Control Engineering Practice 60 (2017) 51–62

Contents lists available at ScienceDirect

Control Engineering Practice

journal homepage: www.elsevier.com/locate/conengprac

Rapid development of modular and sustainable nonlinear model predictive control solutions

Sergio Lucia[a,b,*], Alexandru Tătulea-Codrean[b], Christian Schoppmeyer[b], Sebastian Engell[b]

[a] Institute for Automation Engineering, Otto-von-Guericke University Magdeburg, Universitätsplatz 2 (Building 07), 39106 Magdeburg, Germany
[b] Process Dynamics and Operations Group, Technische Universität Dortmund, Emil-Figge-Str 70, 44227 Dortmund, Germany

ARTICLE INFO

Keywords:
Nonlinear model predictive control
Robust control
Optimization
Process control
Development support
Numerical methods

ABSTRACT

While computational complexity is often not anymore an obstacle for the application of Nonlinear Model Predictive Control (NMPC), there are still important challenges that prevent NMPC from already being an industrial reality. This paper deals with a critical challenge: the lack of tools that facilitate the sustainable development of robust NMPC solutions. This paper proposes a modularization of the NMPC implementations that facilitates the comparison of different solutions and the transition from simulation to online application. The proposed platform supports the multi-stage robust NMPC approach to deal with uncertainty. Its benefits are demonstrated by experimental results for a laboratory plant.

1. Introduction

Model Predictive Control (MPC) is a popular control strategy that has been successfully applied especially in the process industries as reported e.g. in Qin and Badgwell (2003). The most important reason for this success is its ability to handle coupled multivariable systems with constraints. Its nonlinear variant, Nonlinear Model Predictive Control (NMPC), has been studied intensively by the research community and many simulation results have been published in the last years, including large-scale and highly nonlinear systems based on rigorous models (e.g. Huang, Zavala, & Biegler, 2009; Idris & Engell, 2012 or Toumi & Engell, 2004). Although some companies develop industrial NMPC implementations for some processes (see Cybernetica, 2014; IPCOS, 2014; Pluymers, Ludlage, Ariaans, & Van Brempt, 2008), its practical use is still in the early stages.

Some years ago, one of the main reasons that prevented NMPC from being applied in practice was the computation time needed to solve the resulting large-scale nonlinear programming problems. The progress made in the last years on optimization algorithms and on computational power has made it possible to significantly reduce the computation times needed to solve NMPC problems even to the microsecond range as shown in Houska, Ferreau, and Diehl (2011b). With computational power no longer being a problem in many cases, at least three major challenges remain as main obstacles for the application of NMPC to real processes: The high cost of the development of models, the lack of tools that provide a rapid and sustainable implementation of NMPC and the presence of significant uncertainty in the available models. The focus here lies on the two last challenges. Dealing with model errors in a systematic but not overly conservative manner also reduces the effort that is needed for the development of high-fidelity models.

Regarding the implementation of NMPC, many software tools have recently been developed in academia such as MUSCOD II (Diehl, Leineweber, & Schäfer, 2001), ACADO (Houska, Ferreau, & Diehl, 2011a), NMPC tools (Amrit & Rawlings, 2008), OptCon (Nagy, 2008) or the MPT Toolbox (Herceg, Kvasnica, Jones, & Morari, 2013) among others. These tools can solve different kinds of problems including NMPC formulations. It is very common however that different applications require different software tools, depending on their complexity. Currently most of these tools require an implementation of a model in a particular syntax and an interface to other necessary components such as a simulator or an observer. If the model is changed, the entire implementation has to be modified. If one wants to test a new tool, most of the code has to be rewritten. This lack of modularity results in complex and non-sustainable implementations, making it also difficult to compare the computational performance and solution qualities of different algorithms or to combine parts of different tools.

This paper extends the results presented in Lucia, Tatulea-Codrean, Schoppmeyer, and Engell (2014) by describing in more detail a new concept for the modularization of a NMPC implementation, dividing it into four main components: model and problem description, optimizer, observer and simulator. Such modular design makes it possible to

* Corresponding author at: Institute for Automation Engineering, Otto-von-Guericke University Magdeburg, Universitätsplatz 2 (Building 07), 39106 Magdeburg, Germany.
E-mail addresses: sergio.lucia@ovgu.de (S. Lucia), alexandru.tatulea-codrean@bci.tu-dortmund.de (A. Tătulea-Codrean), christian.schoppmeyer@bci.tu-dortmund.de (C. Schoppmeyer), s.engell@bci.tu-dortmund.de (S. Engell).

http://dx.doi.org/10.1016/j.conengprac.2016.12.009
Received 25 April 2016; Received in revised form 18 December 2016; Accepted 21 December 2016

compare different solutions (e.g. different discretizations or estimators) just by exchanging the corresponding module, but maintaining the rest of the solution. The effort of going from simulation to an online application is also reduced to the exchange of a simulator module by an application module. The platform supports the multi-stage NMPC (Lucia, Finkler, & Engell, 2013) approach to deal with uncertainties in a systematic way. Both contributions have been combined in the environment do-mpc to provide a framework for the rapid and sustainable development of standard and multi-stage NMPC solutions, which can be easily transferred to online applications, as illustrated by experimental results obtained for a laboratory reactor.

do-mpc provides a unique environment that can be useful to all kinds of users due to the flexibility of the implementation. The use of the CasADi (Andersson, Åkesson, & Diehl, 2012) tool set makes it possible to modify the NMPC formulation provided within do-mpc in a simple manner to include specific elements necessary for alternative formulations of the resulting optimal control problems (e.g., conditions for stability or robustness) rather than providing a black-box NMPC tool. Practitioners can benefit from the existence of templates to develop state-of-the-art implementations of both multi-stage and standard NMPC with low effort. The modularized implementation leads to sustainable NMPC solutions that can be reused when models are updated, or that can be easily transferred to the real system once the desired performance is achieved in simulations, as shown by the experimental results presented in this paper.

The remainder of this paper is structured as follows. Section 2 explains the concepts and the main components of the modular implementation of NMPC, as well as the do-mpc environment. The experimental setup is described in Section 3. The simulation and experimental results, together with a short review of multi-stage NMPC, are presented in Section 4. The paper is concluded in Section 5.

2. do-mpc: an environment for the rapid development of modular and sustainable NMPC solutions

One of the main outcomes of the European research project EMBOCON (Embedded Optimization for Resource Constrained Platforms) was the development of the open source software platform GEMS (Generic EMBOCON Minimal Supervisor) (Schoppmeyer, 2013).

The central idea of GEMS is to offer a set of general and standardized interfaces to simplify the process of developing and deploying a model-based control algorithm for a real system, and to offer a so-called supervisor that manages the flow of information between the different parts of the implementation. The implementation of a model-based control approach in GEMS is divided into four main components: the model, the optimizer, the observer and the simulation or real application (see inner blocks in Fig. 1). The exchange of information between the different modules is managed and logged by a supervisor. Using this conceptual idea, existing or newly developed algorithms for control, for simulation or for state estimation of a system can be implemented based on the GEMS interfaces. The interfaces are structured so as to encapsulate all the information needed by GEMS at run-time and they are implemented in plain C-code, which offers the possibility to integrate other tools and to expand the functionality with basic programming techniques.

This modularization idea has been extended and implemented in the environment do-mpc. do-mpc is a platform that uses the main ideas of GEMS to provide users with an easy, modular and robust way to realize sustainable implementations of NMPC, with a special focus on multi-stage NMPC. Within do-mpc, it is proposed to structure the four different modules in the following manner.

The *model and problem description* module contains the right-hand side of the ordinary differential equations or differential algebraic equations, including all model parameters and the definition of model states and inputs. Here also the control and estimation tasks (initial

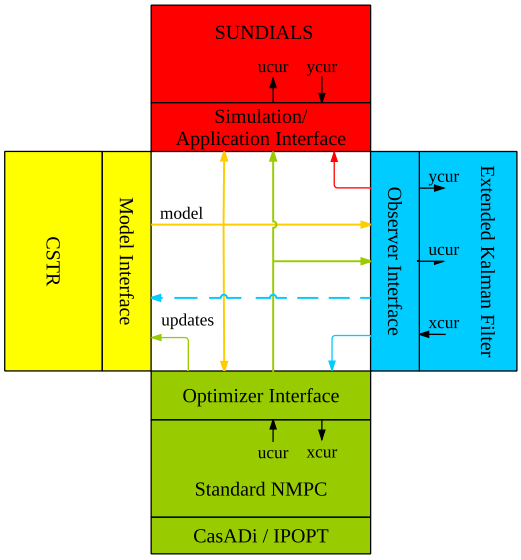

Fig. 1. Modular scheme of an NMPC implementation with four main blocks: Model, optimizer, observer and simulator. The arrows communicating different modules indicate the information exchange and possible updates performed between modules. A combination of the four modules forms a do-mpc configuration.

condition, objective, constraints) should be defined. do-mpc provides templates using the scripting language Python as a user-friendly environment for the representation of the information in the model and problem description module. It is not necessary to use this template, i.e., it is possible to define this information using any existing code or software as long as it passes the necessary information to the model interface so that the other modules can make use of it. It is also possible to define models of different complexity to be used with each one of the other modules. For example, one can use a simple model for the optimization, but a more detailed one for the simulation of the system.

The *optimizer* module contains the implementation of the solution method and of the optimizer for the NMPC problem. For example in the case of a simultaneous NMPC approach it contains the algorithm for the discretization of the dynamics so that a Nonlinear Programming Problem (NLP) is generated (using the information provided in the model and problem description module) and then passed to any available solver (as e.g. IPOPT Wächter & Biegler, 2006) or a self-implemented one. As a major contribution of do-mpc, a Python template that generates the NLP resulting from the robust multi-stage NMPC formulation (Lucia et al., 2013) is provided, or – if chosen by the user – the NLP resulting from standard NMPC formulations. This module also contains the controller parameters (sampling time, prediction horizon, etc.).

The *observer* module contains an algorithm in which, given the measurements of the plant or simulation, the states of the system are calculated to initialize the optimizer. Standard observers such as the Extended Kalman Filter or the Moving Horizon Estimator can be implemented in a generalized manner so that the information of the other modules is used and there is no need to re-implement the observers for different models or optimizers.

The *simulation* module contains an integrator which can be self-coded or interfaced with existing software such as the SUNDIALS (Hindmarsh et al., 2005) toolbox. For a real test case, this module contains an interface to the Input–Output device that is used for the communication with the real plant. The implementation of the module is application specific and is able to send the calculated control inputs and receive the measurements from the real system.

do-mpc is built upon CasADi (Andersson et al., 2012) as a building block for the necessary modules and interfaces. CasADi is a tool for