Nonlinear Model Predictive Control

of a combined power and district heating production portfolio

Joakim Børlum Petersen
Control and Automation, MSc. thesis
Department of Electronic Systems
June 2017

Abstract

This thesis aims to evaluate the use of Nonlinear Model Predictive Control (NMPC) as a control concept for production planning and balance control, for a fictitious combined power and district heating production portfolio, in a component-based modeling context. A comparison to Linear Model Predictive Control (LMPC) is used as basis for this evaluation. The desire to use NMPC is due to the desire of eliminating the need for linearization; potentially loosing valuable information and minimizing the need for manual preconditioning labor.

An optimization-friendly first-principle nonlinear model of the production portfolio and consumers is constructed in Modelica, heavily relying on the component-based capabilities of the language. This model is linearized; thus both a nonlinear model usable in NMPC and a linear model usable in LMPC is obtained. Through simulations, the linear model was found comparable to the nonlinear model – but with deviations when using the accumulator included in the production portfolio.

The MPC control scheme is designed around an economical cost function, derived through basic economical considerations of the use of production units and a simplified power market model. The resulting optimal control problem is shared between both NMPC and LMPC; the only difference being the model employed in the constraints enforcing system dynamics. To provide full state information an Extended Kalman Filter (EKF) is designed, under the assumption that consumer states are not measurable.

The optimal control problems are solved using JModelica.org, a framework that allows optimization directly on Modelica models. Thus, a simulation framework is designed and implemented on top of JModelica.org, allowing for simulations with both NMPC and LMPC.

Simulation studies show, that NMPC uses the accumulator more actively. The extensive use of the accumulator by NMPC, is performance-wise better, considering long simulations with historical power prices and ambient temperatures. The method of using first-principle nonlinear models directly in MPC is thus, at least on a conceptual level, highly encouraged.

AALBORG UNIVERSITY
DENMARK

Preface

This thesis is written as the final project under the MSc in Control and Automation at Aalborg University. The thesis has been written in the period between the 1. September 2016 and the 8. June 2017 in collaboration with Added Values P/S. My supervisor through the project has been Associate Professor, PhD Jan. D. Bendtsen from Aalborg University.

I would like to thank the people at Added Values P/S for providing me with the opportunity of writing my thesis in collaboration with them; allowing me to conduct my work in a helpful and resourceful environment. A special thanks is extended to Rene Just Nielsen for his guidance and for introducing me to the wonders of component-based modeling using Modelica.

Thanks is also extended to; Per-Ola Larsson from Modelon AB, for providing me with insight around using JModelica.org for optimization on district heating systems and to both PhD Kasper Vinther from Added Values P/S and Associate Professor Palle Andersen from Aalborg University, for providing valuable feedback on my project half-way through.

—Joakim Børlum Petersen, 8. June 2017

Abbreviations

AMR	Automatic Meter Reading
AST	Abstract Syntax Tree
CHP	Combined Heat and Power
COP	Coefficient of Performance
DAE	Differential Algebraic Equation
DOP	Dynamic Optimization Problem
EDP	Economic Dispatch Problem
EKF	Extended Kalman Filter
FMI	Functional Mockup Interface
FMU	Functional Mockup Unit
HP	Heat Pump
IDE	Integrated Development Environment
LMPC	Linear Model Predictive Control
LQR	Linear Quadratic Regulator
MPC	Model Predictive Control
MSL	Modelica Standard Library
NLP	Nonlinear Program
NMPC	Nonlinear Model Predictive Control
OCP	Optimal Control Problem
ODE	Ordinary Differential Equation
QoS	Quality-of-Service
UCP	Unit Commitment Problem

Notation

The following notation is used throughout this thesis:

$\dot{x}$	Time derivatives are denoted with an 'over-dot'.
$\mathrm{sgn}(x)$	The sign operator; -1 for x negative, and 1 for x positive.
$\mathbb{E}(x)$	The expectation operator.
$\boldsymbol{x}$	Vectors are denoted in lower-case bold.
$\boldsymbol{A}$	Matrices are denoted in upper-case bold.

Also, the convention for denoting heat flows in MJ/s and electrical power in MW is used. *Electrical power* will often just be refered to as *power*.

Contents

Introduction 1

Power production in Denmark is increasingly stochastic, especially since more and more energy is produced by wind turbines and photovoltaics. At the time of writing, wind turbines make up for more than 40 % of the total energy production in Denmark, with the goal of hitting 50 % by 2020[1]. But even more so, for a total of 1460 hours in of 2015, wind turbines in Western Denmark (Jutland and Funen) delivered more energy than what was consumed in Western Denmark[1]. The production surplus can in many cases be exported to bordering countries and thus still provide an income – but even so, the energy produced by renewable sources is reaching a critical mass, introducing effects such as negative electricity prices; meaning that suppliers have to pay to get rid of the electricity they produce. Instead of simply getting rid of this excess power, it would be beneficial to use it, to avoid expenses and wasting resources.

In Denmark, district heating is responsible for supplying heat to approximately 65 % of all households[2]. One use of the excess electricity, would be to convert it to district heating. Conversion to district heating is beneficial for one reason in particular; storage. Even if at the time of conversion, heat is not in demand, it is possible with the current district heating infrastructure to store it in accumulator tanks for later use. The problem with this method, is, that energy is converted several times, before delivered as heat to the consumer, resulting in undesirable losses; e.g. from thermal energy, to mechanical energy, to electrical energy and finally back to thermal energy again. Converting excess electricity to district heating is therefor not the ideal solution, but combined with new modes of operation (e.g. turbine bypass) in the production of heat and power, it can provide desirable added flexibility.

The increasingly stochastic electricity production is imposing new requirements to production planning and balance control, where new control strategies are required to match the change in dynamics. Production planning involves distributing load amongst units in a production portfolio. This is often done, by the use of static optimization, solving what is commonly known as the Unit Commitment Problem (UCP). The solution to a UCP determines the active subset of portfolio units and a load plan. Often the UCP is formulated as minimizing/maximizing an economical cost function and is often solved on a daily or weekly basis. The load plan is not guaranteed to hold, due to prediction errors, disturbances and fluctuations in production – hence, a balance controller is introduced, regulating load to balance production and consumption.[3]

In [4], short-term production planning is considered for a district heating production portfolio. The approach taken divides the problem into two separate optimization problems; the UCP and the Economic Dispatch Problem (EDP). In [4], the solution to the UCP is used to determine whether a unit is online or not, and as such, it is modeled by mixed-integer linear models and solved by standard mixed-integer solvers. Determining the load for the online units is handled by solving the EDP. The solution to the EDP includes dynamic optimizing of nonlinear first principle models, and determines the load and different setpoints for the individual units. Furthermore, in [5], the method is suggested to be extended to having the EDP solved in a loop, utilizing Model Predictive Control (MPC). It is concluded that this would further improve on the obtained results[5]. In [3], a balance controller for a portfolio of electricity producing units is also designed using an MPC scheme, which was found to give significant improvements over the standard industry approach of distributed PID controllers; both in terms of minimizing cost and rejecting disturbances[3].

The thermo-hydraulic models necessary to describe both a power production and district heating system are inherently *nonlinear*. The nonlinearities stem from several different phenomena. These phenomena are for example:

- Transmission lines feature a quadratic relation between flow and pressure.
- Multiplication of conjugate variables in energy balances [1].
- The general mathematical description of fluid properties.

The majority of literature around controller design is based upon *linear* systems. Thus, when a control law is desired for a system including nonlinearities, the nonlinearities are often handled by linearizing the system. The linearization will be around a suitable operating point, under the assumption that small perturbations around this operating point are adequately described by linear models. This approach is however tedious, especially when one operating point is not enough. It is *highly* desirable, to be able to work directly with the physical models – including their nonlinearities – to avoid both the loss of information and the tedious work, credited to linearization.

Using MPC as a control concept, it is to some extent possible to work directly with the nonlinear physical models. In general, one distinguishes between Linear Model Predictive Control (LMPC) and Nonlinear Model Predictive Control (NMPC). For the case of LMPC, the system model and all constraints are linear and the Optimal Control Problem (OCP) can be cast as a quadratic problem. This has the benefit of guaranteed efficient solutions and is also the most common approach. For the case of NMPC, the OCP can be cast as an Nonlinear Program (NLP); and as such, the system model and constraints are no longer required to be linear, making it possible to avoid linearization and the resulting loss of information. An NLP is however computationally more demanding to solve and with the added risk for an optimizer, of not finding a feasible solution.[7]

[1] Conjugate variables are sets of intensive (X) and extensive variables (x), where the product, $X\mathrm{d}x$, is the change in internal energy of a system. These products arise in energy conservation in all domains, and is thus not specific to thermodynamics or hydraulics, as is the case in this thesis. An intensive variable, is a material property which is independent of the amount of material; e.g. a concentration or a temperature. An extensive variable, is a material property which is dependent on the amount of material; e.g. mass or energy.[6]

1.1 Problem statement

This thesis is done in collaboration with the danish company Added Values P/S. At Added Values P/S, dynamic modeling and control of power and district heating production portfolios is an essential part of their work. Added Values P/S are interested in investigating, whether they can minimize manual preconditioning labor; one example hereof would be linearization and the overhead it includes. An optimal approach, for Added Values P/S to provide a complete modeling and control concept, would include reusing models of complete production units, in a combined system model. An MPC would then be added, requiring no changes to the model, for it to be used in an OCP. Thus, Added Values P/S are very much interested in evaluating the use of NMPC together with component-based modeling.

The aim of this thesis, is to evaluate the use of NMPC as a control concept, when considering production planning and balance control, for a given combined power and district heating production portfolio, in a component-based modeling context.

1.2 Production portfolio

This thesis will consider a fictitious production portfolio. For inspiration, the portfolio of Sønderborg Fjernvarme is used. Sønderborg Fjernvarme is a district heating supplier in the southern part of Jutland. At the time of writing, Sønderborg Fjernvarme is responsible for delivering district heating to its approximately 10 000 customers. In 2016, 253 GW h heat was sold to the consumers and 28 GW h electricity was sold to the grid. This gives an average heat output of approximately 28 MJ/s and an average power output of 3 MW – not considering losses. The heated water is supplied at a temperature of 75 °C to 82 °C and the pressure difference across the consumer is maintained at 0.3 bar to 0.6 bar.[8]

Sønderborg Fjernvarme is comprised of several distributed production units, including a Combined Heat and Power (CHP) Waste-to-Energy plant responsible for 49 % of the heat production and almost all the power production. The secondary production unit delivering 38 % of the heat, is a plant fueled by biomass and utilizing geothermal heat as a reservoir for four absorption Heat Pumps (HPs). The last 13 % is from peak load boilers (fueled by gas and oil) and from a solar heating plant. An accumulator tank is also present in the portfolio, allowing excess heated water to be stored and supplied to consumers at a later point in time.[8]

Just as the Sønderborg Fjernvarme portfolio, the portfolio in this thesis will include a CHP. Instead of the secondary biomass and absorption HP plant, this thesis will include a compression HP, driven by electrical energy, in the portfolio. This inclusion is interesting, as it will allow power, produced from the CHP and/or bought, to be converted into heat instead. This will increase the flexibility of the portfolio, which will allow it to adapt to the fluctuating electricity prices. A peak load boiler is also considered, fueled by e.g. oil, as well as an accumulator tank. The production portfolio is depicted in **Figure 1.1**. The sizing and parameterization of the individual units will be done, to approximately match the scale of the production at Sønderborg Fjernvarme, but with some design freedom.

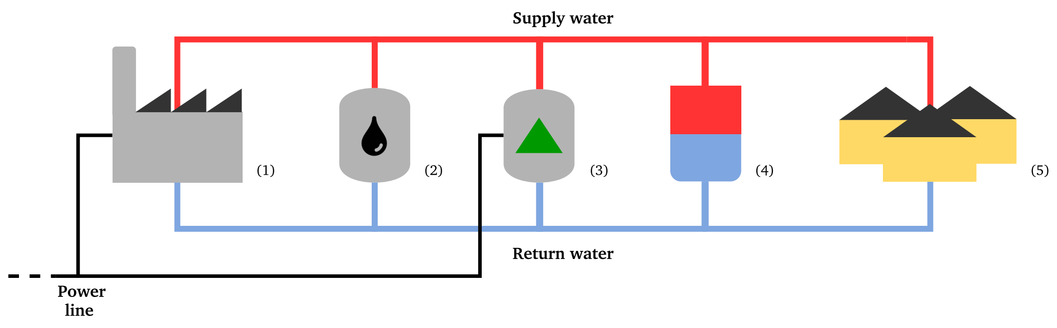


Figure 1.1: *(1) CHP, (2) peak load boiler, (3) compression HP, (4) accumulator tank and (5) consumers.*

1.3 Scope and approach

This thesis will focus on evaluating the use of NMPC for a combined production planning and balance controller for the fictitious production portfolio described in **Section 1.2**. Specifically, how first principle nonlinear models can be used directly in the optimization part of MPC, eliminating linearization. To be able to evaluate the performance of NMPC, a comparison to LMPC will be made.

The work of this thesis will include:

- Building a nonlinear model of the production portfolio.
- Designing both an NMPC and a comparable LMPC control concept.
- Evaluation; including the design method and the performance of the two controllers.

Production planning and balance control is complex, including among others: regulations, taxation, pricing heat production and trading on the power market. These elements will, in this thesis, be greatly simplified, as the focus is purely conceptual. The following simplifications have been made:

Pricing heat production

Cost of running the CHP and peak load boiler will be a fixed price pr. MW h. Cost of running the compression HP will be based entirely on the selling price of power.

Power trading

Power prices will be fixed, based on the ELSPOT day-ahead market – but power can be traded at all times.

This thesis will also not include scheduling units as online or offline; it will assume all productions units to be running all the time, as if the UCP was already solved. As such, the production planning part is reduced to load distribution; and as such, focus is on the EDP.

The specific design approach taken in this thesis, will draw on the approach taken in [4]. In [4], the EDP is solved using a particular set of tools. The object-oriented modeling language, Modelica, is used to create first principle based models of the district heating production portfolio. These models are then used, with no modifications, in solving the EDP, by employing *JModelica.org*. JModelica.org is an open source Modelica platform, that handles both modeling, simulation and optimization. The optimization part is key here, as it allows for using Modelica models directly in a optimization problem, posed using language constructs added on top of the standard Modelica language. This is an interesting approach for Added Values P/S, as they already use Modelica.

Thus, the same method will be applied in this thesis; using Modelica and JModelica.org to undertake the modeling and optimization on the nonlinear models – and as such, a major part of this thesis is therefor also an evaluation of this specific approach and choice of tools.

Methods 2

The chapter will first account for modeling using the Modelica language and the benefits that stem from acausal component-based modeling. Then, optimal control considering nonlinear models is introduced; including the underlying principles in how an optimal control problem is solved. With the introduction of optimal control, MPC is accounted for and the chapter will then move on to introduce JModelica.org and how it can be used to solve optimization problems involving Modelica models directly.

2.1 Modelica

The classical approach to modeling dynamic systems, is causal block-oriented modeling[9]. Causal block-oriented modeling deals with Ordinary Differential Equations (ODEs), to which efficient numerical solutions are available[9]. A more modern approach is that of acausal modeling. Acausal modeling allows for posing models, by declarative equations, without the overhead of having to reformulate the equations, to adhere to a certain signal path.

Acausal modeling is historically very domain-specific, e.g. SPICE which only deals with modeling of electrical circuits[9]. However, system modeling in general also includes cross-overs between domains, and thus a holistic approach to modeling is desired. One example is that of a car; here one needs to deal with both mechanics, thermodynamics, electronics etc. Another is that of a district heating system, which for instance deals with both thermodynamics, fluid dynamics and power system dynamics.

Modelica is a general purpose modeling language for acausal modeling[10]. The Modelica language allows for multi-domain modeling with a focus on modularity by being object-oriented. Modelica deals with Differential Algebraic Equations (DAEs) instead of ODEs, but also allows for the formulation of discrete equations, such that hybrid systems can be modeled. The Modelica language is open-source and developed by the Modelica Association[10].

One disadvantage to causal modeling, lies in the complexity, which arises in large hierarchical models. To allow for efficient simulation using numerical methods, a significant amount of preprocessing is required[9]. This is however handled behind the scenes by the chosen Modelica compiler, and as such it does not result in extra work for the modeler.

This project will employ Modelica, to construct a model of the district heating production portfolio. The model will rely heavily on the object-oriented approach, to allow for an iterative design procedure. For developing the Modelica models for this project, Dymola has been chosen as the Integrated Development Environment (IDE). Dymola is a proprietary tool, including a commercial Modelica compiler. Open alternatives also exist. It is for example possible to rely solely on the JModelica.org suite of tools – this does however not feature a graphical editor for constructing the Modelica models, which is a valuable tool, dealing with large models.

2.2 Optimal control

Optimal control deals with the formulation of control laws, based on the solution to optimization problems. One concept of optimal control, is that of trajectory optimization. Trajectory optimization seeks to find an optimal trajectory for some dynamical system; whether that be by minimizing control action, minimizing the final time of the trajectory or perhaps maximizing profit. A general trajectory optimization problem in continuous time can be formulated as:

$$\underset{\boldsymbol{x}^*,\, \boldsymbol{u}^*}{\text{minimize}}\, J(\boldsymbol{x}, \boldsymbol{u}) = \int_{t_0}^{t_f} g(t, \boldsymbol{x}, \boldsymbol{u})\, \mathrm{d}t \tag{2.1}$$

subject to:

$$\dot{\boldsymbol{x}} = \boldsymbol{f}(t, \boldsymbol{x}, \boldsymbol{u}), \quad \boldsymbol{x}_{\min} \leq \boldsymbol{x} \leq \boldsymbol{x}_{\max}, \quad \boldsymbol{u}_{\min} \leq \boldsymbol{u} \leq \boldsymbol{u}_{\max}$$

Here g is some function that maps from states, $\boldsymbol{x} \in \mathbb{R}^n$, and inputs, $\boldsymbol{u} \in \mathbb{R}^m$, to $\mathbb{R}$. The dynamics of the system are given by $\boldsymbol{f}$ (possibly nonlinear), as the optimal solution has to confine itself to what is physically possible. Furthermore, constraints can optionally be posed on states and inputs, but also on boundary conditions; e.g. on the final time or the value of a state to the final time – not all types of constraints are shown in **Equation (2.1)**.

There are several approaches to solving such an OCP, but two approaches are worth mentioning, as they scale well for high-dimensional nonlinear systems with constraints; one is by *indirect methods* the other is by *direct methods*. A direct method minimizes the objective function, by constructing a sequence of points[11]:

$$(\boldsymbol{x}_2, \boldsymbol{u}_1),\ (\boldsymbol{x}_3, \boldsymbol{u}_2),\ \ldots,\ (\boldsymbol{x}^*, \boldsymbol{u}^*)$$

Such that the following typically holds:

$$J(\boldsymbol{x}_2, \boldsymbol{u}_1) > J(\boldsymbol{x}_3, \boldsymbol{u}_2) > J(\boldsymbol{x}^*, \boldsymbol{u}^*)$$

The direct method does this, by *transcribing* the above infinite dimensional optimization problem (infinite because it is continuous), to a finite dimensional NLP – effectively discretizing the continuous problem to a discrete problem. The NLP is then solvable by a wide-range of available solvers.[11]

An indirect method on the other hand, attempts to find a root for the necessary condition:

$$\nabla J(\boldsymbol{x}, \boldsymbol{u}) = 0 \tag{2.2}$$

An indirect method must therefore compute the slope, ∇J, and then decide whether it is sufficiently close to zero. This optimality condition, together with the problem itself is then transcribed to a NLP and then solved. The distinction between the two, can roughly be boiled down to:

- Indirect: "optimize, then transcribe"
- Direct: "transcribe, then optimize"

Both methods feature transcription, but they differ in how they determine an optimal solution. Indirect methods can also be numerically unstable and difficult to both implement and initialize[11]. For these reasons, direct methods are preferable. In JModelica.org, optimization is handled by an implementation of a direct method and therefor these will be further accounted for.

2.2.1 Transcription

The solution to an optimization problem such as **Equation (2.1)**, is by a direct method, a two-step procedure. First, the continuous OCP is transcribed into a NLP. The transcription is basically a discretization, where the problem is sampled such that a finite dimensional problem arises. By transcribing the problem:

- Decision variables change from **vector function** into **real numbers**.
- **Differental** equations change to **algebraic** equations.

The methods for transcribing an infinite dimensional optimization problem, as the one in **Equation (2.1)**, can be divided into two classes. *Shooting methods* and *simultaneous methods*. The difference lies in how the system dynamics are enforced. Shooting methods use a simulation, to explicitly enforce the dynamics and a constraint in the resulting NLP is that the endpoints for two consecutive simulations touch; giving rise to the notion of defect constraints. The simultaneous methods enforce the dynamics only at a series of points along the trajectories and the trajectories are then approximated as piece-wise polynomial. A constraint in the resulting NLP is thus, that the dynamics at these specific points match the actual dynamics. The two methods are conceptually visualized in **Figure 2.1**.[9]

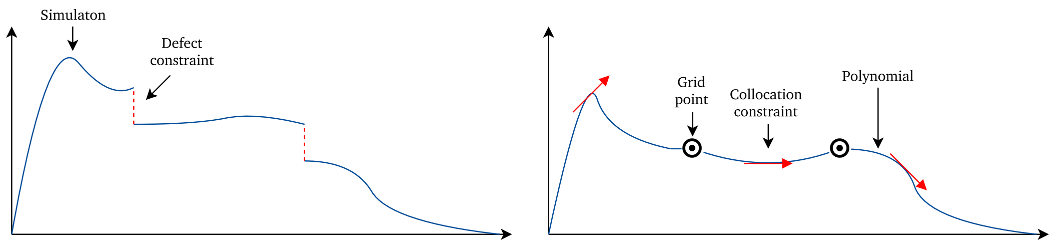


Figure 2.1: *(Multiple) shooting method on the left. Simultaneous method on the right. For shooting methods, the dynamics are enforced by a simulation. For simultaneous methods, the problem is approximated as piece-wise polynomial, with the dynamics enforced at specific* collocation *points.*

The transcription method used in JModelica.org is a simultaneous method, also known as a collocation method. But before diving into the specific method in JModelica.org, a basic example using a simple simultaneous transcription method is considered, to grasp the underlying concept.

Generally, for simultaneous methods, the problem is first sampled at N grid points:

$$\begin{array}{l} \boldsymbol{x}_k = \boldsymbol{x}(t_k) \\ \boldsymbol{u}_k = \boldsymbol{u}(t_k) \end{array} \Rightarrow \begin{array}{rcl} t & \rightarrow & [t_0, t_1, \dots, t_N] \\ \boldsymbol{x}(t) & \rightarrow & [\boldsymbol{x}_0, \boldsymbol{x}_1, \dots, \boldsymbol{x}_N] \\ \boldsymbol{u}(t) & \rightarrow & [\boldsymbol{u}_0, \boldsymbol{u}_1, \dots, \boldsymbol{u}_N] \end{array} \tag{2.3}$$

Now for a very basic approach, the control action and the dynamics can be estimated to be linear between the samples. This is known as collocation using a trapezoid method. By this, the objective function can be approximated as:

$$\begin{aligned} J(\boldsymbol{x}, \boldsymbol{u}) &= \int_{t_0}^{t_f} g(t, \boldsymbol{x}, \boldsymbol{u}) \, \mathrm{d}t \\ &\approx \sum_{k=0}^{N-1} \frac{h_k}{2} \left(g(t_k, \boldsymbol{x}_k, \boldsymbol{u}_k) + g(t_{k+1}, \boldsymbol{x}_{k+1}, \boldsymbol{u}_{k+1}) \right) \end{aligned} \tag{2.4}$$

Where h_k is the time between the two consecutive samples; $t_{k+1} - t_k$. This approximation is illustrated in **Figure 2.2**.

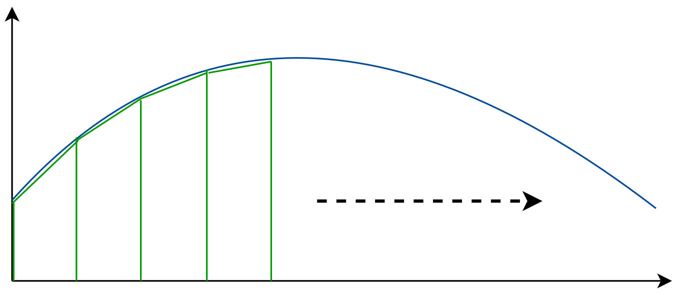


Figure 2.2: *Linear approximation between the selected grid points, gives rise to an objective function, that can be approximated as the sum of the area of a series of trapezoids.*

The constraints imposed by the dynamics, can be transcribed by:

$$\dot{\boldsymbol{x}} = \boldsymbol{f}(t, \boldsymbol{x}, \boldsymbol{u}) \Leftrightarrow \dot{\boldsymbol{x}}(t) = \boldsymbol{f}(t, \boldsymbol{x}(t), \boldsymbol{u}(t))$$

$$\boldsymbol{x}(t_{k+1}) - \boldsymbol{x}(t_k) = \int_{t_k}^{t_{k+1}} \boldsymbol{f}(t, \boldsymbol{x}(t), \boldsymbol{u}(t)) \, \mathrm{d}t \tag{2.5}$$

$$\boldsymbol{x}(t_{k+1}) = \boldsymbol{x}(t_k) + \int_{t_k}^{t_{k+1}} \boldsymbol{f}(t, \boldsymbol{x}(t), \boldsymbol{u}(t)) \, \mathrm{d}t \tag{2.6}$$

Now, employing the same method as for the objective function, the following *collocation constraints* are obtained:

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \frac{h_k}{2}\,(\boldsymbol{f}_{k+1} + \boldsymbol{f}_k) \tag{2.7}$$

Using the above, an NLP is obtained:

$$\underset{[\boldsymbol{x}_0,\ldots,\boldsymbol{x}_N],\,[\boldsymbol{u}_0,\ldots,\boldsymbol{u}_N]}{\text{minimize}} \sum_{k=0}^{N-1} \frac{h_k}{2}\,\big(\,g(t_k, \boldsymbol{x}_k, \boldsymbol{u}_k) + g(t_{k+1}, \boldsymbol{x}_{k+1}, \boldsymbol{u}_{k+1})\,\big) \tag{2.8}$$

subject to:

$$\boldsymbol{x}_{k+1} - \boldsymbol{x}_k = \frac{h_k}{2}\,(\boldsymbol{f}_{k+1} + \boldsymbol{f}_k) \tag{2.9}$$

$$\boldsymbol{x}_{\min} \leq \boldsymbol{x}_k \leq \boldsymbol{x}_{\max}$$

$$\boldsymbol{u}_{\min} \leq \boldsymbol{u}_k \leq \boldsymbol{u}_{\max}$$

This simple example of a transcription method provides the basis of understanding direct collocation in general, which is used in JModelica.org.

2.3 Direct collocation

This section will account for the method implemented in JModelica.org, to solve optimal control problems; obtaining an NLP through direct collocation, after which the NLP is solvable by a wide range of solvers. The basis of this section is [9], which gives a detailed description of the specific implementation.

The general optimal control problem considered, is given as:

minimize:

$$\int_{t_0}^{t_f} L(\boldsymbol{x}(t), \boldsymbol{y}(t), \boldsymbol{u}(t))\,\mathrm{d}t \tag{2.10}$$

with respect to:

$$\boldsymbol{x} : [t_0; t_f] \to \mathbb{R}^{n_x}, \quad \boldsymbol{y} : [t_0; t_f] \to \mathbb{R}^{n_y}$$

$$\boldsymbol{u} : [t_0; t_f] \to \mathbb{R}^{n_u}$$

subject to:

$$\dot{\boldsymbol{x}}(t) = \boldsymbol{f}(t, \boldsymbol{x}(t), \boldsymbol{y}(t), \boldsymbol{u}(t))$$

$$\boldsymbol{g}(t, \boldsymbol{x}(t), \boldsymbol{y}(t), \boldsymbol{u}(t)) = \boldsymbol{0}$$

$$\boldsymbol{h}(t, \dot{\boldsymbol{x}}(t), \boldsymbol{y}(t), \boldsymbol{u}(t)) \leq \boldsymbol{0}$$

$$\forall t \in [t_0; t_f]$$

The objective function includes the definite integral of the function L; where L maps from states, outputs and inputs to $\mathbb{R}$. The constraints are comprised of system dynamics, equality constraints and inequality constraints.

The optimization time horizon, $t_f - t_0$, is divided into N_e elements, where h_i denotes the length of the i-th element – and the length, h_i, is normalized, such that the sum of all element lengths is one. Furthermore, a local time, τ, for each element is introduced. The local time is also normalized, so that it is 0 at t_{i-1} and 1 at t_i. The corresponding unnormalized time, $\bar{t}_i$, is then described by:

$$\bar{t}_i(\tau) := t_{i-1} + h_i\,(t_f - t_0)\,\tau \quad \forall\, \tau \in [0; 1], \quad \forall\, i \in [1..N_e] \tag{2.11}$$

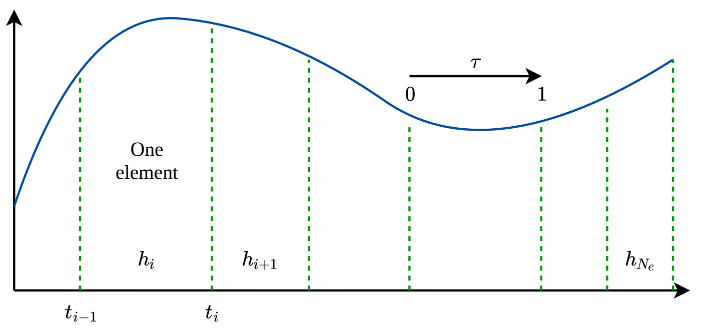


Figure 2.3: *The optimization horizon is divided into N_e elements; each element i features a normalized length, h_i – normalized, such that the sum of all N_e element lengths is equal 1.*

Within element i, a time-dependent system variable z is defined as:

$$\boldsymbol{z}_i = [\dot{\boldsymbol{x}}_i, \boldsymbol{x}_i, \boldsymbol{y}_i, \boldsymbol{u}_i] \tag{2.12}$$

Where $\boldsymbol{x}$ is the state, $\boldsymbol{y}$ is the output and $\boldsymbol{u}$ is the input. Where in the trapezoid method, the input and the dynamics were estimated to be linear – they are now estimated as a polynomial of given degree. The desire is, to estimate the system variable, $\boldsymbol{z}$, by a polynomial, which maps from τ to $\mathbb{R}^{n_z}$ – where n_z, is the number of variables in $\boldsymbol{z}$. The polynomial is denoted the collocation polynomial, for that given element.

The collocation polynomials are created, by choosing a number of collocation points, N_c, for each element – using the collocation points as interpolation points. Consider collocation point $k \in [1..N_c]$, which is located at local time τ_k, then:

$$\boldsymbol{z}(\tau_k) = \boldsymbol{z}_{i,k} = [\dot{\boldsymbol{x}}_{i,k}, \boldsymbol{x}_{i,k}, \boldsymbol{y}_{i,k}, \boldsymbol{u}_{i,k}] \tag{2.13}$$

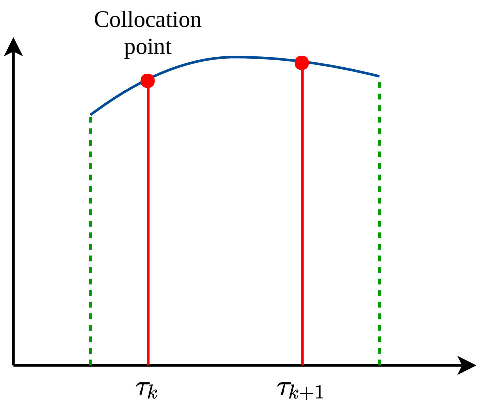


Figure 2.4: *Collocation polynomials are formed on the basis of N_c collocation points for each element.*

The collocation polynomials are formed as:

$$\boldsymbol{x}_i(\tau) = \sum_{k=0}^{N_c} \boldsymbol{x}_{i,k}\, \bar{\ell}_k(\tau) \tag{2.14}$$

$$\boldsymbol{y}_i(\tau) = \sum_{k=1}^{N_c} \boldsymbol{y}_{i,k}\, \ell_k(\tau) \tag{2.15}$$

$$\boldsymbol{u}_i(\tau) = \sum_{k=1}^{N_c} \boldsymbol{u}_{i,k}\, \ell_k(\tau) \tag{2.16}$$

Where $\bar{\ell}_k$ and ℓ_k are Lagrange basis polynomials, defined as:

$$\bar{\ell}_k(\tau) := \prod_{l \in [0..N_c] \setminus \{k\}}^{N_c} \frac{\tau - \tau_l}{\tau_k - \tau_l} \quad \forall k \in [0..N_c] \tag{2.17}$$

$$\ell_k(\tau) := \prod_{l \in [1..N_c] \setminus \{k\}}^{N_c} \frac{\tau - \tau_l}{\tau_k - \tau_l} \quad \forall k \in [1..N_c] \tag{2.18}$$

There is one subtle difference, for the collocation polynomial for $\boldsymbol{x}_i(\tau)$. Since the state has to be continuous on the interval $[t_0; t_f]$, an extra collocation point is added to each element, at $\tau = 0$. This will ensure continuity of $\boldsymbol{x}$, as the start collocation point of an element will be situated at the same place as the end collocation point for the previous element. This is visible in the initialization of k in the summation in **Equation (2.14)**.

Since the time is normalized, the basis polynomials are identical for all elements. Furthermore, the polynomials satisfy the important property that:

$$\ell_k(\tau_j) = \begin{cases} 1 & \text{if } j = k \\ 0 & \text{if } j \neq k \end{cases} \tag{2.19}$$

This property makes sure, that exactly at the collocation points, the value of the collocation polynomial is equal to the respective system variable; state, output or input. Consider the following:

$$\boldsymbol{x}_i(\tau_k) = \sum_{k=0}^{N_c} \boldsymbol{x}_{i,k}\, \bar{\ell}_k(\tau_k) \tag{2.20}$$

$$\boldsymbol{x}_i(\tau_k) = \boldsymbol{x}_{i,0}\, \bar{\ell}_0(\tau_k) + \boldsymbol{x}_{i,1}\, \bar{\ell}_1(\tau_k) + \ldots + \boldsymbol{x}_{i,N_c}\, \bar{\ell}_{N_c}(\tau_k) \tag{2.21}$$

Now, as an example, let $\tau_k = \tau_0$:

$$\boldsymbol{x}_i(\tau_0) = \boldsymbol{x}_{i,0}\, \bar{\ell}_0(\tau_0) + \boldsymbol{x}_{i,1}\, \bar{\ell}_1(\tau_0) + \ldots + \boldsymbol{x}_{i,N_c}\, \bar{\ell}_{N_c}(\tau_0) \tag{2.22}$$

$$\boldsymbol{x}_i(\tau_0) = \boldsymbol{x}_{i,0} \cdot 1 + \boldsymbol{x}_{i,1} \cdot 0 + \ldots + \boldsymbol{x}_{i,N_c} \cdot 0 = \boldsymbol{x}_{i,0} \tag{2.23}$$

Equation (2.14) through **Equation (2.16)** does not cover all of $\boldsymbol{z}$, as the state derivative, $\dot{\boldsymbol{x}}$, has not been given an explicit collocation polynomial. This can be obtained, by differentiating the collocation polynomial for $\boldsymbol{x}_i$ with respect to time:

$$\dot{\boldsymbol{x}}_i(\tau) = \frac{\mathrm{d}\boldsymbol{x}_i}{\mathrm{d}\bar{t}_i}(\tau) \overset{\text{chain rule}}{=} \frac{\mathrm{d}\tau}{\mathrm{d}\bar{t}_i}\frac{\mathrm{d}\boldsymbol{x}_i}{\mathrm{d}\tau} \tag{2.24}$$

From the definition of $\bar{t}_i$, τ can be isolated and differentiated with respect to $\bar{t}_i$:

$$\tau = \frac{\bar{t}_i}{h_i\,(t_f - t_0)} - t_{i-0} \quad \Rightarrow \quad \frac{\mathrm{d}\tau}{\mathrm{d}\bar{t}_i} = \frac{1}{h_i\,(t_f - t_0)} \tag{2.25}$$

And differentiating the collocation polynomial for the state, $\boldsymbol{x}_i$, with respect to τ, gives:

$$\frac{\mathrm{d}\boldsymbol{x}_i}{\mathrm{d}\tau} = \sum_{k=0}^{N_c} \boldsymbol{x}_{i,k}\,\frac{\mathrm{d}\bar{\ell}_k(\tau)}{\mathrm{d}\tau} \quad \text{(As } \boldsymbol{x}_{i,k} \text{ is constant)} \tag{2.26}$$

And thus:

$$\dot{\boldsymbol{x}}_i(\tau) = \frac{1}{h_i\,(t_f - t_0)} \sum_{k=0}^{N_c} \boldsymbol{x}_{i,k}\,\frac{\mathrm{d}\bar{\ell}_k(\tau)}{\mathrm{d}\tau} \tag{2.27}$$

2.3.1 Placement of collocation points

The above account of the underlying math of the collocation method, does not include how collocation points are placed. A number of methods are available, to decide upon the location of the collocation points. The most common being *Radau* quadrature and *Legendre-Gauss*. Both of these are supported in JModelica.org, with Radau being the default. By using Radau, the collocation points within an element are placed at the approximate local times, τ, given by **Table 2.1**. One collocation point is always placed at beginning of the element; $\tau_1 = 0$. Leaving $N_c - 1$ free collocation points. These are placed at the roots of the following polynomial:

$$\frac{P_{N_c-1}(\tau) + P_{N_c}(\tau)}{1+\tau} \tag{2.28}$$

Where $P(\tau)$ are Legendre polynomials – the first few given as:

$$P_0(\tau) = 1, \quad P_1(\tau) = \tau \tag{2.29}$$

$$P_2(\tau) = \frac{1}{2}(3\tau^2 - 1), \quad P_3(\tau) = \frac{1}{2}(5\tau^3 - 3\tau) \tag{2.30}$$

$$P_4(\tau) = \frac{1}{8}(35\tau^4 - 30\tau^2 + 3) \tag{2.31}$$

By increasing the number of collocation points, the accuracy of the transcription increases; but so does the dimension of the resulting NLP. The number of $\mathbb{R}$ optimization variables, n_Z, is determined as:

$$n_Z = (1 + N_e\, N_c)\, n_z + (N_e - 1)\, n_x \tag{2.32}$$

Where N_e is the number of elements, N_c the number of collocation points, n_z the number of system variables and n_x the number of states[9].

N_c	τ_1	τ_2	τ_3	τ_4
2	0	2/3		
3	0	0.35	0.84	
4	0	0.22	0.59	0.91

Table 2.1: *Approximate placement of collocation points using Radau quadrature; at local times, τ, within a single element.*

2.4 Model Predictive Control

Many control problems stem from the desire to design stabilizing feedback that minimizes a performance criterium and does not violate given constraints. A closed solution to such a problem is often not obtainable – even without considering constraints[7]. One approach is solving an open-loop OCP to a given state, applying only a part of the optimal control input to the system, and then repeating this process. This control strategy is what defines MPC. Thus, MPC ties up open-loop optimal control in a loop consisting of the following steps:

- Obtain state or state estimate, $\boldsymbol{x}_k$
- Find control input, $\boldsymbol{u}_k$, by solving an OCP
- Apply control input to process

The OCP is identical to **Equation (2.10)**, but with a shifting optimization horizon. That is, t_0 and t_f, which define the start and end of the definite integral, change with the current time, t. Generally, the notion of a prediction horizon, t_H, is introduced. The prediction horizon determines, how far out in the future, the process behavior is to be predicted. Thus, at time t_k, the optimal control problem is considered from $t_0 = t_k$ to $t_f = t_k + t_H$. The concept of MPC and the shifting optimization horizon is depicted in **Figure 2.5**.

As MPC is a discrete control method, it features a sample time, $t_s = t_k - t_{k-1}$. During a sample period, the control input, $\boldsymbol{u}_k$, is often held constant. As the solution to the OCP is an optimal input trajectory, $\boldsymbol{u}(t)$, one has to decide upon how to pick $\boldsymbol{u}_k$. This can however be handled, by specifying in the OCP, that $\boldsymbol{u}(t)$ should be piecewise constant, only changing at specific time instants. The optimal control input will thus be a discrete input sequence $\boldsymbol{u} = \{\boldsymbol{u}_1, \boldsymbol{u}_2, ..., \boldsymbol{u}_N\}$. A common approach, is to specify $\boldsymbol{u}(t)$ to be piecewise constant, only changing at time instant that correspond to the MPC sample period of t_s; then $\boldsymbol{u}_k$ is chosen as $\boldsymbol{u}_1$.

The algorithm is identical for both NMPC and LMPC – the only difference is whether the models and constraints are nonlinear or linear.

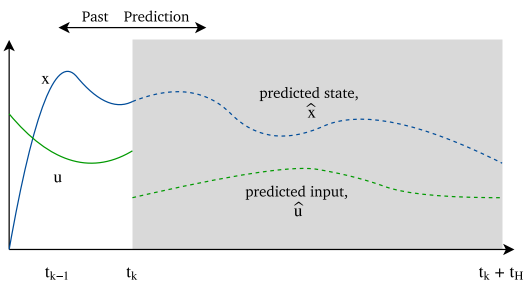


Figure 2.5: *The concept of MPC, graphically represented by the fictive state, x, and input signal, u.*

2.5 State estimation

Generally, it can not be expected, in a control application, that all states are measurable. Therefore, state estimation is almost always required for a successful design. In this particular case of production portfolio control, one reason to apply state estimation is due to the assumption that not everything is measurable at the consumers, even with the introduction of Automatic Meter Reading (AMR) for both power, heating and water supply. Several options for state estimation are available, but a popular choice in control applications is the Kalman Filter.

Given a discrete-time linear system with state and measurement noise:

$$\boldsymbol{x}_{k+1} = \boldsymbol{A}\boldsymbol{x}_k + \boldsymbol{B}\boldsymbol{u}_k + \boldsymbol{w}_k \quad \wedge \quad \boldsymbol{w}_k \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{Q}) \tag{2.33}$$

$$\boldsymbol{y}_k = \boldsymbol{C}\boldsymbol{x}_k + \boldsymbol{D}\boldsymbol{u}_k + \boldsymbol{v}_k \quad \wedge \quad \boldsymbol{v}_k \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{R}) \tag{2.34}$$

Where $\boldsymbol{A} \in \mathbb{R}^{n\times n}, \boldsymbol{B} \in \mathbb{R}^{n\times m}, \boldsymbol{C} \in \mathbb{R}^{p\times n}$ and $\boldsymbol{D} \in \mathbb{R}^{p\times m}$; letting n be the number of states, m the number of inputs and p the number of outputs. The matrices, $\boldsymbol{Q}$ and $\boldsymbol{R}$, are covariance matrices. The Kalman Filter finds a state estimate, $\hat{\boldsymbol{x}}_k$, that minimizes the mean square estimation error:

$$\underset{\hat{\boldsymbol{x}}_k}{\text{minimize}} \; \mathbb{E}[(\boldsymbol{x}_k - \hat{\boldsymbol{x}}_k)^T \, \boldsymbol{M} \, (\boldsymbol{x}_k - \hat{\boldsymbol{x}}_k)] \quad \wedge \quad \boldsymbol{M} > 0 \tag{2.35}$$

The following notation is used, where $\boldsymbol{Y}_k$ denotes all, including the k-th, measurements:

$$\hat{\boldsymbol{x}}_{k|k} \triangleq \mathbb{E}[\boldsymbol{x}_k | \boldsymbol{Y}_k] \tag{2.36}$$

$$\hat{\boldsymbol{x}}_{k|k-1} \triangleq \mathbb{E}[\boldsymbol{x}_k | \boldsymbol{Y}_{k-1}] \tag{2.37}$$

$$\boldsymbol{P}_{k|k} = \mathbb{E}[(\boldsymbol{x}_k - \hat{\boldsymbol{x}}_{k|k})(\boldsymbol{x}_k - \hat{\boldsymbol{x}}_{k|k})^T] \tag{2.38}$$

$$\boldsymbol{P}_{k|k-1} = \mathbb{E}[(\boldsymbol{x}_k - \hat{\boldsymbol{x}}_{k|k-1})(\boldsymbol{x}_k - \hat{\boldsymbol{x}}_{k|k-1})^T] \tag{2.39}$$

The Kalman Filter works like an observer:

$$\hat{\boldsymbol{x}}_{k|k-1} = \boldsymbol{A}\hat{\boldsymbol{x}}_{k-1} + \boldsymbol{B}\boldsymbol{u}_k \tag{2.40}$$

$$\hat{\boldsymbol{y}}_{k|k-1} = \boldsymbol{C}\hat{\boldsymbol{x}}_{k|k-1} + \boldsymbol{D}\boldsymbol{u}_k \tag{2.41}$$

$$\hat{\boldsymbol{x}}_{k|k} = \hat{\boldsymbol{x}}_{k|k-1} + \boldsymbol{K}_k(\boldsymbol{y}_k - \hat{\boldsymbol{y}}_{k|k-1}) \tag{2.42}$$

Where the observer gain, $\boldsymbol{K}_k$, is denoted the Kalman gain.

The implementation of a Kalman Filter is often done, by splitting it up in two steps; a *time update* step and a *measurement update* step:

Time update:

$$\hat{\boldsymbol{x}}_{k|k-1} = \boldsymbol{A}\hat{\boldsymbol{x}}_{k-1|k} + \boldsymbol{B}\boldsymbol{u}_k \tag{2.43}$$

$$\boldsymbol{P}_{k|k-1} = \boldsymbol{A}\,\boldsymbol{P}_{k-1|k}\,\boldsymbol{A}^T + \boldsymbol{Q} \tag{2.44}$$

Measurement update:

$$\hat{\boldsymbol{y}}_{k|k-1} = \boldsymbol{C}\hat{\boldsymbol{x}}_{k|k-1} + \boldsymbol{D}\boldsymbol{u}_k \tag{2.45}$$

$$\boldsymbol{K}_k = \boldsymbol{P}_{k|k-1}\boldsymbol{C}^T(\boldsymbol{C}\boldsymbol{P}_{k|k-1}\boldsymbol{C}^T + \boldsymbol{R})^{-1} \tag{2.46}$$

$$\hat{\boldsymbol{x}}_{k|k} = \hat{\boldsymbol{x}}_{k|k-1} + \boldsymbol{K}_k(\boldsymbol{y}_k - \hat{\boldsymbol{y}}_{k|k-1}) \tag{2.47}$$

$$\boldsymbol{P}_{k|k} = (\boldsymbol{I} - \boldsymbol{K}_k\,\boldsymbol{C})\boldsymbol{P}_{k|k-1}(\boldsymbol{I} - \boldsymbol{K}_k\,\boldsymbol{C})^T + \boldsymbol{K}_k\,\boldsymbol{R}\boldsymbol{K}_k^T \tag{2.48}$$

As such, the implementation is straight forward, given a linear system – and the design itself will deal with the choice of appropriate covariance matrices.

However, problems arise when dealing with nonlinear systems; as assumed in this thesis. One approach could be to linearize the system in question, and use the Kalman Filter as is. Performance may be acceptable, but it may also happen, that critical information is lost in the linearization, which will influence the estimation performance.

The Extended Kalman Filter (EKF) can be used for state estimation on nonlinear systems. The EKF builds upon the linear Kalman Filter with two additions: using the nonlinear model when possible and linear models where necessary – by employing online linearization. Thus, for each iteration of the EKF, a new linear system is obtained.

For the EKF, **Equation (2.43)** in the time update step is replaced by:

$$\hat{\boldsymbol{x}}_{k|k-1} = \boldsymbol{f}(\hat{\boldsymbol{x}}_{k|k-1}, \boldsymbol{u}_k) \tag{2.49}$$

And **Equation (2.45)** in the measurement update step is replaced by:

$$\hat{\boldsymbol{y}}_{k|k-1} = \boldsymbol{g}(\hat{\boldsymbol{x}}_{k|k-1}, \boldsymbol{u}_k) \tag{2.50}$$

Where $\boldsymbol{f}$ and $\boldsymbol{g}$ constitute the nonlinear model. The EKF is not optimal in the sense the linear Kalman Filter is and stability is not guaranteed for the observer – but this thesis will however consider the EKF for state estimation, given the straight-forward implementation.

2.6 JModelica.org

JModelica.org is a tool including functionality for modeling, simulating and optimization, for large-scale dynamic systems[12]. The most prominent feature being the possibility of solving Dynamic Optimization Problems (DOPs), involving models written in the Modelica modeling language. This is done, by utilizing the Modelica language extension, Optimica. Optimica adds language constructs to Modelica, that allow the formulation of optimization problems. The functionality of JModelica.org is exposed to the user, via an API in Python. Relevant parts of the process, from Modelica and Optimica code, to a solved optimization problem, are depicted in **Figure 2.6**.

First, the JModelica.org compiler creates an Abstract Syntax Tree (AST), an internal representation of the Modelica and Optimica code. An AST is a data structure, representing all language constructs found in the given source files, as a tree – it is an essential part of any compiler. Using the AST, several steps are taken; including resolving class inheritance to obtain one flat model and alias elimination to minimize the number of equations; general symbolic transformations, handled by any Modelica compiler. The resulting AST is a symbolic representation of the DOP.[12]

This AST is then used, to create a CasADi representation of the DOP. CasADi is a symbolic framework for algorithmic/automatic differentiation for use in numeric optimization; effectively allowing the construction of symbolic expressions that can be efficiently differentiated. CasADi expressions are created, mapping to the expressions of the DOP. The steps until now, has effectively mapped Modelica and Optimica code to CasADi expressions, and the CasADi expressions can now be used to obtain derivatives.[12]

Transcription to an NLP, is now handled by a direct collocation algorithm, working with the CasADi expressions. The resulting NLP is then numerically solved using an external solver – with IPOPT being the default – and the result is delivered to the user.[12]

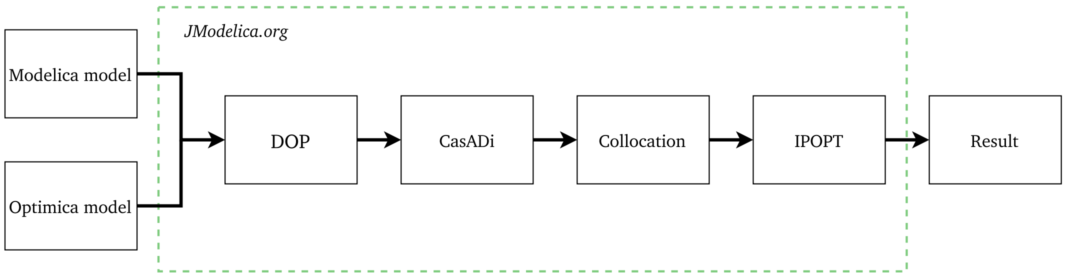


Figure 2.6: *From Modelica and Optimica code to a solved optimization problem. The box outlines what is handled behind the scenes by JModelica.org.*

2.6.1 Functional Mockup Interface

The simulation part of JModelica.org is tightly coupled to the Functional Mockup Interface (FMI). The FMI is a tool independent standard for model exchange and co-simulation of dynamic models. One particular benefit is that e.g. a Modelica model created using Dymola, can be easily packaged as an Functional Mockup Unit (FMU) and then used in different tools, employing this standard. One use case would be to use Dymola and Modelica purely for modeling, then package the model up as an FMU and use the FMU in a control design, for e.g. performance verification through simulations on given FMU.

JModelica.org includes a Python API, *pyfmi*, to handle FMUs; including simulation and the possibility of extracting linear models.

Modeling 3

The modeling in this thesis is undertaken using the object-oriented modeling language, Modelica. The object-oriented approach is especially beneficial for a model scenario as this one, as the combined model includes several sub-models that can be modeled independently. By keeping to a fixed set of interfaces, production units can later be changed for different ones, e.g replacing them with refined models including added dynamics or interesting nonlinearities worth investigating. This approach is a big incentive, to keep the models very simple at first and then iteratively expand.

To allow for optimization on nonlinear models, the problem has to be *well-posed*. In this case, well-posed is a requirement to the model, as it has to be C^2 continuous; first and second order derivatives are continuous. This requirement stems from the fact, that JModelica.org employs a gradient-based method, based on Newton's method, to find a solution to first order optimality conditions[12].

This requirement raises an issue for the modeling part of the project. Modeling of the system is done in the Modelica language. The advantages in using Modelica especially lies in the object-oriented modeling approach with great emphasis on reuse of components and the Modelica Standard Library (MSL) with already available components and interfaces for a wide range of applications. Thus, to model a district heating system in Modelica, one could use the components of the Fluid Library and Media Library (part of the MSL) to model the fluid dynamics of water in pipes. This would both lower the amount of work needed to put up a model, but by using interfaces in the MSL, lock-in to custom interfaces is avoided, which makes the project much more available for future work.

The problem is, however, that the MSL does not adhere to the C^2-requirement, eliminating its apparent use in the project. A well-acknowledged third-party library would be a viable solution to limit the lock- in; however none seem to exist. Thus, basic fluid components and interfaces have to be constructed for this project, to allow for the use together with JModelica.org.

Now, other than lock-in to a set project defined components and interfaces, a certain level of accuracy will also be lost, in terms of how well the model fits reality. This is also undesirable. Thus, to still have a *reality* to simulate against, two models will be considered; a C^2 *model* that is usable in an optimization context and a *simulation model*. The simulation model will be modeled using components from the MSL. Due to the object-oriented nature of Modelica, a certain level of reuse between the two models is possible (e.g. sharing parameters).

The chapter will focus on describing the C^2 model, as this has been the primary model in the project. Afterwards, the differences between the C^2 model and the simulation model will be outlined. Throughout the modeling, it has been attempted to apply some of the principles of obtaining fast simulation speeds for Modelica models, discussed in [13]. Among these principles, is avoiding algebraic loops by decoupling through additional states; this is sometimes referred to, as employing a *staggered grid*. Algebraic loops, formed by interdependent equations, require an iterative solution which is time expensive[13].

3.1 Fluid library

A Modelica fluid component library is built, to allow the construction of a model on which optimization is possible – adhering to the C^2 requirement. It is required to model the behavior of subcooled water flowing in the system and the heating of said water. As only subcooled water is modeled, it is assumed that the specific heat capacity, c_p, and density, ρ, are constant.

3.1.1 Connectors

The Modelica component library created contains a set of Modelica connectors, defining the interface. These connectors are to be used on any component that is to take part in the combined district heating system. The fluid connectors define the three states; p, $\dot{m}$ and h at that connector – pressure, mass flow and specific enthalpy, respectively.

Specific enthalpy is defined as:

$$h \triangleq c_p\,(T - T_0) + \frac{p}{\rho} \quad \wedge \quad T_0 = 273.15\,\mathrm{K} \tag{3.1}$$

The above equation does not violate the requirement of C^2 models. It would however violate the C^2, if one considers phase change, as this introduces discontinuities. Disregarding phase changes limits the case to constant c_p and constant ρ. Considering the case of constant pressure as well[1], the equation can be rewritten as:

$$h = c_p\,(T - T_0) + \text{constant} \tag{3.2}$$

The constant relates to the compressibility of the liquid, and is unnecessary considering energy balances. The notion of enthalpy can thus be simplified to the following very operational and linear function of temperature:

$$h(T) = c_p\,(T - T_0) \tag{3.3}$$

3.1.2 Pipe

The pipe is connected to boundary components using the liquid water connectors – water flows in and water flows out. The pipe component is modeled as a component with mass storage and energy storage; it thus potentially has two states, but the pipe is modeled with a constant volume and the density is assumed constant, thus the mass is also constant. The equations governing the pipe model are as follows:

$$m = V\,\rho \tag{3.4}$$

$$U = m\,c_p\,(T - T_0) \approx m\,h_{\text{out}} \tag{3.5}$$

[1]Water is approximately incompressible.

The total energy in the pipe, U, is the mass, m, multiplied with the enthalpy of the water leaving the pipe. The state equations are given as:

$$\dot{m} = \dot{m}_{\text{in}} + \dot{m}_{\text{out}} \quad (= 0) \tag{3.6}$$

$$\dot{U} = \dot{m}_{\text{in}}\, h_{\text{in}} + \dot{m}_{\text{out}}\, h_{\text{out}} + \dot{Q} \tag{3.7}$$

The change in energy allows for heat added to or removed from the water through $\dot{Q}$. Given the linear relationship between U and T, with m and c_p constant, T can be used in the state equations instead of U:

$$\dot{T}\; m\; c_p = \dot{U} \tag{3.8}$$

The pipe, complete with two fluid connectors and a heat port, is depicted in **Figure 3.1** with the icon representation in Modelica.

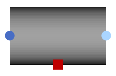


Figure 3.1: *The pipe model's graphical representation.*

3.1.3 Valve

A simple valve with a linear opening characteristic is modeled as a static component, connected to boundary components using the liquid water connectors. The valve behavior is given by the following equations:

$$\dot{m} = u\, \sqrt{k\, |\Delta p|}\; \text{sgn}(\Delta p) \tag{3.9}$$

Where the valve opening, u, is in the interval $[0;\, 1]$, k is given as:

$$k = \frac{\dot{m}^2_{\text{nominal}}}{u^2_{\text{nominal}}\; \Delta p_{\text{nominal}}} \tag{3.10}$$

The pressure drop is given as the pressure difference between the two connectors:

$$\Delta p = p_{\text{in}} - p_{\text{out}} \tag{3.11}$$

As the valve represents an isenthalpic process, there is no change in energy and thus:

$$h_{\text{in}} = h_{\text{out}} \tag{3.12}$$

The valve, complete with two fluid connectors and an opening signal, is depicted in **Figure 3.2**.

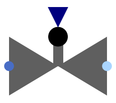


Figure 3.2: *The valve model's icon representation in Modelica.*

3.1.4 Pump

A simple pump is modeled as a static ideal component and is governed by the following equations:

$$h_{\text{out}} = h_{\text{in}} \tag{3.13}$$

$$\dot{m}_{\text{out}} = -\dot{m}_{\text{in}} = \dot{m}_{\text{set}} \tag{3.14}$$

Where $\dot{m}_{\text{set}}$ is an input to the pump. The pump, complete with two fluid connectors and a mass flow setpoint signal, is depicted in **Figure 3.3**.

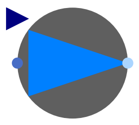


Figure 3.3: *The pump model's icon representation in Modelica.*

3.1.5 Sources and sinks

To allow for non-cyclic simulation experiments, a set of boundary conditions in the form of source/sink components have been created. These components have a single liquid water connector with two of the states fixed. Specifically, a mass flow boundary and a pressure boundary is constructed. The mass flow boundary has the mass flow and the outlet temperature (and thus enthalpy) fixed. The pressure boundary has the pressure and the outlet temperature fixed. The icon representation is depicted in **Figure 3.4**.

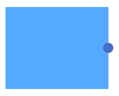


Figure 3.4: *The graphical representation of a source or sink model.*

3.2 Consumer

Each consumer is modeled as a house with floor heating. The floor heating is supplied with hot water from the district heating system. Heat is then transferred from the floor heating pipes to the floor mass itself and from the floor to the surrounding air. The consumer has a variable desired room temperature. To be able to control the temperature to reach the desired temperature, a valve is included in the floor heating, to change the mass flow of the water in the pipes. A simple SISO controller can then be implemented on top of the floor heating system, and the controller can be adjusted to the desired consumer dynamics.

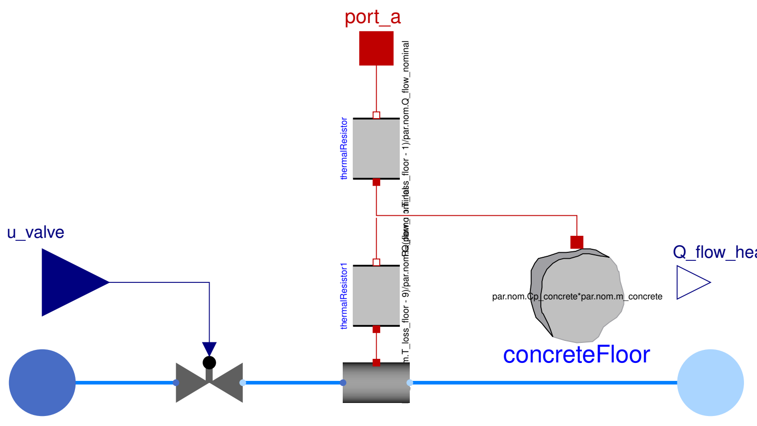
Several types of consumers exists, but as stated, this consumer will be a house. The average annual heat consumption for detached houses, terraced houses and flats, in Denmark, is 12 524 kW h[14]. From this, the average heat flow can be computed:

$$\dot{Q}_{\text{consumer}} = \frac{12.524\,\text{MW h}}{1\,\text{year}} \approx 1.429\,68\,\text{kW h/h} \approx 1.4\,\text{kW} \tag{3.15}$$

This heat flow is the basis of the consumer model; from this the consumer model will be nominally parameterized.

3.2.1 Floor heating

The floor heating model has one input and one output; a valve opening signal, u_{valve}, and the heat flow leaving the floor, $\dot{Q}$. Furthermore, it has a heat flow interface and fluid interfaces. The graphical component-based model of the floor heating is depicted in **Figure 3.5a** together with the icon representation in **Figure 3.5b**.



(a) *Model*

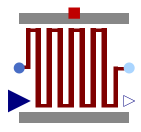

(b) *Icon*

Figure 3.5: *The floor heating model's component model and icon representation in Modelica.*

The floor heating model contains a valve component, described in **Section 3.1.3**, characterized by the constant k, given in **Equation (3.9)**. The constant is determined by nominal mass flow, nominal opening of the valve and the nominal pressure difference.

The nominal mass flow is determined by the nominal heat consumption for the consumer, 1400 W, the nominal temperature drop over the consumer, and by knowing the specific heat capacity of water. The nominal temperature drop is estimated to be 40 °C. The choice of 40 °C is due to the fact, that a minimum cooling requirement is posed by the district heating supplier. The minimum cooling requirement from Sønderborg Fjernvarme is set at 30 °C[8].

$$\dot{m}_{\text{nominal}} = \frac{\dot{Q}_{\text{nominal}}}{C_{\text{p, water}}\,\Delta T_{\text{nominal}}} = \frac{1400\,\text{W}}{4186\,\text{J/(kgK)}\;40\,^{\circ}\text{C}} = 8.36 \times 10^{-3}\,\text{kg/s} \tag{3.16}$$

The nominal pressure drop over the valve is given to be 0.6 bar and the nominal opening of the valve, u_{nominal}, is set to 50 %. Choosing a nominal supply temperature of $T_{\text{supply,nominal}} = 80\,^{\circ}\text{C}$, equivalent to Sønderborg Fjernvarme, the nominal return temperature thus becomes $T_{\text{return,nominal}} = 40\,^{\circ}\text{C}$.

The floor heating piping, modeled by a pipe component described in **Section 3.1.2**, is parameterized by volume, V, which is specified from the length and the diameter of the piping. These are loosely chosen as 100 m and 0.02 m, respectively. Furthermore, the floor heating includes a heat capacitance which models the concrete floor, in which the piping is laid out. This is parameterized by the mass of the concrete and by knowing the specific heat capacity of concrete to be:

$$c_{\text{p, concrete}} = 960\,\text{J/(kgK)} \tag{3.17}$$

The mass of the concrete is chosen, based on a desired temperature response. It should take approximately 10 h to introduce a 10 °C temperature change of the concrete. This is, by simulations, obtainable at a mass of $m_{\text{concrete}} = 360\,\text{kg}$.

The concrete floor also includes two thermal resistors[2], modeling a temperature difference over the concrete floor. The thermal resistors model a nominal temperature difference of 10 °C; this together with the nominal return temperature determines the nominal temperature of the floor:

$$T_{\text{floor,nominal}} = T_{\text{return,nominal}} - 10\,^{\circ}\text{C} = 40\,^{\circ}\text{C} = 30\,^{\circ}\text{C} \tag{3.18}$$

Knowing that the heat transfer through the resistors is nominally 1400 W, the total resistance becomes:

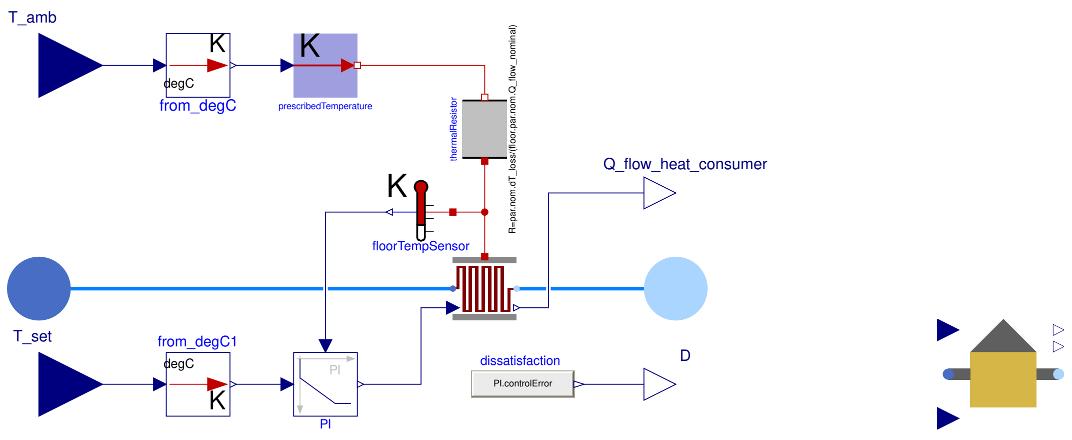
$$R_{\text{tot,concrete}} = \frac{10\,\text{K}}{1400\,\text{W}} = 7.1 \times 10^{-3}\,\text{K/W} \tag{3.19}$$

[2]The reason for employing two resistors, instead of just one, is to use the staggered grid approach, briefly introduced in the chapter introduction.

3.2.2 House

The floor heating model is used in a house model, with the graphical component-based model of the house depicted in **Figure 3.6a** together with the icon representation in **Figure 3.6b**. The house model includes two inputs and two outputs. The first input is a setpoint temperature, T_{set}, which is the desired temperature at the surface of the floor. The second input is an ambient temperature input, T_{amb}, determining the temperature outside the house. The first output gives the current heat consumed and the second output is the error between desired temperature and actual temperature, at the surface of the floor – useful as a simple measure of Quality-of-Service (QoS).

The house model is connected to the rest of the district heating system by a set of fluid interfaces. These fluid interfaces are routed directly to a floor heating model included in the house model.



(a) *Model* **(b)** *Icon*

Figure 3.6: *The house model's component model and icon representation in Modelica. The PI regulator uses the output from the floor temperature sensor as measurement signal.*

To model a temperature difference of $T_{\text{floor,nominal}} - T_{\text{amb,nominal}} = 20\,^{\circ}\text{C}$, between outside and the top of the floor, a thermal resistance is used – again knowing the nominal heat consumption:

$$R_{\text{wall}} = \frac{20\,^{\circ}\text{C}}{1400\,\text{W}} = 14.3 \times 10^{-3}\,\text{K/W} \tag{3.20}$$

The floor heating is controlled using a PI controller, tuned until a satisfactory response was obtained. The PI controller is given a gain of 1×10^{-3} and a time constant for the integrator of 360 s.

The entire consumer model only contains a single constraint; limiting u_{valve} to be between 0 and 1. This inequality constraint is posed in the Modelica code, by utilizing `min` and `max` attributes.

3.2.3 Scaling

The above model considers a single consumer, but it is desirable to include the possibility of scaling the model, to represent several thousand consumers – e.g. 10 000 as is the case for Sønderborg Fjernvarme. This is handled by scaling the nominal heat flow by $N_{\text{consumers}}$, which is the basis of determining several other nominal characteristics, scaling the thermal capacitance of the concrete floor by $N_{\text{consumers}}$ and the floor heating pipe length by $N_{\text{consumers}}$.

The reason why the thermal capacitance also has to be scaled, lies in the desire not to alter the dynamics in the model, but only the amplitude. Here the amplitude relates to the heat consumption and the mass flows. In **Figure 3.7**, the thermal capacitance modeling the concrete floor is depicted, in conjunction with one of the thermal resistors. A direct analogy can be drawn to an electrical circuit, which is also depicted in **Figure 3.7**. Both systems in their respective domain, are characterized as first order systems, governed by a differential equation of the same form. For the thermal system, the equation is given as:

$$C\dot{T} = \frac{(T_{\text{in}} - T)}{R} \tag{3.21}$$

Where R is the thermal resistance and C is the thermal capacitance. By rearranging the equation, the following is obtained:

$$RC\dot{T} = -T + T_{\text{in}} \tag{3.22}$$

Thus, a time constant can be identified, as $\tau = RC$, just as for the electrical equivalent. Now, if only scaling the heat flow by $N_{\text{consumers}}$, the thermal resistance will become $N_{\text{consumers}}$ times larger, resulting in τ being $N_{\text{consumers}}$ times smaller. To avoid this, C has to be $N_{\text{consumers}}$ larger, such that the time constant is unchanged through the scaling.

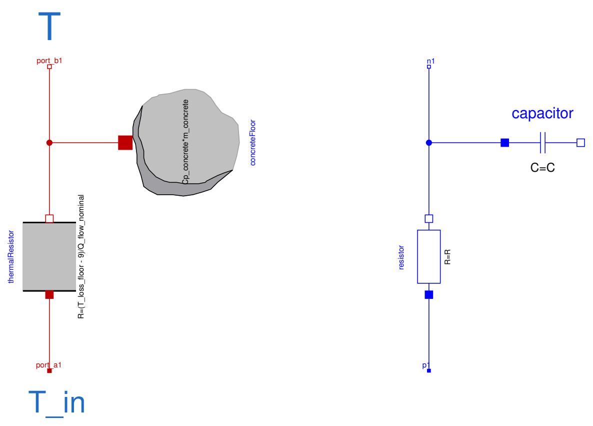


Figure 3.7: *The thermal capacitance modeling the concrete floor, together with one of the thermal resistors, modeling the temperature drop through the floor. Comparison to an electrical equivalent; a standard RC circuit.*

3.2.4 Simulations

A few simulations of the house model with fixed boundary conditions are undertaken, to investigate the response to some known inputs. The boundary conditions are set, so that the house is supplied with heated water at a temperature of 80 °C at 6 bar and so that the pressure at the outlet of the house is at 5.4 bar, providing the nominal pressure drop of 0.6 bar. The ambient temperature is fixed to 10 °C and the setpoint temperature is provided as a step, which starts at 30 °C and jumps down to 20 °C at $t = 11$ h.

The simulation is performed for both a single consumer ($N_{\text{consumers}} = 1$) and for 10000 consumers ($N_{\text{consumers}} = 10000$), to validate that the model can be scaled successfully. Results are given in **Figure 3.8**. There is a noticeable undershoot, which is undesirable for an actual consumer – but nothing which conflicts with the purpose of this model. The results indicate that the scaling is successful, as the responses are identical for one and for 10000 consumers – it only changes the total heat flow consumed and the total mass flow. To compare the two simulations, the mass flow and the heat flow is, for the case with just one consumer, multiplied with 10000.

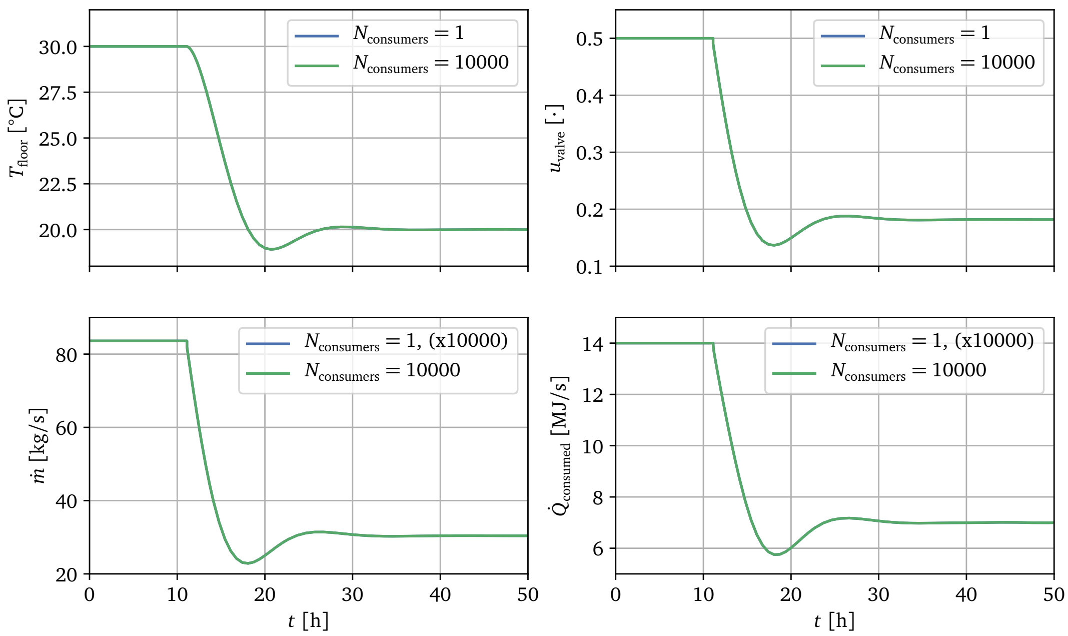


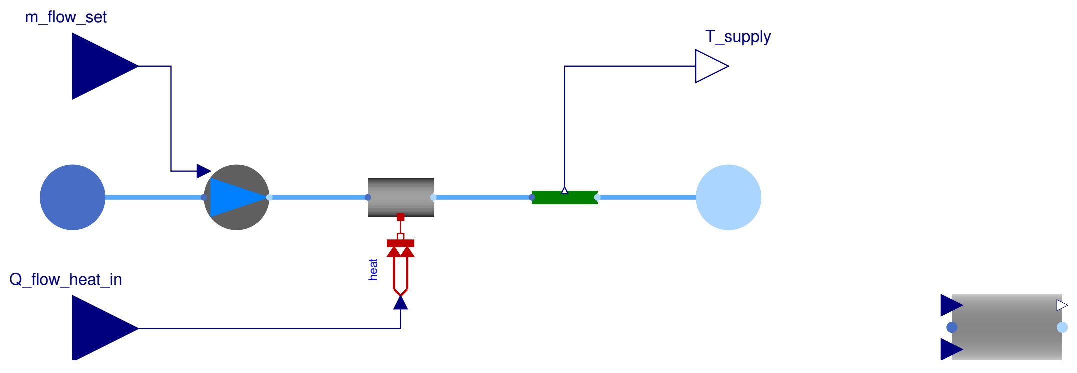
Figure 3.8: *Response to a step in desired floor surface temperature, from* 30 °C *to* 20 °C*. Showing both the floor temperature, the valve control signal – the responding mass flow and consumer heat flow. The response is both for a single consumer and 10000 consumers.*

3.3 Base production unit

The district heating production units share the same base component, described in this section. The base production unit contains two inputs and a single output; a mass flow setpoint, a heat flow input and a measurement of the forward temperature available for control. Included in the base production unit is a pump, a pipe and a temperature sensor. The model is depicted in **Figure 3.9** and is purely based on instantiating and connecting existing models.

The mass flow setpoint is used directly in the pump component, to control the mass flow through the base production unit. Together with the forward temperature measurement, this allows a controller on top of the unit, to control forward temperature by changing the mass flow. The heat flow input is used directly on the pipe component, and determines the amount of heat supplied to it.

As no dynamics are included in the pump model or the temperature sensor model, the production unit is only parameterized by the volume of the pipe. This is chosen as $V = 0.5\,\mathrm{m}^3$.



(a) *Model* **(b)** *Icon*

Figure 3.9: *The base production unit model's component model and icon representation in Modelica.*

3.4 Combined heat and power plant

The production portfolio contains a CHP, a unit which generates both heat and power. The CHP contains two inputs, a load setpoint and a forward temperature setpoint. The output is the produced power and the produced heat. The CHP is connected to the rest of the system via a set of fluid connectors. The model of the CHP is given in **Figure 3.10** and includes a base production unit component, described in **Section 3.3**, and a PI controller.

The amount of power produced and the amount of heat produced is directly determined by the load. The CHP has been dimensioned, such that for each W power produced, the plant produces twice the amount of heat. With a fixed maximum power production of 10 MW the plant can produce a maximum of 20 MJ/s heat. The load determines the heat produced, such that a load of 100 % correspond to 20 MJ/s heat – and in turn also 10 MW power. This mode of operation, where heat production is primary and power production is a secondary product, is often referred to as back-pressure.

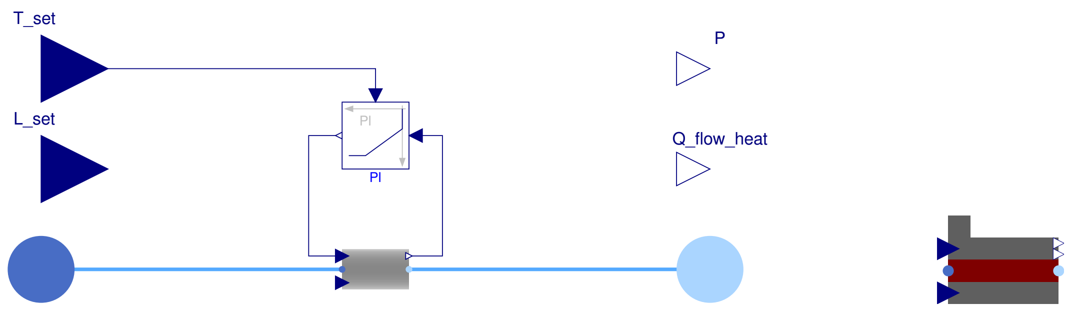
The load variable has been constrained to be within 0.3 and 1; thereby complying with the maximum heat and power productions – and setting a minimum heat and power production, to model that the CHP is always running. The maximum and minimum load constraints, are imposed, by setting `min` and `max` attributes on the variable in Modelica.

The load is not static, but is governed by the following differential equation:

$$\tau \, \dot{L} = -L + L_{\text{set}} \tag{3.23}$$

Thus, by adjusting the time constant τ, it can be determined how fast the unit will react to a change in load setpoint. The time constant is chosen as $\tau = 3600\,\text{s}$. The heat produced is added to the pipe, and consequently added to the water flowing through it.

The forward temperature is controlled by adjusting the mass flow. Thus, the forward temperature is measured and fed into a PI controller where the control signal is a mass flow setpoint. The PI controller is tuned to obtain a satisfactory response. The PI controller is given a gain of 1 and a time constant for the integrator of 360 s.



(a) *Model* **(b)** *Icon*

Figure 3.10: *The CHP model's component model and icon representation in Modelica. The unconnected inputs and outputs, are governed by equations specified in the underlying Modelica code.*

3.4.1 Simulations

Two simulations are performed, to see how the CHP responds to known inputs and to verify that it behaves as desired. In both cases, the boundary conditions are set to: an inlet temperature of 40 °C and inlet pressure of 5.4 bar – the outlet pressure is fixed at 6 bar.

The first simulation starts with $T_{\text{set}} = 80\,°\text{C}$ and $L_{\text{set}} = 1$. At $t = 5\,\text{h}$, L_{set} is ramped down to 0.5, over a period of 30 min. Results are given in **Figure 3.11**. The first graph shows how the production of both power and heat is ramped down, as the load is ramped down. The second graph shows how the mass flow is lowered to half; this is the temperature controller acting in response to less heat being added to the water.

The second simulation has the load setpoint fixed at $L_{\text{set}} = 1$ and instead introduces a step in the temperature setpoint. The step is introduced at $t = 5\,\text{h}$, where the temperature setpoint is changed from 80 °C to 90 °C. Results are given in **Figure 3.12**. The change in reference has the temperature controller turn down the mass flow, in order to reach the desired temperature.

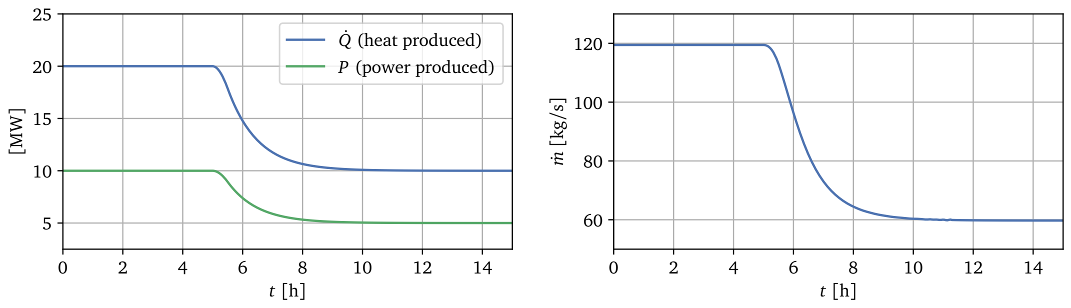


Figure 3.11: *Simulation where CHP load is ramped down from 1 to 0.5 over 5 h. Production of both power and heat is correspondingly lowered – and to keep the desired forward temperature, the mass flow through the CHP is turned down.*

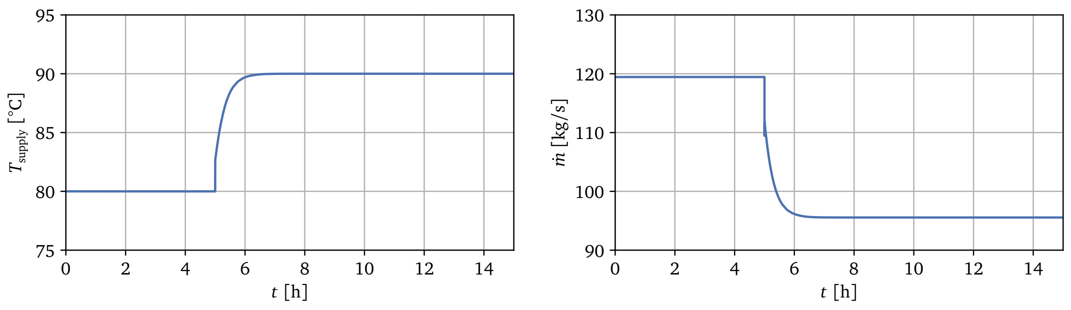


Figure 3.12: *Simulation where the forward temperature setpoint for the CHP is changed from 80 °C to 90 °C. This has the the temperature controller lower the mass flow through the CHP, to reach the new desired temperature.*

3.5 Compression heat pump

The production portfolio contains a compression HP, which can produce heat by using electricity. The HP model, depicted in **Figure 3.13**, has two inputs and one output. One input is a forward temperature setpoint, T_{set}. The other input is the electrical power supplied to it, P. The output is heat produced, $\dot{Q}_{\text{produced}}$. The HP is connected to the rest of the district heating system by fluid connectors, but it also includes a heat connector, as an interface to a heat reservoir. The HP model closely resembles the CHP model, by including the same base production unit, described in **Section 3.3**.

The main equations governing the HP:

$$\dot{Q}_{\text{produced}} = P + \dot{Q}_{\text{reservoir}} \tag{3.24}$$

$$\dot{Q}_{\text{reservoir}} = P\,(\text{COP} - 1) \tag{3.25}$$

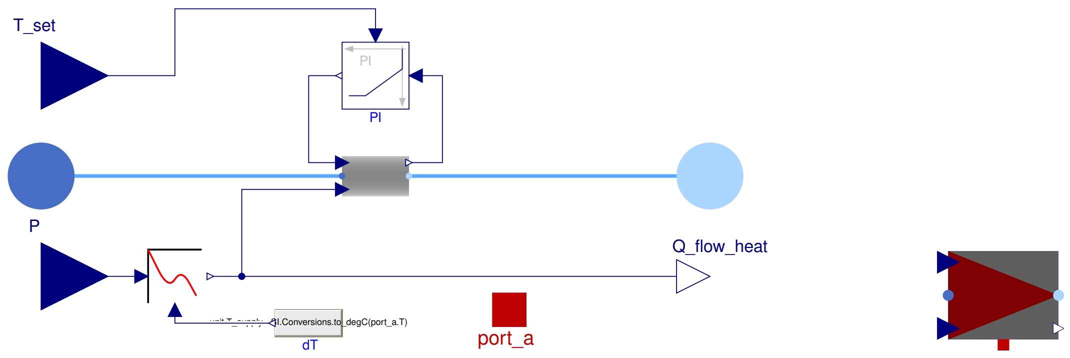
Where $\dot{Q}_{\text{reservoir}}$ is the amount of heat taken from the connected reservoir. The connected reservoir will for this portfolio be the ambient temperature. The Coefficient of Performance (COP), is modeled as a replaceable Modelica function; allowing both for constant COP or a COP dependent on e.g. temperature. As default, the COP will be given by the following linear function:

$$\text{COP}(T_{\text{supply}}, T_{\text{reservoir}}) = -0.1\,(T_{\text{supply}} - T_{\text{reservoir}}) + 10 \tag{3.26}$$

$$\text{COP}(80,10) = 3, \quad \text{COP}(80,0) = 2, \quad \text{COP}(80,20) = 4 \tag{3.27}$$

Thus, in nominal conditions with the ambient temperature varying in the interval 0 °C to 20 °C, the COP will vary between 2 and 4.

The power input, P, has been limited to be within 100 kW and 5 MW; thus modeling that the HP is always running. These constraints have been imposed by setting `min` and `max` attributes on the power input variable in Modelica.



(a) *Model* **(b)** *Icon*

Figure 3.13: *The HP model's component model and icon representation in Modelica.*

3.5.1 Simulations

A simulation is performed, with a fixed forward temperature setpoint of $T_{\text{set}} = 80\,^\circ\text{C}$ and a change in power input. The boundary conditions are, just as for the case with the CHP simulations, an inlet with nominal return water conditions; temperature of 40 °C and a pressure of 5.4 bar and an outlet where the pressure is fixed at 6 bar. The heat reservoir, from where the HP takes the additional heat it needs, is set as the ambient with a temperature of 10 °C.

The power input, P, is ramped from 5 MW to 2.5 MW over 30 min starting at $t = 5\,\text{h}$. From the response given in **Figure 3.14**, the produced heat ramps down as the power input ramps down. The produced heat is however not always 3 times as large as the power input; as the produced heat decreases, the mass flow required to maintain $T_{\text{supply}} = 80\,^\circ\text{C}$ also has to decrease. As the temperature control is not ideal, the forward temperature drops, until the required mass flow is reached. During this period, with a lower forward temperature, the COP is equivalently higher, as the difference between forward temperature and ambient temperature is lower. The magnitude of this effect is however very dramatic due to the over-simplified COP model, but it provides an interesting element, perhaps exploitable by a controller.

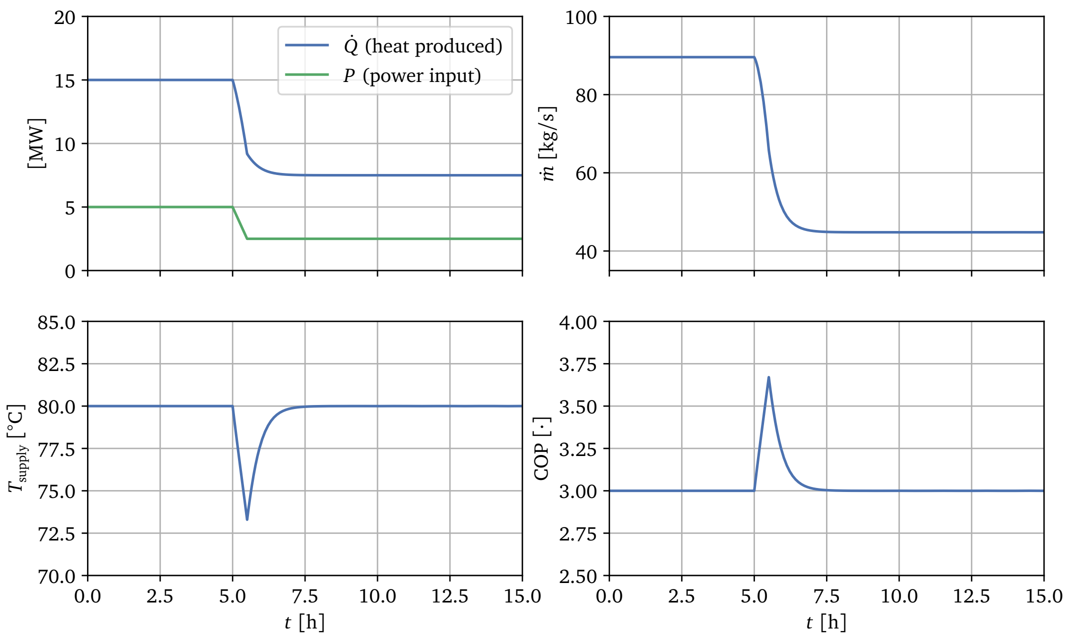


Figure 3.14: *Simulation where HP power input is ramped down from* 5 MW *to* 2.5 MW *over* 30 min*. The production of heat is correspondingly lowered – and to keep the desired forward temperature, the mass flow through the HP is turned down.*

3.6 Boiler

The production portfolio also includes a boiler, which produces heat using an arbitrary fuel. Furthermore, this boiler model also serves the purpose of pressure control in the entire production portfolio.

The boiler is modeled using a different approach than the other production units; where the other production units have embraced the component-based modeling paradigm, the boiler model is purely based on explicit equations. This is due to the fact, that the boiler is modeled with an ideal pressure difference controller. The reason for this was to simplify the system, by abstracting away yet another controller. It includes two inputs; a pressure difference setpoint, Δp_{set}, and a forward temperature setpoint, T_{set}. It has a single output; the heat produced, $\dot{Q}$. No graphical model in Modelica is given but the icon is shown in **Figure 3.15**.

Figure 3.15: *The boiler model's icon representation.*

There is no mass storage in the boiler:

$$\dot{m}_{\text{in}} + \dot{m}_{\text{out}} = 0 \tag{3.28}$$

And there is no energy storage:

$$\dot{m}_{\text{in}}\, h_{\text{in}} + \dot{m}_{\text{out}}\, h_{\text{out}} + \dot{Q} = 0 \tag{3.29}$$

The boiler is ideally controlled regarding forward temperature and the pressure difference. However, first order dynamics have been added for the temperature control, in the form of a time constant parameter, which can be adjusted. The ideal control gives the following two equations:

$$\tau\, \dot{T}_{\text{out}} = T_{\text{out,set}} - T_{\text{out}} \tag{3.30}$$

$$p_{\text{out}} = p_{\text{in}} + \Delta p_{\text{set}} \tag{3.31}$$

Given the affine relation between temperature and enthalpy, due to the assumption of constant specific heat capacity and constant density of the water, the outlet temperature can be related to the outlet enthalpy as:

$$T_{\text{out}} = T(h_{\text{out}}) = \frac{h_{\text{out}}}{c_p} + T_0 \quad \wedge \quad T_0 = 273.15\,\text{K} \tag{3.32}$$

This means, that the outlet enthalpy is independent from the inlet enthalpy, which implies that $\dot{Q}$ must be the difference between the inlet enthalpy and the outlet enthalpy.

The energy flow *required* by the boiler, to comply with the desired outlet temperature, is thus $\dot{Q}$.

The heat production is limited, to model that the boiler is always running, by setting a minimum and maximum value for $\dot{Q}$. Minimum heat production is set at 1 MJ/s and maximum at 10 MJ/s. These minimum and maximum constraints, are imposed, by setting `min` and `max` attributes on the heat flow variable in Modelica.

This is a very basic boiler model, but sufficient for the scope of this project, allowing for focus to be placed elsewhere.

3.6.1 Simulations

For the boiler, a single simulation is made, with a step in the forward temperature setpoint. The step is from 80 °C to 65 °C and is introduced at $t = 5$ h. Also, a step in the pressure difference is made, from 0.6 bar to 1.0 bar, at $t = 10$ h. The boundary conditions are nominal return water conditions; temperature of 40 °C and a fixed inlet mass flow of 30 kg/s and the outlet pressure is fixed at 6 bar. The results are given in **Figure 3.16**.

From the results, it can be concluded, that the boiler functions as intended. When a lower forward temperature is requested, less heat is produced, and the pressure difference is ideally controlled.

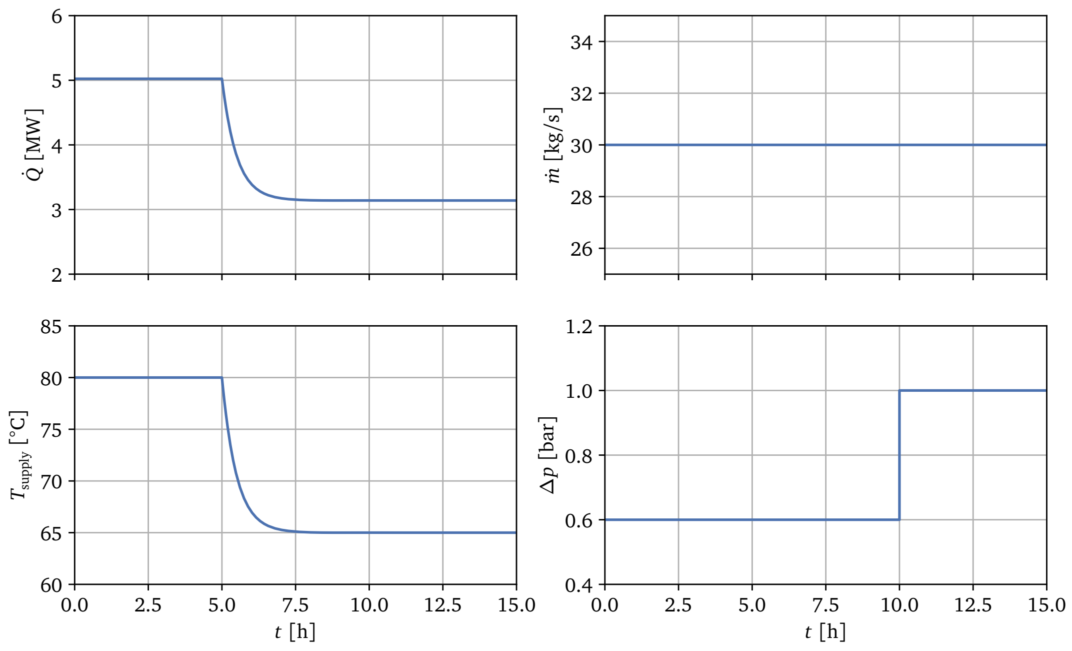


Figure 3.16: *Simulation where the forward temperature setpoint for the boiler is changed from 80 °C to 65 °C. To decrease the forward temperature, the heat produced is decreased. Step in pressure difference setpoint has the ideal controller respond accordingly.*

3.7 Accumulator

The production portfolio also includes an accumulator. The accumulator is a tank, which can store a limited amount of heated water. The accumulator is connected to both the supply pipe and the return pipe of the district heating system. Thus enabling it to let both supply water and return water, enter and exit the accumulator. The principle is, that the accumulator is always full – it is just the ratio between supply and return water that can change. The notion of charge will be used to describe the accumulator. When the accumulator is fully charged, it only contains supply water (hot water) – and when it is fully discharged, it only contains return water (cold water).

Just as the boiler model, the accumulator does also not rely on the component-based approach to modeling in Modelica. Instead, it is solely based on explicit equations. It includes a single input; a charge rate, $\dot{Q}_{\text{charge}}$, and a single output; the charge. The charge rate determines whether the accumulator is charging or discharging and how fast this is done. No graphical model in Modelica is given, but the icon is shown in **Figure 3.17**.

In the real world, the accumulator would be driven by the pressure difference; working as buffer. But, as the pressure difference is kept constant in this district heating system, by the simplified boiler model, it would effectively only allow the accumulator to charge and not discharge. Thus, the introduction of an input signal. This also has the added benefit of more direct control over the accumulator in a control design.

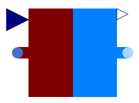


Figure 3.17: *The accumulator model's icon representation. It has a 'hot side' and a 'cold side', which illustrates that it contains a mixture of both supply water and return water.*

The accumulator is governed by the following energy balance and mass balance:

$$\dot{E} = \dot{m}_{\text{in}}\, h_{\text{in}} + \dot{m}_{\text{out}}\, h_{\text{out}} = \dot{Q}_{\text{charge}} \tag{3.33}$$

$$\dot{m}_{\text{in}} + \dot{m}_{\text{out}} = 0 \tag{3.34}$$

Thus, by setting a desired charge rate, $\dot{Q}_{\text{charge}}$, the energy stored in the accumulator is changed at this rate. The accumulator is limited to a maximum energy storage, E_{max}, which is equivalent to a charge $c = 1$. This is handled by defining the charge as:

$$c = E/E_{\text{max}} \tag{3.35}$$

The maximum energy storage is chosen as 100 MW h and the minimum is 0 MW h. The maximum and minimum constraints, are imposed, by setting `min` and `max` attributes on the charge and energy variables in Modelica.

One thing *not* considered for the accumulator is loss. When storing supply water, it can not maintain the temperature which the water had when it entered the accumulator – heat will transfer to the surroundings of the accumulator.

3.7.1 Simulations

The accumulator is initialized with a charge of 0.0 and thus contains 0 MJ. The boundary conditions are set to nominal case, where the supply water has a temperature of 80 °C at a pressure of 6 bar and the return water has a temperature of 40 °C at 5.4 bar.

The charge rate is at $t = 2.5\,\text{h}$ set to $10\,\text{MJ/s}$, thus charging the accumulator. At $t = 7.5\,\text{h}$, the charge rate is set to $0\,\text{MJ/s}$ for 5 h, where after at $t = 12.5\,\text{h}$, the charge rate is set to $-10\,\text{MJ/s}$ for 5 h. The expected outcome is, that the accumulator will, at $t = 7.5\,\text{h}$ have reached a charge of $c = 0.5$. And that the charge will again be $c = 0.0$ at $t = 17.5\,\text{h}$. As seen in **Figure 3.18**, this is exactly what happens.

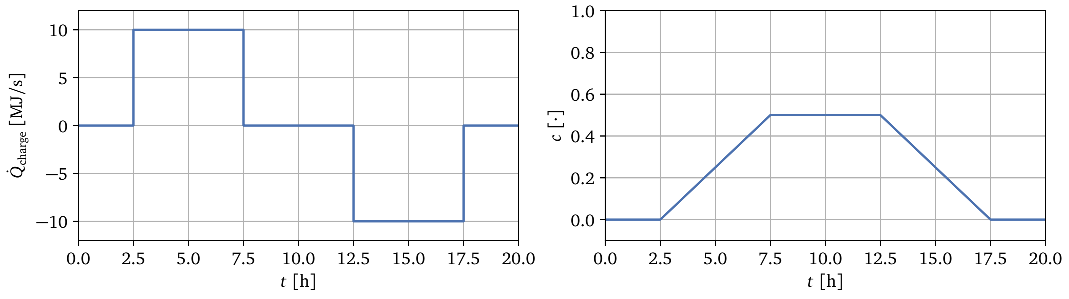


Figure 3.18: *The accumulator is first charged to $c = 0.5$, at a charge rate of $10\,\text{MJ/s}$. Then at $t = 12.5\,\text{h}$, the accumulator is again discharged, with a charge rate of $-10\,\text{MJ/s}$ until reaching a charge of $c = 0.0$ at $t = 17.5\,\text{h}$.*

3.8 District heating system

The production units and the consumer are connected in parallel, to form the complete district heating system. The number of consumers, $N_{\text{consumers}}$, will be kept at 10 000 and the HP uses the ambient as heat reservoir. This combined system is depicted in **Figure 3.19**. Besides connecting the units in parallel, some additional connections are given in the diagram. To handle inputs and outputs for each of the production units and the consumer, buses have been laid out. These are realized using Modelica's `expandable connector`. An input bus and an output bus is available on each component. These buses are connected to two master buses; one for inputs and one for outputs, thus making all inputs available on a single connector and all outputs available on a single connector. This is simply a Modelica technicality, to minimize overhead when having to apply different inputs.

The pressure on the return side of the system is fixed to 5.4 bar. This is necessary to avoid singularity issues, when initializing the system, since all components consider pressure differences and not absolute pressure. A direct analogy is an electrical circuit, where a reference voltage – or ground – is needed.

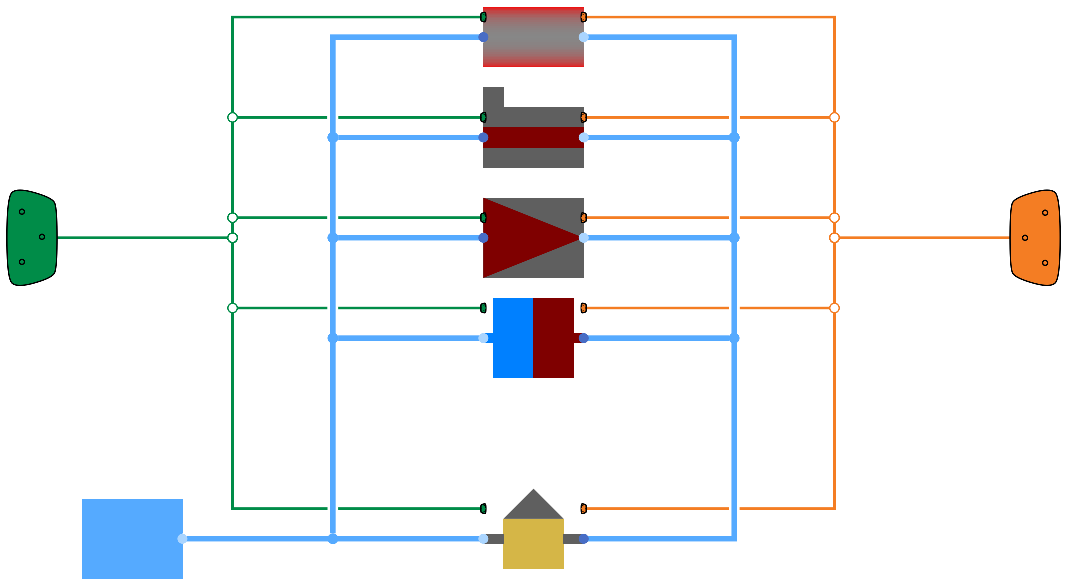


Figure 3.19: *The combined district heating system, with all production units and the consumer included. The green bus is the input bus, the orange bus is the output bus. The blue connections represent water flowing to/from the different components, using the constructed fluid connectors.*

3.8.1 Simulations

The combined system is simulated with known inputs to the different units, to investigate whether the behavior is as desired. The number of inputs available has been minimized, by connecting all the forward temperature setpoints together on the input bus; exposing a single forward temperature setpoint that is used by all production units. Thus, the following controllable inputs are available:

- L_{set}: Load setpoint for the CHP
- P_{HP}: Power input for the HP
- Δp_{set}: Pressure difference setpoint for the boiler
- $T_{\text{set,supply}}$: Forward temperature setpoint for all production units
- $\dot{Q}_{\text{charge}}$: Accumulator charge rate

The setpoint temperature, $T_{\text{set,consumer}}$ for the floor temperature controller inside the consumer is regarded as a disturbance, along with the ambient temperature, T_{amb}. For this simulation, $T_{\text{set,consumer}}$ is kept constant at 30 °C. The system has been simulated for 110 h with input signals as given in **Figure 3.21**. These input signals utilize all units in the portfolio, but allows for the system to settle, in between applying different inputs, such that their effect can be identified in the overall system response. The inputs are applied in the following sequential manner:

1. At $t = 3$ h: Ramp down L_{set} from 0.5 to 0.3 over 1 h.

2. At $t = 10$ h: Ramp down $T_{\text{set,supply}}$ from 80 °C to 70 °C over 1 h.

3. At $t = 30$ h: Ramp up P_{HP} from 0.1 MW to 1 MW over 1 h.

4a. At $t = 40$ h: Step in $\dot{Q}_{\text{charge}}$ from 0 MW to 2 MW (charging).

4b. At $t = 40$ h: Step in $\dot{Q}_{\text{charge}}$ from 2 MW to 0 MW.

5. At $t = 60$ h: Ramp up Δp_{set} from 0.6 bar to 1.0 bar over 1 h.

6. At $t = 80$ h: Ramp down T_{amb} from 10 °C to 5 °C over 5 h.

In this context, the overall system response is a plot of the produced and consumed heat for each of the components in the system – and a sum of all the heat flows, to verify that production and consumption is balanced. A plot of the heat flows in the system is given in **Figure 3.20**.

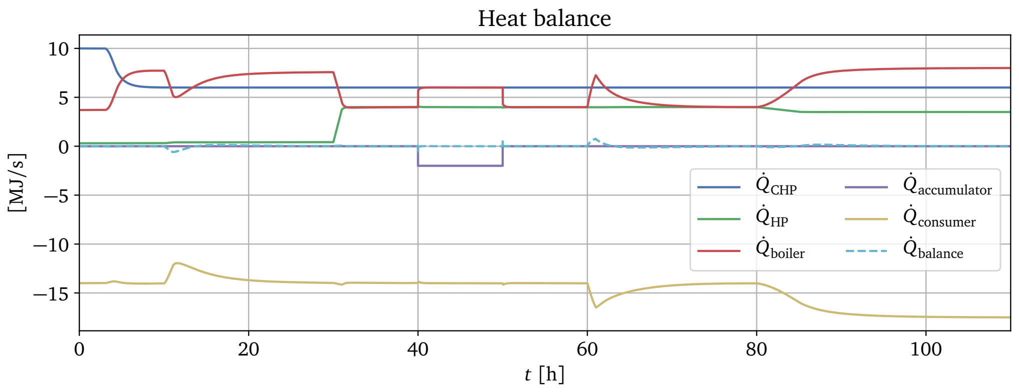


Figure 3.20: *Heat production and consumption for the different units in the district heating system. Production and consumption is balanced, as the sum of heat flows, $\dot{Q}_{balance}$, always converges to zero. Note that $\dot{Q}_{accumulator} = -\dot{Q}_{charge}$.*

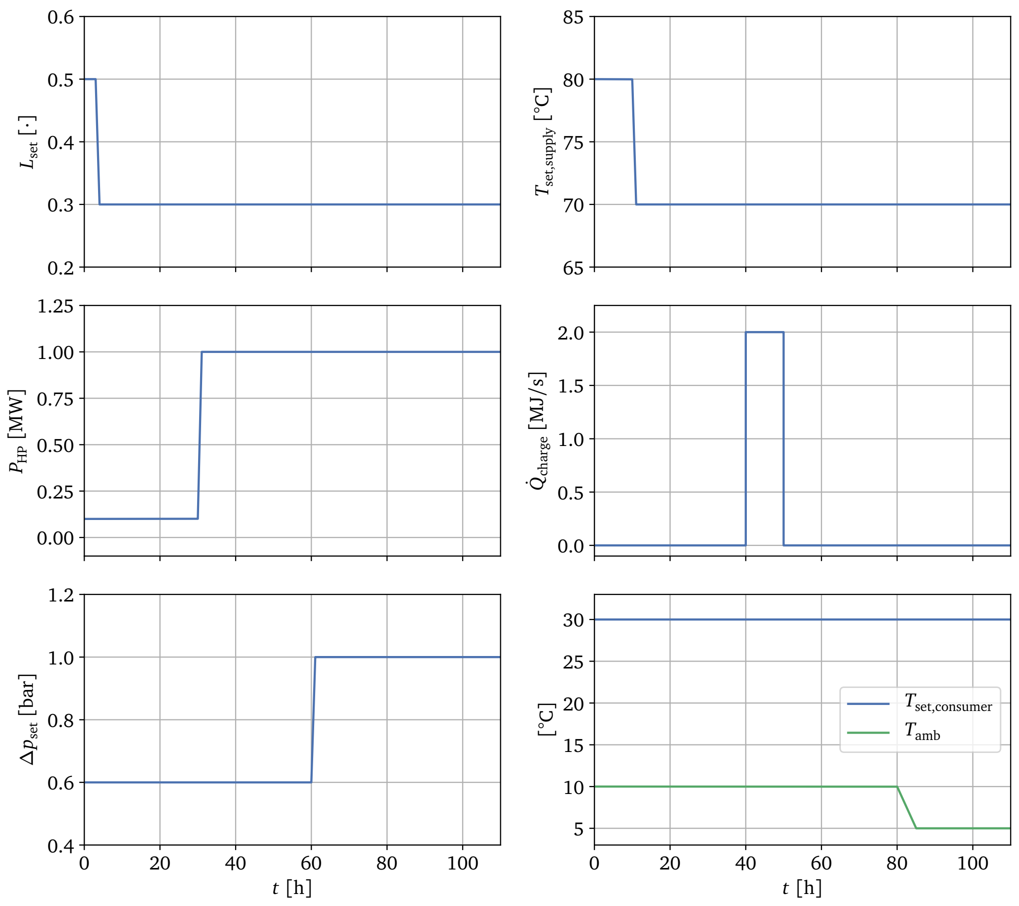


Figure 3.21: *The input signals applied to the district heating system. The inputs are changed sequentially, such that their effect on the overall system response can be identified.*

Event 1

At $t = 3$ h the load setpoint for the CHP is ramped down from 0.5 to 0.3 over 1 h as is seen in **Figure 3.21**. This has the effect, that the heat produced from the CHP is lowered from 10 MJ/s to 6 MJ/s, which is seen in **Figure 3.20**.

Before the change in input, the boiler is producing 4 MJ/s, the HP is producing 0.1 MJ/s and the accumulator is neither charging nor discharging. As the total heat consumption is approximately 14 MJ/s, the drop in production from the CHP has to be matched by an equal rise in production from another unit, in order to balance the system. Due to the drop in CHP load, the following occurs:

- The mass flow through the CHP has to be decreased, to keep the supply temperature at 80 °C – handled by the PI regulator in the CHP.
- This decrease in mass flow propagates to the consumer, where a decrease in mass flow induces a drop in floor temperature.
- The PI regulator in the consumer responds, by opening the consumer valve more.
- Opening the consumer valve more results in a pressure drop.
- The boiler, with ideal pressure difference control, counteracts the pressure drop; increasing the mass flow and thus the heat flow – ramping it up to ≈ 8 MJ/s.

The transfer of load from the CHP to the boiler is seen in **Figure 3.20**. The transfer is not entirely bump-less, inducing a small drop in floor temperature at the consumers, visible in **Figure 3.20** as a small drop in the heat supplied to the consumers. The increase in mass flow through the boiler is seen in **Figure 3.25**

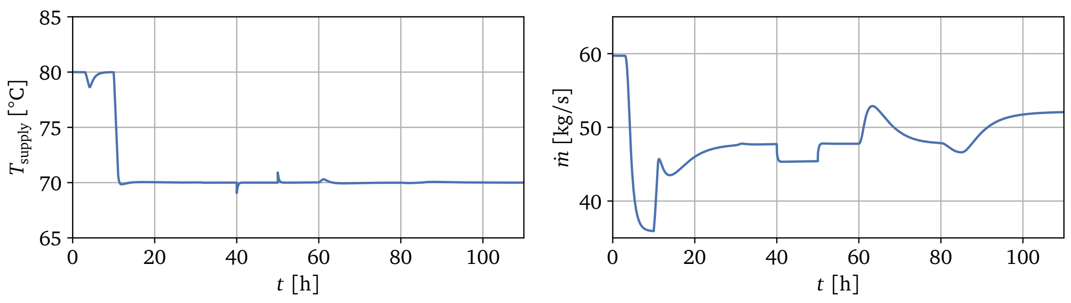


Figure 3.22: *The outlet temperature and the mass flow through the CHP.*

Event 2

At $t = 10$ h the forward temperature setpoint is lowered from 80 °C to 70 °C over 1 h, which is seen in **Figure 3.21**. This is reflected in **Figure 3.20** by an equivalent ramp down of the heat consumption, from 14 MJ/s to approximately 12 MJ/s. This is because the temperature controllers in the production units are significantly faster than the temperature controller in the consumer model. Thus, it takes a noticeable amount of time, for the PI regulator in the consumer model to increase the mass flow, so that the reference floor temperature is again met. This is reflected in the T_{floor} and mass flow responses in **Figure 3.26**.

The lower forward temperature setpoint has the CHP raise the mass flow through it, since the heat flow is fixed, given the load setpoint. This is seen in **Figure 3.22**.

Event 3

At $t = 30$ h the HP is introduced, by ramping up the power input to it, from 0.1 MW to 1 MW over 1 h. This change in input is seen in **Figure 3.21**. Given the COP at the current forward temperature and ambient temperature, this correspond to 4 MJ/s heat produced. This introduction of extra heat flow, has the boiler reduce the mass flow through it, such that heat balance is preserved.

The HP PI regulator introduces a spike in the outlet temperature, before it settles at the desired 70 °C. This increase in temperature propagates to the consumer, and is visible as a very small increase in the floor temperature (see **Figure 3.26**) and thus also the heat consumption. This is however attenuated by the consumer PI regulator.

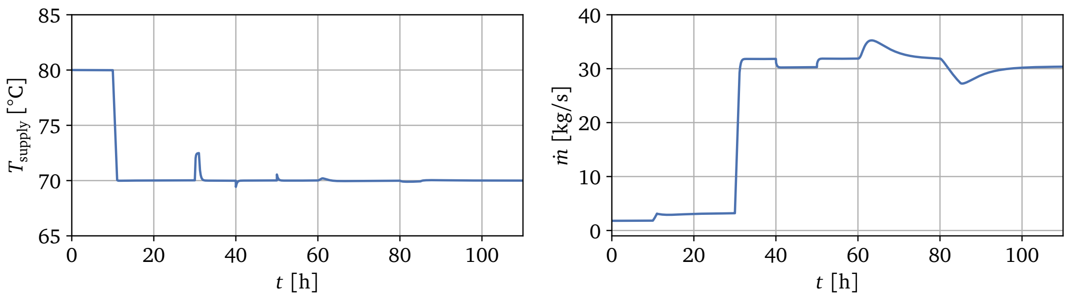


Figure 3.23: *The outlet temperature and the mass flow through the HP.*

Event 4

At $t = 40\,\text{h}$ the accumulator is charged at a rate of $2\,\text{MJ/s}$. The input signal is seen in **Figure 3.21**. This increase in heat consumption is handled by the boiler, responding with an increase in heat production, by increasing the mass flow. After 10 h of charging, the accumulator has reached a charge of 0.7 and stops charging. The boiler responds by lowering the heat production.

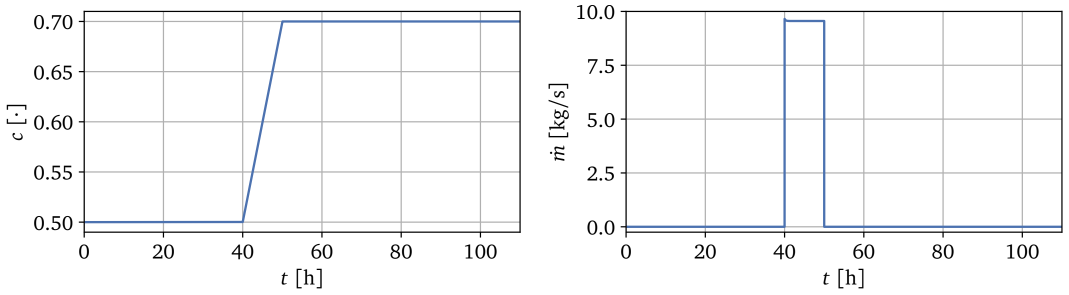


Figure 3.24: *Charge of the accumulator and the mass flow through the accumulator.*

Event 5

At $t = 60\,\text{h}$ the pressure difference is raised from 0.6 bar to 1.0 bar over 1 h. This change in input is seen in **Figure 3.21**. The effect is closely related to the effect of changing the forward temperature. The pressure increase means that the consumer valve can handle the same mass flow at a lower opening. The pressure increase therefore induces a sudden increase in the mass flow through the consumer, visible in **Figure 3.26**. This has the heat consumption rise, giving an undesirable higher floor temperature. The higher temperature has the PI regulator in the consumer react, lowering the mass flow, such that the floor temperature once again reaches 30 °C.

The increase in consumption, which is apparent as long as the floor temperature has not yet fallen back to 30 °C, has the boiler increase the heat production, by increasing the mass flow. The effect also propagates to the mass flow through the CHP and the HP, through a brief increase return temperature.

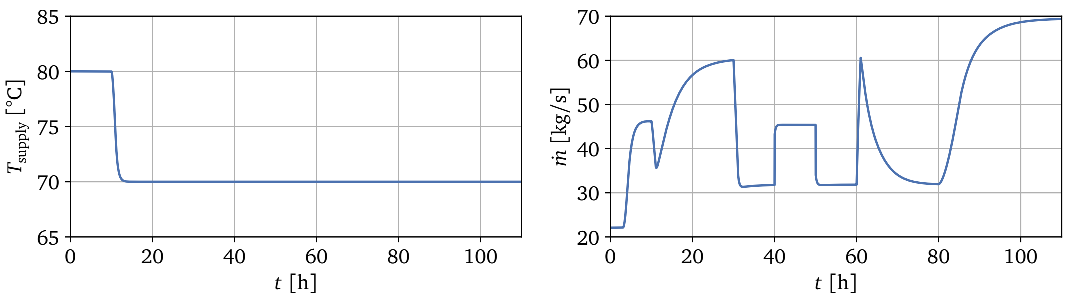


Figure 3.25: *The outlet temperature and the mass flow through the boiler.*

Event 6

At $t = 80\,\text{h}$, the ambient temperature is lowered from 10 °C to 5 °C over 5 h. The input signal is seen in **Figure 3.21**. This results in a slowly increasing heat consumption – and an equivalent increasing production. The temperature drop is too large, for the consumer PI regulator to suppress it, and thus, the floor temperature drops, which is seen in **Figure 3.26**. At $t = 85\,\text{h}$, when the ambient temperature has settled on 7 °C, the consumer PI regulator slowly pulls the floor temperature back up to 30 °C.

The increase in heat demand, caused by an increase in mass flow through the consumer, is reflected mainly in the increase of mass flow through the boiler (see **Figure 3.25**), but is also visible for both the CHP (see **Figure 3.22**) and for the heat pump (see **Figure 3.23**).

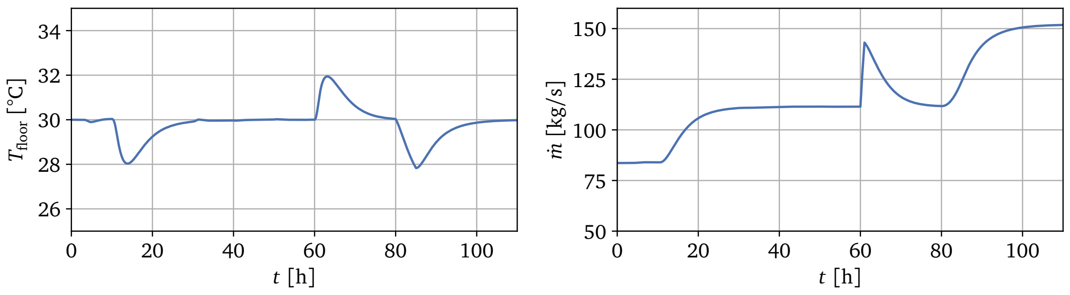


Figure 3.26: *Floor temperature for a consumer and the mass flow through all consumers.*

Comment

A general comment is, that it is always the boiler, which without any change in input, adapts the produced heat to meet the consumer demand, such that heat balance is maintained. This is the property of not having the heat production as a boundary condition. In reality, the balance between production and consumption can be handled by an accumulator tank working as a buffer and with a series of pumps spread around the network to maintain a constant pressure difference. Letting this be handled by the boiler is however deemed acceptable for this project.

3.9 Linear model

To allow for linear MPC a linear model is necessary. The linear model is obtained by linearizing the C^2 continuous nonlinear model described in the previous sections. Linearization is undertaken using the built-in methods of the `pyfmi` Python module, included in JModelica.org (see **Section 2.6.1**), by first compiling the nonlinear model as an FMU.

The FMU is first simulated to steady-state, using the desired operating point input signals. Here, nominal values have been used, for a general purpose operating point:

$$T_{\text{set,consumer}} = 30\,^{\circ}\text{C}, \quad T_{\text{amb}} = 10\,^{\circ}\text{C}$$

$$L_{\text{set}} = 0.5, \quad P_{\text{HP}} = 1\,\text{MW}, \quad \dot{Q}_{\text{charge}} = 0\,\text{MJ/s}$$

$$\Delta p_{\text{set}} = 0.6\,\text{bar}, \quad T_{\text{set,supply}} = 80\,^{\circ}\text{C}$$

The linearized system is, on state-space form, given as:

$$\dot{x} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}\boldsymbol{u} \tag{3.36}$$

Where $\boldsymbol{A} \in \mathbb{R}^{10\times10}$, $\boldsymbol{B} \in \mathbb{R}^{10\times7}$. The states have, by the underlying Modelica compiler, been selected as[3]:

$$\boldsymbol{x} = \left[\, T_{\text{floor}}\ T_{\text{floor,pipe}}\ \dot{m}_{\text{I,consumer}}\ T_{\text{supply,boiler}}\ L\ \dot{m}_{\text{I,CHP}}\ T_{\text{supply,CHP}}\ E\ \dot{m}_{\text{I,HP}}\ T_{\text{supply,HP}} \,\right]^T \tag{3.37}$$

Where some of the states have been mentioned previously, except for $T_{\text{floor,pipe}}$ which is the temperature of the water leaving the floor heating pipe (effectively also the return temperature) and $\dot{m}_{\text{I,x}}$ are integrator states for the PI regulators situated in both the consumer, the CHP and the HP.

The system matrix, $\boldsymbol{A}$, has the following eigenvalues:

$$\boldsymbol{w} = \begin{bmatrix} -0.20 \\ -0.16 \\ -14.8 \times 10^{-3} \\ -1.37 \times 10^{-3} \\ -1.09 \times 10^{-3} \\ -0.55 \times 10^{-3} \\ -0.28 \times 10^{-3} \\ (-92.0 \pm 55.0\text{j}) \times 10^{-6} \\ 0 \end{bmatrix} \tag{3.38}$$

Which characterize the system as being stable – except for one single eigenvalue; the one in zero. This eigenvalue, or the corresponding pole, is due to the integrator in the accumulator. The system contains no transmission zeros.

[3]As default, a Modelica compiler will choose states, based on the usage of `der()`, the operator for derivatives of variables. It is possible, through the attribute `stateSelect`, to provide the compiler with information on whether to use a specific variable as a state or not.

It is desirable to see, whether the system is controllable or not, given the available controllable input signals. These are all the input signals, excluding $T_{\text{set,consumer}}$ and T_{amb}. As such, controllability is tested using a reduced $\boldsymbol{B} \in \mathbb{R}^{10\times 5}$ matrix, with these inputs removed, after which the controllability matrix is computed as:

$$\boldsymbol{C_o} = \begin{bmatrix} \boldsymbol{B} & \boldsymbol{AB} & \boldsymbol{A}^2\boldsymbol{B} & \dots & \boldsymbol{A}^{n-1}\boldsymbol{B} \end{bmatrix} \tag{3.39}$$

The controllability matrix is found to have $\text{rank}(\boldsymbol{C_o}) = 10$, meaning that all states are controllable. It is worth checking the condition number of the controllability matrix, since it may be ill-conditioned, given the large differences in magnitudes of the states. The condition number of $\boldsymbol{C_o}$ is computed as:

$$\kappa(\boldsymbol{C_o}) = \frac{\sigma_1}{\sigma_{10}} = \frac{1}{40.0 \times 10^{-9}} = 2.50 \times 10^7 \tag{3.40}$$

Where σ_1 and σ_{10} are the largest and smallest singular values of $\boldsymbol{C_o}$, respectively. The condition number is large, but not so large as to not trust the rank computation, thus concluding, that the system is controllable.

3.9.1 Simulation

A simulation has been performed, comparing the nonlinear to the linear system. The simulation is comprised of the following events:

1. At $t = 5\,\text{h}$: Ramp down L_{set} from 0.5 to 0.3 over 1 h.

2. At $t = 20\,\text{h}$: Ramp up P_{HP} from 1 MW to 2 MW over 1 h.

3. At $t = 30\,\text{h}$: Charge up accumulator, with a charge rate of 5 MJ/s, for 5 h.

4a. At $t = 40\,\text{h}$: A change of −10 °C in $T_{\text{supply,boiler}}$ (disturbance).

The simulation results are given in **Figure 3.27**, where all states are shown. Generally, by inspection of the simulation results, the linear model fits the nonlinear model, especially considering the magnitudes of the states – but with noticeable differences in dynamics. Mainly states located in the consumer model differ, where the linear model fails to capture some of the faster dynamics. Another noticeable difference, is the effects induced by using the accumulator.

When the accumulator suddenly charges at $t = 30\,\text{h}$, a drop in the return temperature is introduced, as cold water leaves the accumulator to make room for hot water. Due to the dynamics of the PI regulators in the production units, it takes time to settle on a new mass flow, which again gives the desired supply temperature. The same happens, when the accumulator again stops charging, as the cold water stops leaving the accumulator. This effect is lost through linearization.

Together with the simulation results in **Figure 3.27**, the heat production and consumption is for the same simulation given in **Figure 3.28**. One difference is, that here the simulation time is extended to also show the effect of a step in T_{amb}, from 10 °C to 5 °C, at $t = 45$ h. From the comparison in **Figure 3.28**, the conclusion can be drawn, that the linear model is suitable for use in a controller design, given how well the trajectories fit. The main difference appears when using the accumulator and when introducing a step in T_{amb}.

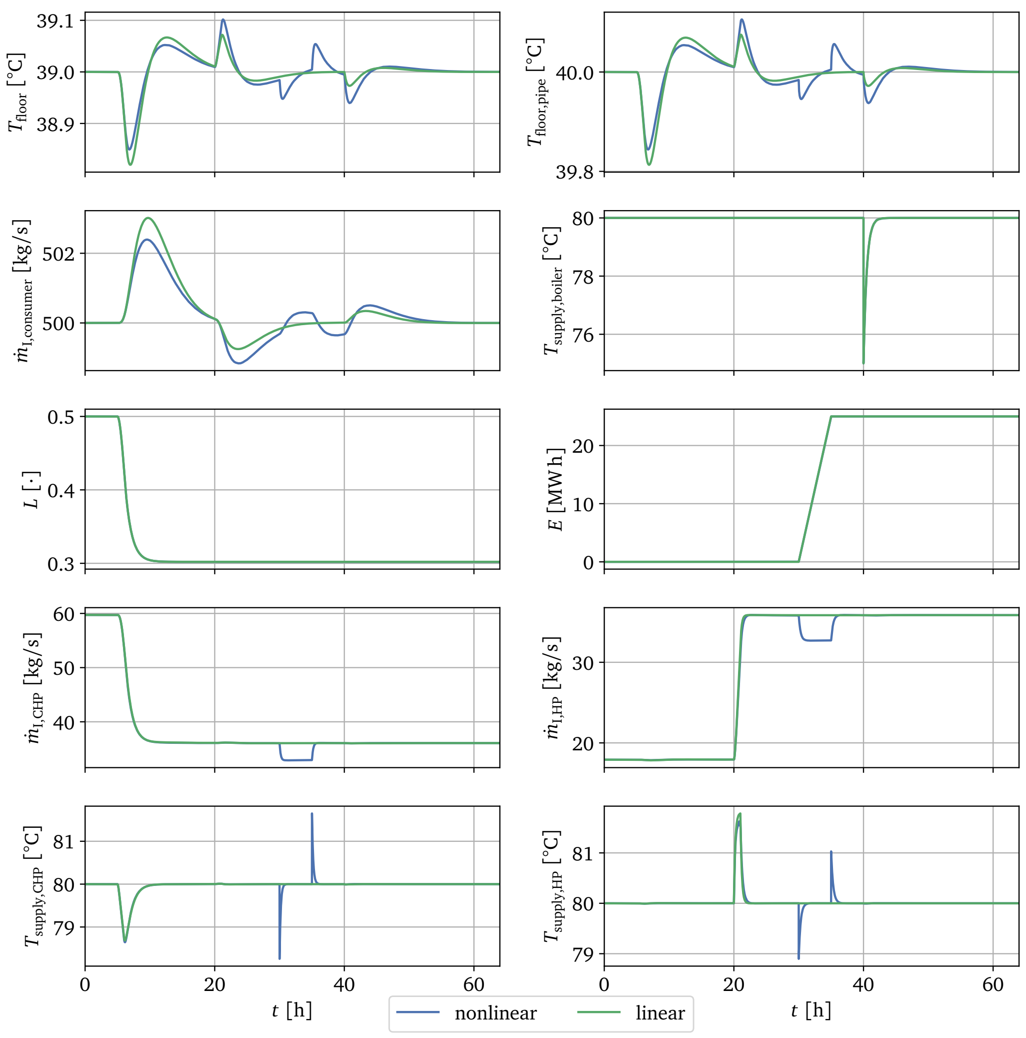


Figure 3.27: *Simulation, comparing the nonlinear C^2 continuous model to a linearized model. Noticeable dynamics have been lost through linearization, especially regarding effects of the accumulator, but nothing that introduces large deviations in magnitude.*

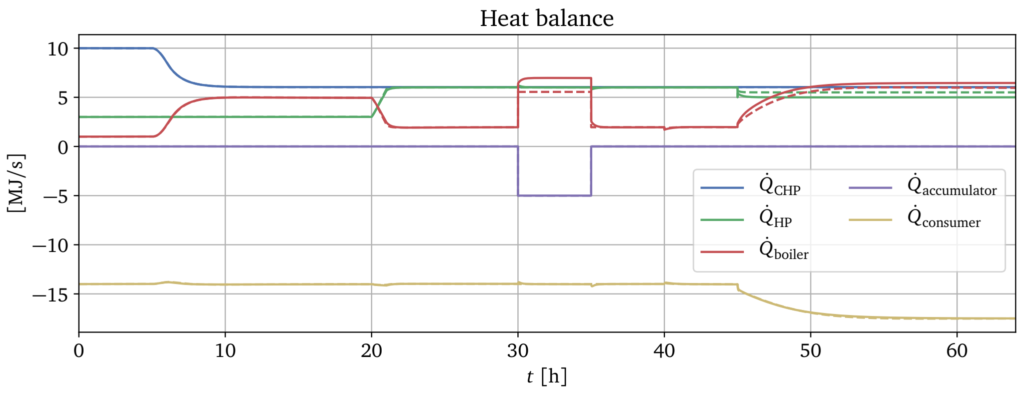


Figure 3.28: *Heat production and consumption for the different units in the district heating system – comparing the nonlinear C^2 continuous model (solid) to a linearized model (dashed). Noticeable differences when using the accumulator and when changing the ambient temperature. Note that $\dot{Q}_{accumulator} = -\dot{Q}_{charge}$.*

3.10 Simulation model

This section will outline the differences between the nonlinear C^2 continuous model on which optimization is possible and a nonlinear simulation model. Where the C^2 model uses a simplified fluid library constructed for the purpose of this project, the simulation model uses the fluid and media libraries from the MSL. Together with using components from the MSL, the consumer model is expanded in the simulation model. Instead of considering a single scaled consumer model, the simulation model introduces stochastic parameter variation on groups of consumers.

The $N_{\text{consumers}} = 10000$ have been equally divided into $N_{\text{types}} = 5$ groups with $N_{\text{consumers}}/N_{\text{types}}$ consumers in each group. Each group – effectively a single scaled consumer – is given a parameter set, drawn from a normal distribution. The parameters considered, together with their mean and standard deviation, are given in **Table 3.1**. These are chosen to provide enough interesting variations for a noticeable effect in simulations.

Parameter	**Description**	μ	σ	**Unit**
$\Delta T_{\text{nominal}}$	Temperature drop over consumer	40	3	°C
$\Delta T_{\text{nominal,amb}}$	Temperature drop between floor and ambient	20	2	°C
K_{PI}	PI regulator gain	0.001	0.0001	1/°C
T_{PI}	PI regulator time constant	360	10	s/°C
$c_{\text{p,concrete}}$	Concrete specific heat capacity	960	20	J/kgK

Table 3.1: *Parameter variations used in the stochastic consumer models in the simulation model. Variations on the specific heat capacity of concrete allow change in the heat capacity, independently of mass, which is scaled by the number of consumers.*

3.10.1 Simulation

A simulation is made, to compare the C^2 model to the simulation model. The simulation features the same input signals, as for the comparison with the linearized model. For the simulation model, the consumer now includes $N_{\text{types}} = 5$ times more states. In **Figure 3.29**, the results for all states have been shown and in **Figure 3.30**, the heat production and consumption is shown, where a step in T_{amb} is also introduced at $t = 45\,\text{h}$. Generally, the two models are almost identical, except for, naturally, the stochastic consumer model. This result is important, as it validates the further use of the C^2 continuous model – and provides a simulation model with a stochastic element to test both LMPC and NMPC against.

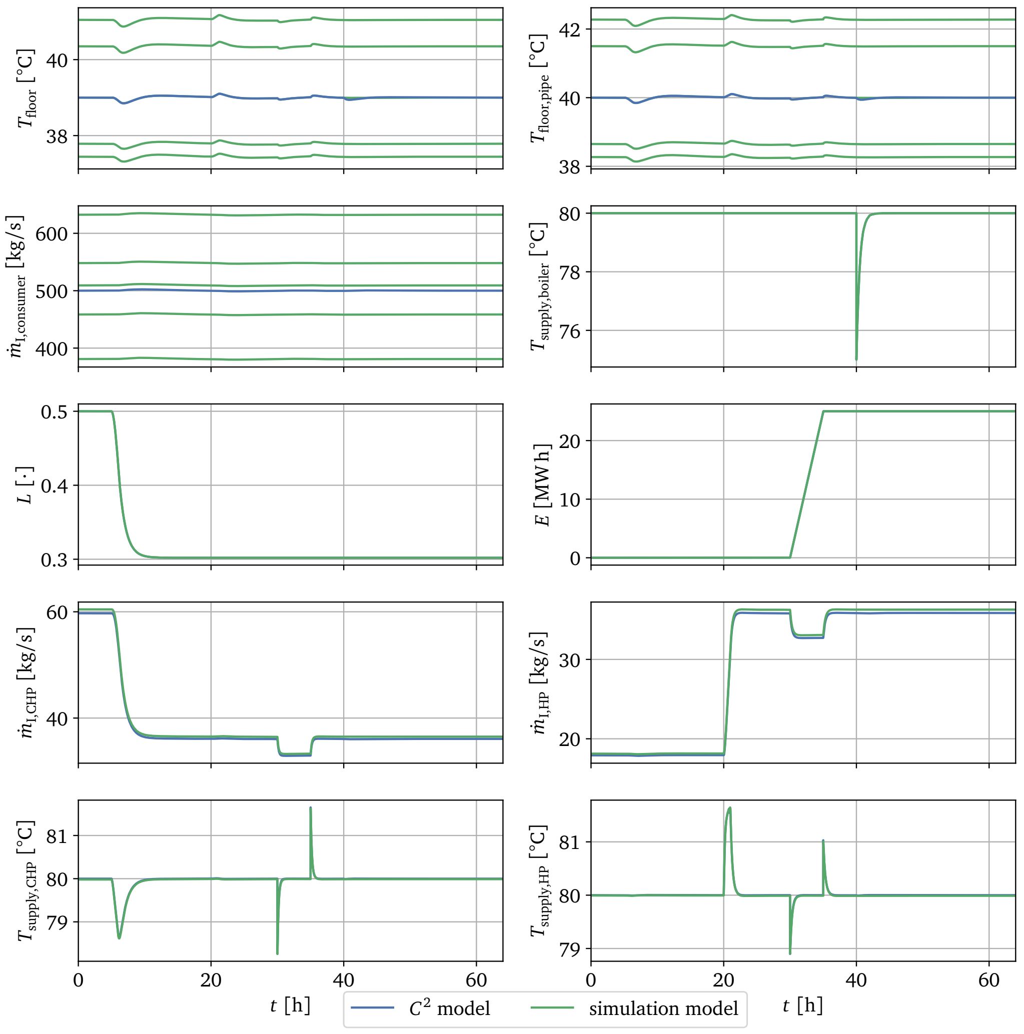


Figure 3.29: *Simulation, comparing the nonlinear C^2 continuous model to the nonlinear simulation model. The trajectories are almost identical; the only difference is, naturally, in the extended stochastic consumer model. The stochastic consumer model features $N_{types} = 5$ times more states, and they are all plotted.*

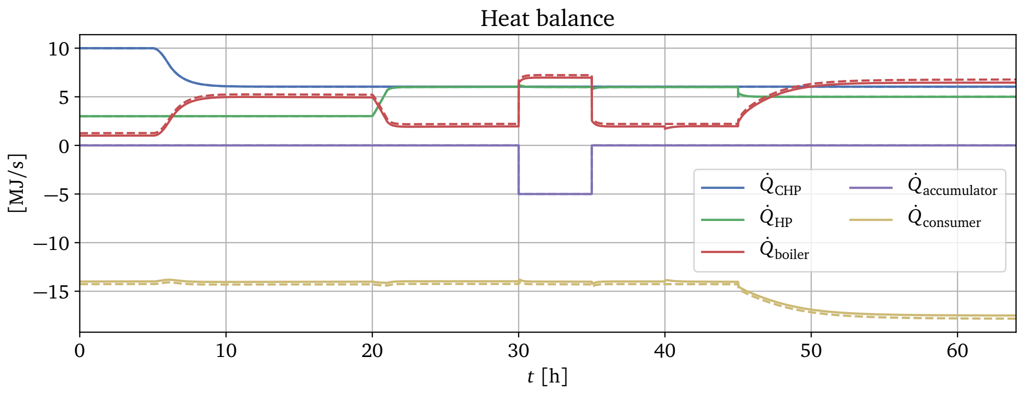


Figure 3.30: *Heat production and consumption for the different units in the district heating system – comparing the nonlinear C^2 continuous model (solid) to the nonlinear simulation model (dashed). The simulation model response gives a slightly higher heat consumption for the consumers, which propagates to the boiler providing the additional heat. The simulation model consumer heat flow, is the sum of the heat flow for the 5 consumer groups. Note that $\dot{Q}_{accumulator} = -\dot{Q}_{charge}$.*

Control 4

This chapter describes the design of MPC for production planning and balance control of the production portfolio. The design of a MPC is mainly the formulation of a suitable optimization problem; including cost function and constraints. The cost function and constraints will be the same for both NMPC and LMPC, for comparison purposes – only the model will differ; whether it is nonlinear or linear. Furthermore, this chapter describes the design of an EKF for state estimation, as it is assumed that not all states are measurable. A diagram of the control scheme is given in **Figure 4.1**.

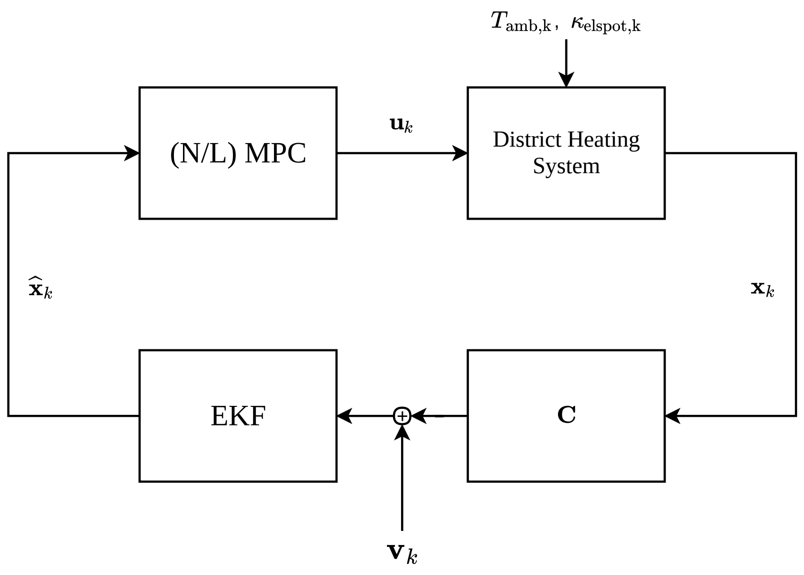


Figure 4.1: *MPC control scheme with an EKF for state estimation. Here* $\boldsymbol{C}$ *illustrates that measurements are picked from the full state and* $\boldsymbol{v}_k$ *illustrates that noise is added to these measurements.*

4.1 Cost function design

The objective of the controller is two-fold, as it includes both production planning and balance control. First it has to plan and execute production, in an economical optimal sense. This could either be by maximizing revenue or minimizing cost. Secondly, production and consumption has to be balanced – especially in a sense, where the consumers get the heat they desire, such that a certain QoS is maintained.

The inputs available are given in **Table 4.1**. They are divided in control inputs and exogenous inputs (disturbances). To simplify matters, it has been decided to keep $T_{\text{set, supply}}$ and Δp_{set} constant, reducing the degree of freedom for the optimizer, even though e.g. changing $T_{\text{set, supply}}$ could be economically beneficial[4]. Three control inputs are therefor considered. To allow for slew rate constraints, the actual control inputs will, in the case of L_{set} and P_{HP}, be their derivatives; $\dot{L}_{\text{set}}$, $\dot{P}_{\text{HP}}$. The accumulator input is already a rate input ($\dot{Q}_{\text{charge}}$), and will be used as is. As changes in consumer heat demand, can be achieved solely through changes in ambient temperature, it has been decided to keep $T_{\text{set, consumer}}$ constant at the nominal 30 °C.

Input	**Description**	**Unit**
	Control inputs	
L_{set}	Load setpoint for the CHP	·
P_{HP}	Power input for the HP	W
$\dot{Q}_{\text{charge}}$	Charge rate input for the accumulator	J/s
$T_{\text{set, supply}}$	Supply temperature setpoint for all production units	°C
Δp_{set}	Pressure difference setpoint for the boiler	bar
	Exogenous inputs	
$T_{\text{set, consumer}}$	Temperature setpoint for the consumer (floor heating)	°C
T_{amb}	Ambient temperature	°C
κ_{elspot}	Power price	DKK/MWh

Table 4.1: *Available control inputs and exogenous inputs; for the control design, only a subset of the available inputs is considered.*

4.1.1 Production portfolio economics

In [4] and [5], the economical objective is to maximize revenue; the costs of running the production portfolio subtracted from the income. This is also the objective considered in this thesis. The following costs of running the production portfolio are considered:

- Cost of running CHP, η_{CHP} [DKK/s]
- Cost of running boiler, η_{boiler} [DKK/s]

In the case of the CHP and the boiler, the cost is defined by the amount of heat produced; a fixed price, κ_{CHP}, multiplied with the heat output, $\dot{Q}_{\text{CHP}}$. The fixed price is thus given in DKK/J. The (instantaneous) cost terms are given by η:

$$\eta_{\text{CHP}} = \dot{Q}_{\text{CHP}}\,\kappa_{\text{CHP}} \tag{4.1}$$

$$\eta_{\text{boiler}} = \dot{Q}_{\text{boiler}}\,\kappa_{\text{boiler}} \tag{4.2}$$

Income is given by the heat sold to the consumers, with a fixed price. The (instantaneous) income terms are given by ξ:

$$\xi_{\text{heat}} = \dot{Q}_{\text{consumer}}\, \kappa_{\text{heat}} \tag{4.3}$$

The fixed price the consumers pay is chosen as $\kappa_{\text{heat}} = 500\,\text{DKK/MWh}$, this is comparable to the price set by Sønderborg Fjernvarme, where the price is approximately 400 DKK/MWh[8]. The prices for producing heat have been chosen as:

- $\kappa_{\text{CHP}} = 300\,\text{DKK/MWh}$
- $\kappa_{\text{boiler}} = 500\,\text{DKK/MWh}$

Making the boiler the more expensive production unit.

Power trade defines the cost of running the HP and a possible income or expense from selling/buying power. As the portfolio allows power produced by the CHP to be used directly in the HP, the power traded is given by the following equation:

$$P_{\text{trade}} = P_{\text{produced}} - P_{\text{consumed}} \tag{4.4}$$

Where P_{produced} is the power produced by the CHP and P_{consumed} is the power consumed by the HP. Thus, when producing more than consuming, power is sold (P_{trade} is positive), and when consuming more than produced, power is bought (P_{trade} is negative). Selling price and buying price will in this thesis be the same; η_{elspot}. Power trade will therefor not be regarded as either income or cost:

$$\lambda_{\text{power}} = P_{\text{trade}}\, \kappa_{\text{elspot}} \tag{4.5}$$

The power prices considered, are ELSPOT prices for Western Denmark. In 2015, the average price was 170 DKK/MWh[15], thus making the HP the cheapest unit on average.

With the above, (instantaneous) revenue, R [DKK/s], is defined as:

$$R = \lambda_{\text{power}} + \xi_{\text{heat}} - (\eta_{\text{CHP}} + \eta_{\text{boiler}}) \tag{4.6}$$

And from an optimization view, the desire is now to maximize the integral of R. The above economical break-down does not necessarily reflect reality, but it shows how basic economic considerations can be used in the formulation of a cost function.

4.1.2 Weighting control action

Through preliminary evaluation of JModelica.org for optimization and [4], it has been concluded, that weighting control action is necessary, to obtain a well-behaved optimization problem. If no weight is present, the input can practically move arbitrarily without cost. The input weight term introduced:

$$W = \boldsymbol{u}^T \boldsymbol{W} \boldsymbol{u} \tag{4.7}$$

$$\boldsymbol{u} = \begin{bmatrix} \dot{L}_{\text{set}} & \dot{P}_{\text{HP}} & \dot{Q}_{\text{charge}} \end{bmatrix}^T \tag{4.8}$$

$$\boldsymbol{W} = \begin{bmatrix} w_{\dot{L}_{\text{set}}} & 0 & 0 \\ 0 & w_{\dot{P}_{\text{HP}}} & 0 \\ 0 & 0 & w_{\dot{Q}_{\text{charge}}} \end{bmatrix} \tag{4.9}$$

And the desire is, to minimize W, penalizing control action. The weights, w, are initially determined by Bryson's Rule, which is normally applied when synthesizing a Linear Quadratic Regulator (LQR), where the cost function is given as:

$$J = \int_{t_0}^{t_f} \boldsymbol{x}^T \boldsymbol{Q} \boldsymbol{x} + \boldsymbol{u}^T \boldsymbol{R} \boldsymbol{u} \, \mathrm{d}t \tag{4.10}$$

Disregarding how the cost function is quadratic in the state, $\boldsymbol{x}$, the input term is equivalent. Bryson's Rule states, that weights (on either states or inputs), q_x, should be calculated as:

$$q_x = \frac{1}{x_{\text{max}}^2} \tag{4.11}$$

Where q_x are diagonal entries in the given weight matrix and x_{max} is the maximum acceptable value of x.

4.1.3 Ensuring Quality of Service

It is critical, that the consumers receive the heat they desire. As such, an attempt has been made to include the *satisfaction* of the consumers, in the optimization problem. One very direct measure of satisfaction, is the deviation between the desired temperature at the consumers and the actual temperature; the error signal, e, in the local floor heating PI regulator running at the consumer. A measurement of this signal is not available for the producer, but an estimate is obtainable, through state estimation. One disadvantage to using this measure of dissatisfaction, is that it is not decoupled, from the performance of the local PI regulator.

Defining the *dissatisfaction*, D, as the error signal for the local PI regulators, enables the inclusion in the cost function. Three different ways of including it:

- As a direct term in the cost function, minimizing D
- As constraints, requiring D to be within given requirements
- Both of the above

It will be included only as a term in the cost function, to avoid constraint violation because of the consumer PI controller.

4.1.4 Constraints

Constraints are given by the system dynamics:

$$\dot{\boldsymbol{x}} = \boldsymbol{f}(t, \boldsymbol{x}, \boldsymbol{u}) \tag{4.12}$$

Together with equality constraints and inequality constraints, as stated in the general OCP, in **Section 2.3**. The model contains, besides constraints given by the dynamics, also inequality constraints by limits on certain variables, posed as `min`/`max` attributes in Modelica. These are noted in **Chapter 3 Modeling**. Constraints on inputs have also been posed in **Chapter 3 Modeling**, but since derivatives of two input signals have been introduced, slew rate constraints have also been posed.

The following slew rate constraints have been introduced:

- $|\dot{L}_{\text{set}}| < 1 \times 10^{-3}\,\text{s}^{-1}$
- $|\dot{P}_{\text{hp}}| < 1\,\text{kW/s}$
- $|\dot{Q}_{\text{charge}}| < 15\,\text{MJ/s}$

By limiting L_{set}, to not change more than 1×10^{-3}/s, ramping from minimum load (0.3) to maximum load (1.0) is given an upper limit of $0.7/1 \times 10^{-3}\,\text{s} = 700\,\text{s} \approx 12\,\text{min}$. This is deemed acceptable, given the capacity of the modeled CHP.

By limiting P_{hp}, to not change more than 1 kW per s, going from minimum load (0.1 MW) to maximum load (5 MW), is limited to take $4.9\,\text{MW}/1\,\text{kW/s} = 490\,\text{s} \approx 8\,\text{min}$.

4.1.5 Resulting optimal control problem

Combining the economical term, the control action term and the QoS term, results in the following cost function:

$$J = \int_{t_0}^{t_f} R(t) - W(t) - D(t)\,\mathrm{d}t \tag{4.13}$$

And the optimization problem can then, subjected to the constraints, be posed as:

$$\underset{\boldsymbol{u}^*}{\text{maximize}}\, J \tag{4.14}$$

Or equivalently, posed as a minimization problem:

$$\underset{\boldsymbol{u}^*}{\text{minimize}}\ -J \tag{4.15}$$

The minimization problem is what will be implemented in practice, as the Optimica language constructs only allow for posing minimization problems.

4.1.6 Stability

The general NMPC problem, considers a cost function, where the distance to a predefined equilibrium is penalized. This is not the case in this thesis, where an economical cost function is considered. When using an economical cost function, the domain is often referred to as economic NMPC. The disadvantage, to such a free-form cost function, is that it is not clear, whether MPC yields a well-performing closed-loop solution[16]. Progress has however been made, to establish grounds for a stability analysis of economic NMPC; e.g. [17] and [18] – but these results have not been applied in this thesis.

4.2 Choosing MPC settings

A sample time, t_s, a prediction horizon, t_H and the number of collocation points has to be determined. The sample time has to be chosen small enough, to allow the controller to respond to changes in power prices, κ_{elspot}, changes in ambient temperature, T_{amb} and attenuate the effect of model deviations between the C^2 model and the simulation model. The trade-off lies in computational effort. As power prices are given per h[15], it has been chosen to let $t_s \leq 1\,\text{h}$. A longer prediction horizon gives better performance, since the controller can take into account more information, when determining optimal inputs. But a longer prediction horizon also increases memory requirements and solution times.

Transcribing the OCP by direct collocation requires the selection of the number of elements, N_e, and the number of collocation points per element, N_c. For convenience, each element will be t_s long. The number of elements will make up the entire prediction horizon, and will thus be given by:

$$N_e = \frac{t_H}{t_s} \tag{4.16}$$

The number of collocation points determines the accuracy of the transcribed system, but also greatly influences solution time, by increasing the number of variables in the NLP. The number of variables in the NLP is given by the following equation, introduced in **Section 2.3**:

$$n_Z = (1 + N_e\, N_c)\, n_z + (N_e - 1)\, n_x \tag{4.17}$$

By replacing the number of elements with **Equation (4.16)**, the equation becomes:

$$n_Z = (1 + \frac{t_H}{t_s}\, N_c)\, n_z + (\frac{t_H}{t_s} - 1)\, n_x \tag{4.18}$$

Relating the number of collocation points, the prediction horizon and the sample time to the size of the NLP; serving as a measure for the computational effort.

A sample time $t_s = 30\,\text{min}$ has been chosen, given the power price resolution of 1 h and the slow dynamics of the consumer. With the sample time fixed, the effect of changing the number of collocation points is investigated, through a series of open loop simulations, solving the specified OCP only once and applying the optimal inputs in a simulation of the C^2 model. This will reveal the deviations due to collocation. The OCP is solved with $N_c = 2$, $N_c = 3$ and $N_c = 4$. The final time of the OCP is set to 10 h. The results are given in **Figure 4.2**, where the comparison is of the consumer heat flow, $\dot{Q}_{\text{consumer}}$.

The effect of increasing the number of collocation points is visible, as the collocation response better approximates the simulation response for larger N_c. The corresponding solution time for the OCP also increases. For $N_c = 2$, the solution time, t_{sol} was 0.82 s, while for $N_c = 4$ the solution time was 2.31 s; almost 3 times higher[1].

To be able to keep the low sample time, without sacrificing prediction horizon, the number of collocation points has been chosen as 2. It was then possible to successfully solve the OCP with a final time of 10 h; decreasing sample time, increasing collocation points or prediction horizon beyond these settings and JModelica.org would exit with errors regarding memory consumption. No further investigation into the memory issues have been conducted, as it was concluded, that a 10 h prediction horizon is long enough, to evaluate the performance MPC as a control concept for the production portfolio.

As such; the base settings will be $t_s = 30\,\mathrm{min}$, $t_H = 10\,\mathrm{h}$ and $N_c = 2$.

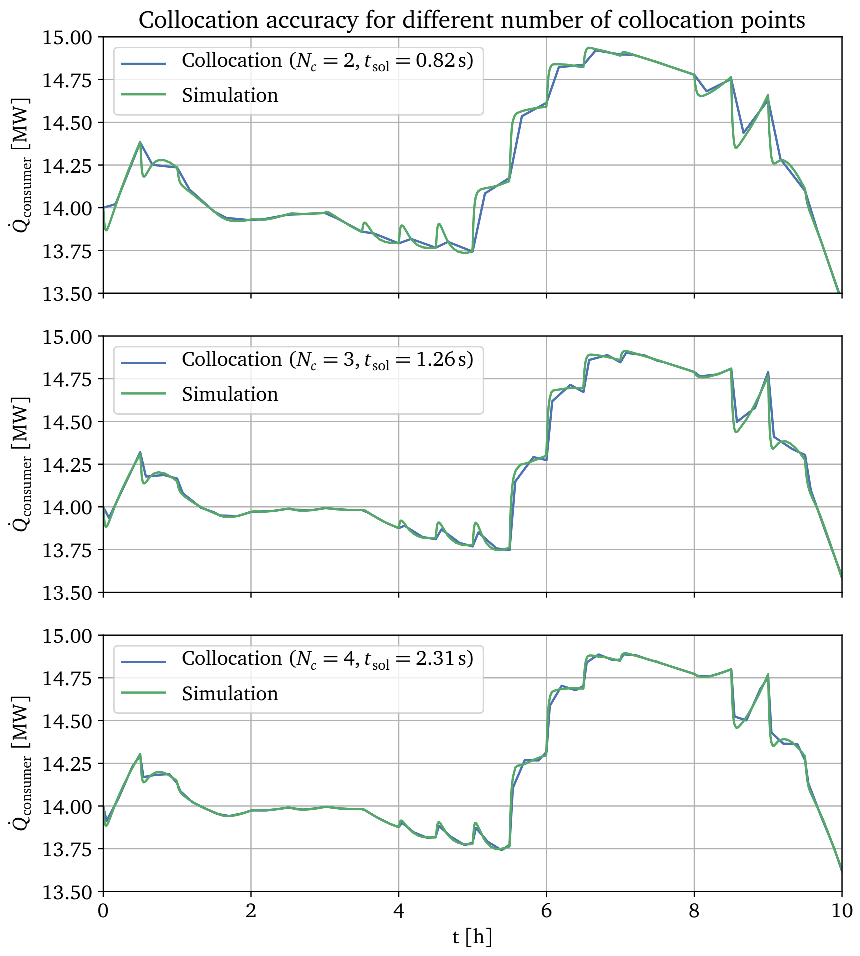


Figure 4.2: *Comparing the results of collocation, through the solution of the specified OCP, and a simulation run with the resulting optimal inputs.*

[1]Performed on the same piece of hardware; a Lenovo ThinkPad T530, Intel i5-3320M @ 2.60 GHz and 4GB RAM.

4.3 Observer design

As discussed in **Section 2.5**, it is a fair assumption, that no measurements are available at the consumers. This assumption is embraced, by designing an observer to provide estimates of all states. It will further be assumed, that measurements of all states of the production units are available. From **Section 3.9**, the states were found as:

$$\boldsymbol{x} = \begin{bmatrix} T_{\text{floor}} & T_{\text{floor,pipe}} & \dot{m}_{\text{I,consumer}} & T_{\text{supply,boiler}} & L & \dot{m}_{\text{I,CHP}} & T_{\text{supply,CHP}} & E & \dot{m}_{\text{I,HP}} & T_{\text{supply,HP}} \end{bmatrix}^T \tag{4.19}$$

Here, T_{floor}, $T_{\text{floor,pipe}}$ and $\dot{m}_{\text{I,consumer}}$ are not measurable, while the rest are. By not considering measurements at the consumer, the fact that the simulation model – where the measurements would be taken – contains N_{types} more states in the consumer model does not require any extra care in the estimator. No specifications of measurement noise is known, as the project is purely conceptual. Thus, the the standard deviation has been set to be 5 % of the nominal values, for each state measured. Refer to **Table 4.2** for the actual values.

Considering the linearized system, obtained and described in **Section 3.9**, an observability analysis is performed. It has to be noted, that this is only valid for the general purpose operating point considered for the linear model, but it will still provide information as to if the measurements available are enough to provide estimates of all states, necessary for control. On the basis, that no states at the consumers are measurable, a $\boldsymbol{C} \in \mathbb{R}^{7\times10}$ matrix is formed as:

$$\boldsymbol{C} = \begin{bmatrix} 0&0&0&1&0&0&0&0&0&0 \\ 0&0&0&0&1&0&0&0&0&0 \\ 0&0&0&0&0&1&0&0&0&0 \\ 0&0&0&0&0&0&1&0&0&0 \\ 0&0&0&0&0&0&0&1&0&0 \\ 0&0&0&0&0&0&0&0&1&0 \\ 0&0&0&0&0&0&0&0&0&1 \end{bmatrix} \tag{4.20}$$

Observability is checked, by forming the observability matrix as:

$$\boldsymbol{O_b} = \begin{bmatrix} \boldsymbol{C} & \boldsymbol{CA} & \boldsymbol{CA}^2 & \dots & \boldsymbol{CA}^{n-1} \end{bmatrix}^T \tag{4.21}$$

The observability matrix is found to have $\text{rank}(\boldsymbol{O_b}) = 10$, meaning that all states are observable for the linear system. Just as for the controllability matrix in **Section 3.9**, the condition number of the observability matrix is computer. It is found as:

$$\kappa(\boldsymbol{O_b}) = \frac{\sigma_1}{\sigma_{10}} = \frac{9.80}{11.1 \times 10^{-9}} = 8.83 \times 10^8 \tag{4.22}$$

Again, the condition number is large, but not large enough to reject the rank computation – concluding that the system is indeed observable.

The observer will be implemented as an EKF, as noted in **Section 2.5**. Generally, the design of a Kalman Filter deals with the choice of appropriate covariance matrices; $\boldsymbol{Q}$ and $\boldsymbol{R}$. By weighting the matrices differently, the observer can be tuned to either trust the model or the measurements, more or less. Generally, a good approach is to only consider diagonal elements in $\boldsymbol{Q}$ and $\boldsymbol{R}$. For a first design iteration, the choice has been to let the diagonal elements of $\boldsymbol{Q}$ reflect a standard deviation of 1 % of the nominal values of the states. The diagonal entries in $\boldsymbol{R}$ have been given the measurement noise standard deviations; 5 % of the nominal values. Thus, the observer trusts the model more than the measurements. The actual values are given in **Table 4.2**.

The EKF will run with the same sample time, $t_s = 30\,\text{min}$, as the MPC; thus the linear system used in the EKF will be discretized with this sample time.

State	Measurement noise σ	State noise σ	Unit
	Not measurable		
T_{floor}		0.3	°C
$T_{\text{floor,pipe}}$		0.4	°C
$\dot{m}_{\text{I,consumer}}$		0.5	kg/s
	Measurable		
$T_{\text{supply,boiler}}$	4.0	0.8	°C
L	25×10^{-3}	5×10^{-3}	·
$\dot{m}_{\text{I,CHP}}$	3.0	0.6	kg/s
$T_{\text{supply,CHP}}$	4.0	0.8	°C
E	1.8×10^{9}	360×10^{6}	J
$\dot{m}_{\text{I,HP}}$	1.0	0.2	kg/s
$T_{\text{supply,HP}}$	4.0	0.8	°C

Table 4.2: *Standard deviations for measurement noise and state noise, used in the observer design, parameterizing* $\mathbf{Q}$ *and* $\mathbf{R}$ *for the EKF.*

Implementation 5

A simulation framework has been designed and implemented around the APIs provided by JModelica.org. The main goal of the framework, is to provide easy and configurable simulations with both NMPC and LMPC. The simulation framework consists of several different components. In **Figure 5.1**, an overview of all the components is given. Two domains are considered when discussing the implementation; *the Modelica domain* and the *JModelica.org domain*. The JModelica.org domain is in practice a collection of Python modules. The following sections will account for the different components in **Figure 5.1**, together with implementation considerations regarding MPC and EKF. A more practical introduction to the software is given in **Appendix B**, together with setup and run instructions in **Appendix A**.

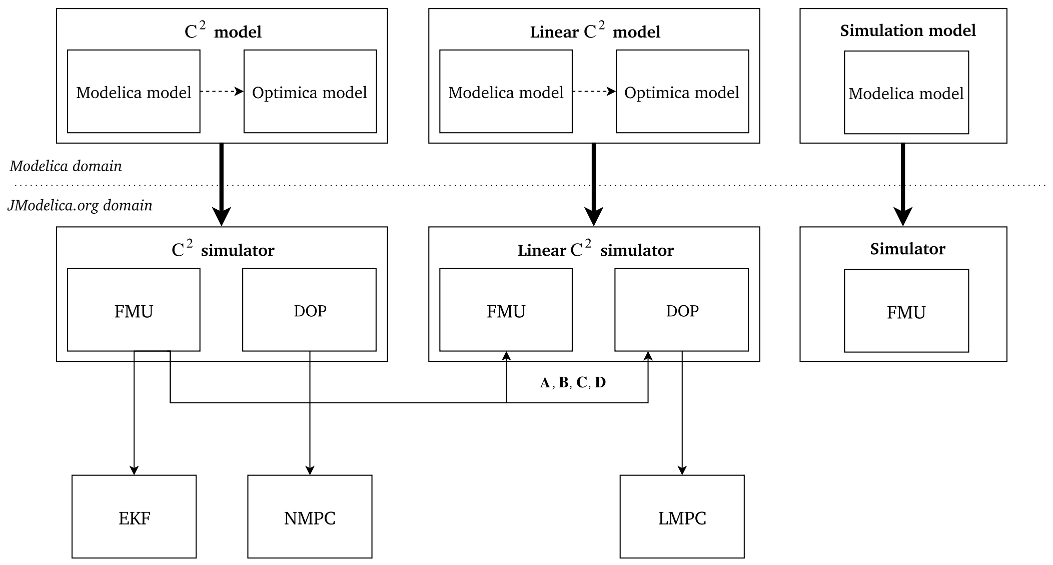


Figure 5.1: *Overview of the different components included in the implementation.*

5.1 Modelica domain

The Modelica domain, depicted in **Figure 5.1**, shows the three models considered in this project; the C^2 model, the linear C^2 model and the simulation model. The C^2 model and the linear C^2 model include an OCP formulation in Optimica, as they are used in optimization. The simulation model only includes a Modelica model, as it only serves the purpose of simulation.

The Modelica domain consists of the models described in **Chapter 3 Modeling**.

However, when employing JModelica.org for optimization purposes, a single model with clearly defined inputs and outputs has to be provided. The combined district heating model described in **Section 3.8**, uses an input and output bus, to apply input signals and extract output signals, for all production units and the consumers. Due to limitations in the JModelica.org Modelica compiler, it is required that the top-level inputs, for a model used in an optimization problem, are defined with the type *Real*. Thus, a specific Modelica model has been created, for the sole purpose of use in JModelica.org, which instantiates the combined district heating system, and routes the bus I/O to *Real* inputs. The Modelica model is depicted in **Figure 5.2**.

This JModelica.org-exposed Modelica model also handles the implementation of the cost function, defined in **Section 4.1**. Another possibility would have been to implement it in the Optimica code, which handles the formulation of the OCP, but by implementing it in Modelica code, the cost function is always evaluated, even when simulating to predefined input signals. This very useful, considering the fact, that a specific simulation model has been constructed. Thus, when simulating using the simulation model, the cost function will also be evaluated. As Modelica is object-oriented, the JModelica.org-exposed Modelica model is shared between the C^2 model, the linear C^2 model and the simulation model; the only difference lies in which district heating system model is instantiated. This has the benefit, that the three models share the same interface, making it easy to apply the same set of input signals to all models.

Everything related to the cost function, is implemented in the MPC-block, depicted in **Figure 5.2**. By handling the cost function in the Modelica model, the Optimica code very conveniently boils down to extending the Modelica model and simply pointing to the cost function defined in the MPC-block.

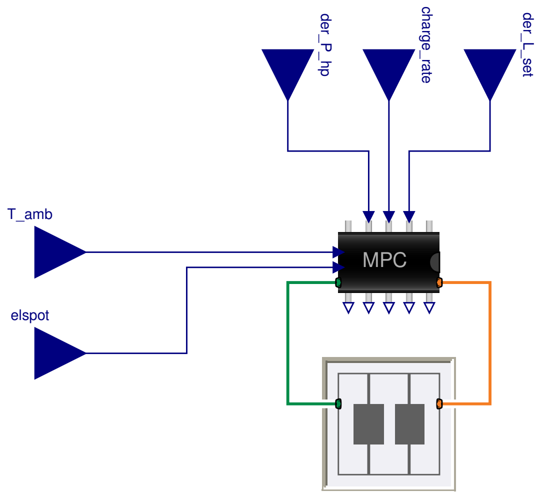


Figure 5.2: *The JModelica.org-exposed Modelica model; routing inputs to Real signals, through an MPC-block. The MPC-block implements the cost function is Modelica code; this enables the evaluation of the cost function, even when just simulating the system to fixed input signals.*

5.2 JModelica.org domain

Moving from the Modelica domain to the JModelica.org domain, is handled by compiling FMUs for each model and by compiling Modelica models and Optimica models to DOP using the Modelica compiler in JModelica.org. This is depicted in **Figure 5.1**. The linear C^2 model is derived by linearizing the C^2 model using methods available through `pyfmi` in the JModelica.org domain. This is illustrated in **Figure 5.1**, by the arrowing from the C^2 simulator FMU to the linear C^2 simulator FMU and DOP. The *simulator* objects in **Figure 5.1**, are effectively implemented as a Python *Simulator class*, wrapping simulation functionality available through `pyfmi` and extending with data aggregation functionality. Thus, the three simulator objects expose almost the same interface. MPC and EKF have also been implemented as Python classes.

5.2.1 MPC class

The purpose of the MPC class, it to wrap the optimization tools provided by JModelica.org in an appropriate abstraction level. The desire is construct a class, that boiled down exposes two methods; an `update` method that takes state information and sets up a new optimization problem to be solved and a `sample` method, that solves the optimization problem and returns optimal inputs to be applied for a single sample. Upon instantiating an object of the class, everything else is setup behind the scenes, given options provided in a a suitable data structure – thus allowing an MPC simulation loop with minimum code overhead.

The implementation of MPC is divided into three step:

- Initialization step: OCP is transcribed to NLP
- Update step: Initial conditions, start time (t_0) and final time (t_f) are updated
- Solution step: NLP is solved

It is desirable to limit the computational effort on-line, even though the sample time has been chosen as $t_s = 30\,\text{min}$, which in a real-world scenario gives plenty of time for computations to complete. By limiting the amount of computations on-line, the simulation loop can be executed faster. Thus, the initialization step is kept off-line, only transcribing the OCP to an NLP once. This is possible, as JModelica.org provides methods to alter the resulting NLP, manipulating parameters such as initial conditions for states, start time, t_0, and final time, t_f.

A good initial guess for the optimizer is essential[12]. To provide a good initial guess on-line, JModelica.org provides *warm starting* of the optimizer; using the previous solution as an initial guess for the subsequent optimization. A first initial guess is provided by using nominal trajectories.

Numerical optimization performance is highly dependent on how well scaled the problem is; with poor scaling resulting in slow convergence or divergence. Scaling is especially important for thermo-hydraulic model, as the difference in magnitude of the states (e.g. temperature in °C versus pressure in Pa) often provides ill-conditioned systems. Scaling is handled automatically in JModelica.org, by providing nominal trajectories in the form of a previous simulation. The maximum absolute value of all variables is then used in a scaling. Scaling is handled before transcription; this can prove a problem if the trajectories move to far away from the nominal trajectories used for scaling. It is however possible to update scaling factors on-line directly in the NLP, if deemed necessary.

It is desirable to provide trajectories with power price data, κ_{elspot}, and ambient temperature, T_{amb}. JModelica.org allows specifying external data (e.g. references, disturbances) for an optimization problem before transcription. These exogenous inputs are thus also transcribed. Extra care had to be given here, since the NLP is discrete, the exogenous inputs are after transcription not specified by time, t, but by sample number, k.

Consider an example, of a price increase at $t = 1\,\text{h}$. With a sample time of $t_s = 30\,\text{min}$, the price increase would in the NLP happen at sample $k = 2$. This works the first time the NLP is solved. For the next iteration, the start time, t_0, and final time, t_f, have been updated to reflect the moving horizon, but the price increase is still fixed at $k = 2$, thus happening one hour into the future (given the specified sample time). The exogenous inputs have to be shifted one sample per iteration, to adhere to the moving horizon. This feature is essential for this thesis and is not implemented in existing MPC frameworks, e.g. the one described and implemented in [19].

5.2.2 EKF class

The purpose of the EKF class is equivalent to the MPC class; to wrap the underlying algorithm in an appropriate abstraction level. The desire is to construct a class that, boiled down, exposes three methods; a method computing the *time update step*, a method computing the *measurement update step* and a method that returns the current state estimate, $\hat{\boldsymbol{x}}$. The EKF implementation is straight forward, given the account of the algorithm in **Section 2.5**. Especially when using FMUs, as *pyfmi*, introduced in **Section 2.6.1**, provides methods to obtain linearized systems.

Instantiating an object of the EKF class, requires an FMU – together with covariance matrices, sample time and configurations related to measurable states. The FMU is used to both provide on-line linearization but it is also used to simulate the nonlinear model; providing both $\hat{\boldsymbol{x}}_{k|k-1}$ and $\hat{\boldsymbol{y}}_{k|k-1}$; state and measurement estimates given no new measurement information.

5.3 Simulation setup

The simulation loop is conceptually implemented, as the control loop depicted in **Figure 5.3**. The exogenous inputs, T_{amb} and κ_{elspot} used in simulation, is historical data provided by Added Values P/S. The data provided is sampled with a sampling time of 1 h. The power price, κ_{elspot}, is for Western Denmark in 2015. The ambient temperature is also for 2015, but is sampled in Wales. This inconsistency is acceptable, given that the entire project is on a conceptual basis and since the T_{amb} data fits with the designed nominal ambient temperature of 10 °C. To distinguish between using T_{amb} and κ_{elspot} for prediction in MPC and for simulation, noise is applied before applying the inputs in simulation. Each T_{amb} sample is applied noise sampled from a normal distribution with standard deviation $\sigma = 0.1$ and each κ_{elspot} sample is applied noise sampled for a normal distribution with standard deviation $\sigma = 10$.

Before running the simulation loop, all states are initialized in steady-state. The entire simulation loop is run with a sampling time of 30 min, equivalent to that of the designed MPC and EKF. A single iteration consists of the following steps:

Update MPC Updating initial conditions given $\hat{\boldsymbol{x}}_k$, final time, shifting exogenous input

Sample MPC Solving the OCP, obtaining input $\boldsymbol{u}_k$

EKF time update EKF time update step, given the new input signal

Apply $\boldsymbol{u}_k$ Simulate all three models with $\boldsymbol{u}_k$ and noisy exogenous inputs

Obtain noisy measurement Select measured states and add noise accordingly

EKF measurement update Use measurement in EKF to obtain state estimate, $\hat{\boldsymbol{x}}_k$

All three models are always simulated, to always provide the necessary data to perform comparisons.

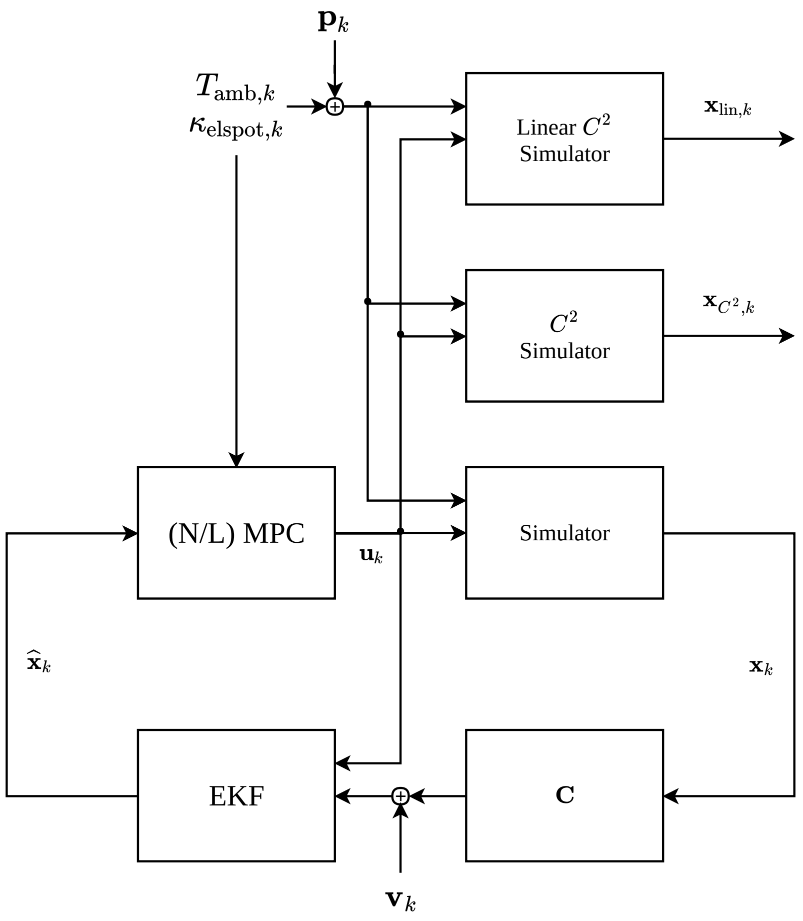


Figure 5.3: *The implemented simulation loop. All three models are always simulated for comparison purposes. Noise is added to exogenous inputs before applying them, to distinguish from the inputs used for prediction. Measurement is obtained, by selecting measured states through* **C** *and applying noise.*

Results 6

This chapter will account for and discuss simulations performed, using the designed control strategy described in **Chapter 4 Control** and the designed simulation framework described in **Chapter 5 Implementation**. First, a simulation has been performed, to a set of predefined ambient temperature and power price sequences, in order to provide a well-defined simulation scenario, where causality is more easily identified. The main performance evaluation, will however be by a very long simulation, using historical data for both ambient temperature and power prices.

6.1 Preliminary simulation

The preliminary simulation, has been run with the exogenous inputs given in **Figure 6.1**. These inputs define clearly distinguishable events, with low and high prices and low and high ambient temperatures, without overlap. The preliminary simulation has been performed – both for NMPC and LMPC – with exactly the parameterization derived in **Chapter 4 Control**. The preliminary simulation was run on a Lenovo ThinkPad T530, Intel i5-3320M @ 2.60 GHz and 4GB RAM.

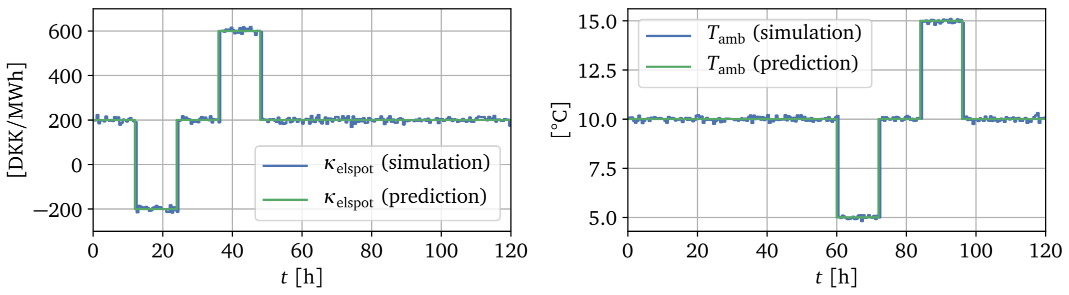


Figure 6.1: *Exogenous input sequences used for preliminary simulation.*

6.1.1 Control inputs

In **Figure 6.2**, the resulting control input trajectories are shown. Generally, NMPC uses more control input than LMPC. Looking at the load setpoint for the CHP, L_{set}, it is from $t = 0$ h drawn to its minimum value of 0.3, as the CHP is more expensive to use comparing to the HP. However, when the power price rises to 600 DKK/MWh at $t = 36$ h, the CHP load is ramped up, to produce more power to sell. NMPC uses the CHP more than LMPC. When the price drops down to 200 DKK/MWh again, the CHP is ramped down to minimum load again.

The HP is used extensively through the simulation, as it is on average the cheapest production unit. At the base power price of 200 DKK/MWh, the HP in powered with approximately 2 MW. When the price goes negative, the incentive to use the HP increases (as it will cost money to send produced power to the grid) and both the NMPC and LMPC increases power to the HP; more in the case of NMPC. When the price rises to 600 DKK/MWh, the incentive to use the HP is very low and it it almost powered down entirely. At $t = 60$ h, the ambient temperature drops, increasing the heat consumption of the consumers. As a result of this, the HP power input is ramped up, because it is at that time the cheapest production unit. Equivalently, when the ambient temperature rises, the heat consumption decreases and the HP power input is lowered.

The accumulator is used very differently by NMPC and LMPC. The overall takeaway is, that the NMPC uses the accumulator more aggressively, charging and discharging more often. Both discharge during the first 12 h; the accumulator tank is half full, so plenty of heat is available at almost no cost[1]. When the price drops below zero, both the NMPC and the LMPC start charging, using excess heat produced by the HP, as it is free. When the price reaches 200 DKK/MWh again, the accumulator is in both cases again discharged. At $t = 36$ h the price reaches 600 DKK/MWh and the CHP load is increased to produce more power to sell, however increasing the load also has the CHP produce more heat; the heat production and consumption is balanced, by charging the accumulator with the excess heat. In the LMPC case, the CHP load is simply kept lower, to not produce excess heat. The LMPC keeps discharging the accumulator until zero charge is reached – except for when the temperature rises, decreasing the consumer heat consumption.

[1]Only control input penalty.

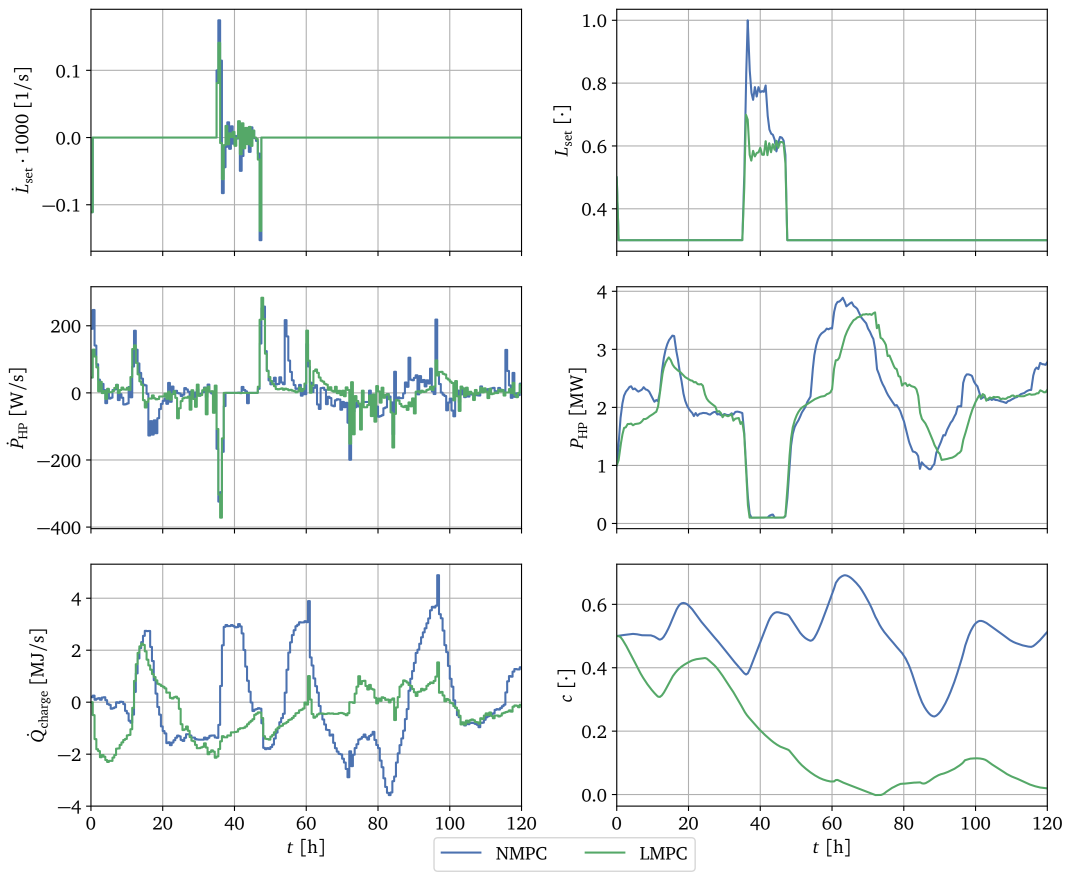


Figure 6.2: *Control inputs from preliminary simulation, for both NMPC and LMPC. Showing both derivatives (the actual manipulated variables) and the correspondingly integrated signals.*

6.1.2 EKF performance

In **Figure 6.3** the EKF provided state estimates of the three non-measurable consumer states are shown, together with the estimated consumer heat flow. To investigate the performance of the EKF, the actual states in the simulation model are shown. As discussed in **Section 3.10**, the simulation model includes a stochastic consumer model, featuring five groups of consumers, resulting in five actual states pr. state shown in **Figure 6.3**. The EKF however, is based upon the C^2 nonlinear model and thus only includes a single group of consumers.

The EKF provides reasonable state estimates, given no tuning of the covariance matrices. The consumer heat flow estimate shows a steady-state error but nothing too alarming. It is concluded, that no tuning is required and that the estimates are usable for control. Especially since both NMPC and LMPC works with the EKF state estimates.

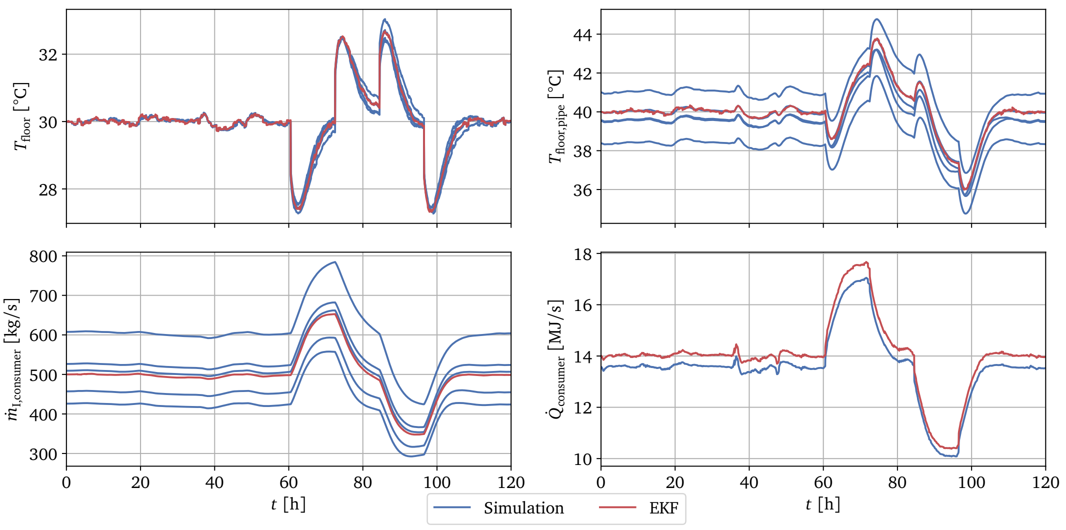


Figure 6.3: *EKF performance by a comparison to non-measurable consumer states. Deviations are acceptable, considering no measurements. A small steady-state error appears on the consumer heat flow. Only showing response from simulation with NMPC; no difference in performance when using LMPC.*

6.1.3 Heat production and consumption

In **Figure 6.4**, an overall view of heat production and consumption is given. Just as noted in **Section 6.1.1**, the responses provided by NMPC and LMPC are similar, the main difference being the use of the accumulator. The boiler is generally not very active, given the high cost of heat production, but due to the ideal pressure difference control, the response is very jumpy. The boiler is mainly active when the accumulator is charged or discharged. Another interesting effect visible, is that the LMPC HP response almost seems delayed compared to the NMPC response. **Figure 6.4** also shows how the NMPC uses excess heat from the CHP, when power prices are high, to charge the accumulator.

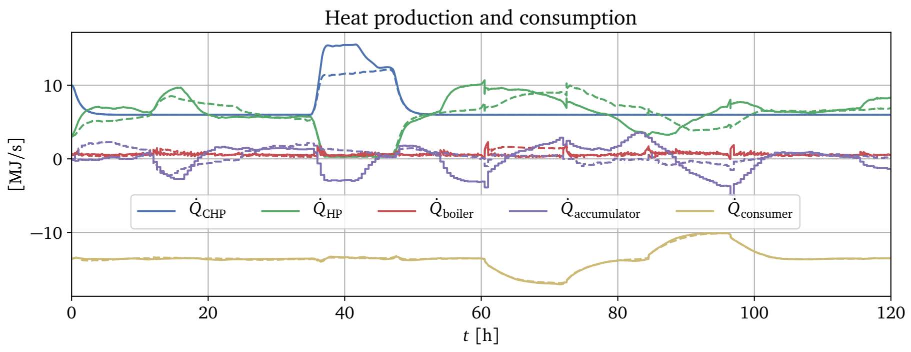


Figure 6.4: *Heat production and consumption; using NMPC (solid) and LMPC (dashed). Noticeable difference in use of accumulator.*

6.1.4 Supply and return temperature

In **Figure 6.5** the supply and return temperatures are given. The supply temperature is kept very close to the desired 80 °C by the temperature controllers in the production units. The return temperature is close to the nominal return temperature of 40 °C. Also shown in **Figure 6.5** is the COP for the HP, as it depends on the supply temperature and the ambient temperature; with a seemingly steady supply temperature, the changes in COP is mostly due to the changes in ambient temperature. The change is COP does not seem to motivate either controller to use the HP more, as it is already the cheapest production unit.

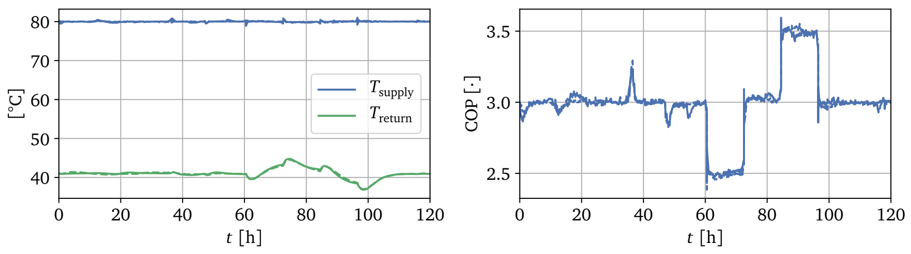


Figure 6.5: *Supply temperature, return temperature and COP; using NMPC (solid) and LMPC (dashed).*

6.1.5 Power trade

In **Figure 6.6** variables related to power trade are shown. The traded power, P_{trade}, is the difference between power produced by the CHP, P_{produced}, and the power consumed by the HP, P_{consumed}. During the time where the power price is 200 DKK/MWh, $P_{\text{produced}} = 3$ MW and $P_{\text{consumed}} \approx 2$ MW; leaving ≈ 1 MW to be sold. When the price drops to -200 DKK/MWh, it is undesirable to sell power letting more power be consumed instead. When the price increases to 600 DKK/MWh, it is desirable to produce and sell as much power as possible; and thus $P_{\text{consumed}} \approx 0$ MW and $P_{\text{trade}} \approx P_{\text{produced}}$.

When the ambient temperature drops to 5 °C, the heat consumption increases. The cheapest way to provide more heat is through the HP, buying the necessary extra power from the grid. As the ambient temperature rises to 15 °C, the heat consumption decreases, thus P_{consumed} is lowered and P_{trade} is equivalently increased.

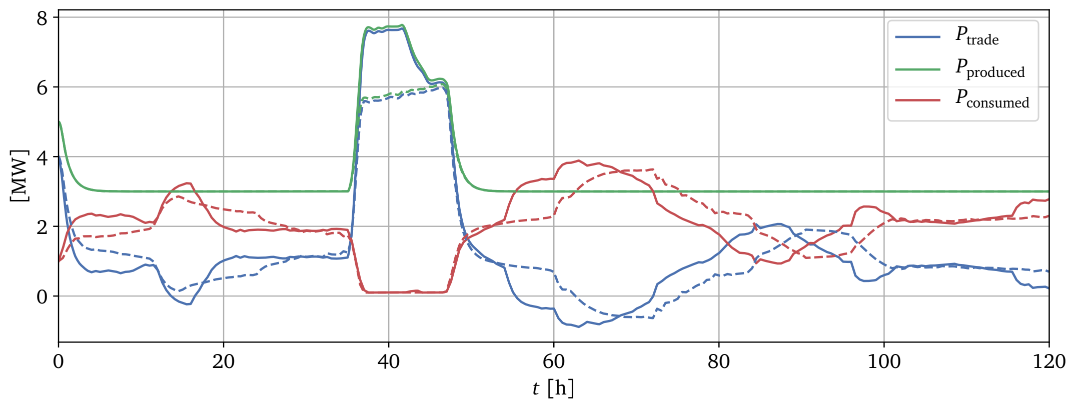


Figure 6.6: *Power trade; using NMPC (solid) and LMPC (dashed). Power consumption is power used by the HP and power production is power produced by the CHP.*

6.1.6 Cost function breakdown

In **Figure 6.7** a breakdown of the cost function is depicted. Here J is the combined cost function, described in **Section 4.1**:

$$J = \int_{t_0}^{t_f} R(t) - W(t) - D(t)\,\mathrm{d}t \tag{6.1}$$

The performance of NMPC and LMPC, is for this simulation almost identical, when looking at the value of the cost function. The same goes for a comparison of the cumulated revenue, $\int R$, showing that using either controller results in almost the same revenue trajectory. Comparing the control input penalty term, W, NMPC is generally punished more, which is due to the more aggressive use of the accumulator. Finally, the consumer QoS term, D, shows that the consumer floor temperature is kept close to the desired floor temperature – for both NMPC and LMPC; only deviating when the ambient temperature changes. The deviation is however only $\approx 2\,°\mathrm{C}$, which is deemed acceptable.

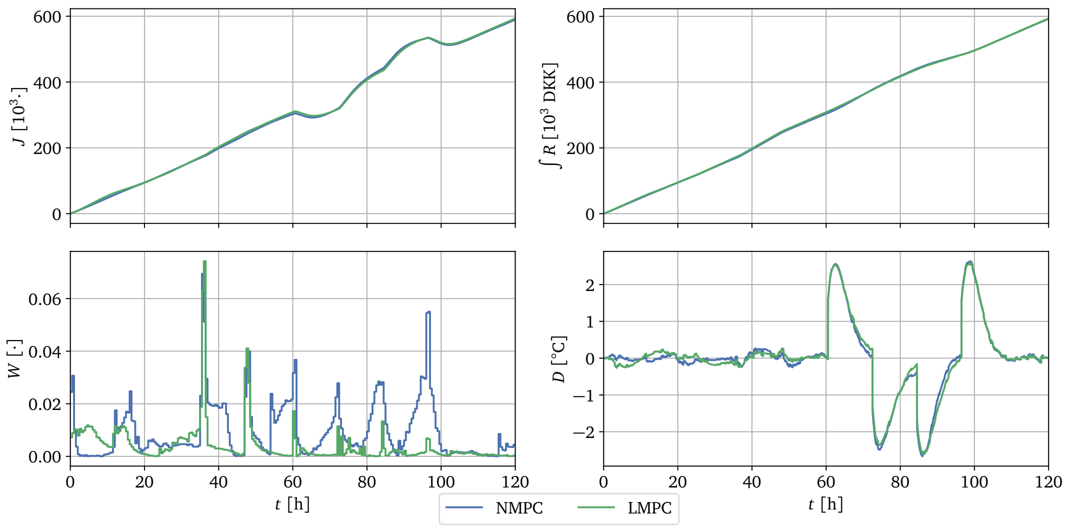


Figure 6.7: *Comparing NMPC and LMPC through the cost function and terms involved. Almost identical performance, except for the use of inputs.*

6.1.7 Solution time

In **Figure 6.8**, a comparison of MPC solution time has been made. It shows, for both NMPC and LMPC, the solution time, t_{sol} for each sample/iteration. NMPC is significantly slower than LMPC, requiring almost double the time to obtain a solution.

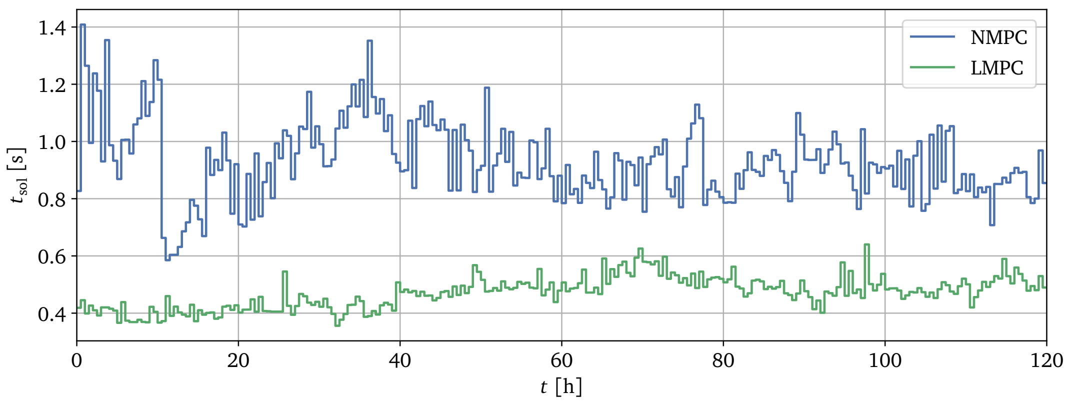


Figure 6.8: *Comparing NMPC and LMPC solution times. The plot shows the solution time, t_{sol}, for each iteration throughout the simulation.*

6.2 Long simulation

The long simulation has been run with exogenous inputs given in **Figure 6.9**. These inputs are from historical data of ELSPOT prices for Western Denmark in 2015 and temperatures from Wales in 2015, provided by Added Values P/S. The simulation time is set to 80 d; both to investigate robustness through real-world data and to provide a longer simulation on which to evaluate performance.

Also, due to the long simulation time, the hardware on which the simulation was run, has been changed, to reflect a significant increase in memory requirement. Thus, the long simulation was run on a desktop computer featuring an Intel i7-6700 @ 3.40 GHz and 16GB RAM. This has to be noted, since the processor change is reflected in faster solution times.

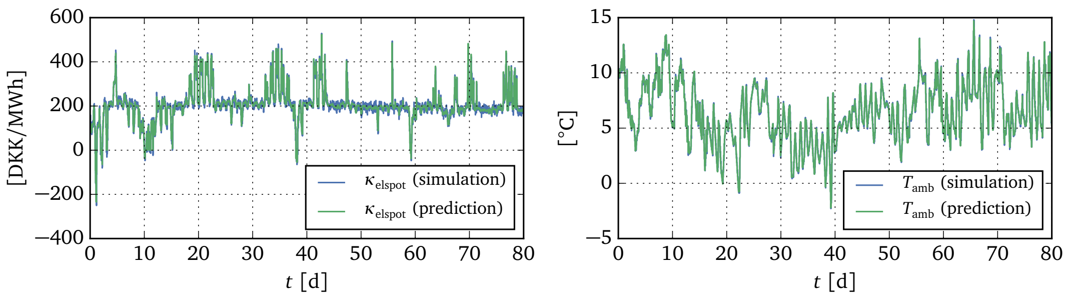


Figure 6.9: *Exogenous input sequences used for long simulation.*

6.2.1 Control inputs

In **Figure 6.10**, the resulting control input trajectories are shown. Generally, as for the preliminary simulation, NMPC uses more control input than LMPC. Looking at the load setpoint for the CHP, L_{set}, it is mainly kept at minimum load, except times with significantly higher power prices, e.g. at $t \approx 20$ d and $t \approx 35$ d, where the power price reaches 400 DKK/MWh.

The HP use is similar between the two controllers; the trajectories feature the same overall response, except for e.g. at $t \approx 20$ d, where NMPC turns down the HP, as the CHP is used at almost maximum load providing more than enough heat for the consumers. The reason for this difference may stem from the difference in use of the accumulator, between NMPC and LMPC.

The accumulator is almost neglected when running with LMPC. It is discharged from the beginning, leaving it almost empty through the entire simulation. On the other hand, NMPC uses it extensively, charging and discharging on a daily basis, following the daily change in ambient temperature – and when power prices are significantly high and low.

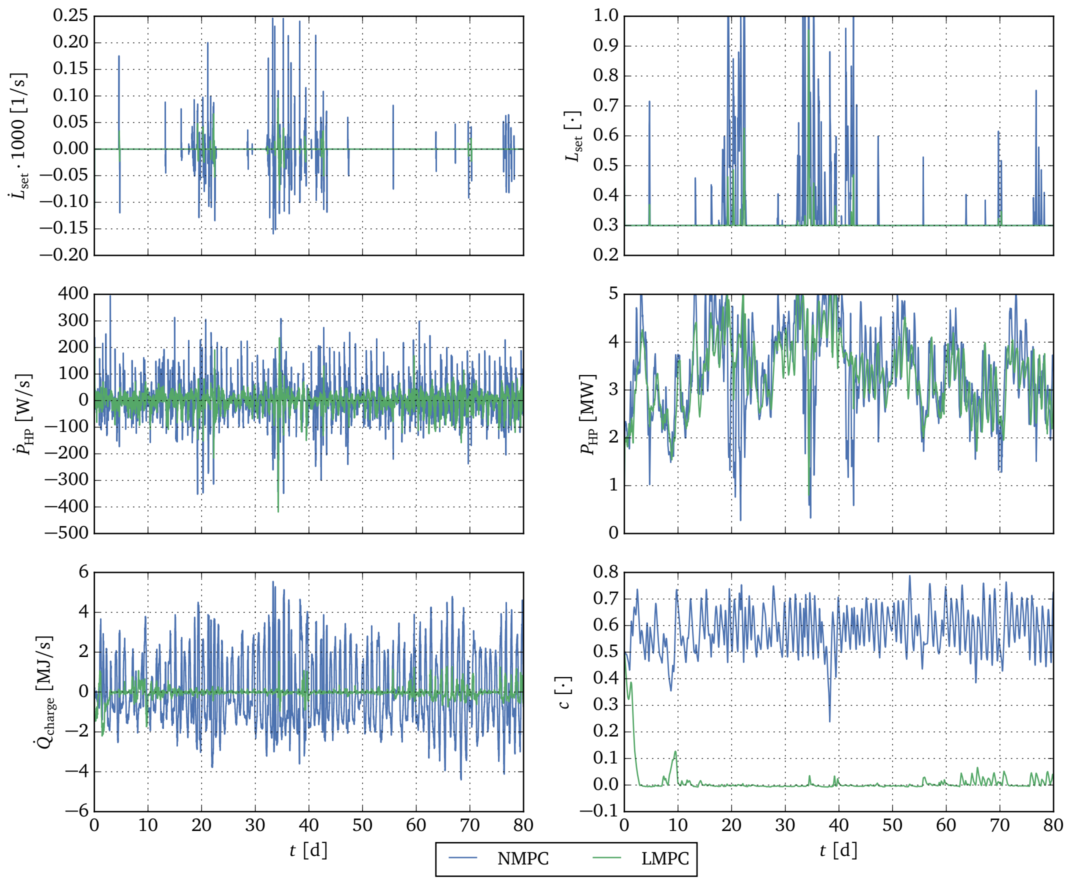


Figure 6.10: *Control inputs from long simulation, for both NMPC and LMPC. Showing both derivatives (the actual manipulated variables) and the correspondingly integrated signals.*

6.2.2 EKF performance

In **Figure 6.11** the EKF provided state estimates of the three non-measurable consumer states are shown, together with the estimated consumer heat flow. The EKF provides acceptable state estimations, just as shown for the preliminary simulation. The most notable performance issue, is the steady-state error on the consumer heat flow.

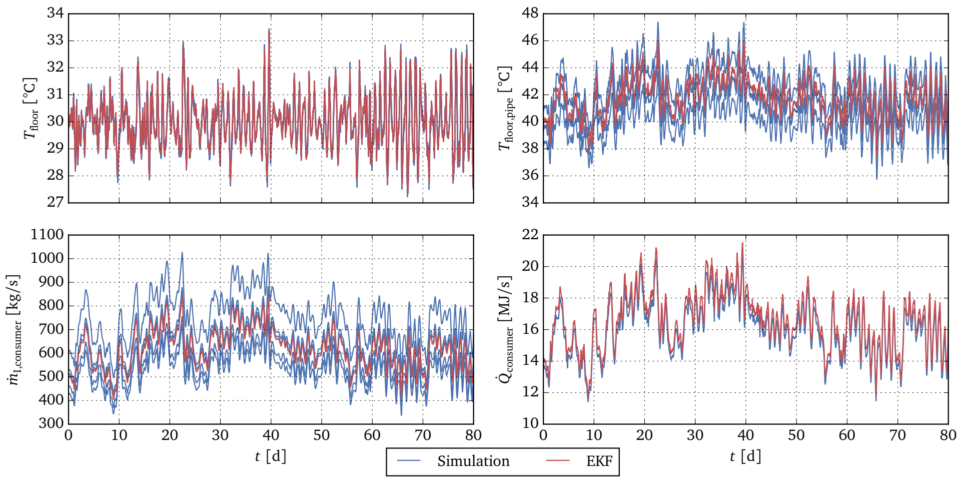


Figure 6.11: *EKF performance by a comparison to non-measurable consumer states. Only showing response from simulation with NMPC; no difference in performance when using LMPC.*

6.2.3 Heat production and consumption

In **Figure 6.12**, an overall view of heat production and consumption is given. Just as noted in **Section 6.1.1**, the responses provided by NMPC and LMPC are similar, the main difference being the use of the accumulator; this is reflected in the increased use of the boiler by LMPC.

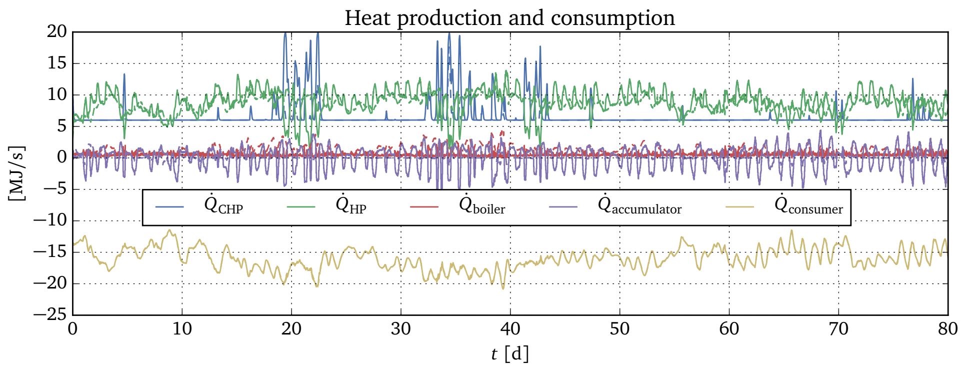


Figure 6.12: *Heat production and consumption; using NMPC (solid) and LMPC (dashed).*

6.2.4 Supply and return temperature

In **Figure 6.13** the supply and return temperatures are given. The supply temperature is kept close to the desired 80 °C by the temperature controllers in the production units. The return temperature is also close to the nominal return temperature of 40 °C.

The varying COP is also given in **Figure 6.13**. At $t \approx 40$ d, the ambient temperature drops below 0 °C, which has the COP drop below 2. It would be interesting to see a corresponding drop in use of the HP, but a drop in power price is featured at the same time; motivating the use of the HP, making it difficult to identify the effect of the COP on the use of the HP.

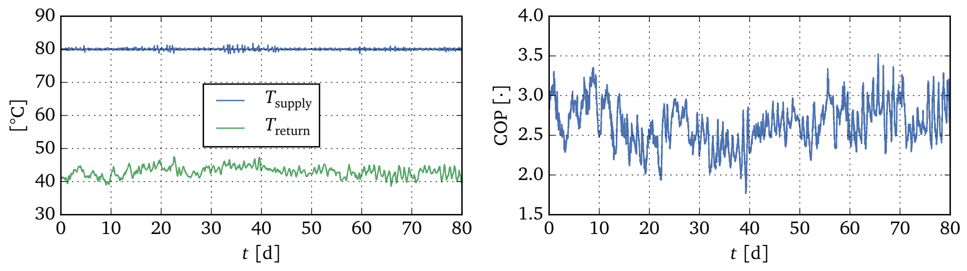


Figure 6.13: *Supply and return temperature; using NMPC (solid) and LMPC (dashed). The COP is dependent on ambient temperature and supply temperature.*

6.2.5 Power trade

In **Figure 6.14**, power trade is shown, together with power produced and power consumed. The most notable difference is given at high power prices, where NMPC uses the CHP more, to produce more power to sell.

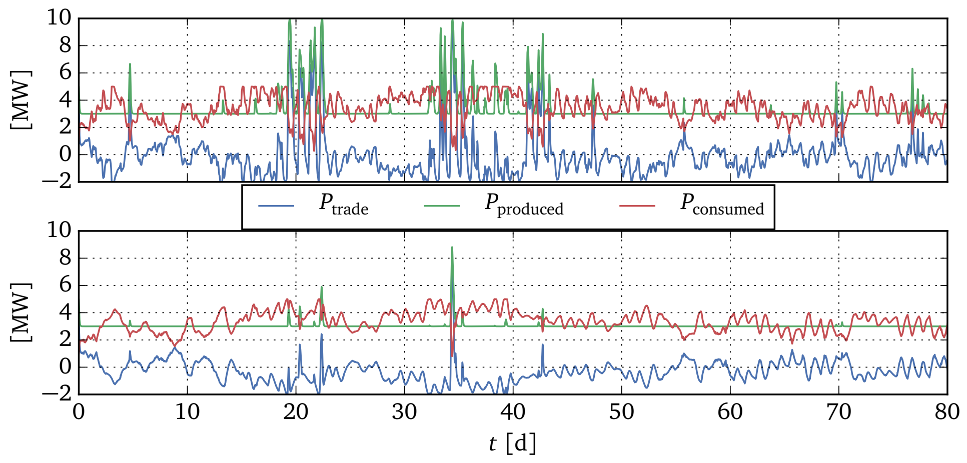


Figure 6.14: *Power trade; using NMPC (top) and LMPC (bottom). Power consumption is power used by the HP and power production is power produced by the CHP.*

6.2.6 Cost function breakdown

In **Figure 6.15** a breakdown of the cost function is depicted. The most interesting result, compared to the preliminary simulation, is that in the long run, NMPC features better performance, reflected in both the combined cost function and the cumulated revenue. The extensive use of control input by NMPC is also identifiable in the control input penalty term, W.

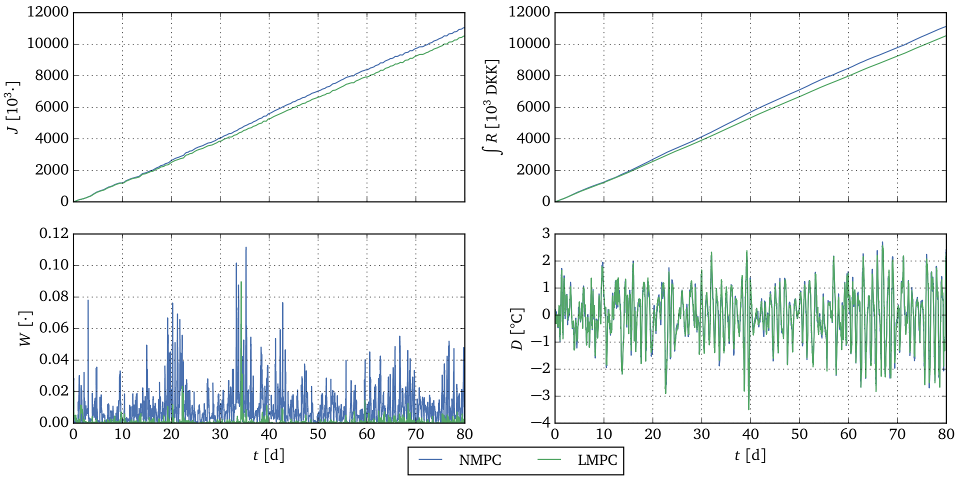


Figure 6.15: *Comparing NMPC and LMPC through the cost function and terms involved.*

6.2.7 Solution time

In **Figure 6.8**, a comparison of MPC solution time, t_{sol} for each sample/iteration, has been made. The comparison shows, that LMPC is faster and computationally more robust. Several iterations show very long solution times for NMPC and at $t \approx 38$ d, NMPC fails to converge to a solution. This incident could be related, to a low ambient temperature and a low power price happening at the same time.

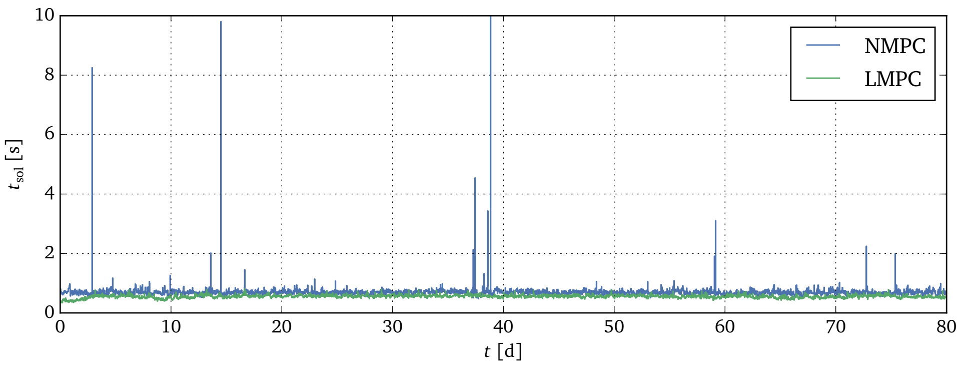


Figure 6.16: *Comparing NMPC and LMPC solution times. The plot shows the solution time, t_{sol}, for each iteration throughout the simulation.*

Conclusion 7

The aim of this thesis was to evaluate the use of NMPC as a control concept for production planning and balance control, for a fictitious combined power and district heating production portfolio, in a component-based modeling context. The fictitious production portfolio was chosen to include a CHP (producing power and heat), a compression HP (electrical power to heat), a peak-load boiler and an accumulator tank. These units were chosen with inspiration from the production portfolio at Sønderborg Fjernvarme.

The idea was to investigate the benefits of MPC, considering first-principle nonlinear models in the optimization problem, eliminating the need for linearization. This is desirable, as linearization potentially involves the loss of valuable information in an optimization context – and because linearization involves manual preconditioning labor. To evaluate the benefits, a comparison of NMPC and LMPC was necessary; the only difference being the use of either nonlinear or linear models.

A nonlinear C^2 continuous model of the production portfolio together with consumers has been built in Modelica. The model has to be C^2 continuous in order to enable its use in an optimization context. For the model to be C^2 continuous, extended use of the MSL was not possible, and a set of simplified Modelica fluid connectors and components were built to meet the requirement. A scalable consumer model was designed as a simplified house with floor heating, to reflect average heat consumption. Being scalable, the consumer model could parametrically be set to model the heat consumption of 10 000 consumers, comparable to the amount of consumers supplied by Sønderborg Fjernvarme. The production units were modeled equally simple, featuring only the most basic dynamics – but they were all built using the object-oriented capabilities of Modelica, allowing exchange for more detailed models.

A C^2 continuous model of a district heating system was obtained, complete with control inputs for the different production units, by connecting the production portfolio together with the consumers. This model was linearized around a single general-purpose operating point, obtaining a linear model of the system for use in LMPC. Through simulation studies, the linear model gave a response similar to the nonlinear model – but with a noticeable difference for two states when charging and discharging the accumulator.

Together with the C^2 continuous nonlinear model and the linear model, a third model was built; a simulation model. The simulation model was, to distinguish it from the C^2 model, constructed using components from the MSL, providing a more detailed model. For more interesting simulations, the simulation model includes an extended consumer model, featuring several groups of consumers with stochastic parameter variations. Simulations showed comparable responses between the C^2 model and the simulation model, with the most notable difference being the consumer response, as expected.

The MPC control concept, was designed to feature an economical cost function, maximizing revenue and consumer QoS, subjected to a changing power price and ambient temperature and with a simplified power market, allowing power to be bought or sold at the same price. To solve the included OCP, JModelica.org, a framework providing optimization tools on Modelica models, was employed. Using JModelica.org, direct collocation was used to transcribe the infinite dimensional OCP into a finite dimensional NLP. Under the assumption that consumer states are not measurable, an EKF was designed to provide full state information. A simulation framework was built around the APIs provided by JModelica.org, allowing simulations with both NMPC and LMPC, by providing it with Modelica models.

Simulation studies were conducted, to investigate the differences between NMPC and LMPC. A preliminary simulation, featuring a fabricated power price and ambient temperature with clearly distinguishable events – and a long simulation, featuring historical power prices and ambient temperatures, simulating the system over several months. The two simulations showed acceptable performance of the EKF, with generally good state estimates but with a noticeable steady-state error on the estimated heat consumption.

Comparing NMPC and LMPC, an overall similar response was obtained through both the preliminary simulation and the long simulation. The main difference was in the use of the accumulator; where NMPC showed aggressive use of the accumulator, LMPC showed the desire to always empty the accumulator – only very special cases (e.g. very low power prices) motivated the LMPC to charge. This difference in accumulator use also propagated to the use of other production units. One noteworthy example; when power prices were high, the incentive was high for the CHP to run at a high load, producing power to be sold. In the NMPC case, it would run the CHP at full load, charging the accumulator with the excess heat. In the LMPC case, it would run the CHP at a lower load, not utilizing the capacity of the accumulator. It is very interesting to note, that it was also accumulator use, that featured the most noticeably difference in simulation response, when comparing the nonlinear model to the linear model.

Looking at the preliminary simulation, no difference in performance between NMPC and LMPC was identifiable, when looking at the cost function. However, when looking at the long simulation, it was very clear that NMPC was better performing; both in term the entire cost function but also considering only the revenue term. This performance difference is likely due to the more active use of the accumulator by NMPC. The lack of accumulator use by LMPC could perhaps be alleviated by choosing a different operating point – or by configuring the accumulator differently – but the likely conclusion to be drawn, is that information lost through linearization meant that NMPC had the upper hand in the performance comparison.

There is however also a drawback when using NMPC, as a comparison of MPC solution times showed the LMPC to provide solutions significantly faster and more robust, with the NMPC sometimes not converging to a solution at all. The solution time aspect is however, in a real-world implementation not as important, considering the sample time chosen – but numerical robustness is important and contingency strategies have to be considered.

Considering the resulting MPC control concepts in a real-world context, the simplifications made do not allow for any noteworthy results. Important aspects not included in the models are e.g. transport delays and heat loss, which would be very interesting to add. The power market model would also need to be updated. Thus, the results are only interesting in a purely conceptual sense.

However, the method of posing first-principle models in Modelica and using these models directly in an OCP for MPC – with a minimum amount of preconditioning labor – was by the author found to be very productive and it is highly encouraged to actively pursue this approach. The biggest disadvantage, considering the specific tools, currently lies in the lack of a MSL fluid interface, that supports the C^2 constraint, to avoid lock-in to a custom interface. If the interface could allow for both detailed medium models and C^2 models, a greater level of component-reuse is achievable.

Bibliography

[1] Energinet.dk. *New record-breaking year for Danish wind power*. URL: `http://energinet.dk/EN/El/Nyheder/Sider/Dansk-vindstroem-slaar-igen-rekord-42-procent.aspx` (visited on 11/11/2016).

[2] Energitilsynet. *Fjernvarmestatistik december 2015*. URL: `http://energitilsynet.dk/varme/statistik/fjernvarmestatistik/december-2015/` (visited on 11/11/2016).

[3] Kristian Edlund. *Dynamic Load Balancing of a Power System Porfolio*. 2010. ISBN: 978-87-92328-34-2.

[4] Stéphane Velut et al. *Non-linear and Dynamic Optimization for Short-term Production Planning*. 2013.

[5] Per-Ola Larsson and Stéphane et al. Velut. "Production Planning for Distributed District Heating Networks with JModelica.org". In: *11th International Modelica Conference* (2015).

[6] Michael Tiller. *Introduction to Physical Modeling with Modelica*. 2001. ISBN: 0-7923-7367-7.

[7] Frank Allgöwer, Ralf Findeisen, and Zoltan Nagy. "Nonlinear Model Predictive Control: From Theory to Application". In: *Journal of the Chinese Institute of Chemical Engineers* (2004).

[8] Sønderborg Fjernvarme. *About*. URL: `http://www.sonderborg-fjernvarme.dk/kort-om-os/` (visited on 11/11/2016).

[9] Fredrik Magnusson. *Numerical and Symbolic Methods for Dynamic Optimization*. 2016. ISBN: 978-91-7753-005-3.

[10] Modelica Association. *Modelica Website*. URL: `http://modelica.org/` (visited on 12/16/2016).

[11] John T. Betts. *Practical Methods for Optimal Control Using Nonlinear Programming*. 2001. ISBN: 0-89871-488-5.

[12] Fredrik Magnusson and Johan Åkesson. "Dynamic Optimization in JModelica.org". In: *Processes* (2015).

[13] Filip Jorissen, Michael Wetter, and Lieve Helsen. "Simulation Speed Analysis and Improvements of Modelica Models for Building Energy Simulation". In: *11th International Modelica Conference* (2015).

[14] Kirstine Nærvig Petersen and Kirsten Gram-Hanssen. *Husholdningers energi- og vandforbrug*. 1st. 2005. ISBN: 87-563-1231-8.

[15] Nord Pool Markets. *Market Data*. URL: `http://www.nordpoolspot.com/` (visited on 05/30/2017).

[16] Lars Grüne and Jürgen Pannek. *Nonlinear Model Predictive Control*. 2nd. 2016. ISBN: 978-3-319-46023-9.

[17] M. Diehl, R. Amrit, and J.B. Rawlings. “A Lyapunov function for economic optimizing model predictive control”. In: *IEEE Transactions on Automatic Control* (2011).

[18] D. Angeli, R. Amrit, and J.B. Rawlings. “On average performance and stability of economic model predictive control”. In: *IEEE Transactions on Automatic Control* (2012).

[19] Magdalena Axelsson, Fredrik Magnusson, and Toivo Henningsson. “A Framework for Nonlinear Model Predictive Control in JModelica.org”. In: *11th International Modelica Conference* (2015).

Software setup and run A

This appendix contains as guide to setting up the designed simulation framework. The guide assumes the host PC to be running GNU/Linux 64-bit; attempts at using the software under Windows or OS X has not been made.

A.1 Dependencies

The software depends on JModelica.org and Dymola. For this thesis, JModelica.org 1.17 (`r9313`) was used. During the work on the thesis, JModelica.org 2.0 was released; however first attempts at moving to the new release showed issues, when compiling Modelica models. Thus, for reproduction of results, it is advised to use `r9313`. JModelica.org was compiled to use IPOPT 3.12.4 and Java 7 – it was later found that Java 8 was also compatible.

Dymola 2017 has been used.

A.2 Setup

The software is available in the `.zip` archive, handed in together with the thesis. It has the following tree structure when extracted:

```
/
├── design/
│   ├── control/
│   ├── data/
│   ├── model/
│   └── start.mos
├── documentation/
├── setup.sh
└── README.md
```

The `documentation` folder contains notes on installing JModelica.org and Dymola. After extraction, the setup consists of setting up environment variables. This is conveniently handled, by sourcing the `setup.sh` file in the root folder. Sourcing is done by issuing the following command:

```
$ . setup.sh
```

The `setup.sh` file assumes that JModelica.org is installed in `/opt/jmodelica` and that IPOPT is installed in `/opt/ipopt`. This has everything setup for single session use; for continuous use, ensure that the shell of choice sets up the environment variables upon startup.

A.3 Running

Simulations are started, using the `design/control/run.py` script. The script can be run without arguments; this will start a simulation using NMPC, with $t_f = 1\,\mathrm{d}$:

```
$ cd design/control
$ ./run.py
```

NOTE; *JModelica.org depends on Python 2.7, thus ensure that the current environment uses Python 2.7 as default, otherwise, start the script using the Python 2.7 interpreter directly.*
The `run.py` script exposes several command-line arguments, to configure the simulation:

```
# Simulate 5 days w. LMPC, w. historical data from day 20
$ ./run.py -c LMPC --start 20 --length 5
# Simulate 10 days w. NMPC, w. historical data from day 100
$ ./run.py -c NMPC --start 100 --length 10
```

The software employs caching of compiled FMUs, these are stored in `design/control/FMUs`. To clear the cache (if changes to Modelica model has been made), delete the contents of the folder or use the `-clear` flag, when simulating:

```
# Simulate w. clear cache; forcing re-compilation of FMUs
$ ./run.py --clear -c NMPC --start 10 --length 10
```

Simulation output, is provided in the *HD5* format, with trajectories for all model variables. Configurations to simulation parameters; e.g. MPC parameters or EKF covariances are done in `design/control/helpers/config.py`. A successful run, will exit with the following output:

```
$ ./run.py
...
0: Solve_Succeeded in 28 iterations
1: Solve_Succeeded in 40 iterations
2: Solve_Succeeded in 39 iterations
...
47: Solve_Succeeded in 33 iterations
Simulation complete (setup: 13.0 s, sim: 73.0 s)
```

Software documentation B

The software is available in the `.zip` archive, handed in together with the thesis. When extracted, the following source tree is available:

```
/
├── design/
│   ├── control/
│   ├── data/
│   ├── model/
│   └── start.mos
├── documentation/
├── setup.sh
└── README.md
```

The `design` folder holds the software, consisting of two parts; a Modelica model residing in `model` and a simulation framework built using JModelica.org, residing in `control`. The `data` folder contains the historical data used in simulations.

B.1 Simulation framework

The simulation framework, is constructed by a series of Python modules and classes. The source tree is given as:

```
control/
├── controllers/
│   ├── MPC.py
│   └── EKF.py
├── dhsim/
│   └── simulators.py
├── helpers/
├── simulation.py
├── run.py
├── problem.mop
└── gen_dhp_fmu.mos
```

The `controllers` module contains classes for MPC and EKF and the `simulators` contains simulator classes, extending on the functionality provided by `pyfmi` to simulate FMUs. The `helpers` module contain several helper functions, used extensively in the other modules. In `simulation.py`, the main simulation loop is given and `run.py` is a convenient script, that starts the simulation loop exposing simulation parameters as command-line arguments. The optimization problem is formulated in Optimica in `problem.mop` and `gen_dhp_fmu.mos` is a Dymola script, compiling the simulation model to an FMU.

B.1.1 MPC class example

The MPC class is instantiated and used as:

```
from controllers import MPC
# Instantiation:
mpc = MPC(
    problem, data, init_trajectory, nom_trajectory,
    settings
)
# k = sample number, x_k = state
mpc.update(k, x_k)
# time = solution time, u_k = new control input
time, u_k = mpc.sample(k)
```

Parameters used for instantiating an MPC object:

`problem`	a DOP compiled with JModelica.org
`data`	exogenous input trajectories
`init_trajectory`	initial solution guess as simulation trajectory
`nom_trajectory`	nominal trajectory for variable scaling
`settings`	sample time, prediction horizon and collocation points

B.1.2 EKF class example

The EKF class is instantiated and used as:

```
from controllers import EKF
# Instantiation:
ekf = EKF(C2_simulator, settings)
# k = sample number, x_hat_k = state estimate
# u_k = control input
ekf.time_update(k, x_hat_k, u_k)
# y = measurements
ekf.measurement_update(y)
x_hat_k = ekf.get_x_hat()
```

Parameters used for instantiating an EKF object:

`C2_simulator`	Simulator object (FMU abstraction)
`settings`	Sample time, measured states, $\boldsymbol{P}_0$, $\boldsymbol{Q}$, $\boldsymbol{R}$

B.1.3 Simulator class example

The `Simulator` classes, provides an appropriate abstraction on top of the FMU class. The main functionality being steady-state initialization per default, providing trajectory aggregation functionality, maintaining a compiled FMU and DOP and exposing initial and nominal trajectories, for easy use together with the MPC class. The `Simulator` classes are instantiated and used as:

```python
# One class for each model considered in the project,
# but they all extend the same base class
from dhsim import SimulatorC2, LinearSimulatorC2, Simulator
import helpers

models = helpers.get_models()
# Instantiation (models is a dict() pointing to Modelica models):
simulators = {}
simulators['C2'] = SimulatorC2(models['C2'])
simulators['linC2'] = LinearSimulatorC2(models['linC2'])
simulators['sim'] = Simulator(models['sim'])

u_k = helpers.constant_input(
    {'T_amb': 10, 'der_L_set': 0, 'der_P_hp': 0, 'charge_rate': 0}
)
# Simulations, starting in steady-state (state parameter)
# with the input u_k. The append parameter, ensures that
# resulting trajectories are appended to an underlying data structure
simulators['C2'].simulate(
    state=simulators['C2'].x_ss,
    start_time=0, final_time=3600,
    input=u_k, append=True
)
simulators['linC2'].simulate(
    state=simulators['linC2'].x_ss,
    start_time=0, final_time=3600,
    input=u_k, append=True
)
simulators['sim'].simulate(
    state=simulators['sim'].state_ss,
    start_time=0, final_time=3600,
    input=u_k, append=True
)

# Get state
x_C2_k = C2.get_state()
x_linC2_k = linC2.get_state()
sim_state_k = sim.get_fmu_state()
# Get aggregated trajectories for all variables
C2_res = C2.aggregated_result()
linC2_res = linC2.aggregated_result()
sim_res = sim.aggregated_result()
```

B.1.4 Minimal NMPC loop example

A condensed example, of how an NMPC simulation loop is put together, using the above mentioned modules, simulating only on the simulation model – using historical data.

```
from dhsim import SimulatorC2, Simulator
from controllers import MPC, EKF
import helpers

models = helpers.get_models()
settings = helpers.get_settings(24) # 24 hour simulation

SIM = Simulator(models['sim'])
C2 = SimulatorC2(models['C2'])
SIM.build_state_map(C2.x_ss)

data = {}
data['elspot'] = helpers.create_external_data(
    *helpers.get_elspot(0, 24 + 10)
)
data['T_amb'] = helpers.create_external_data(
    *helpers.get_tamb(0, 24 + 10)
)

mpc = MPC(
    C2.problem, data, C2.initial_trajectory,
    C2.nominal_trajectory, settings['MPC']
)
ekf = EKF(C2, settings['EKF'])

sim_state_k = SIM.state_ss
x_hat_k = C2.x_ss
u_k = ()

for k in range(settings['N_samples']):
    mpc.update(k, x_hat_k)
    time, u_k = mpc.sample(k)
    ekf.time_update(k, x_hat_k, u_k)

    SIM.simulate(
        state=sim_state_k,
        start_time=k*mpc.Ts, final_time=(k+1)*mpc.Ts,
        input=u_k, append=True
    )
    sim_state_k = SIM.get_fmu_state()

    ekf.measurement_update(
        SIM.get_measurement(
            ekf.outputs, settings['measurement_noise_std']
        )
    )
    # Get state estimate
    x_hat_k = ekf.get_x_hat()

sim_res = SIM.aggregated_result()
```

B.2 Modelica model

The Modelica model is found in `design/model`, and is divided into two component libraries; one for the district heating models (`model/DHP`) and one for the simplified C^2 continuous media model (`model/FluidJM`). The district heating component library has the following tree structure:

```
DHP/
├── Components/
├── Optimization/
├── DATA/
└── Icons/
```

The `Components` include all production portfolio models and consumer models, divided into the C^2 models and the simulation models. For each model in the production portfolio, `Variants` are available, allowing easy substitution with more detailed models. Also, the Modelica model is complete with `UnitTests` for each model, subjecting the models to known inputs, with fixed boundary conditions. The `Loops` models consists of the combined district heating system; using both C^2 models, simulation models and the linear model. The `Interfaces` contain the interfaces shared among the different models, e.g. the input and output bus.

```
Components/
├── C2/
│   ├── Consumers/
│   │   ├── Consumer.mo
│   │   ├── Variants/
│   │   └── UnitTests/
│   └── Producers/
│       ├── PowerPlants/
│       │   ├── CHP.mo
│       │   ├── Variants/
│       │   └── UnitTests/
│       ├── HeatPumps/
│       ├── Boilers/
│       └── Accumulators/
├── SIM/
│   ├── Consumers/
│   └── Producers/
├── Loops/
│   ├── LoopC2.mo
│   ├── LoopSIM.mo
│   └── LoopLIN.mo
└── Interfaces/
    ├── InputBus.mo
    └── OutputBus.mo
```

`DATA` contains shared parameters, `Optimization` contains the models exposed to JModelica.org, complete with the cost function.