nonlinear setpoint tracking 91
A.6 NMPC algorithm 93

Chapter 1

Introduction

The main aim of this work is to design a model predictive controller that controls a Selective Catalytic Reduction (SCR) process efficiently. The SCR model that is used in this thesis is developed by Milver Colmenares in his master's thesis [11]. The model is nonlinear, and therefore, a nonlinear model predictive controller is proposed. Model predictive control (MPC) is an advanced control process based on optimization. The goal is to reduce emissions from the SCR process and keep them beneath the emission regulations. MPC can handle constraints on states and on the control signal, so the emission restrictions are easily implemented as constraints for the controller.

In MPC the system behaviour is predicted, using a prediction model of the real system or process. The future behaviour is then optimized, and the first optimal control decision is used as input for the next time step.

Often, linearization is used for a nonlinear model, since linear MPC is computationally less demanding than NMPC. However, the SCR model is highly nonlinear, and linearization would be inaccurate. As hardware is becoming faster, NMPC is gaining more popularity. Computers are now much faster than decades ago, which could make nonlinear model predictive control cost-effective.

The book *Nonlinear Model Predictive Control* [1], written by Grüne and Pannek is theoretical and mathematical, so it suits well as the primary source for the NMPC theory. State estimation is required for the controller, since some states are not measured, and all states are important for the MPC controller design. In this thesis, moving horizon estimation is proposed, since it is suited for both nonlinear models and linear models. The primary source for this chapter is the book [2] by Rawlings, Mayne and Diehl, in particular Sections 1.4 and 4.3. Theory about nonlinear MHE is still difficult to find, since this area has not been extensively researched. The material in Chapter 4 of [2] is, as the authors state, up to date with the current literature and includes the latest research in the area.

Emission regulations keep becoming stricter and this puts pressure on the industry, since new solutions for regulating emissions more efficiently must be found quickly. One major contaminant is nitrogen oxides which are produced in the combustion process of diesel and gas engines. A way to reduce emissions in diesel and gas-powered engines is by a process called Selective Catalytic Reduction.

The SCR is a process that reduces the nitrogen oxides in the exhaust gas to nitrogen and water using a reducing agent. Ammonia or a urea-water solution is usually used as the reducing agent. At first, this process was used in stationary power plants and in industrial equipment, but now the process is widely used in other applications, since the SCR process has developed tremendously [12]. Almost every new diesel-powered car relies on this process to reduce emissions to match the Euro 6 standards. The AdBlue liquid that is added to a diesel-powered car consists of a urea-water mixture which is used for the SCR process [19].

The results presented in this thesis are generated by simulation. A simulator is developed in Matlab to test the control and estimation of the SCR process. The NMPC software developed in this thesis is an extensively developed version of the NMPC routine by Grüne and Pannek, their NMPC algorithm can be found on [18]. Figure 1.1 describes how the simulator is organized.

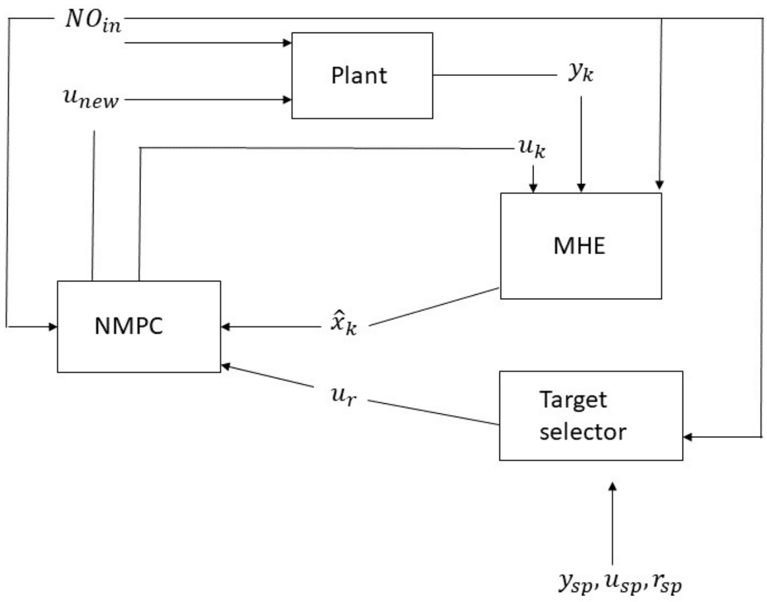

Figure 1.1: Block diagram of the simulator

The plant describes the process that is controlled and measurements from the plant y_k, control signals u_k and disturbance NO_{in} are passed to the estimator MHE, which estimate the plant state for the NMPC regulator. The NMPC optimizes the future behaviour of the system, starting from the estimated state $\hat{x}_k$. The NMPC determines the optimal control signal u_{new}, which is then applied to the plant and the process restarts. The Target selector determines a reference value u_r, which is used in the NMPC cost function.

In every block, a model is used. A detailed model is used to describe the plant, and a simplified model is used by the estimator as an estimation model and by the NMPC regulator as a prediction model. The target selector also use the simplified model to determine the reference for the control signal. These models are presented and discussed in detail in Chapter 6.

The contribution of this work is the simulator developed for the SCR control. The modified NMPC software and the implementation of the MHE in Matlab is the major progress in this thesis. The theory for the implementation is based on [1] and [2]. Mathematical results of linear estimator convergence and uniqueness of the linear setpoint tracking problem is also proved.

Chapter 2

Linear system theory

Some theory of linear systems is presented together with sampling and discretization. The NMPC in this thesis uses discrete-time models, which is why discretization is presented. Linear system theory is presented to help the reader understand the concepts in the nonlinear case. This chapter is based on Sections 1.2 and 1.5 from [2].

2.1 Continuous-time systems

Usually, models describing real-life applications or processes are modeled in continuous time, as differential equations. Numerical simulation is usually faster with discrete-time models, which is why sampling of the continuous-time systems is desired. Sampling means that the states are determined on sample points. In this thesis, the sampling intervals is chosen to be equidistant. Linear systems are presented beneath together with some definitions.

Definition 2.1. A *continuous time-invariant linear state-space system* is defined as

$$\begin{cases} \dfrac{dx(t)}{dt} &= Ax(t) + Bu(t) \\ y(t) &= Cx(t) + Du(t), \quad x(0) = x_0 \quad \text{given,} \end{cases} \tag{2.1}$$

where $A \in \mathbb{R}^{n\times n}$ is the *transition matrix*, $B \in \mathbb{R}^{n\times m}$ is the *input matrix*, $C \in \mathbb{R}^{p\times n}$ is the *output matrix* and $D \in \mathbb{R}^{p\times m}$ is the *feedthrough matrix*. The time is denoted by $t \in \mathbb{R}$, the *state* at time t is denoted by $x(t) \in \mathbb{R}^n$, the input is denoted by $u(t) \in \mathbb{R}^m$ and the *output* is denoted by $y(t) \in \mathbb{R}^p$.

Linear systems can easily be solved explicitly, given an initial condition x_0 and some input signal u. The system 2.1 is solved by multiplication with the matrix exponential function and the calculations are

$$\begin{aligned} e^{-At}\left(\frac{dx(t)}{dt} - Ax(t)\right) = e^{-At}Bu(t) \iff \quad & \frac{d}{dt}\left(e^{-At}x(t)\right) = e^{-At}Bu(t) \\ \iff \quad & e^{-At}x(t) - x_0 = \int_0^t e^{-As}Bu(s)ds \\ \iff \quad & x(t) = e^{At}x_0 + \int_0^t e^{A(t-s)}Bu(s)ds. \end{aligned}$$

2.2 Discretization using sampling and Zero-Order Hold

For the control and estimation in this thesis, discrete-time systems are used. A linear discrete-time system can be derived from the continuous-time system using sampling. The idea with sampling is to evaluate the system at the sample points $t_d := kT$ where $T > 0$ is the sampling time. The linear discrete-time system is of the form

$$\begin{aligned} x_s(k+1) &= A_s x_s(k) + B_s u_s(k) \\ y_s(k) &= C_s x_s(k) + D_s u_s(k), \quad x_s(0) = x_0 \quad \text{given} \end{aligned} \tag{2.2}$$

where $x_s(k) := x(t_d)$. The matrices A_s, B_s, C_s and D_s can be derived exactly for linear systems and these calculations are demonstrated beneath. For a sample $t_d = kT$ the state is defined as

$$x(t_d) = x(kT) = e^{AkT}x_0 + \int_0^{kT} e^{A(kT-s)}Bu_s(s)ds. \tag{2.3}$$

For the next sample $k+1$ the state is defined as

$$\begin{aligned} x_s(k+1) &= e^{A(k+1)T}x_0 + \int_0^{(k+1)T} e^{A((k+1)T-s)}Bu_s(s)ds \\ &= e^{A(k+1)T}x_0 + \int_0^{kT} e^{A((k+1)T-s)}Bu_s(s)ds + \int_{kT}^{kT+T} e^{A((k+1)T-s)}Bu_s(s)ds \\ &= e^{AT}\left(e^{AkT}x_0 + \int_0^{kT} e^{A(kT-s)}Bu_s(s)ds\right) + \int_{kT}^{kT+T} e^{A((k+1)T-s)}Bu_s(s)ds. \end{aligned}$$

The parenthesis is exactly (2.3). For the second integral we assume that u is constant between each sample time and then we use a variable substitution $v = (k+1)T - s$

$$\begin{aligned} x(k+1) &= e^{AT}x(k) + \int_{(k+1)T-kT}^{(k+1)T-kT-T} -e^{Av}dv \cdot Bu_s(k) \\ &= e^{AT}x(k) + \int_T^0 -e^{Av}dv \cdot Bu_s(k) \\ &= e^{AT}x(k) + \int_0^T e^{Av}dv \cdot Bu_s(k). \end{aligned}$$

For y we define $y(k) := y(t_d)$ and obtain

$$y(k) = C(k) + Du(k)$$

From the calculations above we obtain

$$A_s = e^{AT}, \quad B_s = \int_0^T e^{Av}dv \cdot B, \quad C_s = C, \quad D_s = D,$$

for $k = 0,1,2,\ldots$

The lower index s from the matrices A,B,C and D is removed in further calculations and the notation $x(k)$ is used for discrete-time systems. As can be seen it is possible to derive explicit formulas for discrete-time models that match exactly the continuous model at the sample instances, for linear sampled data systems. This does not apply for nonlinear systems, but if the sampling period is chosen properly and the system considered is suitable for sampling, the discrete-time system should resemble the continuous-time system. In this thesis, discretization of the nonlinear continuous-time plant is done with Zero Order Hold (ZOH).

It is possible to sample the nonlinear system with an ordinary differential equation solver when the sampling points are chosen in advance. In the software that Grüne and Pannek developed [18], sampling of a nonlinear system is done using the Matlab function ODE45 between the sample points.

More information about discrete-time systems, sampling and discretization can be found in [1, Chapter 2], where nonlinear discrete-time systems are presented together with results of stability. In [2, Section 1.2], linear discrete-time systems are presented. Observability is required for some results in the following sections, and it is next defined for linear systems.

Definition 2.2 (Observability)**.** A discrete-time linear system (A,C) with zero input is *observable* if for every $x(0)$ there exists $N > 0$, such that the measurements $y(0),y(1),\ldots,y(N-1)$ determine the initial state $x(0)$ uniquely.

A weaker condition than observability is detectability, which is a property of a system that describes state-to-output interaction [6].

Definition 2.3 (Detectability)**.** A linear discrete-time system with zero input

$$\begin{aligned} x(k+1) &= Ax(k) \\ y(k) &= Cx(k) \end{aligned}$$

is said to be *detectable* if there exists a matrix L such that $A + LC$ is stable, *i.e*

$$\begin{aligned} &x(k+1) = (A + LC)x(k) \implies x(k) \to 0 \\ &\text{when} \quad k \to \infty. \end{aligned}$$

Sometimes a system is not observable and hence, detectability is important. Detectability is used when proving estimator convergence in Chapter 3. The steady state of a system is also an important concept in optimization, and it is presented beneath.

Definition 2.4 (Steady state)**.** If $x_s = f(x_s,u_s)$, then we say that x_s is a *steady state* for $u(k) := u_s$.

This means that state of the system is held constant and not changing over time.

2.3 Setpoint tracking

In control problems, it is usually desired to steer the system output to some specific setpoint; the control problem is known as *setpoint tracking*. In most regulation problems, the target is to bring the state of the system to the origin, and this is referred to as stabilization. Setpoint tracking can be reduced to stabilization using a change of coordinates [2], as will be demonstrated next. Setpoint tracking is demonstrated for linear systems, since it is possible to obtain exact and unique solutions. For nonlinear plants, it is in general challenging to obtain an exact unique solution and hence the theory of nonlinear setpoint tracking is excluded from this work. This section is based on [2, Section 1.5].

Consider the linear unconstrained discrete-time system (2.2) and denote the steady state as (x_s,u_s). Another requirement for the steady state is that it satisfies $Cx_s = y_{sp}$, where y_{sp} is the setpoint. From (2.2) and using the Definition of the steady state 2.4, one obtains that the steady state should satisfy

$$\begin{bmatrix} I - A & -B \\ C & 0 \end{bmatrix} \begin{bmatrix} x_s \\ u_s \end{bmatrix} = \begin{bmatrix} 0 \\ y_{sp} \end{bmatrix}. \tag{2.4}$$

If (2.4) has a solution, then deviation variables can be defined as

$$\begin{aligned} \tilde{x} &= x(k) - x_s \\ \tilde{u} &= u(k) - u_s, \end{aligned}$$

that satisfy

$$\begin{aligned} \tilde{x}(k+1) &= x(k+1) - x_s = Ax(k) + Bu(k) - (Ax_s + Bu_s) \\ \tilde{x}(k+1) &= A\tilde{x}(k) + B\tilde{u}(k). \end{aligned}$$

Now we can find $\tilde{u}(k)$ that takes $\tilde{x}(k)$ to zero, which is equivalent to $x(k) \to x_s$, so that at steady state, $Cx(k) = Cx_s = y_{sp}$, which is the setpoint.

The simplest assumption, which guarantees the solvability of (2.4) for all y_{sp}, is that the rows of the large matrix should be linearly independent. This requires at least as many inputs as outputs of the system. In many applications, however, this is not the case. It is possible to have more measured outputs than inputs that can be manipulated. For these cases, a matrix H is introduced and a new variable is denoted $r = Hy$, which is the selection of linear combinations of the measured output. In this case, setpoints are assigned to r and the setpoints are denoted r_{sp}.

The theory presented above is for unconstrained systems, but for constrained systems we simply put constraints on the states x_s and on the control signal u_s. The steady state should also satisfy the setpoint r_{sp}. Now an optimization problem can be defined for the setpoint tracking problem.

Problem 2.5. *The optimization problem is defined as*

$$\min_{x_s,u_s} \frac{1}{2}\left(|u_s - u_{sp}|^2_{R_s} + |Cx_s - y_{sp}|^2_{Q_s}\right), \tag{2.5}$$

where $R_s > 0$ *and* $Q_s \geq 0$,
subject to

$$\begin{bmatrix} I-A & -B \\ HC & 0 \end{bmatrix} \begin{bmatrix} x_s \\ u_s \end{bmatrix} = \begin{bmatrix} 0 \\ r_{sp}. \end{bmatrix} \tag{2.6}$$

$$Eu_s \leq e \tag{2.7}$$

$$Fx_s \leq f. \tag{2.8}$$

The idea with Problem 2.5 is to have setpoints r_{sp} that always must be satisfied through (2.6). The objective function (2.5) penalizes the control variable u_s from a soft setpoint u_{sp} for the control variable. The other term then penalizes the states from a soft setpoint y_{sp}, which holds the other states as close as possible to this setpoint.

A result can be proved for Problem 2.5 that guarantees a solution and uniqueness when $R_s > 0$ and $Q_s \geq 0$ hold. But first, a convex set and strictly convex function are defined, these definitions are based on [7] and then a convex optimization problem is defined based on [5].

Definition 2.6. A subset C of $\mathbb{R}^n$ is said to be a *convex set* if $\lambda x_1 + (1-\lambda)x_2 \in C$ for all $x_1 \in C$, $x_2 \in C$ and $0 < \lambda < 1$.

Definition 2.7. A real-valued function f on a convex set C is said to be a *strictly convex function* on C if

$$f(\lambda x_1 + (1-\lambda)x_2) < \lambda f(x_1) + (1-\lambda)f(x_2)$$

holds for $0 < \lambda < 1$, $x_1 \in C$, $x_2 \in C$ and $x_1 \neq x_2$.

Definition 2.8. Consider the optimization problem

$$\begin{aligned} &\min f(x) \\ &\text{s.t } x \in \mathbb{X}. \end{aligned}$$

The optimization problem is *strictly convex* if $f : \mathbb{R}^n \to \mathbb{R}$ is *strictly convex* on $\mathbb{X}$ and $\mathbb{X}$ is a convex set.

A nice feature with strict convex optimization problems is that they have at most one optimal solution [5].

Theorem 2.9. *Consider Problem 2.5 with p controlled variables and m manipulated variables u. For all setpoints r_{sp}, the steady-state solution (x_s,u_s) exists if the inequality constraints (2.7) and (2.8) are absent and*

$$\operatorname{rank} \begin{bmatrix} I-A & -B \\ HC & 0 \end{bmatrix} = n + p. \tag{2.9}$$

Any solution is unique if

$$\operatorname{rank} \begin{bmatrix} I-A \\ HC \end{bmatrix} = n. \tag{2.10}$$

Proof. By assumption

$$\begin{bmatrix} I-A & -B \\ HC & 0 \end{bmatrix} \in \mathbb{R}^{(n+p)\times(n+m)}$$

has $n+p$ independent rows, which means that the matrix is surjective. This means that

$$\forall r_{sp} \in \mathbb{R}^{n+p} \quad \exists (x,u) \in \mathbb{R}^n \times \mathbb{R}^m : \quad \begin{bmatrix} I-A & -B \\ HC & 0 \end{bmatrix} \begin{bmatrix} x_s \\ u_s \end{bmatrix} = \begin{bmatrix} 0 \\ r_{sp} \end{bmatrix},$$

i.e. there exist a solution for every $r_{sp} \in \mathbb{R}^p \times \mathbb{R}^m$. Since the matrix in (2.10) has full column rank, there exists a left inverse $V = \begin{bmatrix} V_1 & V_2 \end{bmatrix}$ with

$$V \begin{bmatrix} I - A \\ HC \end{bmatrix} = I \in \mathbb{R}^{n \times n}.$$

Now we have

$$\begin{bmatrix} I - A \\ HC \end{bmatrix} x_s = \begin{bmatrix} Bu_s \\ r_{sp} \end{bmatrix} \quad \Longrightarrow \quad x_s = V \begin{bmatrix} Bu_s \\ r_{sp} \end{bmatrix},$$

which means that Bu_s and r_{sp} determine x_s uniquely. Now we need to prove that u_s is uniquely determined by the optimization problem. The cost can be rewritten as

$$\begin{aligned} f(u_s) :=& \frac{1}{2} \left(|u_s - u_{sp}|^2_{R_s} + |Cx_s - y_{sp}|^2_{Q_s} \right) \\ =& \frac{1}{2}((u_s - u_{sp})^T R_s (u_s - u_{sp}) + \\ & \frac{1}{2} \left(CV \begin{bmatrix} Bu_s \\ r_{sp} \end{bmatrix} - y_{sp} \right)^T Q_s \left(CV \begin{bmatrix} Bu_s \\ r_{sp} \end{bmatrix} - y_{sp} \right) \\ =& \frac{1}{2}(u_s^T R_s u_s - 2u_{sp}^T R_s u_s + u_{sp}^T R_s u_{sp} \\ & + \frac{1}{2}(CV_1 Bu_s + CV_2 r_{sp} - y_{sp})^T Q_s (CV_1 Bu_s + CV_2 r_{sp} - y_{sp}) \\ =& \frac{1}{2}(u_s^T (R_s + B^T V_1^T C^T Q_s C V_1 B) u_s - 2\omega^T u_s + 2k), \end{aligned}$$

where ω is a vector and k is a scalar, neither of which depends on u_s. Denoting

$$G := \frac{1}{2}(R_s + B^T V_1^T C^T Q_s C V_1 B), \tag{2.11}$$

which is positive definite, since $Q_s \geq 0$ and R_s is positive definite, we obtain

$$f(u_s) = u_s^T G u_s - \omega^T u_s + k. \tag{2.12}$$

Using Definition 2.7 of a strictly convex function, we obtain for u, v and $0 < \lambda < 1$

that

$$
\begin{aligned}
& f(\lambda u+(1-\lambda)v) < \lambda f(u)+(1-\lambda)f(v) \\
\iff \quad & 0 < \lambda f(u)+(1-\lambda)f(v) - f(\lambda u+(1-\lambda)v) \\
& = \lambda u^T G u - \lambda\omega^T u + \lambda k + (1-\lambda)(v^T G v - \omega^T v + k) \\
& \quad - (\lambda u+(1-\lambda)v)^T G(\lambda u+(1-\lambda)v) + \omega^T(\lambda u+(1-\lambda)v) - k \\
& = \lambda u^T G u - (\lambda u+(1-\lambda)v)^T G(\lambda u+(1-\lambda)v) \\
& \quad + (1-\lambda)(v^T G v - \omega^T v) + \omega^T(1-\lambda)v \\
& = \lambda u^T G u + (1-\lambda)v^T G v - \lambda u^T G\lambda u - \lambda u^T G(1-\lambda)v \\
& \quad - (1-\lambda)v^T G\lambda u - (1-\lambda)v^T G(1-\lambda)v \\
& = (1-\lambda)(\lambda u^T G u + v^T G v - 2v^T G\lambda u - (1-\lambda)v^T G v) \\
& = (1-\lambda)(\lambda u^T G u - 2\lambda v^T G u + \lambda v^T G v) \\
& = \lambda(1-\lambda)(u-v)^T G(u-v),
\end{aligned}
$$

Now we have

$$0 < \lambda(1-\lambda)(u-v)^T G(u-v),$$

which holds, since G is positive definite, $u \neq v$ and $0 < \lambda < 1$. Now we still need to prove that the set that we are optimizing over is convex. The set that has to be convex is

$$C := \{u_s \in \mathbb{R}^m | \exists x_s \in \mathbb{R}^n : (2.6),(2.7) \text{ and } (2.8) \text{ hold}\}.$$

Now let $u,v \in C$ and let $x,z \in \mathbb{R}^n$ be such that (2.6), (2.7) and (2.8) hold with (u_s,x_s) replaced by (u,x) or (v,z). According to Definition 2.6 of a convex set, we need to verify that for all $\lambda \in (0,1)$, there exists a $w \in \mathbb{R}^n$ such that $(\lambda u+(1-\lambda)v,w) \in C$. For the inequality constraint (2.7) we have

$$E(\lambda u+(1-\lambda)v) = \lambda Eu + (1-\lambda)Ev \leq \lambda e + (1-\lambda)e = e,$$

since $u,v \in C$ and $\lambda \in (0,1)$. For (2.6) we have, with $w := \lambda x + (1-\lambda)z$, that

$$\begin{aligned}
&\begin{bmatrix} I-A & -B \\ HC & 0 \end{bmatrix} \begin{bmatrix} w \\ \lambda u + (1-\lambda)v \end{bmatrix} \\
&= \lambda \begin{bmatrix} I-A & -B \\ HC & 0 \end{bmatrix} \begin{bmatrix} x \\ u \end{bmatrix} + (1-\lambda) \begin{bmatrix} I-A & -B \\ HC & 0 \end{bmatrix} \begin{bmatrix} z \\ v \end{bmatrix} \\
&= \lambda \begin{bmatrix} 0 \\ r_{sp} \end{bmatrix} + (1-\lambda) \begin{bmatrix} 0 \\ r_{sp} \end{bmatrix} = \begin{bmatrix} 0 \\ r_{sp} \end{bmatrix},
\end{aligned}$$

which holds, since $u,v \in C$, $x,z \in \mathbb{R}^n$ and $\lambda \in (0,1)$. For (2.8) we have

$$Fw = F(\lambda x + (1-\lambda)z) = \lambda F x + F z - \lambda F z \leq \lambda f + f - \lambda f = f,$$

which holds, since $x,z \in \mathbb{R}^n$ and $\lambda \in (0,1)$. This proves that the set is convex. Hence, we have a strictly convex optimization problem, since the objective function is strictly convex, and the set is convex. This means that u_s is determined uniquely by r_{sp}, and then x_s is also determined uniquely by r_{sp}. □

Two examples of how to solve the tracking Problem 2.5 are presented next.

Example 2.10. Consider the two-input, two-output system

$$\begin{aligned} x(k+1) &= Ax(k) + Bx(k) \\ y(k) &= Cx(k), \end{aligned}$$

and A,B,C presented beneath, with the output setpoint $y_{sp} = r_{sp} = \begin{bmatrix} 1 & -1 \end{bmatrix}^T$ and input setpoint $u_{sp} = \begin{bmatrix} 0 & 0 \end{bmatrix}^T$. Calculate x_s and u_s. Is it possible to reach the setpoint y_{sp} for $Q_s = I$, $R_s = I$,

$$A = \begin{bmatrix} 0.5 & 0 & 0 & 0 \\ 0 & 0.6 & 0 & 0 \\ 0 & 0 & 0.5 & 0 \\ 0 & 0 & 0 & 0.6 \end{bmatrix}, \quad B = \begin{bmatrix} 0.5 & 0 \\ 0 & 0.4 \\ 0.25 & 0 \\ 0 & 0.6 \end{bmatrix}, \quad C = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix}, \quad H = I?$$

Equation (2.6), is equivalent to

$$\left(\begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&1&0\\0&0&0&1\end{bmatrix}-\begin{bmatrix}0.5&0&0&0\\0&0.6&0&0\\0&0&0.5&0\\0&0&0&0.6\end{bmatrix}\right)x_s=\begin{bmatrix}0.5&0\\0&0.4\\0.25&0\\0&0.6\end{bmatrix}u_s \tag{2.13}$$

$$\text{and}\quad\begin{bmatrix}1&1&0&0\\0&0&1&1\end{bmatrix}x_s=\begin{bmatrix}1\\-1\end{bmatrix}. \tag{2.14}$$

Multiplying (2.13) with $(I-A)^{-1}$ we obtain

$$x_s=\begin{bmatrix}1&0\\0&1\\0.5&0\\0&1.5\end{bmatrix}u_s.$$

Inserting this in (2.14), we obtain

$$\begin{bmatrix}1&1\\0.5&1.5\end{bmatrix}u_s=\begin{bmatrix}1\\-1\end{bmatrix}\quad\Longleftrightarrow\quad u_s=\begin{bmatrix}2.5\\-1.5\end{bmatrix}.$$

Now, since u_s is determined uniquely, x_s can be determined and y_{sp} is satisfied exactly with $x_s=\begin{bmatrix}2.5\\-1.5\\1.25\\-2.25\end{bmatrix}$ and $u_s=\begin{bmatrix}2.5\\-1.5\end{bmatrix}$. Only one admissible pair (x_s, u_s) exists, since all conditions hold in Theorem 2.9. The conditions (2.9) and (2.10) hold, since both matrices have full rank.

Example 2.11. Consider the same system as in Example 2.10, but now only the first output has a setpoint $y_{sp_1}=1$. What is the solution to the tracking problem, if it exists, for $R_s=I$, $y_{sp}=r_{sp}$, $Q_s=0$ and $u_{sp}=0$?

Now the H matrix is required, since we only have one output setpoint, denote

$H = \begin{bmatrix} 1 & 0 \end{bmatrix}$. The result from Example 2.10 Equation (2.13) can be used to obtain an expression for u_s. Now we only have one setpoint for the output and thus u_s cannot be determined uniquely from the system (2.6).

$$\begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0.5 & 0 \\ 0 & 1.5 \end{bmatrix} u_s = 1 \quad \Longleftrightarrow \quad \begin{bmatrix} 1 & 1 \end{bmatrix} u_s = 1$$

$$\Longleftrightarrow \quad u_{s,1} + u_{s,2} = 1 \quad \Longleftrightarrow \quad u_{s,2} = 1 - u_{s,1}.$$

Now optimization is required, since it is unknown which combination of u_s is the most optimal. Using (2.5), we obtain

$$\begin{aligned} \min_{x_s, u_{s,1}} \frac{1}{2} \left(\begin{bmatrix} u_{s,1} & 1 - u_{s,1} \end{bmatrix} R_s \begin{bmatrix} u_{s,1} \\ 1 - u_{s,1} \end{bmatrix} \right) &= \min_{x_s, u_s} \frac{1}{2} \left(u_{s,1}^2 + (1 - u_{s,1})(1 - u_{s,1}) \right) \\ &= \min_{x_s, u_s} \frac{1}{2} \left(2u_1^2 - 2u_{s,1} + 1 \right) \\ &= \min_{x_s, u_s} \frac{1}{2} \left(2 \left(u_{s,1} - \frac{1}{2} \right)^2 + \frac{1}{2} \right), \end{aligned}$$

which is minimized at $u_{s,1} = u_{s,2} = \frac{1}{2}$. This means that the most optimal way to reach the setpoint y_{sp} is with $u_s = \begin{bmatrix} \frac{1}{2} & \frac{1}{2}, \end{bmatrix}^T$, which determines $x_s = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} & \frac{1}{4} & \frac{3}{4} \end{bmatrix}^T$.

Chapter 3

State estimation

State estimation is a method to estimate states of the plant that cannot be measured. Measurements are usually expensive, and some states are physically impossible to measure. Concentrations are in particular difficult to measure, but they are in fact important for the NMPC in this thesis, since the constraints are on the concentrations. There are many different ways to do state estimations the most famous being the Kalman filter. The Kalman filter is used for linear systems and the extended Kalman filter (EKF) is used for nonlinear systems. The EKF uses linearization of the nonlinear model, which is sometimes usable for plants that are almost linear [2]. The estimation model used in this thesis is highly nonlinear. For nonlinear models [2], proposes other estimation algorithms.

To control the SCR, state estimation is required, since the prediction model relies on the ammonia coverage in the catalyzer. Ammonia coverage cannot be measured and, hence, state estimation is required. In [2], Moving Horizon Estimation (MHE) is proposed, especially combined with MPC and also for nonlinear plants. This is one reason why MHE is considered in this thesis.

The research in nonlinear MHE is still quite thin and complicated; hence, theory for nonlinear MHE remains for further work. The idea of nonlinear MHE is quite similar to linear MHE and hence some theory and results are presented for linear plants. Nonlinear MHE is discussed in Chapter 5 and some simulations are presented in Chapter 7.

3.1 Moving horizon estimation

Moving horizon estimation is useful, for instance, if one wants to apply constraints on the estimates or when using a nonlinear model. Since MHE is an optimization-based estimator, one can use constraints to obtain more accurate estimates. The moving horizon idea is presented in Figure 3.1.

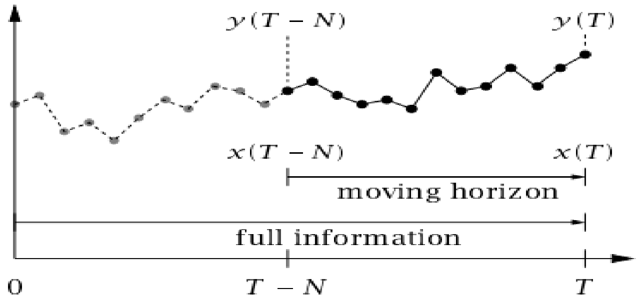

Figure 3.1: The moving horizon estimation problem [2].

The moving horizon idea emerged from full information estimation, *i.e.* estimation using every available measurement. Full information estimation is impractical, since it is computationally intractable in case of a nonlinear system or when considering constraints, which is why the idea of moving horizon estimation arose. For long horizons the optimization problem might become too extensive in full information estimation. In MHE, the idea is to use a fixed amount of measurements $y(T-N), \dots, y(T)$, which makes the MHE problem computationally tractable, see Figure 3.1.

Problem 3.1. *The linear MHE problem can be stated as an optimization problem for $T > N$:*

$$\min_{\hat{X}_N(T)} \hat{V}_T(\hat{X}_N(T)), \tag{3.1}$$

where

$$\hat{X}_N(T) = \begin{bmatrix} \hat{x}(T-N) \\ \hat{x}(T-N+1) \\ \vdots \\ \hat{x}(T) \end{bmatrix} \tag{3.2}$$

and the cost is defined as

$$\begin{aligned} &\hat{V}_T(\hat{X}_N(T)) = \frac{1}{2}\Bigg(|\hat{x}(T-N) - \bar{x}(T-N)|_P^2 + \\ &\sum_{k=T-N}^{T-1} |\hat{x}(k+1) - A\hat{x}(k) - Bu(k)|_Q^2 + \sum_{k=T-N}^{T} |y(k) - C\hat{x}(k)|_R^2 \Bigg). \end{aligned} \tag{3.3}$$

In (3.3), $\hat{x}(k)$ is the estimated state at time k, $y(k)$ the measurement at time k and P, Q and R are weighting matrices. The plant dynamics are approximated with the estimation model and A and B are the matrices from the model. The C matrix gives the estimated outputs, which are compared to the measurements.

The idea with the cost function is to penalize the estimates from the measurements and to penalize the estimates from the plant dynamics. The prior weighting term penalize the estimates from the prior estimation. The prior weighting term is chosen as the second estimate from the previous estimation, *i.e.*

$$\bar{x}(T-N) = \hat{x}_*(T-N+1), \tag{3.4}$$

where $\hat{x}_*$ is the result from the prior estimation. It is possible to choose the weight $P = 0$ and then the problem reduces to zero prior weighting. For $T \leq N$, the MHE problem is usually assumed to be a full information estimation problem, see Figure 3.1 [2].

For nonlinear systems it is difficult to retrieve any explicit results. The nonlinear MHE is still under research and there does not exist many results. Some results about nonlinear MHE (NMHE) stability are presented in [2] in Section 4.3. Results about moving horizon estimator convergence are presented beneath for linear plants. First estimator convergence is proved for a detectable system and then for an observable system. These results are stronger than those proved in [2], since in these results the control variable is considered. In [2], MHE estimator convergence is not proved for detectable plants.

For the next result, the observable canonical form of the system is required. The detectable plant can be written in the observable canonical form

$$\begin{bmatrix} x_1(k+1) \\ x_2(k+1) \end{bmatrix} = \begin{bmatrix} A_{11} & 0 \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} x_1(k) \\ x_2(k) \end{bmatrix} + \begin{bmatrix} B_1 \\ B_2 \end{bmatrix} u(k) \tag{3.5}$$

$$y(k) = \begin{bmatrix} C_1 & 0 \end{bmatrix} \begin{bmatrix} x_1(k) \\ x_2(k) \end{bmatrix}, \tag{3.6}$$

where (A_{11},C_1) is observable and A_{22} is stable due to detectability [6].

Theorem 3.2. *An optimal moving horizon estimator with a perfect estimation model, perfect measurements, prior* (3.4) *weighting* $P = \begin{bmatrix} 0 & 0 \\ 0 & P_{22} \end{bmatrix} \geq 0$, Q *and* R *positive definite, and* $N =$ *dim* $\mathbb{X}$, *is a convergent estimator for a linear* detectable *plant, and the optimal cost is* $\hat{V}_T^0 = 0$.

Proof. Now the goal is to make the cost function (3.3) zero. The sums in the cost function can be rewritten as

$$\begin{aligned} &\frac{1}{2}\left\| \begin{bmatrix} -A & I & 0 & \dots & 0 \\ 0 & -A & I & \dots & 0 \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & \dots & -A & I \end{bmatrix} \hat{X}_N(T) - \begin{bmatrix} B & 0 & \dots & 0 \\ 0 & B & & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & B \end{bmatrix} U_N(T) \right\|_{\tilde{Q}}^2 \\ &+\frac{1}{2}\left\| \begin{bmatrix} C & 0 & \dots & 0 \\ 0 & C & & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & C \end{bmatrix} \hat{X}_N(T) - \begin{bmatrix} y(T-N) \\ y(T-N+1) \\ \vdots \\ y(T) \end{bmatrix} \right\|_{\tilde{R}}^2, \end{aligned} \tag{3.7}$$

where $\hat{X}_N(T)$ and $U_N(T)$ is defined in the same way as (3.2) and $\tilde{Q} := Q\oplus\cdots\oplus Q$, N orthogonal, copies and $\tilde{R} := R\oplus\cdots\oplus R$, $N+1$ orthogonal copies. Since $\tilde{Q} > 0$, the first term in (3.7) is zero if and only if

$$\hat{x}(k+1) = A\hat{x}(k) + Bu(k) \quad \text{for} \quad k = T-N, \dots, T-1, \tag{3.8}$$

which is equivalent to the statement that $\hat{x}$ follows the dynamics of (3.5) exactly on $[T-N,T]$. Now since $\tilde{R} > 0$ and the estimation model is perfect, the second term in (3.7) is equal to zero if and only if

$$\begin{bmatrix} C & 0 & \dots & 0 \\ 0 & C & & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & C \end{bmatrix} \hat{X}_N(T) - \begin{bmatrix} y(T-N) \\ y(T-N+1) \\ \vdots \\ y(T) \end{bmatrix} = 0 \iff$$

$$\begin{bmatrix} C & 0 & \dots & 0 \\ 0 & C & & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & C \end{bmatrix} \hat{X}_N(T) - \begin{bmatrix} C & 0 & \dots & 0 \\ 0 & C & & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & C \end{bmatrix} X_N(T) = 0 \iff$$

$$\begin{bmatrix} C & 0 & \dots & 0 \\ 0 & C & & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & C \end{bmatrix} \begin{bmatrix} I \\ A \\ \vdots \\ A^N \end{bmatrix} (\hat{x}(T-N) - x(T-N)) = 0$$

$$\iff \quad \mathcal{O}_N(\hat{x}(T-N) - x(T-N)) = 0, \tag{3.9}$$

where $\mathcal{O}$ is the observability matrix, which is defined in [2] on page 42. The observability matrix can be calculated for the detectable system

$$\mathcal{O}_N = \begin{bmatrix} C_1 & 0 \\ CA_{11} & 0 \\ CA_{11}^2 & 0 \\ \vdots & \vdots \\ CA_{11}^{N-1} & 0 \end{bmatrix} = \begin{bmatrix} \mathcal{O}_{N,1} & 0 \end{bmatrix},$$

where $\mathcal{O}_{N,1}$ is injective for $N = \dim \mathbb{X}$, since (C_1,A_{11}) is observable. Now applying this to (3.9) we get

$$\begin{bmatrix} \mathcal{O}_{N,1} & 0 \end{bmatrix} (\hat{x}(T-N) - x(T-N)) = \mathcal{O}_{N,1}(\hat{x}_1(T-N) - x_1(T-N)) = 0.$$

The injectivity of $\mathcal{O}_{N,1}$ implies

$$\hat{x}_1(T-N) - x_1(T-N) = 0 \quad \Longleftrightarrow \quad \hat{x}_1(T-N) = x_1(T-N), \tag{3.10}$$

and

$$\hat{x}_1(k) = x_1(k) \quad \text{for} \quad T-N \leq k \leq T, \tag{3.11}$$

since the estimate $\hat{x}$ follows the dynamics of (3.5) exactly on the interval.
Now forcing the prior weighting factor to zero remains:

$$\begin{aligned} |\hat{x}(T-N) - \bar{x}(T-N)|_P^2 &= \left| \begin{bmatrix} \hat{x}_1(T-N) - \bar{x}_1(T-N) \\ \hat{x}_2(T-N) - \bar{x}_2(T-N) \end{bmatrix} \right|_P^2 \\ &= |\hat{x}_2(T-N) - \hat{x}_2(T-N)|_{P_{22}}^2. \end{aligned}$$

The second component $\hat{x}_2$ is still free and we can simply choose

$$\hat{x}_2(T-N) := \bar{x}_2(T-N), \tag{3.12}$$

to make the MHE cost function (3.3) with prior weighting equal to zero. Thus, MHE achieves (3.11), and we use this to prove that the estimation error satisfies

$$e(k+1) = \begin{bmatrix} 0 & 0 \\ 0 & A_{22} \end{bmatrix} e(k); \tag{3.13}$$

then, since A_{22} is stable, $e(k) \to 0$ when $k \to \infty$.
Indeed, the estimation error at time T is

$$e(T) = \hat{x}_*(T) - x(T),$$

where the optimal estimate is defined as $\hat{x}_*$. The optimal estimate and the plant have the dynamics (3.5) and (3.8), and hence

$$e(T) = \hat{x}_*(T) - x(T) = A^N(\hat{x}_*(T-N) - x(T-N)).$$

The estimate is exact for the first component when $N = \dim \mathbb{X}$, $A = \begin{bmatrix} A_{11} & 0 \\ A_{21} & A_{22} \end{bmatrix}$ and $\hat{x}_{*,2}(T-N) = \bar{x}_2(T-N)$ from (3.12). We now obtain

$$\begin{aligned} e(T) &= \begin{bmatrix} A_{11} & 0 \\ A_{21} & A_{22} \end{bmatrix}^N \begin{bmatrix} 0 \\ \bar{x}_2(T-N) - x_2(T-N) \end{bmatrix} \\ &= \begin{bmatrix} 0 & 0 \\ 0 & A_{22}^N \end{bmatrix} \begin{bmatrix} 0 \\ \bar{x}_2(T-N) - x_2(T-N) \end{bmatrix} \\ &= \begin{bmatrix} 0 & 0 \\ 0 & A_{22}^N \end{bmatrix} (\bar{x}(T-N) - x(T-N)). \end{aligned} \tag{3.14}$$

Using (3.4) and the plant dynamics (3.5) we obtain

$$\begin{aligned} e(T) &= \begin{bmatrix} 0 & 0 \\ 0 & A_{22}^N \end{bmatrix} A \left(\begin{bmatrix} x_1(T-N-1) \\ \bar{x}_2(T-N-1) \end{bmatrix} - \begin{bmatrix} x_1(T-N-1) \\ x_2(T-N-1) \end{bmatrix} \right) \\ &= \begin{bmatrix} 0 & 0 \\ 0 & A_{22}^N \end{bmatrix} \begin{bmatrix} 0 & 0 \\ 0 & A_{22} \end{bmatrix} \begin{bmatrix} 0 \\ \bar{x}_2(T-N-1) - x_2(T-N-1) \end{bmatrix} \\ &= \begin{bmatrix} 0 & 0 \\ 0 & A_{22} \end{bmatrix} e(T-1), \end{aligned}$$

which holds since (3.14) holds. Now the estimation error satisfies (3.13) and MHE estimator convergence for a linear detectable plant is obtained. □

For the next result, convergence is proved for an observable system. For this result, the penalty on the prior estimation can be set to zero. The corollary follows from Theorem 3.2.

Corollary 3.3. *A moving horizon estimator with no prior weighting, Q and R positive definite and N sufficiently large is a convergent estimator for a linear* observable *plant*

$$x(k+1) = Ax(k) + Bu(k) \tag{3.15}$$

$$y(k) = Cx(k), \tag{3.16}$$

with exact measurements. The optimal cost is $\hat{V}_T^0 = 0$ *and* $\hat{x}(k) = x(k)$ *for* $T - N \leq k \leq T$*, i.e., the estimates are* exact*, rather than only convergent.*

Proof. The result follows from the proof of Theorem 3.2, for the special case that the observable part of the detectable system is the entire system. □

Chapter 4

Nonlinear Model Predictive Control

Model Predictive Control (MPC) is proposed to solve the infinite horizon optimal control problem, which is defined in this chapter. In this chapter MPC is presented, and the primary sources for this chapter are Chapters 1 and 3 from Grüne and Pannek [1] and Chapter 1 from Rawlings et al. [2]. In the sequel, nonlinear systems are now considered.

4.1 Historical background

Model Predictive Control, also known as receding horizon control, was developed in the late 1970s. The article by Richalet, Rault, Testud and Paponi [8] is one of the first articles that cover model predictive control, and they achieved successful results in industry at the time. They managed to control different processes more accurately and lower limits of different outputs, which resulted in economic advantages. Another early article about MPC is by Cutler and Ramaker, who wrote about Dynamic Matrix Control (DMC) [9]. Since then, MPC has gained increasingly more popularity in industry. The strength of MPC is that it deals with constrained control problems, which arise frequently in industry.

Computers have developed fast during the last decade. Model predictive control benefits from this development, since MPC is computationally demanding. New MPC methods are developed constantly, and the algorithms are made more effective. It is possible to implement the demanding MPC control problem discussed in this thesis on a standard computer today.

Nonlinear model predictive control generated from linear MPC and many of the techniques used in linear MPC were transferred to NMPC [1]. In the article by Chen and Shaw [10], receding horizon feedback control was used on a nonlinear system in 1982. This article can be seen as one of the first steps towards NMPC [1].

4.2 Constraints

The ability to deal efficiently with constraints distinguishes MPC from other control methods. In this thesis, constraints are considered both on the control variable and on the states of the system. A nonempty state constraint set $\mathbb{X} \subseteq X$ is introduced and for each $x \in \mathbb{X}$ a nonempty control constraint set $\mathbb{U}(x) \subseteq U$ is introduced. These sets are used to construct a general definition for the constraints. The constraints are defined as admissibility in the definition beneath.

Definition 4.1 (Admissibility)**.** Consider the control constraint set $\mathbb{U}(x) \subseteq U$, the *state* constraint set $\mathbb{X} \subseteq X$ and the control system (4.1).

1. The states of the system $x \in \mathbb{X}$ are called *admissible states* and the control variables $u \in \mathbb{U}(x)$ are called *admissible control* values for x. The elements of the set $\mathbb{Y} = \{(x,u) \in X \times U | x \in \mathbb{X}, u \in \mathbb{U}(x)\}$ are called *admissible pairs.*

2. For an initial value $x(n) \in \mathbb{X}$ and for $N \in \mathbb{N}$, a *control sequence* $u(\cdot) \in U^N$, which corresponds to the trajectory $x_p(\cdot)$, is called *admissible for* $x(n)$ *up to time* N, if

$$(x_p(k),u(k)) \in \mathbb{Y} \text{ where } x_p(0) = x(n), \text{ for all}$$
$$k = 0, \dots ,N - 1$$

holds. The set of admissible control sequences for $x(n)$ up to time N is denoted by $\mathbb{U}^N(x(n))$.

3. A control sequence $u(\cdot) \in U^{\infty}$ and the corresponding trajectory $x_p(\cdot)$ are called *admissible* for $x(n)$ if they are admissible for $x(n)$ up to every time $N \in \mathbb{N}$. The set of admissible control sequences for $x(n)$ is denoted by $\mathbb{U}^{\infty}(x(n))$.

4. A *feedback law* $\mu : \mathbb{N}_0 \times X \mapsto U$ is called *admissible* if $\mu(x) \in \mathbb{U}(x)$ holds for all $x \in \mathbb{X}$ and all $n \in \mathbb{N}_0$.

4.3 Introduction to NMPC

The idea with MPC is to optimize a cost function to obtain the optimal control signal that satisfies the constraints given the dynamics of the model. This is done in every MPC iteration at every sampling time. Constraints are easily implemented in the optimization and in this thesis, constraints are used on the control signal and on the states of the prediction model. A major advantage compared to other control methods is that MPC also manages control problems with multiple input and multiple output (MIMO).

The basic features of NMPC are the same as for MPC. The difference is that a nonlinear model is used in NMPC, which can result in non-convex optimization. In NMPC and MPC, a process behaviour is predicted and optimized, hence, a model is required to describe the process. The system that is controlled in this thesis is a nonlinear discrete-time system of the form:

$$x(k+1) = f(x(k),u(k)), \tag{4.1}$$

where $x(k+1)$ is the state of the system at the next time instant, $x(k)$ is the state at time k and $u(k)$ the control variable at time k. It is possible to approximate the infinite horizon optimal control problem with MPC and it is defined as:

Problem 4.2 (OPC_{∞}). *Find $u(\cdot)$ that minimizes*

$$J_{\infty}(x_0,u(\cdot)) := \sum_{n=0}^{\infty} l(x(n),u(n)), \tag{4.2}$$

such that $x(n+1) = f(x(n),u(n))$, $x(0) = x_0$ *and* $x(n) \in \mathbb{X}$, $u(n) \in \mathbb{U}(x(n))$ *for all* $n = 0,1,\ldots$

For a nonlinear model the Problem 4.2 does not have any explicit solutions, and hence NMPC is considered. The NMPC idea is now introduced. For the current state $x(n)$, we can iterate the system for any control sequence $u(0),\ldots,u(N-1)$ and construct a *prediction trajectory* x_p defined by

$$x_p(0) = x(n), \quad x_p(k+1) = f(x_p(k),u_p(k)) \quad \text{for } k = 0,\ldots,N-1, \tag{4.3}$$

where $N \geq 2$. The control variables for the prediction trajectory are defined as $u_p(k)$. At every time instant, the future behaviour of the system is obtained for the chosen control sequence $u_p(0),\ldots u_p(N-1)$, on a discrete-time interval, N steps into the future. N is called the *prediction horizon.*

The idea with MPC is to optimize the control sequence $u_p(0),\ldots u_p(N-1)$, such that the predicted behaviour of the system is close to a reference value x_r. The optimization is done by measuring the distance between the state $x_p(k)$ and x_r through a function $l(x_p(k),u_p(k))$. The optimal control is achieved when the distance between $x_p(k)$ and the reference x_r is equal to zero. When this is achieved, the optimizer should hold the state near the reference. It is also possible to penalize the distance between the control variable $u_p(k)$ and a reference u_r. Usually, a quadratic function is chosen; a popular choice according to [1] is

$$l(x_p(k),u_p(k)) = |x_p(k)|^2 + w|u_p(k)|^2,$$

where $|\cdot|$ represents the Euclidean norm and w is a weighting constant that is chosen $w \geq 0$. In this cost function, the reference values x_r and u_r are chosen zero. The optimal control problem is now stated as

Problem 4.3.

$$\textit{minimize } J(x(n),u_p) = \sum_{k=0}^{N-1} l(x_p(k),u_p(k))$$

$$\textit{with respect to } u \in \mathbb{U}^N(x(n)), \;\; \textit{subject to}$$

$$x_p(0) = x(n),\; x_p(k+1) = f(x_p(k),u_p(k))$$

The optimization problem is solved at every time instant. Now assume that Problem 4.3 has a solution, *i.e.* there exists a control sequence $u_p^*(0), \dots u_p^*(N-1)$ that minimizes (4.3). The first optimal control variable $u_p^*(0)$ is used as a feedback control value, at the next time instant, together with the new measurements $x(k+1)$, and the algorithm restarts. In the next section we summarize this as an algorithm.

4.3.1 The NMPC algorithm

Algorithm 4.4. *For a constant reference, the NMPC algorithm can be defined for each sampling time T_n and $n = 0,1,2,\dots,$ in three steps:*

1. *Measure the state $x(n)$ of the system at the current sampling time T_n.*
2. *Solve the optimal control problem (4.3) for the measured state:*

$$\text{Minimize } J(x(n),u_p) = \sum_{k=0}^{N-1} l(x_p(k),u_p(k)),$$

$$\text{with respect to } u \in \mathbb{U}^N(x(n)), \text{ subject to}$$

$$x_p(0) = x(n),\ x_p(k+1) = f(x_p(k),u_p(k)),$$

   and denote the obtained optimal control sequence by $u_p^*(\cdot)$.

3. *Apply the first optimal control value $u_p^*(0)$ as control decision for the next sampling time T_n.*

In Algorithm 4.4, it is assumed that an optimal control sequence $u_p^*(\cdot)$ exists. Now an example of how to use the Algorithm 4.4 is presented.

Example 4.5. Consider the nonlinear discrete-time system and the cost

$$x(k+1) = x(k)^2 - u(k) \tag{4.4}$$

$$l(x(k),u(k)) = x(k)^2 + u(k)^2, \tag{4.5}$$

with the initial value $x(0) = 1$, optimization horizon $N = 2$ and a constraint for the control signal $u \geq 0$. In this example, both x_{ref} and u_{ref} are 0 and not time varying for all k. Now using Algorithm 4.4 we control the system to zero.

NMPC iteration 1

In step 1, we measure the state $x(0) = 1$ and solve the optimal control problem in step 2 of Algorithm 4.4.

$$\begin{aligned}&\text{Minimize } J(x(0),u) = \sum_{k=0}^{1} x(k)^2 + u(k)^2,\\&\text{with respect to } u \in \mathbb{U}^N(x(n)), \text{ subject to}\\&x_p(0) = x(0) = 1,\ x_p(k+1) = x_p(k)^2 - u_p(k).\end{aligned}$$

$$\begin{aligned}\min x_p(0)^2 + u_p(0)^2 + x_p^2(1) + u_p(1)^2 &= \min 1^2 + u_p(0)^2 + (1^2 - u_p(0))^2 + u_p(1)^2\\&= \min 2u_p(0)^2 - 2u_p(0) + 2 + u_p(1)^2\\&= \min 2\left(u_p(0) - \frac{1}{2}\right)^2 + \frac{3}{2} + u_p(1)^2,\end{aligned}$$

Which is minimized at $u_p^*(0) = \frac{1}{2}$ and $u_p^*(1) = 0$. In step 3, we apply the first optimal control value $u_p^*(0) = \frac{1}{2}$ as input for the system and obtain $x(1) = 1^2 - \frac{1}{2} = \frac{1}{2}$.

NMPC iteration 2

For the next sample time we start with step 1 and measure the new initial value $x(1) = \frac{1}{2}$. Now using step 2 from the algorithm,

$$\begin{aligned}&\min x_p(0)^2 + u_p(0)^2 + x_p(1)^2 + u_p(1)^2\\&= \min \left(\frac{1}{2}\right)^2 + u_p(0)^2 + \left(\left(\frac{1}{2}\right)^2 - u_p(0)\right)^2 + u_p(1)^2\\&= \min 2u_p(0)^2 - \frac{u_p(0)}{2} + \frac{5}{16} + u_p(1)^2\\&= \min 2\left(u_p(0) - \frac{1}{8}\right)^2 + \frac{9}{32} + u_p(1)^2,\end{aligned}$$

Wwich is minimized at $u_p^*(0) = \frac{1}{8}$ and $u_p^*(1) = 0$. In step 3, we apply the first optimal control value $u_p^*(0)$ as input for the system and obtain $x(2) = \frac{1}{2}^2 - \frac{1}{8} = \frac{1}{8}$.

NMPC iteration 3

For the next sample time we start with step 1 and measure the new initial value $x(2) = \frac{1}{8}$. Now, using step 2 from the algorithm,

$$\begin{aligned}
&\min x_p(0)^2 + u_p(0)^2 + x_p(1)^2 + u_p(1)^2 \\
&= \min \left(\frac{1}{8}\right)^2 + u_p(0)^2 + \left(\left(\frac{1}{8}\right)^2 - u_p(0)\right)^2 + u_p(1)^2 \\
&= \min 2u_p(0)^2 - \frac{u_p(0)}{32} + \frac{65}{4096} + u_p(1)^2 \\
&= \min 2\left(u_p(0) - \frac{1}{128}\right)^2 + \frac{129}{8192} + u_p(1)^2,
\end{aligned}$$

which is minimized at $u_p^*(0) = \frac{1}{128}$ and $u_p^*(1) = 0$. In step 3, we apply the first optimal control value $u_p^*(0)$ as input for the system which results in $x(3) = \frac{1}{8}^2 - \frac{1}{128} = \frac{1}{128}$.

NMPC iteration 4

For the next sample time we start with step 1 and measure the new initial value $x(3) = \frac{1}{128}$. Now, using step 2 from the algorithm,

$$\begin{aligned}
&\min x_p(0)^2 + u_p(0)^2 + x_p(1)^2 + u_p(1)^2 \\
&= \min \left(\frac{1}{128}\right)^2 + u_p(0)^2 + \left(\left(\frac{1}{128}\right)^2 - u_p(0)\right)^2 + u_p(1)^2 \\
&= \min 2u_p(0)^2 - \frac{u_p(0)}{8192} + \frac{16385}{268435456} + u_p(1)^2 \\
&= \min 2\left(u_p(0) - \frac{1}{32768}\right)^2 + \cdots + u_p(1)^2,
\end{aligned}$$

which is minimized at $u_p^*(0) = \frac{1}{32768}$ and $u_p^*(1) = 0$. In step 3, we apply the first optimal control value $u_p^*(0)$ as input for the system and obtain $x(3) = \left(\frac{1}{128}\right)^2 - \frac{1}{32768} = 0.000031\ldots$. The state $x(3)$ is close to zero and we stop the NMPC algorithm.

As can be seen, the optimization problem can be solved easily, since the horizon N is short. If one increase N to 3 the problem becomes more difficult to solve by hand.

There are different ways to solve the optimal control problem, but these methods are not covered in this thesis. In [1, Section 3.4], the dynamic programming principle is proposed and proved for the optimal control problem. Other algorithms such as the Interior-Point method, Active Set SQP methods and Multiple Shooting are discussed in [1, Chapter 12].

In the software that Grüne and Pannek developed, the optimal control problem is solved with the Matlab function *fmincon*. The function is a nonlinear programming solver that can handle nonlinear constrained multivariable optimization problems. The default optimization algorithm used in fmincon is the interior-point method. More information about fmincon can be found on the Mathworks website [20].

The constraints in Algorithm 4.4 are hard and the controller cannot violate the constraints, which is why constraints are usually set on a critical level in real applications. In practice, it would be desirable to keep the output at a safe level away from the constraints. A way to do that is by setpoint tracking, which was covered in the previous chapter. Nonlinear setpoint tracking combined with NMPC is discussed briefly in Chapter 5. State estimation is usually required for NMPC, since all the states of the model might not be measurable.

Chapter 5

Nonlinear tracking, NMHE and NMPC

In this chapter, nonlinear setpoint tracking and nonlinear MHE are covered briefly before they are combined with nonlinear MPC. See Figure 1.1 for how the components are combined.

5.1 Nonlinear setpoint tracking

The theory of nonlinear setpoint tracking is complicated and it is in general difficult to obtain exact and unique solutions for nonlinear setpoint tracking. Hence, in this section, a nonlinear tracking problem is presented and the implementation for the NMPC is discussed in the next section.

Problem 5.1 (nonlinear tracking problem)**.**

$$\min_{x_s, u_s} \frac{1}{2} \left(|u_s - u_{sp}|^2_{R_s} + |Cx_s - y_{sp}|^2_{Q_s} \right) \tag{5.1}$$

subject to

$$x_s - f(x_s,u_s) = 0 \tag{5.2}$$

$$HCx_s = r_{sp} \tag{5.3}$$

$$Eu_s \leq e \tag{5.4}$$

$$Gx_s \leq g. \tag{5.5}$$

In Problem 5.1, the objective function (5.1) is exactly the same as in the linear case (2.5). The only difference is that the system is replaced with a nonlinear system and for the system, the steady state assumption must hold. The setpoint r_{sp} is determined through an equality constraint. Constraints are available for the steady state x_s and for the control variable u_s. For the nonlinear and linear tracking problem there is no guarantee for feasibility of the solution.

5.1.1 Nonlinear setpoint tracking combined with NMPC

The idea with setpoint tracking, is to find a reference value for the NMPC cost function. In this thesis, a reference value is determined for the control variable. This reference value for the control signal is calculated with Problem 5.1, and the reference is used in the NMPC cost function to penalize the control, when it is far from the reference. In Problem 5.1, the discrete-time simplified model is the function f. The reference value identifies the steady state that satisfies the setpoint, and when used in the NMPC cost function, the NMPC is expected to steer the system to this steady state. The problem with nonlinear setpoint tracking is, that if a solution does not exist, the NMPC regulator cannot achieve optimal control decisions, since it relies on the reference.

The tracking problem is implemented in Matlab and solved using fmincon. The code for the setpoint tracking problem can be found in A.5. A practical example on the benefits of setpoint tracking in NMPC is presented through simulation below.

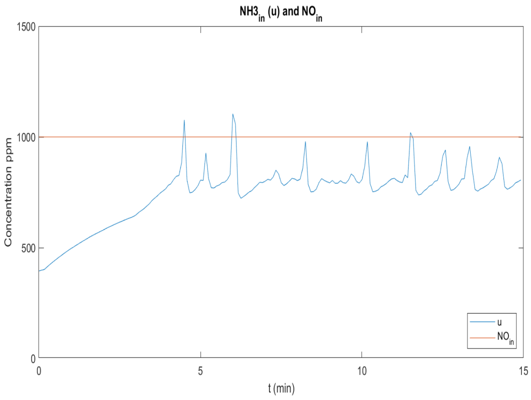

Figure 5.1: Unstable control.

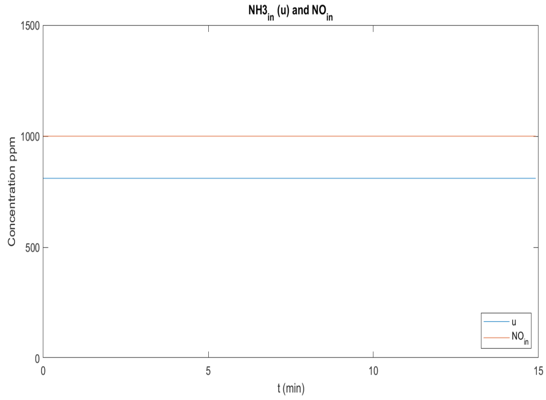

Figure 5.2: Stable control.

In both figures the red curve describes the disturbance and the blue curve describes the control decision made by the NMPC. In Figure 5.1, a quadratic cost-function with reference $u_r = 0$ is chosen for the NMPC and one can observe that when the system reaches steady state, the NMPC does not find steady state. In this example, setpoint tracking is not used. In Figure 5.2, the steady state is determined for a specific output and a reference value for the control signal u is calculated with the nonlinear tracking Problem 5.1 using the Matlab code A.5. The reference value is then used in the NMPC cost function to penalize the control signal when it is far from the reference. Now the control signal is constant, and no oscillations occur, since it is possible to make the NMPC cost function zero. The simulation time is three times faster when using the reference value in the cost function. More information about the simulations can be found in Chapter 7.

Some problems might arise when using nonlinear setpoint tracking. There is no guarantee that the setpoint is feasible for the nonlinear system. Infeasibility might arise when the disturbance changes drastically. For the nonlinear system considered in this thesis, the nonlinear setpoint tracking problem becomes unfeasible when the disturbance goes to zero. When the setpoint tracking problem becomes infeasible, the NMPC cannot make optimal control decisions, since it relies on the reference that the tracking problem provides.

5.2 Nonlinear MHE

In this thesis, nonlinear plant states are estimated with the NMHE cost

$$\hat{V}_T(\hat{X}_N(T)) = \frac{1}{2}\Bigg(|\hat{x}(T-N) - \bar{x}(T-N)|_P^2 + \\ \sum_{k=T-N}^{T-1} |\hat{x}(k+1) - f(\hat{x}(k),u(k))|_Q^2 + \sum_{k=T-N}^{T} |y(k) - h(\hat{x}(k))|_R^2 \Bigg), \tag{5.6}$$

where the weight for the prior weighting term is set to zero. The difference to the linear case is, that a general function f is used to estimate the plant dynamics, where f is the estimation model, and a function h returns the estimates that are compared to the measurements. The cost function uses the simplified nonlinear discrete-time model as f, which should resemble the plant.

In our implementation, the measurements are assumed to be zero before the measurement window is full, *i.e.*, at the first sampling instant the measurement window consists of N zeros. At the next sampling instant, the measurement window consists of $N-1$ zeros and the first measurement, *i.e.* $0,\ldots,0,y_1$. This procedure proceeds until the measurement window is filled. This can be seen as full information estimation until the measurement window is full, since all the available measurements are used.

The results from Theorem 3.2 and 3.3 are not usable for nonlinear plants, since both theorems require linear plants. In [2, Chapter 4], some results of nonlinear estimator stability are proved for nonlinear plants, these results, however, do not consider the control signal in the cost function. In [2, Section 4.3.3] for instance, Theorem 4.37 guarantees estimator stability for moving horizon estimation for a horizon length N, but for this result to hold, five assumptions must hold. The first assumption states that the cost function should be zero when the estimates and the measurements are zero. The assumption does not consider the control variable and hence, for the plant in this thesis, it is possible that the cost is not zero.

In general, most of the results for nonlinear estimation in [2] require many assumptions. This is usually the case when dealing with nonlinear systems, since it is difficult to obtain explicit solutions.

5.2.1 NMHE compared to other estimation methods

In [2], a comparison between MHE, EKF and the Unscented Kalman Filter (UKF) is done. The EKF and UKF are computationally more effective, since the equations update recursively. For nonlinear MHE, the estimation is more demanding, because numerical optimization for long horizons N cause a high workload for the computer. Short horizons on the other hand, might produce inaccurate estimates.

In [2, Section 4.4.4], the comparison is made for the estimation of the states of a nonlinear model using MHE, EKF and UKF. Their results show that MHE produce the most accurate estimates. In the example, some states converge to negative concentrations when using the EKF. This affects the regulation significantly and quite likely results in an unstable regulator. A modified version of the UKF produces slightly more accurate estimates, but the convergence to the plant state is slow. The MHE produces the most accurate estimates, and the estimates converge quickly to the plant state. In the MHE, constraints are available, which can prevent that the estimates converge to negative values.

In a much cited article [14], Haseltine and Rawlings compare the EKF with MHE, for a nonlinear model. They show that MHE produces much more accurate estimates than the EKF, but they also conclude that it comes with a cost in computational efficiency. In their Table 4, the average CPU time per time step (s) is presented and one can see that the EKF is faster to compute.

In another article [13] written by Ubare *et al.*, NMPC is combined with different state estimation methods and the performance is compared. In the article, NMPC is combined with EKF, UKF and nonlinear MHE. The performance between the estimation methods is analysed and they conclude for their example that NMPC combined with nonlinear MHE outperforms both NMPC combined with EKF and NMPC combined with UKF. They conclude that NMHE with horizon 5 is comparable to the other methods in performance, but if one can ignore the computation cost, the NMHE with longer horizon produce even more accurate estimates. Figure 3 in their article illustrate the estimation and Figure 6 illustrate the estimation time, NMPC combined with nonlinear MHE with horizon 5 has the shortest estimation time.

5.2.2 Nonlinear MHE combined with NMPC

In [2, Section 4.5], MHE combined with MPC is discussed. The section is focused on the effect of the estimation error in the MHE when combining control and estimation. Results about the NMPC performance is not covered. They conclude the section with a result (Theorem 4.46) that MHE combined with MPC is robustly asymptotically stable, which means that it is input-to-state stable on a robustly positive invariant set.

For this result to hold many assumptions must hold for the MPC regulator and for the MHE. The first two assumptions for the regulator might be easy to prove. They deal with continuity of the system and cost, and with properties of the constraint set, but the third assumption, which deals with basic stability is more difficult to prove. As for nonlinear MHE stability, the result regarding MPC and MHE combined, also require many assumptions, since nonlinear systems are considered. For more details see Section 4.5 in [2].

In [16], some conditions are proposed to achieve stable control in NMPC based on estimation. In the article, MHE is considered briefly and the main condition to achieve stable control is that the estimate converges fast to the exact plant value. In this article, a mathematical approach is used.

5.3 NMPC, Nonlinear setpoint tracking and nonlinear MHE

In [2], nonlinear setpoint tracking is not covered, even though, linear setpoint tracking is covered in Section 1.5. In [2, Section 5.6], it is only noted that stability of output MPC, when considering a nonlinear system, has not gained much attention, with output MPC meaning MPC combined with state estimation. In [2, Section 5.7], some articles are presented where the tracking problem is discussed and for an interested reader, this section is recommended.

In [15], NMPC, MHE and target tracking is combined in a similar way as in this thesis. Their results are based on simulation and no mathematical results are presented. In their study, a solid oxide fuel cell is controlled based on the estimates. They conclude that the estimator provides good estimates and that it is possible to control the process using NMPC, target tracking and MHE. They also conclude that for a practical implementation, the computational time causes problems, since it requires much time to solve the optimization problems.

To conclude this section, it can be said that there does not exist many results of nonlinear tracking, NMPC and nonlinear MHE combined, and more research in this area is required.

5.4 Conclusion

There exist many results for linear plants considering MPC, MHE and Setpoint tracking separately, since explicit solutions are more easily obtained. For nonlinear plants, the theory becomes much more complicated and results considering stability for instance, require a great deal of work, since it is impossible to obtain explicit solutions.

In [1, Section 2.3], stabilization of discrete-time systems to a reference is presented. In Section 4.1, stability to a reference for the infinite horizon optimal control problem is presented. One result that deals with finite horizon NMPC stabilization to a reference is Theorem 7.41 in [1]. The Theorem does not cover estimation or how the reference is calculated, hence the Theorem is not applicable for this thesis. More information about NMPC feasibility and stabilization can be found in [1, Chapter 7].

Even though many of these applications are used in the industry in the nonlinear case, the theory for different combinations of nonlinear estimation and NMPC is deficient. More research in this area is required to obtain solutions and understanding for these problems.

Chapter 6

Selective catalytic reduction

6.1 The full SCR model

The model that is used to describe the Selective Catalytic Reduction is presented beneath. The model is developed by Milver Colmenares in his Master's thesis [11]. The SCR can be modeled in detail as a partial differential equation. To avoid this, perfect mixture in the catalyst is assumed and the SCR can be modeled as several cells connected in series, which results in a system of nonlinear first order ordinary differential equations:

$$\frac{d\theta}{dt} = k_{ads} c_{NH_3}(1-\theta) - (k_{des} + k_{red} c_{NO} + k_{ox} c_{o_2})\theta \tag{6.1}$$

$$\frac{dc_{NO}}{dt} = \frac{\dot{V}}{V}(c_{NO,in} - c_{NO}) - k_{red} c_{NO} \theta c_{catmax} \tag{6.2}$$

$$\frac{dc_{NH_3}}{dt} = \frac{\dot{V}}{V}(c_{NH_3,in} - c_{NH_3}) - k_{ads} c_{NH_3}(1-\theta) c_{catmax} + k_{des}\theta c_{catmax} \tag{6.3}$$

and this model represents one cell. The SCR model consists of three chemical reactions, (6.1) describes NH_3 adsorption and θ is the ammonia coverage ratio. Equation (6.2) describes NO reduction, where c_{NO} is the concentration of NO. Equation (6.3) describes NH_3 oxidation, where c_{NH_3} is the concentration of NH_3. The constant k_{ads} describes the adsorption rate, k_{des} the desorption rate, k_{red} the reduction rate and k_{ox} describes the oxidation rate. The oxidation rate is assumed to be zero in this thesis, which is why it is removed from the model in further calculations. The NH_3 adsorption capacity of the catalyst is described by c_{catmax}, $\dot{V}$ is the exhaust gas flow and V is the reactor volume. The SCR model is controlled by an ammonia injection $c_{NH_3,in}$ and disturbed by $c_{NO,in}$, which describes the NO, coming from the engine.

To obtain a more realistic model that describes a real catalyst, a sequence of these cells is connected, and the model consists of several systems connected in series. The concentrations in Equations (6.2) and (6.3) are used as input for the next cell, as $c_{NO,in}$ and c_{NH_3in}. In this expanded model, the ammonia c_{NH_3in} injected to the first cell is the only thing that can be regulated.

The full SCR model is highly nonlinear, which is why a nonlinear model predictive controller is used to control the process. In most simulations done in this thesis, a four-cell structure is used to describe the plant, and it consists of four SCR cell models connected in series, which results in 12 states. In the beginning of this project, it was proposed to use the continuous-time full SCR model as prediction model for the NMPC, but it resulted in slow control.

To achieve faster control a discrete-time prediction model is proposed, since discrete-time models are more suitable for numerical calculations. The Zero Order Hold (ZOH) discretization method is implemented in the software, but the control of the full SCR model is slow compared to if the full SCR model would be discretized in advance.

The full continuous-time SCR model was transferred to discrete time using other discretization methods as well. The methods that were tested were the Euler method and the Heun method, but they resulted in unstable control. The discrete-time models have problems capturing the fast dynamics of the Equations (6.2) and (6.3).

In the next section, a simplified version of the full SCR model is determined that is used both as prediction model and estimation model, the simplified model is also used in the setpoint tracking problem. The new prediction model is then used for the control in discrete time. The continuous-time full SCR model is used to simulate the plant in the simulations.

6.2 A simplified SCR model

The Equations (6.2) and (6.3) reach steady state much faster than (6.1). A way to derive a good prediction model for the NMPC is to approximate that the fast reactions reach steady state immediately, *i.e.* to set the derivatives of c_{NO} and c_{NH_3} equal to 0 in the Equations (6.2) and (6.3). The simplified model that is used in both NMPC, MHE and setpoint tracking is derived in the following calculations.

From (6.2), one easily obtains:

$$\begin{aligned} 0 &= \frac{\dot{V}}{V}(c_{NO,in} - c_{NO}) - k_{red}c_{NO}\theta c_{catmax} \iff \\ \frac{\dot{V}}{V}c_{NO,in} &= \left(\frac{\dot{V}}{V} + k_{red}\theta c_{catmax}\right) c_{NO} \end{aligned}$$

and we obtain,

$$c_{NO} = \frac{c_{NO,in}}{1 + k_{red}\theta \frac{c_{catmax}V}{\dot{V}}}. \tag{6.4}$$

From (6.3), one can again calculate:

$$\begin{aligned} 0 &= \frac{\dot{V}}{V}(c_{NH_3,in} - c_{NH_3}) - k_{ads}c_{NH_3}(1-\theta)c_{catmax} + k_{des}\theta c_{catmax} \iff \\ \frac{\dot{V}}{V}c_{NH_3,in} + k_{des}\theta c_{catmax} &= c_{NH_3}\left(\frac{\dot{V}}{V} + k_{ads}(1-\theta)c_{catmax}\right) \end{aligned}$$

and we obtain,

$$c_{NH_3} = \frac{c_{NH_3,in} + k_{des}\theta \frac{c_{catmax}V}{\dot{V}}}{1 + k_{ads}(1-\theta)\frac{c_{catmax}V}{\dot{V}}}. \tag{6.5}$$

In both (6.4) and (6.5) it is assumed that $\dot{V} \neq 0$. The first Equation (6.1) stays the same but can now be rewritten using (6.4) and (6.5).

$$\frac{\theta}{dt} = k_{ads}c_{NH_3}(1-\theta) - (k_{des} + k_{red}c_{NO})\theta. \tag{6.6}$$

The model (6.6) where c_{NO} is replaced with (6.4) and c_{NH_3} is replaced with (6.5), is now used as the prediction model for the NMPC and estimation model for the MHE and as a model for the setpoint tracking. In the new simplified model (6.6) it is assumed that the states for c_{NH_3} and c_{NO} reach steady state immediately. These states are important, since they describe the emissions that we want to control.

We next propose to approximate both states with an exponential moving average value in discrete time, which is calculated by

$$c_a(k+1) = \left(1 - \frac{1}{W}\right) c_a(k) + \frac{1}{W} c, \tag{6.7}$$

where $V > 0$ is an appropriately chosen weighting factor. The constraints for the NMPC are set on the long-term average value of $c_{NH_{3a}}$ and c_{NO_a} calculated by the formula (6.7) and the constraints are set to match the emission regulations. More information about the exponential moving average value can be found in [17, Section 8.1].

The NMPC used in this thesis, use the prediction model in discrete time. To transform the model to discrete time, the Euler method is used. The Euler method can be derived in many different ways, and it is excluded from this thesis. The Euler method is defined as

$$x_s(k+1) \approx x_s(k) + T \cdot f(x_s(k), u_s(k)), \tag{6.8}$$

where T is the sampling time. The simplified model (6.6) in continuous time is transformed to discrete time using the Euler method. The simplified discretized model with the average values is presented beneath

$$\theta(k+1) = \theta(k) + T\left(k_{ads} c_{NH_3}(k)(1-\theta(k)) - (k_{des} + k_{red} c_{NO}(k))\theta(k)\right) \tag{6.9}$$

$$c_{NO_a}(k+1) = c_{NO_a}(k) + \left(\frac{1}{W} c_{NO}(k) - \frac{1}{W} c_{NO_a}(k)\right) \tag{6.10}$$

$$c_{NH_{3a}}(k+1) = c_{NH_{3a}}(k) + \left(\frac{1}{W} c_{NH_3}(k) - \frac{1}{W} c_{NH_{3a}}(k)\right). \tag{6.11}$$

Equation (6.9) describes the ammonia coverage in the catalyzer, (6.10) describes the moving average value of the NO concentration and (6.11) describes the moving average value of the NH_3 concentration. In the simulations $W = \frac{w}{T}$ where w is a constant and T the sample time.

Chapter 7

Simulation results

All simulations are done in Matlab version R2021b and graphs with results from the simulations are presented in this chapter.

7.1 Prediction model validation

At first, the prediction model (6.6) and the full SCR model are compared to see if the prediction model captures the behaviour of the original model. This is important, since MPC relies heavily on the prediction model. If there is a clear difference between the models, the control actions of the NMPC are inaccurate.

Simulations on the simplified prediction model show that it describes the original SCR model well, this simulation is done using the code from A.1. The simplified model is compared both in discrete and continuous time to the original model in continuous time. As can be seen from Figure 7.1, the simplified model follows the θ curve almost exactly in the figure that describes θ. Both discrete and continuous time models capture the behaviour of θ. In the upper right figure one can see that the average value of NO converge to the nominal value. The same behaviour can be observed for NH_3 in the bottom left figure. In the bottom right figure one can observe the control variable $NH_{3_{in}}$ and the disturbance NO_{in}. The parameters used in the simulations are; $k_{ads} = 10$, $k_{red} = 300$, $k_{des} = 0$, $c_{cat,max,NH_3} = 0.1$ and $\frac{\dot{V}}{V} = 2/4$, the same parameters are used in all the following simulations. For this specific simulation the disturbance was set to $c_{NO,in} = 0.001$ and the control signal was set to $c_{NH_3,in} = 0.0008$. The sampling period for the discrete-time simplified prediction model was set to $T = 5$ seconds.

The simplified model is also controlled in both continuous time and discrete time with NMPC. The important observation from this test is that the discrete time model is much faster to evaluate. This simulation is not illustrated in this thesis, since it is quite obvious that the discrete model is faster to evaluate. The interested reader can test this using [18]. In all the NMPC simulations beneath, the discrete time model is used as the prediction model.

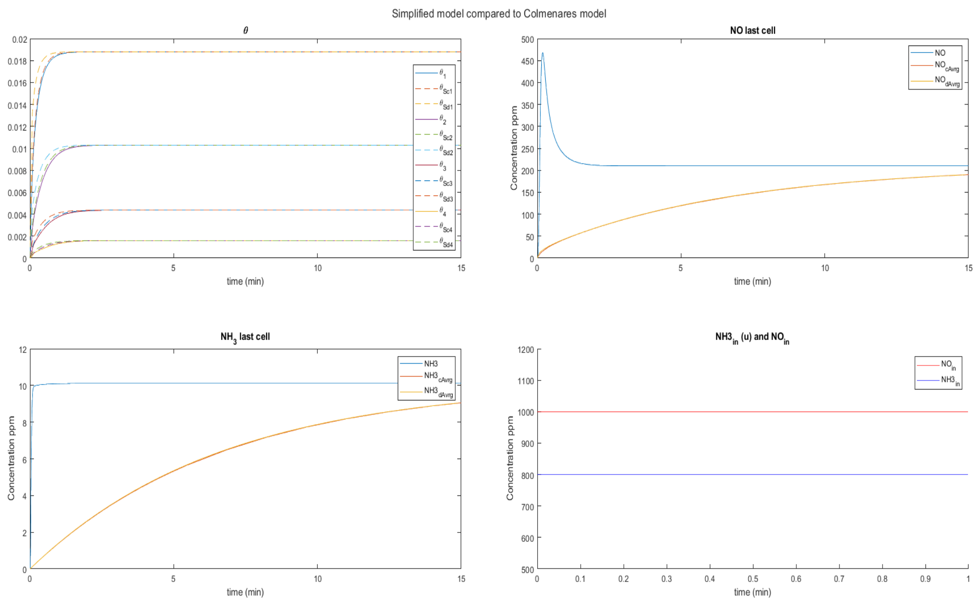

Figure 7.1: Simulations on the simplified model in continuous time and discrete time, compared to Colmenares's model in continuous time.

7.2 NMPC simulation setup

There are many parameters in the NMPC simulator that can be tuned and the most important are presented in the table beneath:

y_{sp}	Soft setpoint for the states
r_{sp}	Hard setpoint for the control variable
u_{sp}	Soft setpoint for the control variable
$nMHE$	NMHE horizon
$mpciterations$	NMPC iterations
N	Optimization horizon
T	Sample time

In the simulations the standard values are $y_{sp} = 0$, $r_{sp} = 0.0002$ (r_{sp} is the setpoint for the $NOout$), $u_{sp} = 0$, $nMHE = 13$, $mpciterations = 180$, $N = 40$ and $T = 5$. If these values change it is mentioned in the section.

To keep the NMPC feasible a slack variable is introduced for the NO constraint. It can be found in the NMPC cost function and in the constraint section. The slack variable has a large weight, and it is activated only in critical situations. All these parameters can be found in A.3 and the simulation is launched from this file.

7.3 Validation of the NMHE

In this section, the nonlinear Moving Horizon Estimation is tested. The code for the NMHE can be found in A.4. An NMPC simulation is done and the NMHE is executed at the same time, at every NMPC iteration. The code for the NMPC algorithm can be found in A.6. In this simulation, the NMPC and NMHE work separately, which means that no control actions are affected by the estimation. Two simulations are done, one with perfect measurement of NO_{out} and one with a measurement that consists of a mixture of NO and NH_3, as well as noise. The formula for the disturbed measurement is

$$NO + 0.5 \times NH_3 + (-1 \times 10^{-5} + (1 \times 10^{-5} + 1 \times 10^-5) \times c), \tag{7.1}$$

where c is a random number generated with the Matlab function randn. This formula can be found in A.6 but it is manipulated from A.3 on row 96 and

97. In the simulation, the NMHE uses 13 measurements. Before the window of measurements is fulfilled, the measurements that are given to the NMHE are zero and updates with the most recent measurement for every sampling instant.

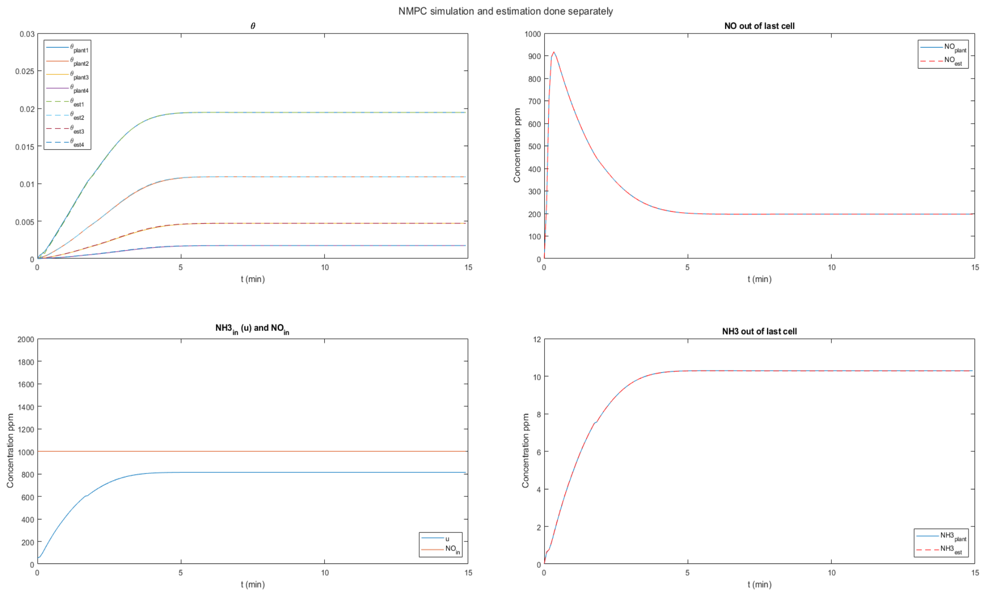

Figure 7.2: NMPC simulation and NMHE done simultaneously. Perfect measurement of NO_{out} and the reference values in the NMPC cost are zero.

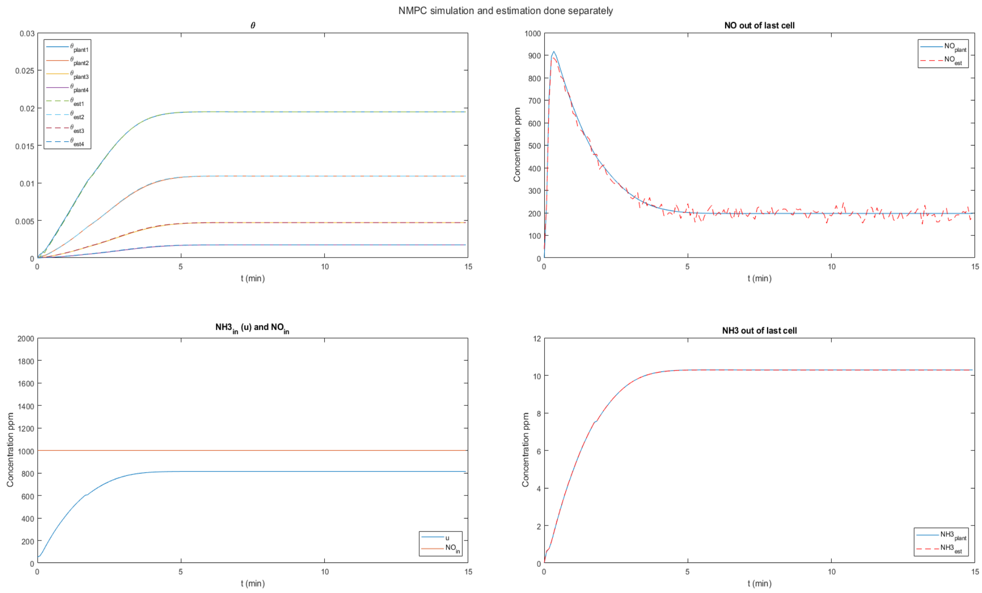

Figure 7.3: NMPC simulation and NMHE done simultaneously. Disturbed measurement of NO_{out} and the reference values in the NMPC cost are zero.

From Figure 7.2, where we have the perfect measurement, can be seen that the exact states and the estimates are almost exactly the same, except from the beginning, where the measurement window is not fulfilled. When noise is added to the NO_{out} measurement, the NO estimate moves around the real measurement. The reason to this is that the NMHE does not know about the noise and estimates the noise exactly. This might be solved by choosing different weights in the MHE cost function. This should not affect the control, since the control is done on the average value of NO. The other estimates remain the same.

For Figures 7.2 and 7.3 a quadratic cost function is used with zero reference for the control variable. The target selector is not in use in these figures. This results in slow control. When using a reference value for the control signal in the NMPC, the control actions become more aggressive and the estimation is not as accurate in the beginning. The ammonia coverage estimates are slightly ahead of the plant values. The results can be seen in Figure 7.4, where the target selector is used to determine the reference for the control signal. The code for the target selector can be found in A.5.

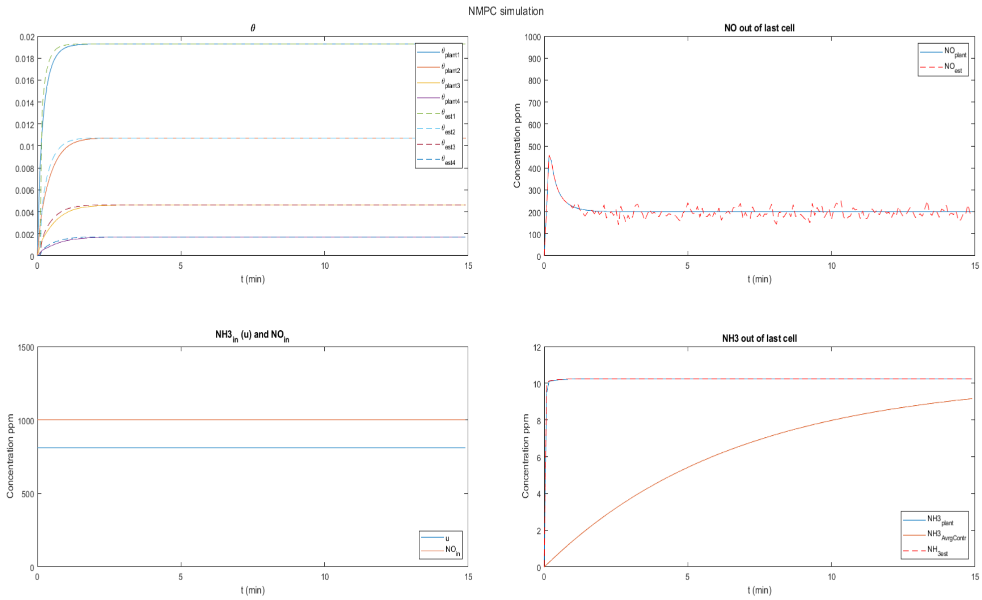

Figure 7.4: NMPC simulation and NMHE done simultaneously. Disturbed measurement of NO_{out} and a reference value for the control signal is calculated using the target selector.

The NO_{in} measurement is vital for the prediction and estimation model, hence the measurement of NO_{in} is assumed perfect in all simulations. Some major changes are required if one wants to conduct simulations, where the NO_{in} measurement has noise. Some estimation might be required for this part.

With these results promising results from the NMHE, a simulator is assembled, where Colmenares's model is used to model the plant and the simplified model is used as the prediction model and estimation model. In the next section, results of NMPC, nonlinear setpoint tracking and NMHE combined are presented and some comparison between NMPC and PI control is also presented. In all simulations with NMPC, the disturbed NO_{out} measurement and a perfect measurement of NO_{in} is used.

7.4 NMPC combined with setpoint tracking and NMHE

Nonlinear model predictive control is dependent on the model of the process. The simplified model has four states of ammonia coverage that the NMPC relies on. As mentioned earlier, the ammonia coverage cannot be measured, which is why the MHE is used to estimate these states. Before every NMPC step, state estimation is done to obtain the missing states. In the simulation beneath, a quadratic cost function is chosen and a reference for the control signal is calculated with the target selector using 5.1. The setpoint r_{sp} is set to 200ppm NO and the reference is calculated such that the ammonia slip is minimized, and the control signal is as low as possible, *i.e.* $u_{sp} = 0$ and $y_{sp} = 0$. The constraints for the NMPC are set on 11ppm for the average value of ammonia and 250ppm for the average value of NO. The *mpciterations* is set to 600.

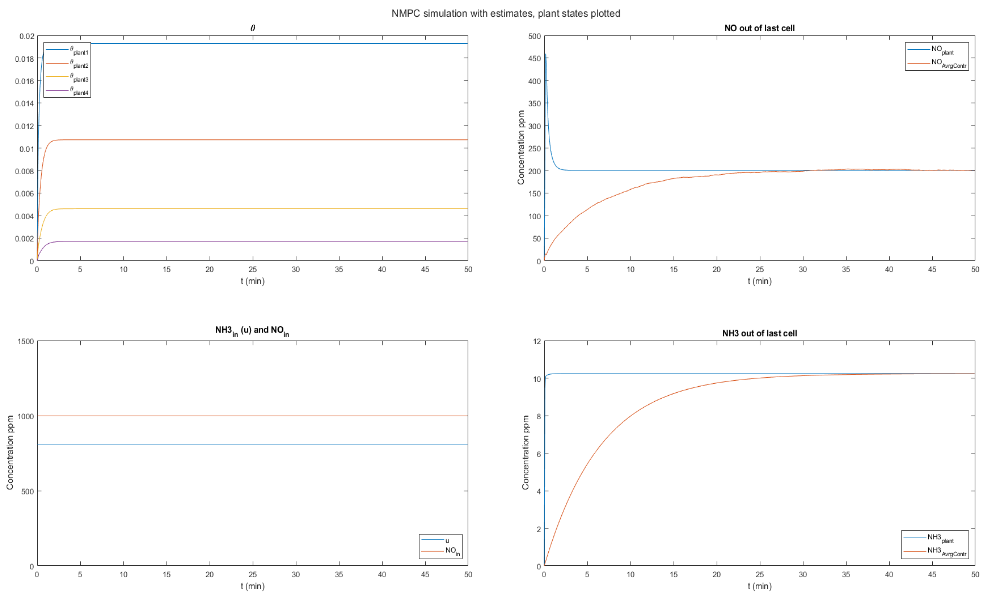

Figure 7.5: NMPC, nonlinear tracking and MHE combined.

From Figure 7.5 can be seen that the control action is constant. The target selector manages to find the reference for the control signal that steer the SCR to steady state with 200ppm NO coming out. The estimate of ammonia slip is also accurate, since the ammonia slip from the plant is kept beneath the limit, which was set to 11ppm. The average values are also plotted and it can be seen that they converge to the plant values. The constraints for the controller are on the average values, which eliminates the oscillations that occur in the NO measurement. Another benefit from the average values is that the controller is more flexible to fast changes in the disturbance.

In the next simulation, the NMPC, nonlinear tracking and MHE are tested with a varying NO_{in}, to see how the controller responds to change, $mpciterations$ is set to 1200.

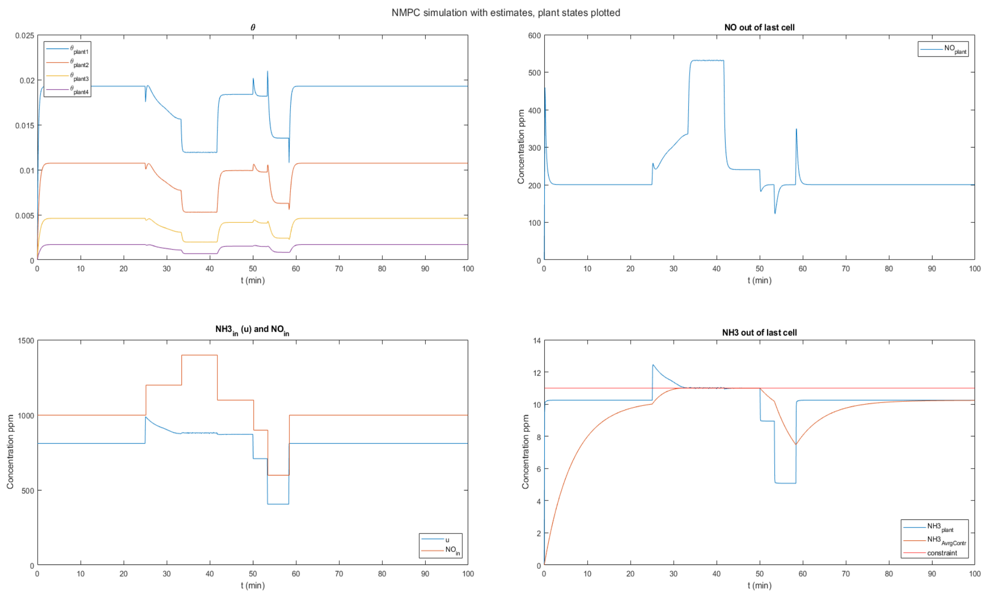

Figure 7.6: NMPC NO_{in} varying.

From Figure 7.6 can be seen that the controller responds quickly to a varying NO_{in}. A hard constraint is set on the average value of NH_3 concentration at 11ppm and one can see that the constraints hold for the average value. The setpoint for NO_{out} is still at 200ppm, which is kept when possible. When the NO_{in} rise, the controller injects the maximum amount of ammonia that hold the NH_3 out constraint. Since the controller cannot inject more ammonia, the NO_{out}

levels rise above the setpoint, which can be seen between 30 and 40 minutes. When this happens the control value is far from the reference and the control is slow.

7.5 PI-control

A proportional-integral-derivative controller (PID-controller) is a frequently used controller in industry. PID-control is popular, since it is easy to implement and use [4]. In this thesis, NMPC is compared to PI-control, which is a PID controller without the derivative term. The P term describes the proportional error value, and the integral term corrects the controller for steady offset from the reference value. The PI-controller compares the measured output to a reference value and then produces a control signal, based on the error. The PI-controller is derived from a continuous-time PID-controller

$$u(t) = K_c \left((r - y(t)) + \frac{1}{T_i} \int_0^t (r - y(s))ds + T_d \frac{d(r - y(t))}{dt} \right).$$

The parameters are explained in the table.

u	Control variable
K_c	Proportional gain
e	Control error
T_i	Integration time
h	Sample time
r	Reference value
y	Measurement

Now by approximating the integral with a rectangular approximation and by setting $T_d = 0$, we obtain by discretization the formula for the PI-controller that is used in the simulation. The formula for the PI-controller is presented beneath, where $e(k) = r - y_s(k)$ and the lower index s indicates a discrete signal

$$u_s(k) = K_c \left(e(k) + \frac{h}{T_i} \sum_{n=1}^{k} e(n) \right)$$

$$u_s(k) - u_s(k-1) = K_c \left(e(k) + \frac{h}{T_i} \sum_{n=1}^{k} e(n) \right) - K_c \left(e(k-1) + \frac{h}{T_i} \sum_{n=1}^{k-1} e(n) \right)$$

$$u_s(k) = u_s(k-1) + K_c \left(e(k) - e(k-1) + \frac{h}{Ti} \cdot e(k) \right). \tag{7.2}$$

In this particular PI-controller, the proportional gain is determined by λ-*tuning*, *i.e.* $K_c = \frac{T}{K\lambda}$, where $T = T_i$. The new parameter λ determines the pace of the regulation [3].

The parameters for the PI-controller are tuned with trial and error and the regulator works toward a long-term average value of NOx. The long-term average value is calculated with the moving exponential average value (6.7). The constants are set to $K_c = -0.9$ and $T_i = 1.9 \cdot 60$ for the simulation. The code for the PI-controller can be found in A.2.

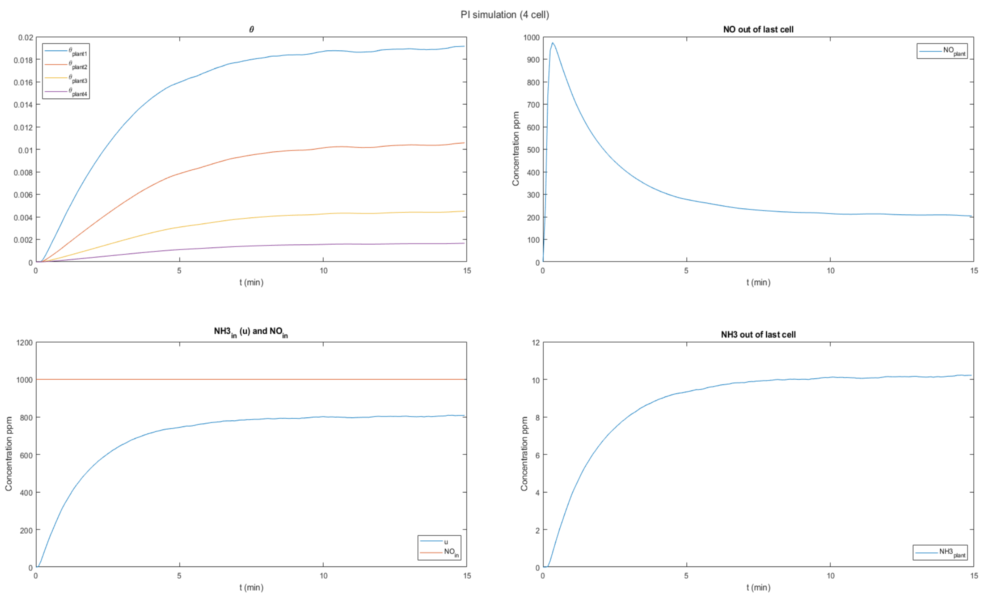

Figure 7.7: PI-control.

From Figure 7.7 can be seen that the PI-controller controls the SCR to steady state. The setpoint for NO was set to an average value of NO, at 200ppm. The simulation took 0.24 seconds, which is much faster than the NMPC which took 35 seconds. There are some situations where the PI-controller struggles and where NMPC work properly. These results are presented in the next section.

7.5.1 NMPC compared to PI-control

The PI-controller is a feedback controller and determines the control signal based on measurements. The NO concentration is the only thing that is measured and hence, the control is based on this measurement. The PI-controller has no information about the ammonia concentration and does not know how much ammonia is coming out. The benefit with NMPC is that the controller is aware of the ammonia slip and it possible to control both ammonia slip and NO emissions. In the simulation *mpciterations* is set to 600.

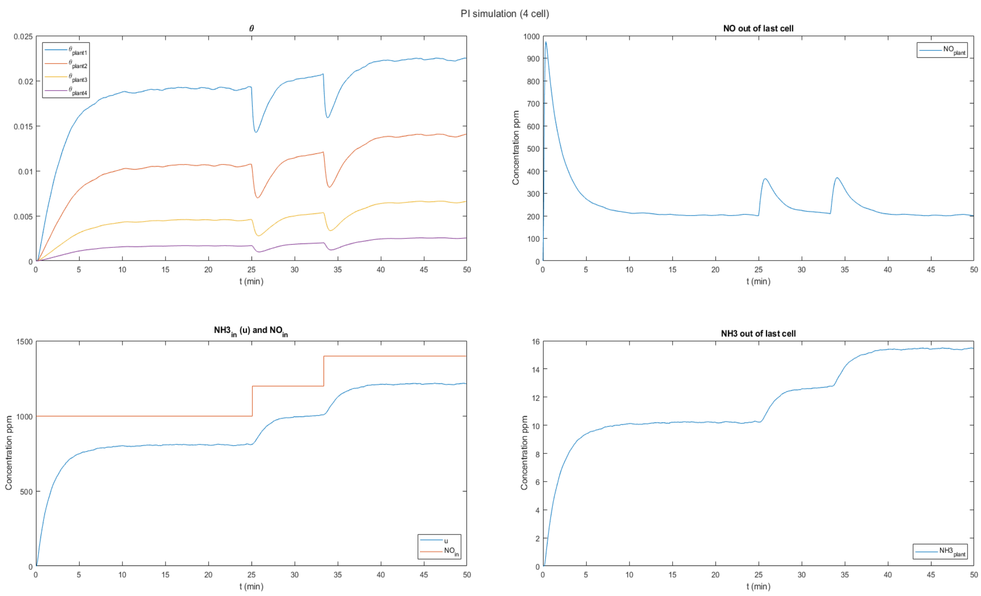

Figure 7.8: PI-control.

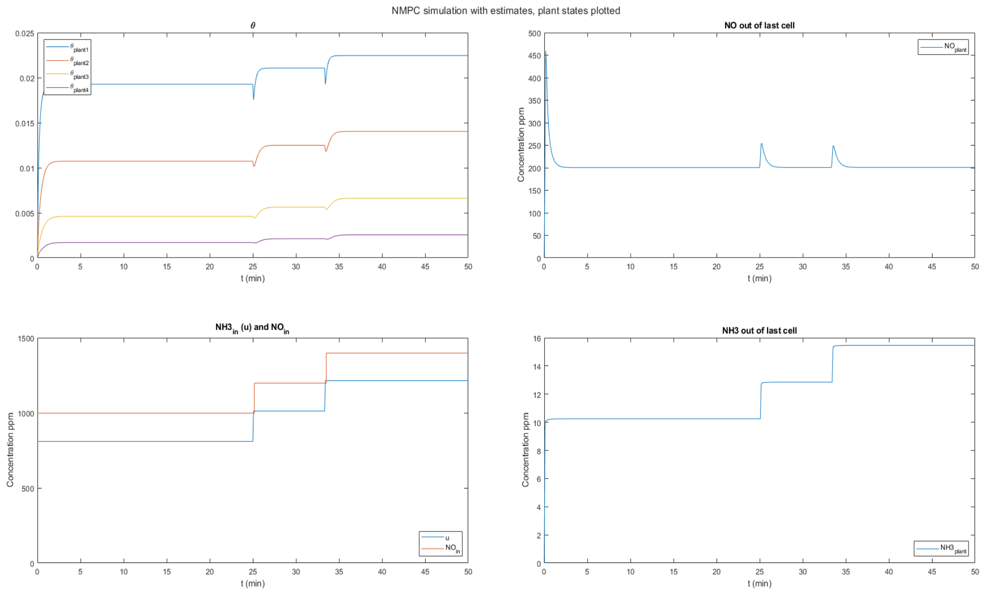

Figure 7.9: NMPC hard constraint on NH3 at 20ppm.

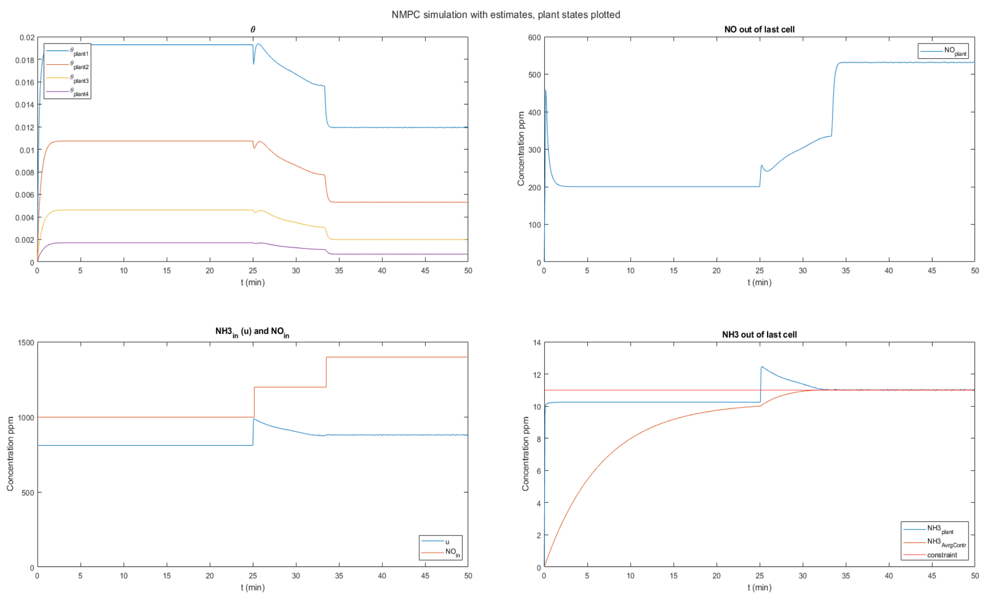

Figure 7.10: NMPC hard constraint on NH3 at 11ppm.

In the simulations presented in Figures 7.8, 7.9 and 7.10, one would like to keep the NO below 200ppm and NH_3 below 11ppm. In Figures 7.8 and 7.9, one can see that the NO limit hold but the NH_3 limit is exceeded. In PI-control, it is impossible to control the NH_3, since there is no measurement of the state. With NMPC it is possible, since the estimate of ammonia is obtained, and one can put a constraint at 11ppm. In Figure 7.10, can be seen that the NH_3 limit hold, but the NO limit is exceeded. The controller works against the long-term average value of NH_3, which is also plotted. One can see that the NH_3 value rise above the constraint, but quickly converge to the constraint. This does not result in infeasibility, since the controller does not know about it. In real applications, one would hold the NH_3 limit instead, since it is much more toxic than NO.

In some applications, one would like to control low NO_{out} levels. In the next simulation, the NO_{out} setpoint is set to 10ppm *i.e.* $r_{sp} = 0.00001$. The number of iterations is increased to $mpciterations = 2160$. In this simulation, cross sensitivity in the NO_{out} measurement is present, and it is increased to 0.8 from 0.5, which means that we have some NH_3 in the NO measurement as well. It is again calculated with the formula (7.1).

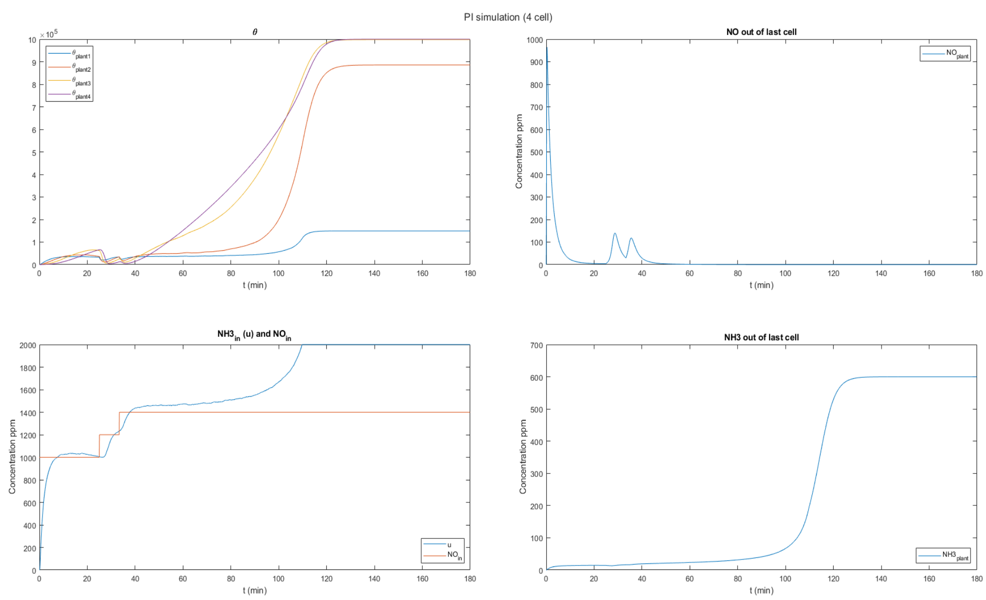

Figure 7.11: PI-control, cross sensitivity in the NO_{out} measurement, note the alarming ammonia slip.

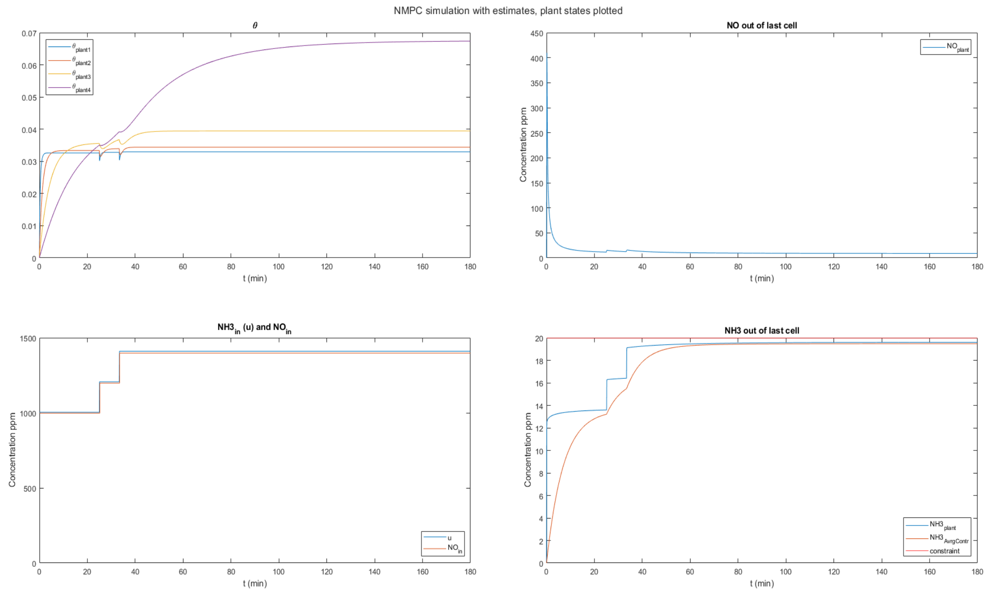

Figure 7.12: NMPC, cross sensitivity in the NO_{out} measurement.

The cross-sensitivity in the measurement causes problems for the PI-controller. When NO concentrations rise, the controller injects more NH_3 to the SCR, which results in a higher NO_{out} measurement, since NH_3 is also present in the measurement. This results in a maximum injection of ammonia and a very high ammonia slip. This is alarming, since high loads of toxics are released. The PI-controller does not know anything about it, since NH_3 is not measured, and the PI-controller can only control the SCR using the measurements that are available.

The NMPC can control multiple outputs and, since there exists an estimate of NH_3, it can be controlled. The cross-sensitivity in the measurement does not affect the estimate of NH_3 noticeably, and hence, a constraint can be set on the NH_3 concentration. The constraint is set at 20ppm and this constraint is satisfied at all times. This is a major advantage with NMPC compared to PI-control. The NMPC does not cause major ammonia slip by blindly injecting ammonia excessively.

7.6 Conclusion

The simulation study shows that it is possible to control the SCR process with NMPC with the assumption that the SCR behave in the same way as the full SCR model. For the control, state estimation is required and the nonlinear MHE provides accurate estimates of the states that cannot be measured. Setpoint tracking is not strictly necessary for the control, but it makes the control much faster, and the controller reach steady state faster, which is why it is used. The major advantage with NMPC is the ability to deal with constraints, which is of critical importance for SCR control governed by emission regulations. NMPC also manages multiple inputs and outputs, which is an advantage in SCR control, since it is possible to control both NO_{out} and $NH_{3_{out}}$. The results can be seen in Figures 7.6, 7.10 and 7.12.

The NMPC control combined with estimation and setpoint tracking is much slower than the PI-control. The PI-control simulation time for in Figure 7.11 was 2 seconds while the simulation time for the simulation in Figure 7.12 was 330 seconds. The PI-controller determines the control signal much faster than the NMPC.

SCR is a process with slow dynamics, which is why the NMPC can manage the control in a real-time application. The time required to calculate one control decision for one sample time varied depending on the NO_{in}, the optimization horizon and estimation horizon, but on average, it was kept around 0.5 seconds on a standard computer. The time to calculate the control decision could be improved by lowering the optimization horizon and estimation horizon and for a physical application, the optimal horizons should be studied.

For a hardware implementation there are still many open questions that should be studied in detail.

1. A study on the effects of measurement error in the NO_{in} measurement is required. The NO_{in} measurement is present in the control, estimation and setpoint tracking, and it plays a critically important role in the control

design. Measurement errors probably affect the control drastically and hence accurate estimation of the NO_{in} measurement might be required.

2. The ammonia injection reaches the catalyst with a time delay. This phenomenon is not captured in the simulations, but it should be studied before a commercial implementation is done.

3. Feasibility problems in the NMPC, MHE and setpoint tracking should also be studied more. Feasibility problems arise *e.g.* when the control is near the constraints. When one of the problems become infeasible, the control is affected and might become infeasible as well.

4. Model mismatch is a central problem in MPC, since MPC requires an accurate model. The model plays a crucial role in MPC, MHE and setpoint tracking and if the model is inaccurate the control actions become inaccurate. Model accuracy should be studied in detail with experiments.

5. The computation time in the simulator could be improved.

Chapter 8

Swedish summary

Olinjär modellprediktiv reglering och estimering tillämpat på selektiv katalytisk reduktion

Modellprediktiv reglering (eng. model predictive control MPC) är en avancerad reglermetod för system som går att reglera. Metoden är baserad på optimering vilket möjliggör att man beaktar fysikaliska begränsningar på tillstånd eller styrsignal. Begränsningarna är den stora fördelen med MPC jämfört med andra reglermetoder. Metoden för olinjär modellprediktiv reglering (NMPC) är mycket lik MPC, skillnaden är att i NMPC är modellen olinjär [1] och optimeringsproblemet kan bli icke-konvext.

I MPC förutses ett systems framtida beteende och det optimeras med avseende på styrsignalen, vilket kräver en modell av systemet. I avhandlingen beaktas system av formen

$$x(k+1) = f(x(k),u(k)),$$

där $x(k)$ är systemets tillstånd vid tidpunkt k och $u(k)$ är styrsignalen vid tidpunkt k. Vid varje tidpunkt k optimeras alltså en följd av styrsignaler $u_p(0), \ldots,$ $u_p(N-1)$ för $N \geq 2$, där horisonten N är antalet diskreta tidssteg i framtiden. Med optimeringen minimeras en kostnadsfunktion så att gränser för tillstånd eller styrsignal tas i beaktande. Den första optimala styrsignalen $u_p^*(0)$ används sedan för nästa tidssteg $k+1$, tillsammans med de nästa tillstånden $x(k+1)$ och algoritmen omstartas.

Estimering används tillsammans med regulatorn eftersom alla tillstånd $x(k)$ inte kan mätas i modellen. Tillstånden är viktiga för prediktionsmodellen i MPC. Det finns olika sätt att estimera tillstånd och Kalmanfiltret är den vanligaste estimatorn för linjära modeller och det utvidgade Kalmanfiltret används vanligen för olinjära modeller. Det utvidgade kalmanfiltret använder linjärisering, vilket enligt [2] inte är optimalt. För olinjära modeller föreslår de en estimator som utnyttjar ett fixt antal mätningar $y(T-N), \ldots, y(T)$, som uppdateras då man rör sig framåt i tiden. På basen av mätningarna optimeras en kostnadsfunktion för att få optimala tillstånds estimat. På engelska heter metoden Moving Horizon Estimation (MHE), vilket beskriver metoden bra. I [2] förespråkas dessutom MHE kombinerat med MPC, vilket används i denna avhandling.

Utsläppsgränserna blir kontinuerligt strängare och nya metoder för att reglera utsläpp mera effektivt behöver snabbt utvecklas. Selektiv Katalytisk Reduktion (SCR) är en kemisk process där kväveoxider (NOx) reduceras till kväve och vatten genom insprutning av ett reduktionsmedel som oftast är en urea-lösning. SCR används i gas- och dieselmotorer. Nästan alla nya bilar som drivs med diesel använder SCR processen för att minska på kväveoxidutsläppen.

SCR modellen som används i denna avhandling är utvecklad av Milver Colmenares i hans diplomarbete [11]. Modellen består av tre olinjära ordinära differentialekvationer, där (6.1) beskriver dynamiken för täckningsgraden θ av ammoniak i katalysatorn, (6.2) beskriver koncentrationen NO och (6.3) beskriver koncentrationen NH_3. SCR-processen går också att modellera med partiella differentialekvationer, men de är svåra att jobba med. För att undvika detta är SCR-processen modellerad med flera seriekopplade celler där utsignalen från tidigare cell används som insignal för nästa cell. Modellen som Colmenares har utvecklat beskriver en cell.

En simulator har utvecklats i denna avhandling för att undersöka och effektivera regleringen av SCR-processen. NMPC algoritmen som används i simulatorn är en vidare utvecklad version av Grünes och Panneks NMPC rutin [18]. En modell med fyra celler är använd i simuleringarna och det visar sig att Colmenares modell med flera celler är mycket tung att optimera, eftersom den har 12 tillstånd. En förenklad modell bestäms genom att approximera ekvationerna (6.2) och (6.3) i deras jämviktstillstånd, det vill säga derivatorna sätts lika med noll.

Den förenklade modellen används både som prediktionsmodell för NMPC, för beräknadet av börvärden för NMPC kostnaden samt som estimeringsmodell för MHE. Den förenklade modellen saknar tillstånd för NO koncentrationen och NH_3 koncentrationen och därför sätts begräsningarna i NMPC på långtidsmedelvärden för respektive koncentration som är beräknade med formel (6.7). Långtidsmedelvärdena ger också lite tid åt regulatorn att agera samt påminner långtidsmedelvärdena de verkliga miljöbegränsningarna. Den förenklade prediktionsmodellen har endast sex tillstånd. Prediktionsmodellen jämförs med simulering mot Colmenares modell och det kan ses att prediktionsmodellens beteende följer Colmenares modell bra, se figur 7.1. Idén med en förenklad modell är att minska på beräkningstiden och göra optimeringen mera effektiv både i NMPC och MHE. I simuleringarna används den förenklade modellen som prediktionsmodell samt estimeringsmodell och Colmenares mer komplicerade modell används för att simulera den riktiga processen.

Colmenares modell är formulerad i kontinuerlig tid och för att göra optimeringen ännu snabbare så transformeras den förenklade modellen till diskret tid med Eulers metod. NMPC algoritmen i Matlab har färdighet för modeller både i kontinuerlig och diskret tid, men optimeringen blir betydligt snabbare då man använder en färdigt diskretiserad modell.

Simuleringarna visar att SCR-processen kan styras effektivt med olinjär modellprediktiv reglering. Regulatorn klarar av att hålla miljögränserna och dessutom styra insprutningen av ammoniak på ett effektivt sätt. Att kunna garantera att miljögränsen för ammoniak hålls är den stora fördelen med NMPC vid styrning av SCR. En PI-regulator klarar inte av att hålla ammoniakgränsen då systemet får ett högt inflöde av NO. Ammoniak är ett större miljöproblem än NO och därför vill man inte överskrida ammoniakgränsen. Estimatorn lyckas noggrant estimera tillstånden som inte enkelt kan beräknas med endast mätning av in- och utflöde NO, samt med kännedom av styrsignalen.

Ett problem med NMPC kombinerat med MHE är lång beräkningstid. Vid varje iteration körs två optimeringar, då systemet får en störning blir beräkningstiden längre, eftersom NMPC:n måste vidta stränga åtgärder för att hålla sig inom begränsningarna. Inga riktiga experiment har gjorts i denna avhandling men riktiga experiment med fysisk hårdvara skulle vara nästa steg för detta arbete.

Appendix A

Matlab code

A.1 Prediction model validation

The simplified prediction model is compared in both continuous-time and discrete-time to the full SCR model with the code below.

```
%Prediction model validation Simulator
%OSCAR AALTONEN
%15.7.2022
clear all
t = 900;
%for continuous models
tMeasure = [0 t];

%for discrete model
T = 5;  %sample time
%Get same time as for the continuous models
evals = t/T;
xaxis = [0:T:t-1];

%evals = t;
%xaxis = [1:t];

```

```
%The original continuous model
options = odeset('RelTol',1e-5,'Stats','on');
[t1,y] = ode15s(@systemM,tMeasure,[0 0 0 0 0 0 0 0 0 0 0 ...
    0],options);

yO1 = y(:,1);        %theta_1
yO2 = y(:,2);        %NO_1
yO3 = y(:,3);        %NH3_1
yO4 = y(:,4);        %theta_2
yO5= y(:,5);         %NO_2
yO6 = y(:,6);        %NH3_2
yO7 = y(:,7);        %theta_3
yO8 = y(:,8);        %NO_3
yO9 = y(:,9);        %NH3_3
yO10 = y(:,10);      %theta_4
yO11 = y(:,11);      %NO_4
yO12 = y(:,12);      %NH3_4



%The steady state continuous time Model
options = odeset('Stats','on');
[t2,y] = ode15s(@systemSSc,tMeasure,[0 0 0 0 0 0],options);

ySSc1 = y(:,1);      %theta1
ySSc2 = y(:,2);      %theta2
ySSc3 = y(:,3);      %theta3
ySSc4 = y(:,4);      %theta4
ySSc5 = y(:,5);      %NO_avr 4th cell
ySSc6 = y(:,6);      %NH3_avr 4th cell


%The Discrete time steady state model
ySSd1 = zeros(evals,1);
ySSd2 = zeros(evals,1);
ySSd3 = zeros(evals,1);
```

```
    ySSd4 = zeros(evals,1);
    ySSd5 = zeros(evals,1);
    ySSd6 = zeros(evals,1);
    aSSd = zeros(evals,1);
    bSSd = zeros(evals,1);

    ySSd1(1)=0;
    ySSd2(1)=0;
    ySSd3(1)=0;
    ySSd4(1)=0;
    ySSd5(1)=0;
    ySSd6(1)=0;

    y = [0 0 0 0 0 0];
    for i = 2:evals
        [y,a,b] = dtSystemSS(0, y, T);
        ySSd1(i) = y(1);
        ySSd2(i) = y(2);
        ySSd3(i) = y(3);
        ySSd4(i) = y(4);
        ySSd5(i) = y(5);
        ySSd6(i) = y(6);
        aSSd(i) = a;
        bSSd(i) = b;
    end
    k = tiledlayout(2,2);
    title(k,'Simplified model compared to Colmenares model')

    nexttile
    plot(t1/60,yO1)
    hold on
    plot(t2/60,ySSc1,'--')
    hold on
    plot(xaxis/60, ySSd1,'--')
    hold on
    plot(t1/60,yO4)
```

```
    hold on
    plot(t2/60,ySSc2,'--')
    hold on
    plot(xaxis/60, ySSd2,'--')
    hold on
    plot(t1/60,yO7)
    hold on
    plot(t2/60,ySSc3,'--')
    hold on
    plot(xaxis/60, ySSd3,'--')
    hold on
    plot(t1/60,yO10)
    hold on
    plot(t2/60,ySSc4,'--')
    hold on
    plot(xaxis/60, ySSd4,'--')
    legend('\theta_{1}','\theta_{Sc1}','\theta_{Sd1}','\theta_{2}', ...
        ...
        '\theta_{Sc2}','\theta_{Sd2}','\theta_{3}','\theta_{Sc3}', ...
            ...
        '\theta_{Sd3}','\theta_{4}','\theta_{Sc4}','\theta_{Sd4}', ...
            ...
        'Location', 'southeast')
    xlabel('time (min)')
    title('\theta')
    hold off

    nexttile
    plot(t1/60,yO11*1000000)
    hold on
    plot(t2/60,ySSc5*1000000)
    hold on
    plot(xaxis/60,ySSd5*1000000)
    legend('NO','NO_{cAvrg}','NO_{dAvrg}','Location', ...
        'northeast')
    xlabel('time (min)')
```

```
    ylabel('Concentration ppm')
    title('NO last cell')
    hold off

    nexttile
    plot(t1/60,yO12*1000000)
    hold on
    plot(t2/60,ySSc6*1000000)
    hold on
    plot(xaxis/60,ySSd6*1000000)
    legend('NH3','NH3_{cAvrg}','NH3_{dAvrg}','Location', ...
        'northeast')
    xlabel('time (min)')
    ylabel('Concentration ppm')
    title('NH_3 last cell')
    hold off

    nexttile
    yline(10^-3*1000000, 'r')
    hold on
    yline(0.8*10^-3*1000000, 'b')
    ylim([0.5*10^-3 1.2*10^-3]*1000000)
    legend( 'NO_{in}','NH3_{in}','Location', 'northeast')
    xlabel('time (min)')
    ylabel('Concentration ppm')
    title('NH3_{in} (u) and NO_{in}')
    hold off


function dx = systemM(t, x) %Milver Model

    kads = 10;
    kred = 300;
    kdes = 0.0;
    catmax = 0.1;
    v = 2/4;
```

```
    NO_in = 10^(-3);
    u = 0.8*10^(-3);
    dx = zeros(4*3,1);

    dx(1) = kads*x(3)*(1-x(1))-(kdes+kred*x(2))*x(1);
    dx(2) = v*(NO_in-x(2))-kred*x(2)*x(1)*catmax;
    dx(3) = ...
        v*(u(1)-x(3))-kads*x(3)*(1-x(1))*catmax+kdes*x(1)*catmax;

    for i = 4:3:4*3
        dx(i) = kads*x(i+2)*(1-x(i))-(kdes+kred*x(i+1))*x(i);
        dx(i+1) = v*(x(i-2)-x(i+1))-kred*x(i+1)*x(i)*catmax;
        dx(i+2) = ...
            v*(x(i-1)-x(i+2))-kads*x(i+2)*(1-x(i))*catmax...
             +kdes*x(i)*catmax;
    end

end


function dxdt = systemSSc(t,x)        %SS continuous model


    kads = 10;
    kred = 300;
    kdes = 0.0;
    catmax = 0.1;
    v = 2/4;
    NO_in = 10^(-3);
    u = 0.8*10^(-3);

    dxdt = zeros(6,1);

    b = NO_in/(1+kred*x(1)*catmax/v);
    a = (u(1)+kdes*x(1)*catmax/v)/(1+kads*(1-x(1))*catmax/v);
    dxdt(1) = (kads*a*(1-x(1))-(kdes+kred*b)*x(1));
```

```

    for i = 2:4
        b = b/(1+kred*x(i)*catmax/v);
        a = (a+kdes*x(i)*catmax/v)/(1+kads*(1-x(i))*catmax/v);
        dxdt(i) = (kads*a*(1-x(i))-(kdes+kred*b)*x(i));
    end

    dxdt(5) = -0.0025*x(5)+0.0025*b;
    dxdt(6) = -0.0025*x(6)+0.0025*a;

end


function [y, a, b] = dtSystemSS(t, x, T)  %Simplified ...
    discrete model
    kads = 10;
    kred = 300;
    kdes = 0.0;
    catmax = 0.1;
    v = 2/4;
    NO_in = 10^(-3);
    u = 0.8*10^(-3);
    avrg = 0.0025;

    y = zeros(1,4+4);
    b = NO_in/(1+kred*x(1)*catmax/v);
    a = (u(1)+kdes*x(1)*catmax/v)/(1+kads*(1-x(1))*catmax/v);
    y(1) = x(1) + T*(kads*a*(1-x(1))-(kdes+kred*b)*x(1));

    for i = 2:4
        b = b/(1+kred*x(i)*catmax/v);
        a = (a+kdes*x(i)*catmax/v)/(1+kads*(1-x(i))*catmax/v);
        y(i) = x(i) + T*(kads*a*(1-x(i))-(kdes+kred*b)*x(i));
    end

    y(5) = x(5)+T*(-avrg*x(5)+avrg*b);
```

```
    y(6) = x(6)+T*(-avrg*x(6)+avrg*a);
end
```

A.2 PI-controller

The Matlab code for the PI-controller is presented below.

```
%__PI-Control of the SCR__
%--OSCAR AALTONEN--
%15.7.2022

%Initate varables
x7m = 0;
eold = 0;
t0 = 0;
T = 5;
x0 = zeros(1,12);
u = 0;
x = [];
t= [];
u1 = [];
NO_in = [];
tic

iter = 180;
%iter = 600;
%iter = 2160;

%Iterate PI-controller
for i = 1:iter
   [¬, NO_in_new] = plant(t0, zeros(1,12),0,0);

   NO_in = [NO_in, NO_in_new];
   x = [ x; x0 ];
```

```
    t = [ t; t0 ];
    u1 = [u1 ; u];

    [u,x7m,eold]=PI(x7m,eold,x0,T,u);
    x0 = dynamicPlant(@plant,T,t0,x0,u,1e-12,1e-12);
    t0 = t0+T;

end
toc
%Plots
     k = tiledlayout(2,2);
     title(k,'PI simulation (4 cell)')
     nexttile
     plot(t/60,x(:,1)*1000000)
     hold on
     plot(t/60,x(:,4)*1000000)
     hold on
     plot(t/60,x(:,7)*1000000)
     hold on
     plot(t/60,x(:,10)*1000000)
     legend('\theta_{plant1}','\theta_{plant2}','\theta_{plant3}',...
         '\theta_{plant4}', 'Location', 'northwest')
     xlabel('t (min)')
     title('\theta')

     nexttile
     plot(t/60,x(:,11)*1000000)
     legend('NO_{plant}','Location', 'northeast')
     xlabel('t (min)')
     ylabel('Concentration ppm')
     title('NO out of last cell')

     nexttile
     plot(t/60,u1*1000000)
     hold on
     plot(t/60,NO_in*1000000)
```

```
    legend('u','NO_{in}','Location', 'southeast')
    xlabel('t (min)')
    ylabel('Concentration ppm')
    title('NH3_{in} (u) and NO_{in}')


    nexttile
    plot(t/60,x(:,12)*1000000)
    hold on
    legend('NH3_{plant}','Location', 'southeast')
    xlabel('t (min)')
    ylabel('Concentration ppm')
    title('NH3 out of last cell')

%Function for the PI-controller
function [u,x7m,em] = PI(x7m,eold,x0,T,u)
Kc=-0.9;
    Ti=1.9*60;
    a= 0.01;
    NOmeas = x0(11)+0.8*x0(12)+(-1*10^-5 + ...
        (1*10^-5+1*10^-5).*randn(1,1));
    x7m = T*a*NOmeas+(1-T*a)*x7m;
    em = 1.9e-4-x7m;
    %em = 1e-5-x7m;

    u=u+Kc*(em-eold+1/Ti*T*em);
    u=max(u,0);
    u=min(u,2e-3);
end

%Function to iterate the Plant
function [x, t_intermediate, x_intermediate] = ...
    dynamicPlant(system, T,...
    t0, x0, u, atol_ode, rtol_ode)
        options = odeset('AbsTol', atol_ode, 'RelTol', rtol_ode);
        [t_intermediate,x_intermediate] = ode45(system, ...
```

```
                [t0, t0+T], x0, options, u);
            x = x_intermediate(size(x_intermediate,1),:);

end
%Model
function [dx, NO_in] = plant(t,x, u,T)
    NO_in = 1e-3;
    kads = 10;
    kred = 300;
    kdes = 0.0;
    catmax = 0.1;
    v = 2/4;
    dx = zeros(12,1);
    if t>1500
    NO_in=1.2e-3;
    end

    if t>2000
    NO_in=1.4e-3;
    end
    dx(1) = kads*x(3)*(1-x(1))-(kdes+kred*x(2))*x(1); ...
                %cell 1
    dx(2) = v*(NO_in-x(2))-kred*x(2)*x(1)*catmax;
    dx(3) = ...
        v*(u(1)-x(3))-kads*x(3)*(1-x(1))*catmax+kdes*x(1)*catmax;

    dx(4) = kads*x(6)*(1-x(4))-(kdes+kred*x(5))*x(4); ...
                %cell 2
    dx(5) = v*(x(2)-x(5))-kred*x(5)*x(4)*catmax;
    dx(6) = ...
        v*(x(3)-x(6))-kads*x(6)*(1-x(4))*catmax+kdes*x(4)*catmax;

    dx(7) = kads*x(9)*(1-x(7))-(kdes+kred*x(8))*x(7); ...
                 %cell 3
    dx(8) = v*(x(5)-x(8))-kred*x(8)*x(7)*catmax;
    dx(9) = ...
```

```
        v*(x(6)-x(9))-kads*x(9)*(1-x(7))*catmax+kdes*x(7)*catmax;

    dx(10)  = kads*x(12)*(1-x(10))-(kdes+kred*x(11))*x(10); ...
            %cell 4
    dx(11)  = v*(x(8)-x(11))-kred*x(11)*x(10)*catmax;
    dx(12)  = ...
        v*(x(9)-x(12))-kads*x(12)*(1-x(10))*catmax+kdes*x(10)*catmax;

end
```

A.3 Main file for the SCR control simulator

This is the main file where the input is set for all the different components. The program starts from this file. The files A.3, A.4, A.5 and A.6 are required for the NMPC control simulation.

```
%Input and output for the NMPC, NMHE and Nonlinear Target ...
    problem
%--OSCAR AALTONEN
%14.7.2022

%--------------

t_Start = tic;
tic
[t, xPlant, u ,xContr, xhat, allU,NO_in, mpciterations, ...
    N, scrCells,...
     nMHE] = input();
toc
t_Elapsed = toc( t_Start );

%Plots
figure(1)
k = tiledlayout(2,2);
```

```
title(k,'NMPC simulation')

%plot of theta
nexttile
for i = 1:3:scrCells*3
plot(t/60,xPlant(:,i))
hold on
end
if nMHE<0
    for i = 1:3:scrCells*3
        plot(t/60,xhat(:,i),'--')
    end
end
ax = gca;
ax.YAxis.Exponent = 0;
legend('\theta_{plant1}','\theta_{plant2}','\theta_{plant3}',...
    '\theta_{plant4}','\theta_{est1}','\theta_{est2}',...
    '\theta_{est3}','\theta_{est4}', 'Location', 'northeast')
xlabel('t (min)')
title('\theta')

%plot of NO
nexttile
plot(t/60,xPlant(:,scrCells*3-1)*1000000)
hold on
if nMHE<0
        plot(t/60,xhat(:,11)*1000000,'--r')
end
ylim([0 1000])
legend('NO_{plant}','NO_{est}','NOcontr','Location', ...
    'northeast')
xlabel('t (min)')
ylabel('Concentration ppm')
title('NO out of last cell')

%plot of NH3in and NOin
```

```
    NH3_in = u(1:2:end);
    nexttile
    plot(t/60,NH3_in*1000000)
    hold on
    stairs(t/60,NO_in*1000000)
    hold on
    ylim([0 1.5*10^-3*1000000])
    legend('u','NO_{in}','u_{exact}','Location', 'southeast')
    xlabel('t (min)')
    ylabel('Concentration ppm')
    title('NH3_{in} (u) and NO_{in}')

    %plot of NH3out
    nexttile
    plot(t/60,xPlant(:,scrCells*3)*1000000)
    hold on
    plot(t/60,xContr(:,end-2)*1000000);
    hold on
    if nMHE<0
            plot(t/60,xhat(:,12)*1000000,'--r')
    end
    legend('NH3_{plant}','NH3_{AvrgContr}','NH_{3est}',...
        'constraint','Location', 'southeast')
    xlabel('t (min)')
    ylabel('Concentration ppm')
    title('NH3 out of last cell')


    %Input for the nmpc algorithm
    function [t, x, u, x1, xhat, allU, NO_in, mpciterations,...
        N, scrCells, nMHE] = input()

    %SCRmodel
    scrCells = 4;  %optimization work only for scrCells=4

    %steadyStateTarget
```

```
    ysp = 0;         %Soft setpoint
    rsp = 0.000200; %Hard setpoint for NOx
    usp = 0;         %Soft setpoint for control variable

    %NMPC with estimates, choose nMHE > 0
    %NMPC without estimation, choose nMHE=0
    %NMPC without estimation, but estimation done separately ...
        choose nMHE<0
    nMHE          = 13;
    ammonia = 0.5; %Ammonia cross sensitivity
    noise = 1;     %1 for noise 0 for no noise

    %NMPC
    mpciterations = 180;        %mpc iterations
    N             = 40;         %prediction horizon length
    uN            = N;          %control variable horizon length
    T             = 5;          %sampling interval
    tmeasure      = 0.0;
    xmeasure      = [0 0 0 0 0 0 0 0 0 0 0 0]; %initial ...
        values for states
    u0            = zeros(2,N);      %initial guess for the ...
        control value
    type ='difference equation';
    tol_opt       = 1e-14;
    opt_option    = 0;
    iprint        = 10;
    atol_ode_real = 1e-12;
    rtol_ode_real = 1e-12;
    atol_ode_sim  = 1e-4;
    rtol_ode_sim  = 1e-4;

    [t, x, u, x1, xhat, allU, NO_in] = ...
        nmpc4cellFinal(@runningcosts,...
         @terminalcosts, @constraints, @terminalconstraints,...
         @linearconstraints, @predMod, @plant, mpciterations, ...
            N, T,...
```

```
        tmeasure, xmeasure, u0, nMHE, uN, ...
            scrCells,ysp,rsp,usp, ammonia,...
        noise, tol_opt, opt_option, type, atol_ode_real,...
        rtol_ode_real, atol_ode_sim, rtol_ode_sim, ...
         iprint,@printHeader, @printClosedloopData);

end

%cost and constraints for nmpc

%stage cost
function cost = runningcosts(t, x, u, ref)
    cost = (u(1)-ref(7))^2+u(2)*10^10; %Cost with reference
    %cost = u(1)^2+u(2)*10^10;          %Cost with 0 reference
end

%terminal cost
function cost = terminalcosts(t, x)
    cost = 0.0;
end

%constraints for the states
function [c,ceq] = constraints(t, x, u)
    c = [];
    c(1) = x(5)-(6*10^-4)-u(2);
    c(2) = x(6)-(1.1*10^-5);
    ceq = [];
end

%terminal constraints for the states
function [c,ceq] = terminalconstraints(t, x)
    c =[];
    ceq = [];
end

%constraints for the control variable
```

```
function [A, b, Aeq, beq, lb, ub] = linearconstraints(t, x, u)
    A   = [];
    b   = [];
    Aeq = [];
    beq = [];
    lb  = [0 0];
    ub  = [0.002 inf];
end


%Discrete time prediction model
function [xPlus, avrg, NO_in]= predMod(t, x, u, T, scrCells)
    %NO in values for different time
    NO_in=1e-3;

    if t>1500
    NO_in=1.2e-3;
    end

    if t>2000
    NO_in=1.4e-3;
    end

    if t>2500
    NO_in=1.1e-3;
    end

    if t>3000
    NO_in=0.9e-3;
    end

    if t>3200
    NO_in=0.6e-3;
    end

    if t>3500
```

```
    NO_in=1e-3;
    end
%}

    %model parameters
    kads = 10;
    kred = 300;
    kdes = 0.0;
    catmax = 0.1;
    v = 2/4;
    avrg = 0.0025;
    xPlus = zeros(1,4+4);
    b = NO_in/(1+kred*x(1)*catmax/v);
    a = (u(1)+kdes*x(1)*catmax/v)/(1+kads*(1-x(1))*catmax/v);
    xPlus(1) = x(1) + T*(kads*a*(1-x(1))-(kdes+kred*b)*x(1));

    for i = 2:scrCells
        b = b/(1+kred*x(i)*catmax/v);
        a = (a+kdes*x(i)*catmax/v)/(1+kads*(1-x(i))*catmax/v);
        xPlus(i) = x(i) + T*(kads*a*(1-x(i))-(kdes+kred*b)*x(i));
    end

    xPlus(5) = x(5)+T*(-avrg*x(5)+avrg*b);
    xPlus(6) = x(6)+T*(-avrg*x(6)+avrg*a);

    xPlus(7) = b;
    xPlus(8) = a;

end

%Model of the plant (Colmenares model)
function dx = plant(t, x, u, scrCells)
    %NO in values for different time
    NO_in=1e-3;

    if t>1500
```

```
    NO_in=1.2e-3;
    end

    if t>2000
    NO_in=1.4e-3;
    end

    if t>2500
    NO_in=1.1e-3;
    end

    if t>3000
    NO_in=0.9e-3;
    end

    if t>3200
    NO_in=0.6e-3;
    end

    if t>3500
    NO_in=1e-3;
    end

    %model parameters
    kads = 10;
    kred = 300;
    kdes = 0.0;
    catmax = 0.1;
    v = 2/4;
    dx = zeros(4*3,1);

    dx(1) = kads*x(3)*(1-x(1))-(kdes+kred*x(2))*x(1);
    dx(2) = v*(NO_in-x(2))-kred*x(2)*x(1)*catmax;
    dx(3) = ...
        v*(u(1)-x(3))-kads*x(3)*(1-x(1))*catmax+kdes*x(1)*catmax;

```

```
    for i = 4:3:scrCells*3
        dx(i) = kads*x(i+2)*(1-x(i))-(kdes+kred*x(i+1))*x(i);
        dx(i+1) = v*(x(i-2)-x(i+1))-kred*x(i+1)*x(i)*catmax;
        dx(i+2) = ...
            v*(x(i-1)-x(i+2))-kads*x(i+2)*(1-x(i))*catmax...
            +kdes*x(i)*catmax;
    end

end

function printHeader()
    fprintf(['   k   |      u(k)        x(1)         x(2)         ...
        x(3)' ...
        '         x(4)          x(5)        x(6)         x(7)  ...
                 x(8)' ...
        '        x(9)         x(10)        x(11)        x(12)  ...
                Time\n']);
    fprintf('---------------------------------------------------\n');
end

function printClosedloopData(mpciter, u, x, t_Elapsed,scrCells)
    fprintf([' %3d  | %+11.6f %+11.6f %+11.6f %+11.6f %+11.6f ...
        %+11.6f' ...
        ' %+11.6f %+11.6f %+11.6f %+11.6f %+11.6f %+11.6f ...
            %+11.6f' ...
        ' %+11.6f'] , mpciter, ...
            u(1,1), x(1), x(2),x(3),x(4), x(5), x(6), x(7), ...
                x(8) ...
            ,x(9),x(10), x(11), x(12), t_Elapsed);
end
```

A.4 Nonlinear MHE

The code below is used for the nonlinear MHE.

```
%Nonlinear Moving Horizon estimation
%--OSCAR AALTONEN--
%14.7.2022

function x = MHE4cellFinal(T,N,Y,u,xInit,NO_in)
 %Input for fmincon
   A = [];
   b = [];
   Aeq = [];
   beq = [];
   lb = [];
   ub = [];
   nonlcon = [];
   options = optimoptions('fmincon','display','none', ...
       'MaxFunctionEvaluations',100000000,...
       'Algorithm','interior-point');
   x0 = xInit;

   %MHE optimization problem
   x = fmincon(@(xhat) costfunctionSimpl(xhat,T,N,Y,u,NO_in), ...
       x0, A, b,Aeq,beq,lb,ub,nonlcon,options);


end


%costfunction for MHE
function obj = costfunctionSimpl(xhat,T,N,Y,u,NO_in)
   sum1 = 0;
   sum2 = 0;

   %sum ||xhat-f(xhat,u)||^2
   for i = 1:N-1
       fxhat = fs([xhat(1,i) xhat(4,i) xhat(7,i) ...
           xhat(10,i)],u(i),T, ...
```

```
           NO_in(i));
        w = [1 1 1 1];
        W = diag(w.^2);
        xhat_k = [xhat(1,i+1) xhat(4,i+1) xhat(7,i+1) ...
           xhat(10,i+1)];
        sum1 = sum1 + (xhat_k-fxhat)*W*(xhat_k-fxhat)';
    end
    %sum ||y-h(xhat)||^2
    for i = 1:N
        st = xhat(:,i);
        h_xhat = h(st);
        w = [1 1 1 1 1 1 1 1];
        conc = concentrationsXhat([xhat(1,i) xhat(4,i) ...
           xhat(7,i) ...
           xhat(10,i)],u(i),NO_in(i));
        y = [conc(1) conc(2) conc(3) conc(4) conc(5) conc(6) ...
           Y(i,:) ...
           conc(8)];
        W = diag(w.^2);
        sum2 = sum2 + (y-h_xhat)*W*(y-h_xhat)';
    end

    obj = sum1+sum2;
end

function hxhat = h(xhat)
    hxhat = [xhat(2) xhat(3) xhat(5) xhat(6) xhat(8) xhat(9) ...
       xhat(11) ...
       xhat(12)];
end



%Simplified model in discrete time
function xPlus= fs(x, u, T, NO_in)

```

```
    kads = 10;
    kred = 300;
    kdes = 0.0;
    catmax = 0.1;
    v = 2/4;
    xPlus = zeros(1,4);
    b = NO_in/(1+kred*x(1)*catmax/v);
    a = (u(1)+kdes*x(1)*catmax/v)/(1+kads*(1-x(1))*catmax/v);
    xPlus(1) = x(1) + T*(kads*a*(1-x(1))-(kdes+kred*b)*x(1));

    for i = 2:4
        b = b/(1+kred*x(i)*catmax/v);
        a = (a+kdes*x(i)*catmax/v)/(1+kads*(1-x(i))*catmax/v);
        xPlus(i) = x(i) + T*(kads*a*(1-x(i))-(kdes+kred*b)*x(i));
    end


end
%model to obtain the steady state values of the concentrations
function concXhat = concentrationsXhat(xhat,u, NO_in)

    kads = 10;
    kred = 300;
    kdes = 0;
    catmax = 0.1;
    v = 2/4;

    b1 = NO_in/(1+kred*xhat(1)*catmax/v);
a1 = (u(1)+kdes*xhat(1)*catmax/v)/(1+kads*(1-xhat(1))*catmax/v);

    b2 = b1/(1+kred*xhat(2)*catmax/v);
    a2 = ...
        (a1+kdes*xhat(2)*catmax/v)/(1+kads*(1-xhat(2))*catmax/v);

    b3 = b2/(1+kred*xhat(3)*catmax/v);
    a3 = ...
```

```
        (a2+kdes*xhat(3)*catmax/v)/(1+kads*(1-xhat(3))*catmax/v);

    b4 = b3/(1+kred*xhat(4)*catmax/v);
    a4 = ...
        (a3+kdes*xhat(4)*catmax/v)/(1+kads*(1-xhat(4))*catmax/v);

    concXhat = [b1 a1 b2 a2 b3 a3 b4 a4];

end
```

A.5 Nonlinear setpoint tracking

This is the Matlab code for the nonlinear setpoint tracking problem.

```
%Nonlinear target problem
%--OSCAR AALTONEN
%14.7.2022

%The optimization problem
function ref = steadyStateTargetFinal(NO_in, T, ysp, rsp, usp)
 obj=@(x) cost(x,ysp,usp);
 x0 = zeros(1,7); %initial value
 A = [];
 b = [];
 Aeq = [];
 beq = [];
 lb = [];
 ub = [];
 nonlcon = @(x) noNlcon(x(1:6),x(7),NO_in,T,rsp);
 options = optimoptions('fmincon','display','none');
 ref=fmincon(obj,x0,A, b,Aeq,beq,lb,ub,nonlcon,options);


end
```

```
%costfunction
function obj = cost(x,ysp,usp)
    Rs=1;               %weight
    Qs=1;               %weight
    obj = ...
        1/2*((x(7)-usp)*Rs*(x(7)-usp)+(x(6)-ysp)*Qs*(x(6)-ysp));

end

%Simplified model
function [xPlus, avrg, NO_in]= predMod(x, u, T, NO_in)

    kads = 10;
    kred = 300;
    kdes = 0.0;
    catmax = 0.1;
    v = 2/4;
    avrg = 0.0025;
    xPlus = zeros(1,4+2);
    b = NO_in/(1+kred*x(1)*catmax/v);
    a = (u(1)+kdes*x(1)*catmax/v)/(1+kads*(1-x(1))*catmax/v);
    xPlus(1) = x(1) + T*(kads*a*(1-x(1))-(kdes+kred*b)*x(1));

    for i = 2:4
        b = b/(1+kred*x(i)*catmax/v);
        a = (a+kdes*x(i)*catmax/v)/(1+kads*(1-x(i))*catmax/v);
        xPlus(i) = x(i) + T*(kads*a*(1-x(i))-(kdes+kred*b)*x(i));
    end

    xPlus(5:6) = [b a];

end

%Steady state constraint
%The system should always satisfy the steady state
```

```
function [c, ceq] = noNlcon(x,u,NO_in,T,rsp)
    c = [];
    ceq(1:6) = x - predMod(x, u, T, NO_in);
    ceq(7) = x(5) - rsp;

end
```

A.6 NMPC algorithm

All the code above is done by me and the code beneath is obtained from [18], but some changes has been done to match the problem that is introduced in this thesis. The function measureInitialValue on row 397 to row 491 has been modified and developed further. New parameters has also been added to some functions to match the requirements for the SCR control.

```
1  %The NMPC algorithm
2  function [t, x, u, xContr, xhatS, allU, NO_in] = ...
       nmpc4cellFinal...
3      (runningcosts, terminalcosts, ...
4                constraints, terminalconstraints, ...
5                linearconstraints, model,plant, ...
6                mpciterations, N, T, tmeasure, xmeasure,...
7                u0,nMHE,uN,scrCells,ysp,rsp,usp,ammonia,noise, ...
8                varargin)
9
10 % Computes the closed loop solution for the NMPC problem ...
       defined by
11 % the functions
12 %   runningcosts:
13 % evaluates the running costs for state and control
14 % at one sampling instant.
15 % The function returns the running costs for one
16 % sampling instant.
```

```
% Usage: [cost] = runningcosts(t, x, u)
% with time t, state x and control u

% terminalcosts:
% evaluates the terminal costs for state at the end
% of the open loop horizon.
% The function returns value of the terminal costs.
% Usage: cost = terminalcosts(t, x)
% with time t and state x

% constraints:
% computes the value of the restrictions for a
% sampling instance provided the data t, x and u
% given by the optimization method.
% The function returns the value of the
% restrictions for a sampling instance separated
% for inequality restrictions c and equality
% restrictions ceq.
% Usage: [c,ceq] = constraints(t, x, u)
% with time t, state x and control u

% terminalconstraints:
% computes the value of the terminal restrictions
% provided the data t, x and u given by the
% optimization method.
% The function returns the value of the
% terminal restriction for inequality restrictions
% c and equality restrictions ceq.
% Usage: [c,ceq] = terminalconstraints(t, x)
% with time t and state x

% linearconstraints:
% sets the linear constraints of the discretized
% optimal control problem. This is particularly
% useful to set control and state bounds.
% The function returns the required matrices for
```

```
% the linear inequality and equality constraints A
% and Aeq, the corresponding right hand sides b and
% beq as well as the lower and upper bound of the control.
% Usage: [A, b, Aeq, beq, lb, ub] = linearconstraints(t, x, u)
% with time t, state x and control u

%  system:
% evaluates the difference equation describing the
% process given time t, state vector x and control u.
% The function returns the state vector x at the next time ...
    instant.
% Usage: [y] = system(t, x, u, T)
% with time t, state x, control u and sampling interval T
% for a given number of NMPC iteration steps (mpciterations). For
% the open loop problem, the horizon is defined by the number of
% time instances N and the sampling time T. Note that the dynamic
% can also be the solution of a differential equation. ...
    Moreover, the
% initial time tmeasure, the state measurement xmeasure and a ...
    guess of
% the optimal control u0 are required.

% Arguments:
% mpciterations: Number of MPC iterations to be performed
% N: Length of optimization horizon
% T: Sampling interval
% tmeasure: Time measurement of initial value
% xmeasure: State measurement of initial value
% u0: Initial guess of open loop control

% Optional arguments:
% iprint= 0 Print closed loop data(default)
% = 1 Print closed loop data and errors of the optimization ...
    method
% = 2 Print closed loop data and errors and warnings of the ...
    method
```

```
% ≥ 5 Print closed loop data and errors and warnings of
% the optimization method as well as graphical
% output of closed loop state trajectories
% ≥10 Print closed loop data and errors and warnings of
% the optimization method with error and warning description

% printHeader: Clarifying header for selective output of closed
% loop data, cf. printClosedloopData

% printClosedloopData: Selective output of closed loop data

% plotTrajectories:
% Graphical output of the trajectories, requires iprint ≥ 4
% tol_opt:          Tolerance of the optimization method
% opt_option: = 0: Active-set method used for optimization ...
    (default)
% = 1: Interior-point method used for optimization
% = 2: Trust-region reflective method used for optimization
% type: Type of dynamic, either difference equation or
% differential equation can be used
% atol_ode_real: Absolute tolerance of the ODE solver for the
% simulated process
% rtol_ode_real: Relative tolerance of the ODE solver for the
% simulated process
% atol_ode_sim: Absolute tolerance of the ODE solver for the
% simulated NMPC prediction
% rtol_ode_sim: Relative tolerance of the ODE solver for the
% simulated NMPC prediction

% Internal Functions:
% measureInitialValue: measures the new initial values for t0
% and x0 by adopting values computed by
% method applyControl.
% The function returns new initial state
% vector x0 at sampling instant t0.

```

```
% applyControl: applies the first control element of u to
% the simulated process for one sampling interval T.
% The function returns closed loop state
% vector xapplied at sampling instant tapplied.

% shiftHorizon: applies the shift method to the open loop
% control in order to ease the restart.
% The function returns a new initial guess
% u0 of the control.

% solveOptimalControlProblem: solves the optimal control ...
    problem of the
% horizon N with sampling length T for the
% given initial values t0 and x0 and the
% initial guess u0 using the specified algorithm.
% The function returns the computed optimal
% control u, the corresponding value of the
% cost function V as well as possible exit
% flags and additional output of the
% optimization method.

% costfunction: evaluates the cost function of the
% optimal control problem over the horizon
% N with sampling time T for the current
% data of the optimization method t0, x0 and u.
% The function return the computed cost function value.

% nonlinearconstraints: computes the value of the restrictions
% for all sampling instances provided the
% data t0, x0 and u given by the
% optimization method.
% The function returns the value of the
% restrictions for all sampling instances
% separated for inequality restrictions c
% and equality restrictions ceq.
```

```
% computeOpenloopSolution: computes the open loop solution ...
    over the
% horizon N with sampling time T for the
% initial values t0 and x0 as well as the control u.
% The function returns the complete open
% loop solution over the requested horizon.

% dynamic: evaluates the dynamic of the system for
% given initial values t0 and x0 over the
% interval [t0, tf] using the control u.
% The function returns the state vector x
% at time instant tf as well as an output
% of all intermediate evaluated time instances.
% printSolution: prints out information on the current MPC
% step, in particular state and control
% information as well as required computing
% times and exitflags/outputs of the used
% optimization method. The flow of
% information can be controlled by the
% variable iprint and the functions
% printHeader, printClosedloopData and plotTrajectories.
%
% Version of May 30, 2011, in which a bug appearing in the ...
    case of
% multiple constraints has been fixed
%
% (C) Lars Gruene, Juergen Pannek 2011

    if (nargin≥22)
        tol_opt = varargin{1};
    else
        tol_opt = 1e-6;
    end;
    if (nargin≥23)
        opt_option = varargin{2};
    else
```

```
        opt_option = 0;
    end;
    if (nargin≥24)
        if ( strcmp(varargin{3}, 'difference equation') || ...
                strcmp(varargin{3}, 'differential equation') )
            type = varargin{3};
        else
            fprintf([' Wrong input for type of dynamic: use ...
                either ', ...
                '"difference equation" or "differential ...
                    equation".']);
        end
    else
        type = 'difference equation';
    end;
    if (nargin≥25)
        atol_ode_real = varargin{4};
    else
        atol_ode_real = 1e-8;
    end;
    if (nargin≥26)
        rtol_ode_real = varargin{5};
    else
        rtol_ode_real = 1e-8;
    end;
    if (nargin≥27)
        atol_ode_sim = varargin{6};
    else
        atol_ode_sim = atol_ode_real;
    end;
    if (nargin≥28)
        rtol_ode_sim = varargin{7};
    else
        rtol_ode_sim = rtol_ode_real;
    end;
    if (nargin≥29)
```

```
            iprint = varargin{8};
        else
            iprint = 0;
        end;
        if (nargin≥30)
            printHeader = varargin{9};
        else
            printHeader = @printHeaderDummy;
        end;
        if (nargin≥31)
            printClosedloopData = varargin{10};
        else
            printClosedloopData = @printClosedloopDataDummy;
        end;
        if (nargin≥32)
            plotTrajectories = varargin{11};
        else
            plotTrajectories = @plotTrajectoriesDummy;
        end;

        % Determine MATLAB Version and
        % specify and configure optimization method
        vs = version('-release');
        vyear = str2num(vs(1:4));
        if (vyear ≤ 2007)
            fprintf('MATLAB version R2007 or earlier detected\n');
            if ( opt_option == 0 )
                options = optimset('Display','off',...
                    'TolFun', tol_opt,...
                    'MaxIter', 20000,...
                    'LargeScale', 'off',...
                    'RelLineSrchBnd', [],...
                    'RelLineSrchBndDuration', 1,'MaxFunEvals',20000);
            elseif ( opt_option == 1 )
                error('nmpc:WrongArgument', '%s\n%s', ...
                     'Interior point method not supported in ...
```

```
                          MATLAB R2007', ...
                       'Please use opt_option = 0 or opt_option = 2');
        elseif ( opt_option == 2 )
            options = optimset('Display','off',...
                'TolFun', tol_opt,...
                'MaxIter', 2000,...
                'LargeScale', 'on',...
                'Hessian', 'off',...
                'MaxPCGIter', ...
                   max(1,floor(size(u0,1)*size(u0,2)/2)),...
                'PrecondBandWidth', 0,...
                'TolPCG', 1e-1);
        end
    else
        fprintf('MATLAB version R2008 or newer detected\n');
        if ( opt_option == 0 )
            options = optimset('Display','off',...
                'TolFun', tol_opt,...
                'MaxIter', 10000000,...
                'Algorithm', 'active-set',...
                'FinDiffType', 'forward',...
                'RelLineSrchBnd', [],...
                'RelLineSrchBndDuration', 1,...
                'TolConSQP', 1e-14);
        elseif ( opt_option == 1 )
            options = optimset('Display','off',...
                'TolFun', tol_opt,...
                'MaxIter', 2000,...
                'Algorithm', 'interior-point',...
                'AlwaysHonorConstraints', 'bounds',...
                'FinDiffType', 'forward',...
                'HessFcn', [],...
                'Hessian', 'bfgs',...
                'HessMult', [],...
                'InitBarrierParam', 0.1,...
                'InitTrustRegionRadius', ...
```

```
                    sqrt(size(u0,1)*size(u0,2)),...
                'MaxProjCGIter', 2*size(u0,1)*size(u0,2),...
                'ObjectiveLimit', -1e20,...
                'ScaleProblem', 'obj-and-constr',...
                'SubproblemAlgorithm', 'cg',...
                'TolProjCG', 1e-2,...
                'TolProjCGAbs', 1e-10);
        %                       'UseParallel','always',...
        elseif ( opt_option == 2 )
            options = optimset('Display','off',...
                'TolFun', tol_opt,...
                'MaxIter', 2000,...
                'Algorithm', 'trust-region-reflective',...
                'Hessian', 'off',...
                'MaxPCGIter', ...
                    max(1,floor(size(u0,1)*size(u0,2)/2)),...
                'PrecondBandWidth', 0,...
                'TolPCG', 1e-1);
        end
    end

    warning off all
    t = [];
    x = [];
    u = [];
    xContr = [];
    xhatS =[];
    allU = [];
    NO_in = [];

    % Start of the NMPC iteration
    %initiate variables
    mpciter = 0;
    xmeasureContr = [xmeasure(1) xmeasure(4) xmeasure(7) ...
        xmeasure(10)...
        xmeasure(11) xmeasure(12) xmeasure(11) xmeasure(12)];
```

```
    x0 = [xmeasure(1) xmeasure(4) xmeasure(7) xmeasure(10) ...
        xmeasure(11)...
        xmeasure(12) xmeasure(11) xmeasure(12)];
    tmeasureContr = tmeasure;
    if nMHE<0
        xhat = zeros(12,-nMHE);
    else
        xhat = zeros(12,nMHE);
    end
    measurementsX = xmeasure;
    measurementsU = u0(1);

    while(mpciter < mpciterations)
        % Step (1) of the NMPC algorithm:
        %   Obtain new initial value with measureInitialValue
        t_Start = tic;
        [t0, x0, xhat, NO_new, ref] = measureInitialValue ( ...
            tmeasure,...
            xmeasure, x0, T, measurementsX, measurementsU, ...
                xhat,...
            nMHE, model,scrCells,ysp,rsp,usp,ammonia,noise);
        % Step (2) of the NMPC algorithm:
        %   Solve the optimal control problem
        [u_new, V_current, exitflag, output] = ...
            solveOptimalControlProblem ...
            (runningcosts, terminalcosts, constraints, ...
            terminalconstraints, linearconstraints, model, ...
            N, t0, x0, u0, T, ...
            atol_ode_sim, rtol_ode_sim, tol_opt, options, ...
                type, uN,...
            ref, scrCells);
        t_Elapsed = toc( t_Start );

        %   Print solution
        if ( iprint ≥ 1 )
            printSolution(plant, printHeader, ...
```

```
                  printClosedloopData, ...
                             plotTrajectories, mpciter, T, ...
                                tmeasure,...
                             xmeasure, u_new, ...
                             atol_ode_sim, rtol_ode_sim, type, ...
                                iprint, ...
                             exitflag, output, t_Elapsed,scrCells);
        end
        %   Store closed loop data
        t = [ t; tmeasure ];
        x = [ x; xmeasure ];
        u = [ u; u_new(:,1) ];
        tContr = [ t; tmeasureContr ];
        xContr = [ xContr; xmeasureContr ];
        xhatS = [xhatS; xhat(:,end)'];
        allU = [allU;u_new(1,:)];
        NO_in = [NO_in,NO_new];

        %   Prepare restart
        u0 = shiftHorizon(u_new);
        % Step (3) of the NMPC algorithm:
        %   Apply control to process
        %Check plant with new u and obtain new measurement
        [tmeasure, xmeasure] = applyControlPlant(plant, T,...
            tmeasure, xmeasure, u_new, ...
              atol_ode_real, rtol_ode_real, type,scrCells);

        measurementsX = [measurementsX ; xmeasure];
        measurementsU = [measurementsU ; u_new(1)];

        %Check model values
        [tmeasureContr, xmeasureContr] = ...
           applyControlModel(model,...
            T, tmeasureContr, x0, u_new, ...
            atol_ode_real, rtol_ode_real, type,scrCells);

```

```
        mpciter = mpciter+1;


    end
end

function [t0, x0, xhat, NO_in, ref] = ...
    measureInitialValue(tmeasure,...
    xmeasure, x0, T, x, u, init, nMHE, ...
    model,scrCells,ysp,rsp,usp,ammonia,noise)
    t0 = tmeasure;
    [¬, avrg, NO_in] = model(tmeasure,x0,0,T,scrCells);
    a = T*avrg;
    ref = steadyStateTargetFinal(NO_in,T,ysp,rsp,usp);

    %MPC with estimation
    if nMHE > 0
        if size(x,1)≥nMHE      %nMHE amount of measurements ...
            required
            meas = zeros(nMHE,1);
            uMeas = zeros(nMHE,1);
            NO_in = zeros(nMHE,1);
            time = tmeasure-nMHE*T;
        for i = 1:nMHE          %pick right amount of ...
            measurements
            r = size(x,1)-nMHE; %with correct index
            meas(i,:) = [x(i+r,11)+ammonia*x(i+r,12)+...
                noise*(-1*10^-5 + ...
                    (1*10^-5+1*10^-5).*randn(1,1))]; %noise
            uMeas(i) = u(i+r);
            [¬, ¬, NO_in(i)] = model(time,x0,0,T,scrCells);
            time = time + T;
        end
```

```
        %all MHE estimates
        xhat = ...
            MHE4cellFinal(T,nMHE,meas,uMeas,shiftHorizon(init), ...
            NO_in);
        %initial value for prediction model
        x0 = [xhat(1,end) xhat(4,end) xhat(7,end) xhat(10,end)...
            a*(xhat(11,end)-(0*10^-6))+(1-a)*x0(5)...
            a*xhat(12,end)+(1-a)*x0(6) xhat(11,end) ...
                xhat(12,end)];


        end
        %virtual measurements before measurement window is ...
            fulfilled
        %all measurements zero before the application starts
        %the last zero updates with the latest measurement
        if size(x,1)<nMHE
            xV = x(:,11);
            uV = u;
            xVirtual = [zeros(nMHE-size(x,1),1);xV];
            uVirtual = [zeros(nMHE-size(x,1),1);uV];
            NO_in = ones(1,nMHE)*NO_in;
            xhat = MHE4cellFinal(T,nMHE,xVirtual,...
                uVirtual,shiftHorizon(init),NO_in);   %All ...
                    MHE estimates
            x0 = [xhat(1,end) xhat(4,end) xhat(7,end) ...
                xhat(10,end)...
                a*xhat(11,end)+(1-a)*x0(5) ...
                    a*xhat(12,end)+(1-a)*x0(6)...
                xhat(11,end) xhat(12,end)]; %x0 for next MPC ...
                    iteration
        end
    end

    %MPC without estimation
    if nMHE==0
```

```
            x0 = [xmeasure(1) xmeasure(4) xmeasure(7) ...
                xmeasure(10)...
                a*xmeasure(11)+(1-a)*x0(5) ...
                    a*xmeasure(12)+(1-a)*x0(6)...
                xmeasure(11) xmeasure(12)];
            xhat = xmeasure';
    end
    %MPC with0 estimation done separately
    if nMHE < 0
        nMHE = -nMHE;
        x0 = [xmeasure(1) xmeasure(4) xmeasure(7) ...
            xmeasure(10)...
             a*xmeasure(11)+(1-a)*x0(5) ...
                 a*xmeasure(12)+(1-a)*x0(6)...
             xmeasure(11) xmeasure(12)];
        if size(x,1)≥nMHE
            meas = zeros(nMHE,1);
            uMeas = zeros(nMHE,1);
            NO_in = zeros(nMHE,1);
            time = tmeasure -nMHE*T;
        for i = 1:nMHE
            r = size(x,1)-nMHE;
            meas(i,:) = [x(i+r,11)+ammonia*x(i+r,12)+...
                noise*(-1*10^-5 + ...
                    (1*10^-5+1*10^-5).*randn(1,1))];
            uMeas(i) = u(i+r);
            [¬, ¬, NO_in(i)] = model(time,x0,0,T,scrCells);
            time = time + T;
        end
        xhat = ...
            MHE4cellFinal(T,nMHE,meas,uMeas,shiftHorizon(init), ...
            NO_in);


        end
        %virtual measurements before measurement window is ...
```

```
                fulfilled
            %all measurements zero before the application starts
            %the last zero updates with the latest measurement
            if size(x,1)<nMHE
                xV = x(:,11);
                uV = u;
                xVirtual = [zeros(nMHE-size(x,1),1);xV];
                uVirtual = [zeros(nMHE-size(x,1),1);uV];
                NO_in = ones(1,nMHE)*NO_in;
                xhat = MHE4cellFinal(T,nMHE,xVirtual,uVirtual,...
                    shiftHorizon(init),NO_in);
            end
        end

        NO_in=NO_in(end); %return latest NO_in measurement
end

function [tapplied, xapplied] = applyControlPlant(plant, T, ...
    t0, x0, u, ...
                                  atol_ode_real, rtol_ode_real, ...
                                      type,scrCells)
        xapplied = dynamicPlant(plant, T, t0, x0, u(:,1), ...
                            atol_ode_real, rtol_ode_real, ...
                                type,scrCells);
        tapplied = t0+T;
end

function [tapplied, xapplied] = applyControlModel(system, T, ...
    t0, x0, u, ...
                                  atol_ode_real, rtol_ode_real, ...
                                      type,scrCells)
        xapplied = dynamic(system, T, t0, x0, u(:,1), ...
                            atol_ode_real, rtol_ode_real, ...
                                type,scrCells);
        tapplied = t0+T;
end
```

```
function u0 = shiftHorizon(u)
    u0 = [u(:,2:size(u,2)) u(:,size(u,2))];
end

function [u, V, exitflag, output] = ...
   solveOptimalControlProblem ...
    (runningcosts, terminalcosts, constraints, ...
       terminalconstraints, ...
    linearconstraints, system, N, t0, x0, u0, T, ...
    atol_ode_sim, rtol_ode_sim, tol_opt, options, type, ...
       uN,ref,scrCells)
    x = zeros(N+1, length(x0));
    x = computeOpenloopSolution(system, N, T, t0, x0, u0, ...
                                atol_ode_sim, rtol_ode_sim, ...
                                   type, uN,scrCells);

    % Set control and linear bounds
    A = [];
    b = [];
    Aeq = [];
    beq = [];
    lb = [];
    ub = [];
    for k=1:N
        [Anew, bnew, Aeqnew, beqnew, lbnew, ubnew] = ...
               linearconstraints(t0+k*T,x(k,:),u0(:,k));
        A = blkdiag(A,Anew);
        b = [b, bnew];
        Aeq = blkdiag(Aeq,Aeqnew);
        beq = [beq, beqnew];
        lb = [lb, lbnew];
        ub = [ub, ubnew];
    end

```

```
    % Solve optimization problem
    [u, V, exitflag, output] = fmincon(@(u) ...
        costfunction(runningcosts, ...
        terminalcosts, system, N, T, t0, x0, ...
        u, atol_ode_sim, rtol_ode_sim, type, uN, ...
            ref,scrCells),...
        u0, A, b, Aeq, beq, lb, ...
        ub, @(u) nonlinearconstraints(constraints, ...
            terminalconstraints, ...
        system, N, T, t0, x0, u, ...
        atol_ode_sim, rtol_ode_sim, type, uN,scrCells), options);
end

function cost = costfunction(runningcosts, terminalcosts, ...
    system, ...
                    N, T, t0, x0, u, ...
                    atol_ode_sim, rtol_ode_sim, type, uN, ...
                        r,scrCells)
    cost = 0;
    x = zeros(N+1, length(x0));
    x = computeOpenloopSolution(system, N, T, t0, x0, u, ...
                                atol_ode_sim, rtol_ode_sim, ...
                                    type,...
                                uN,scrCells);
    for k=1:N
        cost = cost+runningcosts(t0+k*T, x(k,:), u(:,k), r);
    end
    cost = cost+terminalcosts(t0+(N+1)*T, x(N+1,:));
end

function [c,ceq] = nonlinearconstraints(constraints, ...
    terminalconstraints, system, ...
    N, T, t0, x0, u, atol_ode_sim, rtol_ode_sim, type, ...
        uN,scrCells)
    x = zeros(N+1, length(x0));
    x = computeOpenloopSolution(system, N, T, t0, x0, u, ...
```

```
                                        atol_ode_sim, rtol_ode_sim, ...
                                            type,...
                                        uN,scrCells);
    c = [];
    ceq = [];
    for k=1:N
        [cnew, ceqnew] = constraints(t0+k*T,x(k,:),u(:,k));
        c = [c cnew];
        ceq = [ceq ceqnew];
    end
    [cnew, ceqnew] = terminalconstraints(t0+(N+1)*T,x(N+1,:));
    c = [c cnew];
    ceq = [ceq ceqnew];
end

function x = computeOpenloopSolution(system, N, T, t0, x0, u, ...
                                             atol_ode_sim, ...
                                                 rtol_ode_sim,...
                                             type, uN,scrCells)
    x(1,:) = x0;

    for k=1:N
        x(k+1,:) = dynamic(system, T, t0, x(k,:), ...
            u(:,min(k,uN)), ...
                                    atol_ode_sim, rtol_ode_sim, ...
                                        type,scrCells);
    end
end

function [x, t_intermediate, x_intermediate] = ...
    dynamic(system, T, t0, ...
                x0, u, atol_ode, rtol_ode, type,scrCells)
    if ( strcmp(type, 'difference equation') )
        x = system(t0, x0, u, T,scrCells);
        x_intermediate = [x0; x];
        t_intermediate = [t0, t0+T];
```

```
    elseif ( strcmp(type, 'differential equation') )
        options = odeset('AbsTol', atol_ode, 'RelTol', rtol_ode);
        [t_intermediate,x_intermediate] = ode45(system, ...
            [t0, t0+T], x0, options, u,scrCells);
        x = x_intermediate(size(x_intermediate,1),:);
    end
end

%Works only for a continous time plant
function [x, t_intermediate, x_intermediate] = ...
    dynamicPlant(system,...
    T, t0, x0, u, atol_ode, rtol_ode, type,scrCells)
        options = odeset('AbsTol', atol_ode, 'RelTol', rtol_ode);
        [t_intermediate,x_intermediate] = ode45(system, ...
            [t0, t0+T], x0, options, u,scrCells);
        x = x_intermediate(size(x_intermediate,1),:);

end

function printSolution(system, printHeader, ...
    printClosedloopData, ...
             plotTrajectories, mpciter, T, t0, x0, u, ...
             atol_ode, rtol_ode, type, iprint, exitflag, ...
                 output,...
             t_Elapsed,scrCells)
    if (mpciter == 0)
        printHeader();
    end
    printClosedloopData(mpciter, u, x0, t_Elapsed,scrCells);
    switch exitflag
        case -2
        if ( iprint ≥ 1 && iprint < 10 )
            fprintf(' Error F\n');
        elseif ( iprint ≥ 10 )
            fprintf(' Error: No feasible point was found\n')
        end
```

```
        case -1
        if ( iprint ≥ 1 && iprint < 10 )
            fprintf(' Error OT\n');
        elseif ( iprint ≥ 10 )
            fprintf([' Error: The output function terminated ...
                the',...
                      ' algorithm\n'])
        end
        case 0
        if ( iprint == 1 )
            fprintf('\n');
        elseif ( iprint ≥ 2 && iprint < 10 )
            fprintf(' Warning IT\n');
        elseif ( iprint ≥ 10 )
            fprintf([' Warning: Number of iterations ...
                exceeded',...
                      ' options.MaxIter or number of function',...
                      ' evaluations exceeded options.FunEvals\n'])
        end
        case 1
        if ( iprint == 1 )
            fprintf('\n');
        elseif ( iprint ≥ 2 && iprint < 10 )
            fprintf(' \n');
        elseif ( iprint ≥ 10 )
            fprintf([' First-order optimality measure was ...
                less',...
                      ' than options.TolFun, and maximum ...
                         constraint',...
                      ' violation was less than ...
                         options.TolCon\n'])
        end
        case 2
        if ( iprint == 1 )
            fprintf('\n');
        elseif ( iprint ≥ 2 && iprint < 10 )
```

```
                fprintf(' Warning TX\n');
            elseif ( iprint ≥ 10 )
                fprintf(' Warning: Change in x was less than ...
                    options.TolX\n')
            end
            case 3
            if ( iprint == 1 )
                fprintf('\n');
            elseif ( iprint ≥ 2 && iprint < 10 )
                fprintf(' Warning TJ\n');
            elseif ( iprint ≥ 10 )
                fprintf([' Warning: Change in the objective ...
                    function',...
                          ' value was less than options.TolFun\n'])
            end
            case 4
            if ( iprint == 1 )
                fprintf('\n');
            elseif ( iprint ≥ 2 && iprint < 10 )
                fprintf(' Warning S\n');
            elseif ( iprint ≥ 10 )
                fprintf([' Warning: Magnitude of the search ...
                    direction',...
                          ' was less than 2*options.TolX and ...
                              constraint',...
                          ' violation was less than ...
                              options.TolCon\n'])
            end
            case 5
            if ( iprint == 1 )
                fprintf('\n');
            elseif ( iprint ≥ 2 && iprint < 10 )
                fprintf(' Warning D\n');
            elseif ( iprint ≥ 10 )
                fprintf([' Warning: Magnitude of directional ...
                    derivative',...
```

```
                     ' in search direction was less than',...
                     ' 2*options.TolFun and maximum ...
                        constraint',...
                     ' violation was less than ...
                        options.TolCon\n'])
           end
       end
       if ( iprint ≥ 5 )
           plotTrajectories(@dynamic, system, T, t0, x0, u, ...
               atol_ode,...
                rtol_ode, type,scrCells)
       end
end

function printHeaderDummy(varargin)
end

function printClosedloopDataDummy(varargin)
end

function plotTrajectoriesDummy(varargin)
end
```

Bibliography

[1] Lars Grüne, Jürgen Pannek. (2017). *Nonlinear Model Predictive Control Theory and Algorithms second edition.* Springer International Publishing Switzerland

[2] James B. Rawlings, David Q. Mayne, Moritz M. Diehl. (2020). *Model Predictive Control: Theory, Computation, and Design 2nd Edition.* Nob Hill Publishing, LLC.

[3] Krister Forsman. (2010). *Reglerteknik för processindustrin.* Studentlitteratur AB, Lund. Pages 44-52.

[4] Michael A. Johnson, Mohammad H. Moradi. (2005). *PID control New identification and design methods.* Springer Verlag-London. Pages 29-46.

[5] Stephen Boyd and Lieven Vandenberghe. (2004). *Convex Optimization.* Cambridge University Press. Pages 136-145.

[6] P. Lancaster, L. Rodman. (1995). *Algebraic Riccati equations,.* Clarendon Press. Pages 86-92.

[7] R. Tyrrell Rockafellar. (1970). *Convex Analysis.* Princeton University Press New Jersey. Pages 10 and 253.

[8] J. Richalet, A. Rault, J.L. Testud, J. Raponi. (1978). Model Predictive Heuristic Control: Applications to Industrial Processes. Automatica Volume 14, Issue 5, Pages 413-428. doi: `https://doi.org/10.1016/0005-1098(78)90001-8`.

[9] C.R. Cutler, B.L. Ramaker. (1980). Dynamic Matrix Control - A Computer Control Algorithm. Joint Automatic Control Conference. San Francisco.

[10] C.C. Chen, L. Shaw. (1981). On receding horizon feedback control. IFAC Proceedings Volumes, Volume 14, Issue 2, Pages 377-382. doi: `https://doi.org/10.1016/S1474-6670(17)63513-4`.

[11] Milver Antonio Colmenares Ocona. (2013). Development of an environment for virtual testing of SCR systems Master's thesis. Åbo Akademi.

[12] Oliver Kröcher. (2018). Selective Catalytic Reduction of NOx. Catalysts 2018, 8(10), 459. doi: `https://doi.org/10.3390/catal8100459`.

[13] Pramod Ubare, Deepak Ingole, D. N. Sonawane. (2021). Nonlinear Model Predictive Control of BLDC Motor with State Estimation. IFAC-PapersOnLine Volume 54, Issue 6, Pages 107-112. doi: `https://doi.org/10.1016/j.ifacol.2021.08.531`.

[14] Eric L. Haseltine, James B. Rawlings. (2005). Critical Evaluation of Extended Kalman Filtering and Moving-Horizon Estimation. Ind. Eng. Chem. Res. 2005, 44, 8, 2451–2460. doi: `https://doi.org/10.1021/ie034308l`.

[15] X.W. Zhang, S.H. Chan, H.K. Ho, Jun Li, Guojun Li, Zhenping Feng. (2008). Nonlinear model predictive control based on the moving horizon state estimation for the solid oxide fuel cell. International Journal of Hydrogen Energy, Volume 33, Issue 9, Pages 2355-2366, ISSN 0360-3199, doi: `https://doi.org/10.1016/j.ijhydene.2008.02.063`.

[16] Rolf Findeisen, Lars Imsland, Frank Allgöwer, Bjarne A. Foss. (2003). Stability Conditions for Observer Based Output Feedback Stabilization with Nonlinear Model Predictive Control. Proceedings of the 42nd IEEE Conference on Decision and Control, Maui, Hawaii USA, December. doi: `https://ieeexplore.ieee.org/document/1272810`.

[17] Hyndman, R.J., Athanasopoulos, G. (2021). Forecasting: principles and practice, 3rd edition. OTexts: Melbourne, Australia. https://otexts.com/fpp3/. Accessed on Jun 17, 2022.

[18] Nonlinear Model Predictive Control. MATLAB NMPC routine and examples. (2011). http://numerik.mathematik.uni-bayreuth.de/~lgruene/nmpc-book/matlab_nmpc.html
Accessed on Jan 5, 2022.

[19] Volkswagen UK. (2022). AdBlue: Selective Catalytic Reduction (SCR). https://www.volkswagen.co.uk/en/technology/engines/adblue.html. Accessed on Mar 3, 2022.

[20] Matworks Help Center. (2022). fmincon. https://se.mathworks.com/help/optim/ug/fmincon.html#busog7r_vh. Accesed on Jun 29, 2022.