Using Functional Mock-up Units for Nonlinear Model Predictive Control

Manuel Gräber[1] Christian Kirches[2] Dirk Scharff[3] Wilhelm Tegethoff[1,3]

[1]Technische Universität Braunschweig, Braunschweig, Germany
[2]Interdisciplinary Center for Scientific Computing (IWR), Heidelberg University, Germany
[3]TLK-Thermo GmbH, Braunschweig, Germany

Abstract

A software framework for prototyping of Nonlinear Model Predictive Control (NMPC) loops is presented that is based on the standardized model exchange format FMI (Functional Mock-up Interface). Arising optimal control problems are solved by an efficient implementation of the direct multiple shooting method, which is especially suitable for nonlinear and stiff system models. Using co-simulation, an optimizer, plant, and estimator can be coupled to a closed NMPC loop. Several stages of a typical control design process are supported, ranging from virtual simulation experiments to real plants with prototype NMPC controllers. Energy efficient control of vapor compression cycles is presented as example application of the proposed methods.

Keywords: Functional Mock-up Interface; Nonlinear Model Predictive Control; Vapor Compression Cycles

1 Introduction

Nonlinear Model Predictive Control (NMPC) provides promising possibilities to improve control accuracy, stability, as well as energy and economical efficiency of technical systems. The key idea is to utilize rigorous mathematical models of the controlled plant for online computations of appropriate control actions, based on the repeated solution of a dynamic optimization problem. Model-plant mismatch and disturbances are incorporated by updating the mathematical model according to estimates obtained from most recent measurement data. From the point of view of the numerical algorithms employed, these methods are well developed and ready to use. Their application to complex systems however by now is the subject of a few selected research projects only. The most prevalent reason for this may well be the considerably large effort required to develop fast implementations of large-scale accurate nonlinear models. The development of object-oriented and equation-based modeling languages such as Modelica aims at helping to considerably reduce this effort: systems can be conveniently modeled by composition from smaller, reusable subcomponents. Moreover, there no longer is the need to manually transform equations into a signal-oriented form.

In the last few years, Modelica has matured to a modeling language that is widely used for systems simulation in both academics and industry. More recently, research initiatives have come up that helped to extend the scope of Modelica beyond pure systems simulation. For example, [7] gives an overview over current research activities in the area, and shows possible further directions especially from a control design perspective.

Probably the first work reported in literature about dynamic optimization with Modelica models can be found in [12]. Therein the MATLAB S-Function format is used to interface Modelica models with an optimization solver. Dynamic optimization with models generated by the C-code export functionality of the Modelica tool Dymola is described in [16] and [25]. Both approaches suffer from the fact that the used model exchange formats are proprietary. In [26] the development of optimization based controllers in Modelica is addressed. But the authors remain unclear about the technical details how a Modelica model can be reused as internal model of the control algorithms.

A more integrated approach is described in [1]. Based on Optimica, a language extension of Modelica that serves to formulate optimization problems, an open source Modelica simulation and optimization tool has been developed that goes by the name *JMod-*

elica.org, see [2]. Therein, dynamic system models are formulated in the Modelica language and are symbolically transformed into a representation suitable for evaluation by numerical solvers. As is the case for most Modelica tools, not all parts of the Modelica language and the Modelica Standard Library are supported yet. Optimal control problems can be solved in *JModelica.org* by means of a *direct collocation* method.

As part of the ITEA-2 research project Modelisar, the standardized model exchange format FMI (*Functional Mock-up Interface*) [22, 3], has been developed. During the last two years, FMI gained a lot of attention and is now supported by over 20 simulation tools. A detailed list can be found on `http://fmi-standard.org`. The main purpose of FMI is to exchange models between different *simulation* tools. FMI is used to design nonlinear *Kalman Filters* for state and parameter estimation in [6]. To the best of our knowledge, there are no reports of FMI having been applied to *optimization* of dynamic systems, though.

1.1 Contribution

This article addresses the above described situation by presenting a software framework for fast and reliable prototyping of NMPC loops using the FMI standard [22]. The key idea is to use existing specialized software for each task and exchanging models between these tools, relying on FMI for the purpose. Using established modeling and simulation tools such as Dymola, one can conveniently set up large-scale and complex system models. Exported as FMI models, called FMUs (*Functional Mock-up Units*), we import these into the direct optimal control code `MUSCOD-II` [4, 9, 21]. `MUSCOD-II` is a software package for efficient numerical solution of optimal control problems. The implemented *direct multiple shooting method* is favorable especially for large-scale, highly non-linear, and stiff systems.

Using the co-simulation platform TISC [20], we also present a software solution for coupling optimization algorithms with simulation tools to conveniently test designed NMPC loops. Using existing interfaces to measurement and automation software NMPC controllers can also be connected to real plants.

With NMPC of a vapor compression cycle, we present a challenging but promising application and demonstrate the capability of our method.

1.2 Structure of the Paper

The paper starts with a description of the theoretical background of our methods. In Section 2 the underlying model class is defined. Based on this dynamic system model, a class of continuous *Optimal Control Problems* (OCPs) is formulated. The direct multiple shooting method is presented in Section 3 as an efficient numerical approach for the discretization and solution of OCPs. The control loop is closed in Section 4 by taking into account state estimates or measurements and repeatedly solving the OCP. In order to derive an efficient control algorithm, special attention is paid to reinitialization of subsequent optimization iterations and the separation of each iteration into different phases. Technical details of our methods are presented in Sections 5 and 6. We discuss optimization results for an example application in Section 7, using the presented toolchain and algorithms.

2 Problem Class

Starting point is an index-1 system of semi-explicit differential algebraic equations (DAE) describing the dynamic behavior of a controlled plant:

$$\frac{\mathrm{d}x}{\mathrm{d}t}(t) = f\big(x(t), z(t), u(t), p\big), \quad t \in \mathscr{T}, \tag{1a}$$

$$0 = g\big(x(t), z(t), u(t), p\big) \tag{1b}$$

with independent variable time t on the horizon $\mathscr{T} := [0, t_\mathrm{f}]$, differential state variables $x(\cdot) \in \mathbb{R}^{n_x}$, algebraic state variables $z(\cdot) \in \mathbb{R}^{n_z}$, control functions $u(\cdot) \in \mathbb{R}^{n_u}$ and time-invariant model parameters $p \in \mathbb{R}^{n_p}$. Later on, we will show how to use the FMI [22] to conveniently exchange models of type (1) between different modeling software tools.

We may then formulate an OCP based on plant model (1) to find locally optimal control trajectories on the time horizon $\mathscr{T}$ for a given initial process state x_0. To this end, we need to express the performance measure as an OCP objective function, i.e., a combination of a Lagrange-type term L,

$$\int_0^{t_\mathrm{f}} L(x(t), z(t), u(t), p)\,\mathrm{d}t, \tag{2}$$

and a Mayer-type term E that is defined at the end of time horizon only,

$$E(x(t_\mathrm{f}), z(t_\mathrm{f}), p). \tag{3}$$

DOI 10.3384/ecp12076781

With the resulting objective function

$$\Phi(x(\cdot),u(\cdot),z(\cdot),p) := \int_0^{t_\mathrm{f}} L(x(t),z(t),u(t),p)\,\mathrm{d}t + E(x(t_\mathrm{f}),z(t_\mathrm{f}),p), \quad (4)$$

an OCP can be formulated as follows:

$$\min_{\substack{x(\cdot),z(\cdot),\\ u(\cdot),p}} \quad \Phi(x(\cdot),z(\cdot),u(\cdot),p) \quad (5a)$$

$$\text{s.t.} \quad \frac{\mathrm{d}x}{\mathrm{d}t}(t) = f(x(t),z(t),u(t),p),\ t\in\mathscr{T}, \quad (5b)$$

$$0 = g(x(t),z(t),u(t),p),\ t\in\mathscr{T}, \quad (5c)$$

$$0 \le c(x(t),z(t),u(t),p),\ t\in\mathscr{T}, \quad (5d)$$

$$0 \leqq r_i(x(t_i),z(t_i),p), \qquad \{t_i\}_i \subset \mathscr{T}, \quad (5e)$$

$$0 = x(0) - x_0. \quad (5f)$$

We strive to identify trajectories for the controls $u(\cdot)$ and the differential and algebraic states $(x(\cdot),z(\cdot))$ that minimize the cost function Φ, and are a solution to the initial value problem defined by (5b) and (5f). Additionally, mixed state-control inequality constraints (5d) and point constraints on a grid $\{t_i\}_i \subset \mathscr{T}$ (5e) must be satisfied.

3 Direct Multiple Shooting

The OCP presented in Section 2 is an infinite-dimensional optimization problem. The purpose of the Direct Multiple Shooting method [4, 21] is to transform this problem into a finite dimensional nonlinear program (NLP) by discretization of the control functions and path constraints and by parameterization of the state trajectories. To this end, we introduce a *shooting grid* $\{\tau_i\}_{0\le i\le N}$,

$$0 = \tau_0 < \tau_1 < \ldots < \tau_N = t_\mathrm{f}. \quad (6)$$

on the horizon $\mathscr{T}$. Control trajectories are discretized on the shooting grid, e.g. as piecewise constant functions

$$u(t) := u_i, \quad t\in[\tau_i,\tau_{i+1})\subset\mathscr{T}, \quad 0\le i\le N-1. \quad (7)$$

The control space is hence reduced to functions depending on finitely many parameters u_i only.

Multiple shooting state variables s_i are introduced on the time grid to parameterize the differential state trajectories. The node values serve as initial values for an IVP solver computing the state trajectories independently on the shooting intervals $0\le i<N$,

$$\frac{\mathrm{d}x_i}{\mathrm{d}t}(t) = f(x_i(t),z_i(t),u_i,p), \quad t\in[\tau_i,\tau_{i+1}] \quad (8a)$$

$$0 = g\big(x_i(t),z_i(t),u_i,p\big), \quad (8b)$$

$$x_i(\tau_i) = s_i. \quad (8c)$$

Obviously we obtain from the above IVPs N trajectories, which in general will not combine to a single continuous trajectory. Continuity across shooting intervals needs to be ensured by additional matching conditions entering the NLP as equality constraints,

$$s_{i+1} = x_i(\tau_{i+1};\ \tau_i,s_i,z_i,u_i,p), \quad 0\le i\le N-1. \quad (9)$$

Here we denote by $x_i(\tau_{i+1};t_i,s_i,z_i,u_i,p)$ the solution of the IVP on shooting interval i, evaluated in τ_{i+1}, and depending on the initial time t_i, initial states (s_i,z_i), and on control and model parameters u_i and p. Path constraints $c(\cdot)$ are discretized on the shooting grid for simplicity of exposition. Likewise, the point constraint grid is assumed to coincide with the shooting grid.

From this discretization and parameterization, we obtain a highly structured NLP of the form

$$\min_{\xi} \quad \sum_{i=0}^{N} l_i\big(\tau_i,s_i,z_i,u_i,p\big) \quad (10a)$$

$$\text{s.t.} \quad s_{i+1} = x_i(\tau_{i+1};\ \tau_i,s_i,z_i,u_i,p) \quad 0\le i<N, \quad (10b)$$

$$0 = g\big(\tau_i,s_i,z_i,u_i,p\big), \quad 0\le i\le N, \quad (10c)$$

$$0 \le c\big(\tau_i,s_i,z_i,u_i,p\big) \quad 0\le i\le N, \quad (10d)$$

$$0 \leqq r_i\big(\tau_i,s_i,z_i,u_i,p\big) \quad 0\le i\le N, \quad (10e)$$

$$0 = s_0 - x_0, \quad (10f)$$

where all unknowns of the problem are grouped in a single vector $\xi := \big(s_0,z_0,\ldots,s_N,z_n,u_0,\ldots,u_{N-1}\big)$. For the ease of notation, we write $u_N := u_{N-1}$ in (10).

We solve this large-scale but structured NLP by a tailored sequential quadratic programming (SQP) method. This includes an extensive exploitation of the arising structures, in particular using block-wise high-rank updates of the Hessian approximation, a partial null-space reduction to eliminate the algebraic states [21], and condensing techniques for a reduction of the size of this QP to the dimension of the initial values s_0 and controls $(u_0,\ldots,u_{N-1})$ only [4, 21].

Note that the evaluation of the matching condition constraint (10b) requires the solution of an initial value problem with initial values (s_i,z_i) and controls u_i on the time horizon $[\tau_i,\tau_{i+1}]$. For more details on the numerical algorithms and techniques employed we refer the reader to e.g. the textbook [24] for nonlinear programming in general, and to [4, 21] for details on nonlinear programming techniques for Direct Multiple Shooting. An efficient implementation is available with the software package `MUSCOD-II` [9, 21] that has been used for all computations. `MUSCOD-II` for off-line optimal control is publicly available [19] on the *NEOS Server for Optimization* [15].

4 Nonlinear Model Predictive Control Scheme

We now address the issue of solving OCP (10) in an on-line NMPC setting. Key to an efficient numerical algorithm for NMPC is to reuse in every iteration information available from the last problem's solution to initialize the new problem. This is due to the fact that subsequent problems differ only in the real-world process state x_0 (5f). Moreover, the faster the control feedback can be computed and applied to the real-world process, the more similar the subsequent problems will be. If model predictions are sufficiently close to real process behavior, it is reasonable to expect that the information contained in the previous problem's solution already is a very good initial guess close to the desired solution of the new subproblem.

4.1 Initial Value Embedding

In [8, 9] and subsequent works it has been proposed to initialize the current problem with the full solution of the previous optimization run, in particular control variables u_i *and* state variables (s_i, z_i). We refer to [10] for a detailed survey on the topic of initial value embedding. It is a prominent feature of the Direct Multiple Shooting approach that very good state initializers are available not only for $x(0)$ but also for the shooting grid nodes $x(\tau_i)$, $1 \leq i \leq N$.

In using the proposed initialization, the value of s_0 will in general *not* be the value of the current state x_0. By explicitly including the linear initial value constraint (10f) we can however guarantee that s_0 attains the value of x_0 already after the first full Newton–type step computed by the SQP method.

4.2 A Real–Time Iteration in Three Phases

This idea motivates the idea of *real–time iterations* that perform only one SQP iteration per NMPC sample [9]. In this iteration, we can evaluate all derivatives and all function values without requiring knowledge of the current state x_0, the only exception being the linear initial value constraint. Consequently, we can presolve a major part of the direct multiple shooting SQP step as follows:

Preparation All functions and derivatives that do not require knowledge of x_0 are evaluated. This includes ODE solution, sensitivity computation, sparsity analysis, structure exploitation, and matrix factorizations. Note that the preparation phase of the new problem always takes place one sampling period ahead.

Feedback As soon as x_0 is available, the SQP step is readily computed from the prepared data, but only as far as required to give a feedback control to the process. Hence, the feedback delay reduces to the computation time of the SQP step *after preparation* that essentially involves the solution of only a single QP.

Transition The SQP step computation is completed *after* the feedback control has been given to the process.

5 Software Framework

In this section we present our software framework for a convenient setup of simulated and real-world NMPC loops. The basic idea is to use different specialized software for each task and to couple it to a co-simulation master. Using FMI ensures integrity of the underlying plant model that is used in several places, and avoids error-prone and time-consuming model transformations.

5.1 General Structure

A closed NMPC loop consists of three major parts as sketched in Figure 1:

Plant The controlled system. This could be a real-world plant or, in an earlier design stage, a virtual plant based on a simulation model.

Estimator The current value of all state values and parameters of the system model is estimated from available measurement data $y(t)$. The estimator could be realized as a nonlinear Kalman filter or a moving horizon estimator (MHE). If a virtual plant is used wherein all state variables and parameters are accessible, it is also possible to use an ideal estimator with $(x(t), p) = y(t)$.

Optimizer The heart of an NMPC loop is an optimization algorithm that determines the best possible control action for the current system state. This is realized as described in Sections 3 and 4.

DOI 10.3384/ecp12076781

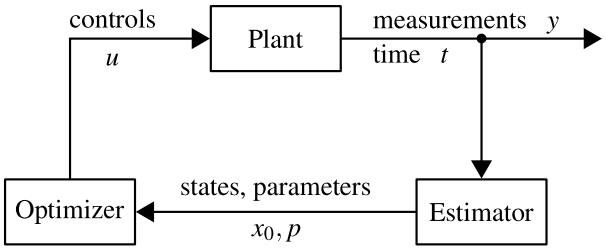


Figure 1: Signal flow diagram of closed NMPC loop.

5.2 Data Exchange

We use the co-simulation platform TISC [20] to set-up a powerful NMPC prototyping environment, keeping the basic structure of Figure 1 in mind. TISC acts as master and manages data exchange between different clients. There already exist interfaces between TISC and a variety of simulation, visualization and measurement tools, e.g. Dymola, LabView, and Simulink. The user has to define types and names of variables to be sent and received for each client. Data routing between clients is automatically managed by matching variable types and names. For our NMPC environment we use a fixed naming and typing convention. Variable names and the direction of information flow are defined according to Figure 1. The TISC type of *time* is *Double*, whereas all other variables are of TISC type *DoubleArray*.

Using this definition it is readily possible to exchange components of an NMPC loop. For example, one could replace a virtual plant that is simulated in Dymola with a real plant interfaced through LabView with just a few mouse clicks.

6 FMI for Optimization

In this section we show some implementation details to shed light on how an FMU can be used in `MUSCOD-II` to formulate and solve OCPs of type (10). We also describe new requirements and demands the FMI standard faces when we desire to use in a consistent derivative-based optimization setting such as direct optimal control, and give recommendations on future enhancements of FMI.

6.1 Interface between MUSCOD and FMI

In order to set up and solve a OPC in `MUSCOD-II` the user has to provide a C++ file that defines the model equations, including differential equations, objectives, and constraints. This source code is compiled into a shared library and dynamically loaded by the main program `MUSCOD-II` during runtime.

Instead of modeling in C++, we link a compiled FMU to a generic `MUSCOD-II` model that calls the appropriate FMI functions. This paradigm has also been followed by [19] to interface `MUSCOD-II` with AMPL [11]. As defined in FMI, some function calls have to be carried out once during startup in order to instantiate and initialize an FMU. This is organized by defining a class, writing the required function calls in its constructor, and instantiating it as a global object. Now, the constructor is called when the resulting dynamic library is loaded into `MUSCOD-II`. The corresponding code is shown in Listing 1. The pointer to the instantiated FMU is defined globally, because we need to call FMU functions in several places.

```
#define  NXD    19
#define  NU     2
#define  NP     0

fmiComponent fmu;
const fmiValueReference uRef[NU] =
    {352321536,  352321537};

class InstantiateFMU {
public:
   InstantiateFMU();
   ~InstantiateFMU();
};

InstantiateFMU::InstantiateFMU ()
{
    // Instantiate fmu
    fmu = fmiInstantiateModel (instanceName,
        GUID, callbacks, fmiFalse);
    // Set Time
    status = fmiSetTime(fmu, 0.0);
    // Set Controls
    const fmiReal uIni[NU] = {2.5, 41.6667};
    status = fmiSetReal (fmu, uRef, NU, uIni);
    // Set Parameters
    const fmiReal pInit[NP] = {};
    fmiSetReal(fmu, pRef, NP, p);
    // Initialize
    fmiEventInfo eventInfo;
    status = fmiInitialize(fmu, fmiFalse, 0.0,
        &eventInfo);
}

InstantiateFMU instantiateFMU;
```

Listing 1: Instantiation and initialization of a FMU in a MUSCOD model source file.

First of all we have to provide the differential right-hand side function of the ODE, as shown in Listing 2. This function is called by a `MUSCOD-II` integrator and is expected to return the right-hand as a function of time, states, controls, and parameters. The objective function is formulated in a similar way. As an example, the source code of a Lagrange term is shown in Listing 3.

```
void ffcn (
    double *t,   double *xd,  double *xa,
    double *u,   double *p,   double *rhs,
    double *rwh, long   *iwh, long   *info
) {
    // Set Time
    fmiSetTime (fmu, *t);
    // Set Controls
    fmiSetReal (fmu, uRef, NU, u);
    // Set Parameters
    fmiSetReal (fmu, pRef, NP, p);
    // Set States
    fmiSetContinuousStates (fmu, xd, NXD);
    // Get Derivatives
    fmiGetDerivatives (fmu, rhs, NXD);
}
```

Listing 2: Right-hand side function.

```
void lfcn (
    double *t,   double *xd,  double *xa,
    double *u,   double *p,   double *lval,
    double *rwh, long   *iwh, long   *info
) {
    // Set Time
    fmiSetTime (fmu, *t);
    // Set Inputs
    fmiSetReal (fmu, uRef, NU, u);
    // Set States
    fmiSetContinuousStates (fmu, xd, NXD);
    // Get Outputs
    const fmiValueReference yRef[2] =
          {905970080, 905971331};
    double y[2];
    fmiGetReal (fmu, yRef, 2, y);

    *lval = (y[1]-283.15) * (y[1]-283.15)
          + 0.01 * y[0] / 1000.0;
}
```

Listing 3: Lagrange term of objective.

A large part of this source code can be generated automatically from the model description xml file of an FMU, but some lines, e.g. objective formulation, currently still need to be coded by hand.

6.2 Directions for Future FMI Developments

In this section we give an outlook on future developments in using FMI for direct dynamic optimization. Ideally, we are interested in realizing FMI access to the full class of DAE-constrained switched systems,

$$\frac{\mathrm{d}x}{\mathrm{d}t}(t) = f_\sigma(t,x(t),z(t),u(t),p), \quad t \in \mathscr{T}, \tag{11a}$$

$$0 = g_\sigma(t,x(t),z(t),u(t),p), \tag{11b}$$

$$\sigma_i(t) = \begin{cases} +1 & s(t,x(t),z(t),u(t),p) > 0, \\ -1 & s(t,x(t),z(t),u(t),p) < 0. \end{cases}, \quad i = 1,\ldots,n_\sigma. \tag{11c}$$

Additional transversality conditions must be satisfied to guarantee that points $s(t,x(t),z(t),u(t),p) = 0$ are isolated and a clear transition between the two alternate right-hand sides occurs in the neighborhood of such points, see e.g. [5].

The principle of internal numerical differentiation (IND) requires a caller-control approach to be used for consistent derivative-based optimization. In such an approach, FMI is responsible for evaluation of the functions f and g, if given a *caller-supplied* switch signature σ, factorization of $\frac{\mathrm{d}g}{\mathrm{d}z}$, iteration count for solving the DAE constraint $0 = g(\cdot)$, etc. The caller is then able to keep these potentially nondifferentiable parts of the evaluation of system (11) fixed for the purpose of computing consistent derivatives and sensitivities of IVP solutions, e.g., as described in [5, 18, 23] for the case of implicit switches.

6.3 FMI Requirements for Consistent Derivatives

The current implementation of the FMI standard has proven sufficient to enable our tools to work with FMI when the problem class is limited to continuous ODEs. DAEs are currently handled internally, and are exposed as ODEs in a reduced space to the caller. This involves iterative solution of the nonlinear DAE constraint that is carried out internally by the FMI. Implicitly discontinuous ODEs, so-called switched or hybrid systems, are supported in an accessible way by the FMI standard. State discontinuities however are handled internally again. This effectively limits our approach to FMI for optimization to ODEs with continuous solutions.

To extend the FMI standard to complement state-of-the-art optimization software, the paradigm of external control over adaptive components needs to be adhered to. This currently is partially the case for switched systems, but needs to be extended to, e.g., state discontinuities, direct linear algebra involving pivoting decisions, and to the use of iterative solvers.

Whenever it is desirable to call such procedures inside an FMI model, all information about control about adaptive components, including pivoting sequences, iteration counts, matrix factors, outcome of conditional evaluations, or choice between alternate functions during implicit switching, should be conveyed to the FMI by the caller. This would grant the caller control over potential sources of non-differentiability inside the FMI. We propose that the caller sould maintain an *FMI state object* that documents the state and outcome of all non-differentiable actions, and would pass this FMI state object to the FMI, to be used for subsequent function evaluation. The caller would fur-

DOI 10.3384/ecp12076781

ther modify this FMI state object whenever appropriate, e.g. exchange functions during implicit switching, but only after the arising non-differentiability or discontinuity has been taken care of on the optimizer's side. Indeed, the FMI 2.0 standard makes considerable progress into this direction.

7 Example Application: Vapor Compression Cycle

To illustrate the applicability of the NMPC tools and algorithms described in the previous sections, we present simulation results for a challenging NMPC application. We desire to control a vapor compression cycle with two goals in mind: good disturbance rejection and maximum energy efficiency.

7.1 System Description

The system under consideration is sketched in Figure 2. Main components are two plate heat-exchangers, a variable-speed scroll compressor, an electronic expansion valve and a suction line accumulator. Working fluids are internally refrigerant R134a and on both secondary sides water-glycol mixtures. This system also exists in reality and is designed as test stand for automotive air-conditioning compressors.

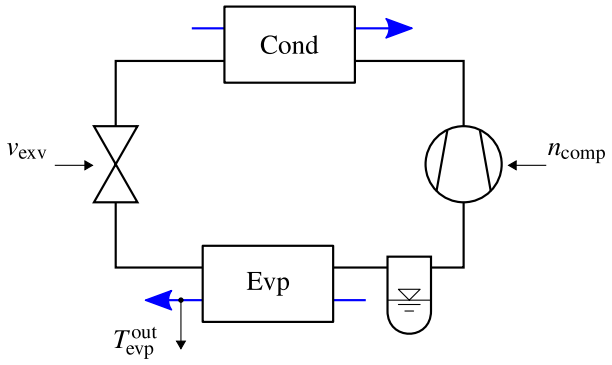


Figure 2: Vapor compression cycle including inputs and controlled outputs of the system.

7.2 System Model and Optimal Control Problem

The system model is derived from first principles only. The condenser is modeled as moving boundary model, details can be found in [14]. Accumulator and evaporator are modeled as lumped volumes.

Refrigerant fluid properties are incorporated using bicubic spline interpolation. This approach leads to improved computational speed and smoothness compared to the commonly used iterative solution of fundamental equations. Further information can be found in [13].

The resulting system model is an explicit ODE system with 17 differential states. There are 2 controls: a voltage signal v_{exv} to the step motor controller actuating the expansion valve and the rotational speed set-point of the compressor n_{comp}.

The main control goal is to keep the evaporator outlet water temperature T_{evp}^{out} at a fixed set point T_{set}. We formulate the squared deviation as first Langrange-type objective term:

$$\int_0^{t_f} (T_{evp}^{out}(t) - T_{set})^2 \, dt. \tag{12}$$

We also want to maximize energy efficiency, in other words, minimize the electrical power P_{comp} needed by the compressor, leading to the second Langrange-type objective term:

$$\int_0^{t_f} P_{comp}(t) \, dt. \tag{13}$$

We also desire to realize a smooth control profile by penalizing excessive control action, adding

$$\int_0^{t_f} (n_{comp} - \tilde{n})^2 \, dt, \tag{14}$$

$$\int_0^{t_f} (v_{exv} - \tilde{v})^2 \, dt \tag{15}$$

to our objective. Where $\tilde{n}$ and $\tilde{v}$ are two additional state variables the original ODE system is augmented by. The corresponding additional equations are

$$\frac{d\tilde{n}}{dt} = n_{comp} - \tilde{n}, \tag{16}$$

$$\frac{d\tilde{v}}{dt} = v_{exv} - \tilde{v}. \tag{17}$$

Weighting factors w_i are introduced and all terms are combined to the objective

$$\Phi := \int_0^{t_f} \left[(T_{evp}^{out}(t) - T_{set})^2 + w_0 P_{comp}(t) + w_1 (n_{comp} - \tilde{n})^2 + w_2 (v_{exv} - \tilde{v})^2 \right] dt \tag{18}$$

We finally obtain a OCP of type

$$\min_{x(\cdot),u(\cdot)} \quad \Phi(x(\cdot),u(\cdot)) \tag{19a}$$

$$\text{s.t.} \quad \frac{dx}{dt}(t) = f(x(t),u(t)) \quad t \in \mathcal{T}, \tag{19b}$$

$$0 \le c(x(t),u(t)) \quad t \in \mathcal{T}, \tag{19c}$$

$$0 = x(0) - x_0, \tag{19d}$$

with 19 differential states x and 2 controls u. In addition to the plant model ODE (19b), fixed upper and lower bounds for all states and controls (19c) as well as initial values for all states (19d) are considered.

7.3 Simulation Results – NMPC

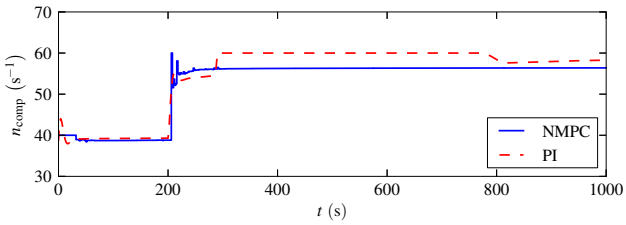
Using the methods and software tools described in previous sections we can set up a closed loop NMPC simulation. The vapor compression system Modelica model is developed, and exported as an FMU using Dymola. As described in section 6.1, the exported FMU is used in `MUSCOD-II` to formulate and solve the arising optimal control problems of type (19).

Investigation of a range of choices for the NMPC controller's parameters, comprising time horizon, number of multiple shooting intervals, and sampling rate, leads to the final choice of 500 s time horizon divided into 10 multiple shooting intervals and a 2 s sampling interval of the closed loop controller. Control trajectories are discretized on the same grid by piecewise constant functions. This setup shows good closed loop performance in terms of stability and disturbance rejection.

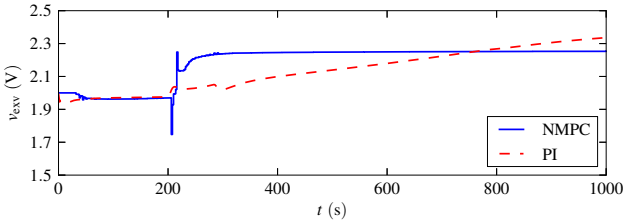
The choosen prediction horizon of 500 s appears to be very large at first sight, but shorter prediction horizons have been found to lead to stability issues. This behavior is mathematically explained by large time constants of the system. A physical explanation can be given by a closer look at the suction line accumulator. In this component, liquid refrigerant is stored in order to compensate for changes of the optimal active refrigerant charge at different working points; see [17] for a detailed discussion. The second task of a suction line accumulator is to separate vapor from liquid and feed the compressor with pure vapor. In steady-state conditions for the whole cycle, the accumulator energy balance forces inlet and outlet refrigerant states to an equilibrium. The accumulator can therefore be seen to act as a passive control unit that drives two points of the cycle (accumulator inlet and outlet) to the dew line. This passive control action takes place comparatively slowly, resulting in large time constants of the system model.

A virtual NMPC experiment is set up by simulating the controlled plant in Dymola and coupling it with `MUSCOD-II` via TISC. The real-time iteration scheme presented in Section 4.2 is applied with a fixed sampling rate of 2 s, assuming zero feedback delay.

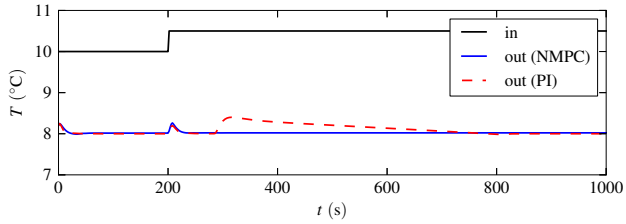
Additional assumptions are no model-plant mismatch, availability of the full process state vector, and uncontrolled input measurements without disturbance.



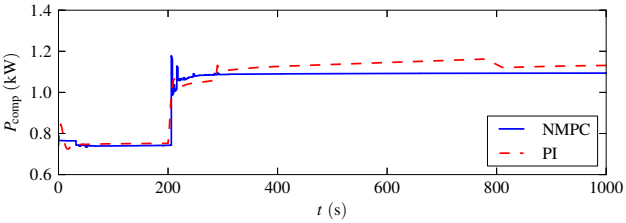
(a) Control input 1: compressor speed.



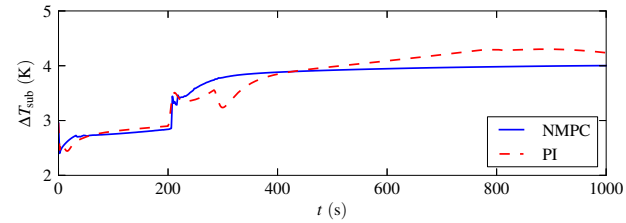
(b) Control input 2: expansion valve voltage signal.



(c) Chilled water temperatures at evaporator inlet and outlet.



(d) Compressor's electrical power consumption.



(e) Refrigerant subcooling at condenser outlet.

Figure 3: Simulation results: NMPC versus PI control of a vapor compression cycle.

Although these assumptions can hardly be satisfied when NMPC is applied to a real plant, this kind of ideal experiment still helps to gain insight into the theoretical performance of an optimally designed NMPC controller. Using our software framework, closed loop performance of extended problems can be studied very conveniently.

DOI 10.3384/ecp12076781

7.4 Simulation Results – Comparison to PI Control

For comparison, we applied a conventional control concept with two continuous PI controllers to the plant. Our primary goal – keeping chilled water outlet temperature at a constant set-point of 8 °C – is achieved by adjusting compressor speed. Contrary to NMPC, we can't take our second goal – maximizing energy efficiency – directly into account. It known, however, that for vapor compression cycles of our type, a certain value of refrigerant subcooling at the condenser outlet is optimal [17]. Hence, we may use the second control input – expansion valve opening – to keep subcooling close to a fixed set-point of 3 K.

In our example experiment we start with near steady-state conditions. At $t = 200$ s the chilled water inlet temperature rises from 10 to 10.5 °C. With chilled water outlet at 8 °C, this results in a cooling load increase of 25%. Figure 3 shows the corresponding response of PI and NMPC closed loops.

In the first 200 seconds there is only little control action. Both control goals, chilled water outlet temperature (Figure 3(c)) and compressor's power consumption (Figure 3(d)), are almost identical for both control concepts. This is because the chosen subcooling setpoint for the PI controller is set to 3 K, which is close to the efficiency optimal working point for these boundary conditions. At $t = 200$ s, when the chilled water inlet temperature rises, things change noticeably. First of all, there is an immediately deviation of the chilled water outlet temperature from its setpoint. Both controllers react by increasing the compressor speed (Figure 3(a)) and drive the temperature back to their setpoints (Figure 3(c)). Looking at Figure 3(b), we see that both controllers react to the disturbance by opening the expansion valve. The NMPC controller however does so much more aggressively, leading to the desired result that water outlet temperature stays at its setpoint for the remaining simulation time. The PI controlled temperature shows a second deviation starting at $t = 300$ s. Because the maximum compressor speed of 60 s^{-1} has already been reached, the temperature deviation lasts until $t = 800$ s.

One could argue that the situation could be improved by tuning the expansion valve PI controller to speed up its reaction. Although we don't claim to have chosen the best possible PI parameters, simulation studies show that the expansion valve PI controller must be comparatively slow in order to ensure stability of the closed loop. This may be due to the large time constants mentioned above. A second reason may be the inverse response behavior of the plant model for expansion valve opening as input and subcooling as output. Besides good disturbance rejection, a second benefit of our NMPC controller becomes clear by looking at the compressor power consumption in Figure 3(d). At $t = 1000$ s the system slowly approaches a new steady state working point with about 4% increased power consumption of the PI controlled compared to the NMPC controlled cycle. Therefore, one can see that a fixed subcooling setpoint is not optimal for all boundary conditions. Figure 3(e) shows that for the new working point, optimal subcooling tracked by the NMPC controller lies somewhere around 4 K. If we continued simulation, the PI controller would steer the cycle back to non-optimal subcooling of 3 K.

8 Conclusion

Although tailored to forward simulation, the FMI format can be used for interfacing Modelica models with state-of-the-art dynamic optimization software. But with the current design of FMI this approach is limited to continuous ODE. To extend the scope of FMI for optimization to hybrid DAE there must be major changes. Instead of solving implicit algebraic equations with embedded solvers internally, the residuum functions should be exposed. The proposed software framework has proven its applicability for setting up NMPC loops in an early design stage. The application vapor compression cycle demonstrates the benefits of NMPC. In the presented scenario, NMPC shows a significantly better performance compared to a conventional PI control concept in terms of energy efficiency and disturbance rejection. Moreover, NMPC is able to identify and track new optimal working points under changed external conditions.

References

[1] Johan Åkesson. *Languages and Tools for Optimization of Large-Scale Systems*. Phd thesis, Lund University, 2007.

[2] Johan Åkesson, Karl-Erik Årzén, Magnus Gäfvert, Tove Bergdahl, and Hubertus Tummescheit. Modeling and optimization with Optimica and JModelica.org–Languages and tools for solving large-scale dynamic optimization problems. *Computers & Chemical Engineering*, 34(11):1737–1749, November 2010.

[3] T. Blochwitz, M. Otter, M. Arnold, C. Bausch, C. Clauß, H. Elmqvist, A. Junghanns, J. Mauss, M. Monteiro, T. Neidhold, D. Neumerkel, H. Olsson, J.-V. Peetz, and S. Wolf. The Functional Mockup Interface for Tool independent Exchange of Simulation Models. In *8th International Modelica Conference*, Dresden, 2011.

[4] H. G. Bock and K. J. Plitt. A Multiple Shooting algorithm for direct solution of optimal control problems. In *Proceedings 9th IFAC World Congress Budapest*, pages 243–247. Pergamon Press, 1984.

[5] H.G. Bock. Numerical treatment of inverse problems in chemical reaction kinetics. In K.H. Ebert, P. Deuflhard, and W. Jäger, editors, *Modelling of Chemical Reaction Systems*, volume 18 of *Springer Series in Chemical Physics*, pages 102–125. Springer, Heidelberg, 1981.

[6] Jonathan Brembeck, Martin Otter, and Dirk Zimmer. Nonlinear Observers based on the Functional Mockup Interface with Applications to Electric Vehicles. In *8th International Modelica Conference*, Dresden, 2011.

[7] Francesco Casella, Filippo Donida, and Marco Lovera. Beyond Simulation: Computer-Aided Control System Design using Equation-based Object-oriented Modelling for the Next Decade. *Simulation News Europe*, 19(1):29–41, 2009.

[8] M. Diehl, H. G. Bock, J. P. Schlöder, R. Findeisen, Z. Nagy, and F. Allgöwer. Real-time optimization and Nonlinear Model Predictive Control of Processes governed by differential-algebraic equations. *Journal of Process Control*, 12(4):577–585, 2002.

[9] Moritz Diehl. *Real-Time Optimization for Large Scale Nonlinear Processes*. Phd thesis, Universität Heidelberg, 2001.

[10] Moritz Diehl, Hans Joachim Ferreau, and Niels Haverbeke. Efficient Numerical Methods for Nonlinear MPC and Moving Horizon Estimation. In Lalo Magni, Davide Martino Raimondo, and Frank Allgöwer, editors, *Nonlinear Model Predictive Control*, Lecture Notes in Control and Information Sciences, pages 391–417. Springer, Berlin, Heidelberg, New York, 2009.

[11] R. Fourer, D. M. Gay, and B. W. Kernighan. *AMPL: A Modelling Language for Mathematical Programming*. Books/Cole—Thomson Learning, 2nd edition, 2003.

[12] Rüdiger Franke. Formulation of dynamic optimization problems using Modelica and their efficient solution. In *2nd International Modelica Conference*, pages 315–323, Oberpfaffenhofen, 2002.

[13] Manuel Gräber, Christian Kirches, Johannes P. Schlöder, and Wilhelm Tegethoff. Nonlinear Model Predictive Control of a Vapor Compression Cycle based on First Principle Models. In *MATHMOD, 7th Vienna International Conference on Mathematical Modelling*, 2012.

[14] Manuel Gräber, Nils Christian Strupp, and Wilhelm Tegethoff. Moving boundary heat exchanger model and validation procedure. In *EUROSIM Congress on Modelling and Simulation*, Prague, 2010.

[15] William Gropp and Jorge J. Moré. Optimization environments and the NEOS Server. In M. D. Buhmann and A. Iserles, editors, *Approximation Theory and Optimization*, pages 167–182. Cambridge University Press, 1997.

[16] L. Imsland, P. Kittilsen, and T. S. Schei. Model-based optimizing control and estimation using Modelica models. *Modeling, Identification and Control*, 31(3):107–121, 2010.

[17] Jørgen Bauck Jensen and Sigurd Skogestad. Optimal operation of simple refrigeration cycles Part I: Degrees of freedom and optimality of sub-cooling. *Computers & Chemical Engineering*, 31(5-6):712–721, May 2007.

[18] C. Kirches. A numerical method for nonlinear robust optimal control with implicit discontinuities and an application to powertrain oscillations. Diploma thesis, Ruprecht–Karls–Universität Heidelberg, October 2006.

[19] Christian Kirches and Sven Leyffer. TACO – A Toolkit for AMPL Control Optimization. Preprint ANL/MCS-P1948-0911, Mathematics and Computer Science Division, Argonne National Laboratory, October 2011.

[20] Roland Kossel, Martin Löffler, Nils Christian Strupp, and Wilhelm Tegethoff. Distributed energy system simulation of a vehicle. In *Vehicle Thermal Management Systems Conference*. Institution of Mechanical Engineers, SAE International, 2011.

[21] D B Leineweber, I Bauer, A A S Schäfer, H G Bock, and J P Schlöder. An Efficient Multiple Shooting Based Reduced SQP Strategy for Large-Scale Dynamic Process Optimization (Parts I and II). *Computers and Chemical Engineering*, 27:157–174, 2003.

[22] MODELISAR. *Function Mock-up Interface for Model Exchange*, 2010. Specification, Version 1.0.

[23] K.D. Mombaur. *Stability Optimization of Open-loop Controlled Walking Robots*. Phd thesis, Universität Heidelberg, 2001.

[24] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. Springer, Berlin, Heidelberg, New York, 2nd edition, 2006.

[25] Andreas Pfeiffer. *Numerische Sensitivitätsanalyse unstetiger multidisziplinärer Modelle mit Anwendungen in der gradientenbasierten Optimierung*. Phd thesis, Martin-Luther-Universität Halle-Wittenberg, 2008.

[26] E. D. Tate, Michael Sasena, Jesse Gohl, and Michael Tiller. Model Embedded Control: A Method to Rapidly Synthesize Controllers in a Modeling Environment. In *6th International Modelica Conference*, pages 493–502, Bielefeld, 2008.

 DOI 10.3384/ecp12076781