Application of the OpenModelica-MATLAB Interface to Integrated Simulation and Successive Linearization Based Model Predictive Control

Mohammad Hadi Alizadeh[1] Ali M. Sahlodin[1] Arunkumar Palanisamy[2] Francesco Casella[3] Peter Fritzson[2]

[1]Process Systems Engineering Laboratory, Department of Chemical Engineering, Amirkabir University of Technology (Tehran Polytechnic), Iran, {m.hadi,sahlodin}@aut.ac.ir
[2]Department of Computer and Information Science (IDA), Linköping University, Sweden, {arunkumar.palanisamy,peter.fritzson}@liu.se
[3]Dipartimento di Elettronica, Informazione e Bioingegneria, Politecnico di Milano, Italy, francesco.casella@polimi.it

Abstract

This paper presents the implementation of successive linearization based model predictive control (SLMPC) efforts through the interfacing of OpenModelica and MATLAB using the OMMatlab tool. The dynamic system (here a chemical process) and the model predictive control (MPC) algorithm are implemented in OpenModelica and MATLAB, respectively. The model linearization procedure is carried out through OMMatlab, which is highly optimized in terms of run-time by using a single executable file and adapting it at each sample time. Also, necessary theories for a continuous model discretization are discussed for both nonlinear Modelica and linearized continuous models. A procedure for constructing an Extended Kalman Filter (EKF) from a continuous Modelica model is also presented. The usability of the OpenModelica-MATLAB interface for SLMPC is demonstrated by control of liquid levels in a tanks-in-series problem.

Keywords: Model predictive control, OpenModelica, OMMatlab, Extended Kalman filter.

1 Introduction

In the recent years, the Modedica language has been widely used for modeling of systems described by differential algebraic equations (DAEs). The object-oriented nature of Modedica facilitates modular model construction and the use of powerful solvers as provided by commercial and open source Modedica-based simulators (e.g., Dymola, OpenModelica (Fritzson et al. 2020), Jmodelica). Some integrated features such as Optimica (Åkesson 2008) and CasAdi are also supplied for solving dynamic optimization problems. The reader is referred to (Ruge et al. 2014; Magnusson and Åkesson 2015) for a thorough discussion of these topics. Moreover, model predictive control (MPC) problems have been implemented in OpenModelica; see e.g., (Bachmann et al. 2012) for MPC implementation of a batch reactor.

Conventional nonlinear model predictive control (NMPC) works by predicting the future behavior of a system and solving a dynamic optimization problem for minimizing a performance index, e.g., tracking error or operational cost; see, for example, (Ellis, J. Liu, and Christofides 2017; Heidarinejad, J. Liu, and Christofides 2013). Although using a rigorous nonlinear model increases the prediction accuracy, it leads to high computational expense (Zhakatayev et al. 2017) and potential convergence issues in the optimization routine. Successive linearization MPC (SLMPC) is an efficient alternative to NMPC (Seki, Ooyama, and Ogawa 2002; Cannon, Ng, and Kouvaritakis 2009; Cortinovis et al. 2014; C. Liu et al. 2015), especially in large-scale applications. In this method, the nonlinear model of the system is linearized successively at each sample time, and the prediction model is updated over time to preserve the prediction accuracy (Vrlić, Ritzberger, and Jakubek 2020). As a result of using a linear model, the dynamic optimization can be performed faster, favoring SLMPC over NMPC for large-scale systems in terms of computational expense, although the prediction accuracy may be undermined to some extent.

Many dynamic optimization or MPC efforts using Modelica models are reported in the literature. Franke (2002) employed Modelica to study optimal startup of a power plant, in which Dymola was used for generating S-Functions that would be imported into Simulink. Gräber et al. (2012) proposed a framework based on functional mockup interface generated from Modelica models for implementing NMPC on a vapor compression cycle. Also, L. Imsland, P. Kittilsen, and T. Schei (2010) integrated Dymola models into commercial software designed for NMPC and did a case study with an offshore oil and gas processing plant. Pandey et al. (2021a) employed OMJulia (Lie et al. 2019) for stochastic MPC of solar and hydropower plants described by a set of small-scale DAEs. Pandey et al. (2021b) implemented MPC for a power grid

system, where a small-scale model was implemented in OpenModelica as the actual plant, and a separate model was developed in Julia for use in the control algorithm that included an unscented Kalman filter for the state estimation. OMJulia was used for interfacing OpenModelica with Julia. Also, when it comes to optimal control problems that require a linearized model, Jmodelica can offer a tool for linearizing the nonlinear model of the plant. Jmodelica supplies the interfacing capability with Python, enabling it to integrate Jmodelica simulation, optimization, and linearization features and design MPC problems that need linearization. It is possible to interact with the Modelica models and functional mockup units through IPython, a command shell for the programming environment (Andersson et al. 2018). The interested readers are referred to, e.g., ((Romero, Goldar, and Garone 2019; Pippia et al. 2021; Lars Imsland, Pål Kittilsen, and T. S. Schei 2009; Jorissen, Boydens, and Helsen 2019)) for other dynamic optimization studies involving Modelica models.

The use of engineering software such as MATLAB for Modelica models offers easy implementation for algorithm prototyping. However, a challenge in the application of MPC for Modelica models is the computational cost of interfacing the Modelica software with engineering software such as MATLAB. This is particularly true in case of SLMPC, where repeated model linearization through Jacobian calculations is required. The computational cost can grow quickly for large-scale models, making it prohibitively expensive to apply SLMPC or similar algorithms through interfacing of Modelica models with MATLAB or other engineering software. Therefore, the procedure needs to be carried out decently ensuring that the Jacobian calculations are undertaken in a time-efficient manner.

In this paper, efficient implementation of SLMPC for Modelica models in OpenModelica is addressed. In particular, the dynamic model of the system (serving as the virtual plant) is implemented in OpenModelica. Also, OMMatlab (*OpenModelica* 2021) is used to take control of the model simulation and make an interconnection between MATLAB and the OpenModelica model, which is transformed into an executable file. Moreover, the OMMatlab linearization method is enhanced to operate considerably faster than its older version, making the new version more favorable for large-scale SLMPC.

These enhancements can also be implemented for other OpenModelica interfaces such as OMOctave, OMPython, and OMJulia so they can support efficient prototyping of advanced control algorithms requiring successive linearization.

The rest of the paper is structured as follows. In section 2, the general structure of the SLMPC problem is briefly introduced. In section 3, the method of assembling the discrete system model from its continuous form is presented. Section 4 discusses the details of the linear prediction model, the pertinent theories, and the enhancements made to OMMatlab to boost the linearization process. The state estimator design algorithm is discussed in section 5. In section 6, the dynamic optimization problem solved at each sample time is formulated. A case study of the SLMPC implementation is demonstrated in section 7. Finally, the paper is concluded in section 8.

2 Overview of Problem Blocks

As depicted in Figure 1, the problem under study includes an often nonlinear model that represents the actual physical system (here a chemical plant). Some of the plant variables are measured at every sample time. Since measuring all the variables is impractical, a state estimator is used to estimate the unmeasured state variables. These values are used to initialize the prediction model, which is a linearized form of the nonlinear model updated at each sample time. It is also possible to increase the frequency of prediction model updates by regenerating the linear model at each step over the prediction horizon instead of updating it only at the beginning of each sample time. The optimal control inputs are calculated over the prediction horizon by minimizing the objective function (e.g., the cumulative tracking error). Then, the first element of the optimal input trajectory is passed to the plant, and the procedure is repeated in the next sample time. The building blocks of the SLMPC are detailed in the following subsections.

3 System Model

The plant model is represented in a continuous, nonlinear, time-invariant state-space form as

$$\dot{x} = f(x,u) \tag{1}$$
$$y = g(x,u) \tag{2}$$

where $x \in \mathbb{R}^{n_x}$ is the set of n_x system states, $u \in \mathbb{R}^{n_u}$ is the vector of n_u control inputs, $y \in \mathbb{R}^{n_y}$ is the vector of n_y system outputs, $f = [f_1, f_2, f_3, ..., f_{n_x}]$ and $g = [g_1, g_2, g_3, ..., g_{n_y}]$ are the sets of equations describing the state evolution and outputs of the system, respectively. OpenModelica automatically generates these functions from a high-level, object-oriented, equation-based model description, and can simulate the nonlinear model and generate an executable file that will be used for the state estimator and prediction model. The existing features of OMMatlab allow the user to take control of the executable file and overwrite the simulation setup and model parameters from MATLAB.

Equation 1 is converted to a discrete state-space form with additive white Gaussian noise for process states and measurements. This is done by the methodology presented in (Brembeck 2019; Brembeck, Otter, and Zimmer 2011), the implementation of which is adapted to the MATLAB-OpenModelica environment. The discrete form reads

DOI 10.3384/ecp204293

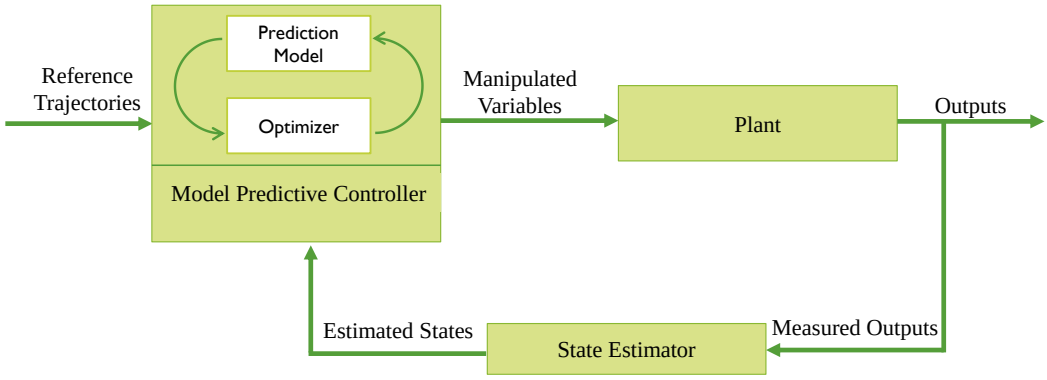

Figure 1. A general schematic of the MPC blocks.

$$x_k = f_{k|k-1} + w_{k-1} \tag{3}$$

$$f_{k|k-1} = x_{k-1} + \int_{t_{k-1}}^{t_k} f(x,u)\,dt \tag{4}$$

$$y_k = g(x_k, u_k) + v_k \tag{5}$$

$$E[w_k] = 0 \tag{6}$$

$$E[v_k] = 0 \tag{7}$$

$$E[w_k w_k^T] = Q_k \tag{8}$$

$$E[v_k v_k^T] = R_k, \tag{9}$$

where w_k and v_k are additive process and measurement noises, respectively; and Q_k and R_k denote the process and measurement noise covariance matrices, respectively.

The LHS of Equation 4 is computed by integration of the system over one sample time. In the continuous model implemented in OpenModelica, initial state values (defined in the `initial equation` section) are set parametrically so that the initial values can be overwritten and Equation 4 can be evaluated from MATLAB at each sample time. This is illustrated in Listing 1 for an arbitrary continuous Modelica model.

Listing 1. Continuous system model in Modelica

```
model List1
...
parameter Real x1_In;
parameter Real x2_In;
...
input Real u1;
input Real u2;
...
output Real y1;
output Real y2;
...
Real x1;
Real x2;
...
initial equation
x1=x1_In;
x2=x2_In;
...
equation
...
end List1;
```

The initial conditions x_{k-1} could be set from MATLAB by the `setParameters()` method of OMMatlab. Similarly, the system inputs u_{k-1} are specified by the `setInputs()` method. Finally, the integration in Equation 4 is performed over one sample time using built-in OpenModelica solvers such as DASSL and IDA.

4 Prediction Model

The system linearization required for the prediction model and its implementation are described in this section.

4.1 System linearization

Equation 1 and Equation 2 can be linearized around an arbitrary operating point (x_{op}, u_{op}) by using the first-order Taylor expansion as follows.

$$\dot{x} = f(x_{op}, u_{op}) + A_c(x - x_{op}) + B_c(u - u_{op}) \tag{10}$$

$$y = g(x_{op}, u_{op}) + C_c(x - x_{op}) + D_c(u - u_{op}), \tag{11}$$

where

$$A_c = \frac{\partial f}{\partial x} = \begin{bmatrix} \frac{\partial f_1(\mathbf{x})}{\partial x_1} & \cdots & \frac{\partial f_1(\mathbf{x})}{\partial x_{n_x}} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_{n_x}(\mathbf{x})}{\partial x_1} & \cdots & \frac{\partial f_{n_x}(\mathbf{x})}{\partial x_{n_x}} \end{bmatrix} \tag{12}$$

Similarly, $B_c = \frac{\partial f}{\partial u} =$, $C_c = \frac{\partial g}{\partial x}$, and $D_c = \frac{\partial g}{\partial u}$.
Considering Equation 1 and Equation 2, the bias values are

$$f(x_{op}, u_{op}) = \left.\frac{dx}{dt}\right|_{(x,u)=(x_{op},u_{op})} \tag{13}$$

$$g(x_{op}, u_{op}) = y|_{(x,u)=(x_{op},u_{op})} \tag{14}$$

Therefore, it is possible to construct continuous linearized models by having A_c, B_c, C_c, D_c, and the bias values. OMMatlab can provide all these matrices along with ($\dot{x}$, y) at any specific simulation time by the `linearize()`

and `getSolutions()` methods. Equation 10 and Equation 11 can then be expressed as

$$\dot{x} = A_c x + B_c u + \mathbb{K}_{xc} \tag{15}$$
$$y = C_c x + D_c u + \mathbb{K}_{yc}, \tag{16}$$

where $\mathbb{K}_{xc} = (f(x_{op}, u_{op}) - A_c x_{op} - B_c u_{op})$, and $\mathbb{K}_{yc} = (g(x_{op}, u_{op}) - C_c x_{op} - D_c u_{op})$. The continuous linearized model can be discretized for a given sample time T_s as (see (Zhakatayev et al. 2017; Vrlić, Ritzberger, and Jakubek 2020) for a detailed proof).

$$x_{k+1} = A_d x_k + B_d u_k + \mathbb{K}_{xd} \tag{17}$$
$$y_k = C_d x_k + D_d u_k + \mathbb{K}_{yd}, \tag{18}$$

in which

$$A_d = e^{A_c T_s} \tag{19}$$
$$B_d = A_c^{-1}(e^{A_c T_s} - I)B_c \tag{20}$$
$$\mathbb{K}_{xd} = A_c^{-1}(e^{A_c T_s} - I)\mathbb{K}_{xc} \tag{21}$$

and $C_d = C_c$, $D_d = D_c$, and $\mathbb{K}_{yd} = \mathbb{K}_{yc}$.

4.2 Improving OMMatlab for successive linearization

Repeated linearization of a nonlinear dynamic model was computationally inefficient in the previous OMMatlab distributions. For example, a single invocation of the `linearize()` method for a system of DAEs with about 1100 equations took around 420 seconds on an Intel Core i5 7200U CPU. This would make implementation of SLMPC impractical as the linearization task should be performed at every sample time. The leading cause for this inefficiency was that the OpenModelica model had to be recompiled each time the `linearize()` command was called. The resulting linearized model had to be rebuilt to create an XML file so that the values of matrices could be extracted into MATLAB.

To solve this problem, OMMatlab is edited by the authors so that the initial executable file can be adapted and used in all invocations without being recompiled. Moreover, instead of using the time-consuming perturbation and finite differences for Jacobian approximation, the built-in automatic differentiation algorithm (invoked by the `-generateSymbolicLinearization` flag) is employed for exact linearization and construction of the model matrices (Braun, Ochel, and Bachmann 2011). The generated linearized model is populated in a `.m` file, from which the matrix values are read. With these improvements, a single invocation of the `linearize()` command for the same DAE system now takes only a fraction of a second, which is a remarkable computational enhancement. This improvement can benefit any application requiring nonlinear Modelica model linearization through MATLAB, including SLMPC.

5 State Estimator

The extended Kalman filter (EKF) is used for state estimation. An EKF for Modelica models can be implemented in MATLAB based on the approach presented in (Brembeck 2019). The EKF uses linearization for state and measurement covariance propagation. The prediction and correction steps of the EKF for Equation 3 to Equation 5 proceed as follows.

- Prediction:

$$\hat{x}_k^- = f_{k|k-1}(x_{k-1}^+, u_{k-1}) \tag{22}$$
$$F_{k-1} = exp(\frac{\partial f}{\partial x}|_{x_{k-1}^+}.T_s) \tag{23}$$
$$P_k^- = F_{k-1}P_{k-1}^+F_{k-1}^T + Q \tag{24}$$

- Correction:

$$G_k = \frac{\partial g}{\partial x}|_{x_k^-} \tag{25}$$
$$K_k = P_k^- G_k^T.(G_k P_k^- G_k^T + R)^{-1} \tag{26}$$
$$\hat{x}_k^+ = x_k^- + K_k(y_k^m - g(x_k^-)) \tag{27}$$
$$P_k^+ = (I - K_k.G_k).P_k^-, \tag{28}$$

where the $-$ and $+$ superscripts respectively denote the predicted and corrected values. The EKF is initialized by setting $\hat{x}_0^+$ and P_0^+ to arbitrary or approximate values. It should be noted that the required Jacobian values F_{k-1} and G_k can be calculated easily by a call to the `linearize()` method through OMMatlab.

6 Optimization Problem

MPC solves the following general optimization problem at the k-th sample time.

$$\min_{r \in U} \quad J_k(X, r) \tag{29}$$
$$\text{s.t.} \quad X_{k+i} = h(X_{k+i-1}, r_{k+i-1}), \quad i = 1, \ldots, N_{H_p} \tag{30}$$
$$M(X, r) \leq 0 \tag{31}$$
$$S(X, r) = 0 \tag{32}$$
$$X_k = \hat{x}_k^+, \tag{33}$$

where J_k is the objective function optimized over the prediction horizon H_p. Note that $N_{H_p} = \frac{H_p}{T_s}$. The decision variable set r is the trajectory of control inputs. Also, in order to reduce the computational cost, the control horizon is set as $H_c < H_p$, and $r_{k+N_{H_c}} = r_{k+N_{H_c}+1} = \cdots = r_{k+N_{H_p}-1}$, with $N_{H_c} = \frac{H_c}{T_s}$. The function h describes the discrete state evolution that is used for prediction. Also, M and S are the inequality and equality constraints, respectively, and X_k is the estimated current state vector used to initialize the prediction model.

The MPC objective can be a tracking index, an economic index, or a combination of the two; see e.g., (Heidarinejad, J. Liu, and Christofides 2013; Ellis, J. Liu,

DOI 10.3384/ecp204293

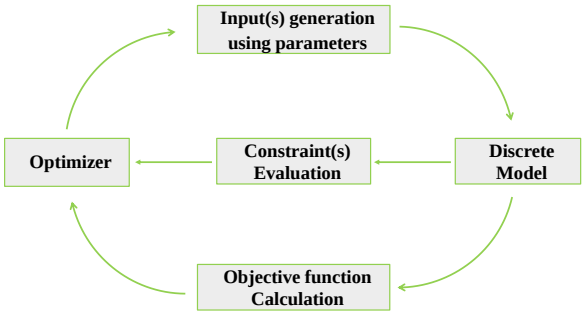

Figure 2. Sequential dynamic optimization method.

and Christofides 2017). In this work, the dynamic optimization problem is solved using the sequential method (Chachuat 2009), where the control input trajectories are parameterized (usually in a piece-wise constant manner), and the dynamic model and the nonlinear optimization problem are solved in sequence as shown in Figure 2. This work uses the active-set optimization algorithm to solve the nonlinear optimization problem.

7 Case Study

A tracking SLMPC is implemented for a system of three interconnected cylindrical tanks as depicted in Figure 3. The flow rates of the three inlet streams are considered the control inputs. The outlet flow rates of the tanks are the measured outputs, and the tanks' liquid levels are regarded as the system states being estimated by the EKF. The tanks are empty at the initial time, and the control scenario is to move the tank levels to a specific set point by adjusting the inlet flow rates. The continuous dynamic model describing the system is as follows.

$$\dot{h_1} = \frac{q_{in1} - q_{o1}}{A_1} \tag{34}$$

$$\dot{h_2} = \frac{q_{o1} + q_{in2} - q_{o2}}{A_2} \tag{35}$$

$$\dot{h_3} = \frac{q_{o2} + q_{in3} - q_{o3}}{A_3} \tag{36}$$

$$q_{o1} = C_v\sqrt{h_1 - h_2} \tag{37}$$

$$q_{o2} = C_v\sqrt{h_2 - h_3} \tag{38}$$

$$q_{o3} = C_v\sqrt{h_3} \tag{39}$$

where A_1, A_2, and A_3 are the vessel cross-section areas, C_v is the valve coefficient; q_{in1}, q_{in2}, and q_{in3} are the volumetric flow rates of the inlet streams, and q_{o1}, q_{o2}, and q_{o3} are the volumetric flow rates of the outlet streams. In case of reverse flow through the valves, the square root term in the valve equations becomes negative, leading to a complex number and solver failure. To avoid this situation and preserve local Lipschitz continuity, the valve equations are regularized as (Barton, Banga, and Galán 2000; Sahlodin 2022; Casella 1998)

$$q_{o1} = C_v\frac{h_1 - h_2}{\sqrt{|h_1 - h_2| + \varepsilon}} \tag{40}$$

$$q_{o2} = C_v\frac{h_2 - h_3}{\sqrt{|h_2 - h_3| + \varepsilon}} \tag{41}$$

$$q_{o3} = C_v\frac{h_3}{\sqrt{|h_3| + \varepsilon}}, \tag{42}$$

where $\varepsilon > 0$. Let $A_1 = A_2 = A_3 = 1m^2$ and $C_v = 0.5$. The model nonlinearity comes from the outlet stream equations. Also, the desired set points for the tank levels are $h^{sp} = [0.56, 0.52, 0.36]^T$. For the dynamic optimization given in Equation 29, the following quadratic tracking objective function with control move penalization is defined.

$$J_k = \sum_{i=1}^{N_{Hp}} (h_{k+i} - h^{sp}_{k+i})^T.W_1.(h_{k+i} - h^{sp}_{k+i}) + \Delta r^T_{k+i-1}.W_2.\Delta r_{k+i-1} \tag{43}$$

The weighting factors W_1 and W_2 are positive-definite matrices set as

$$W_1 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad W_2 = \begin{bmatrix} 18 & 0 & 0 \\ 0 & 18 & 0 \\ 0 & 0 & 18 \end{bmatrix}$$

Also, the following path constraints are added to avoid reverse flow between the tanks.

$$\sum_{i=1}^{N_{Hp}} max(0, -q_{o1}(k+i)) \leq \delta \tag{44}$$

$$\sum_{i=1}^{N_{Hp}} max(0, -q_{o2}(k+i)) \leq \delta, \tag{45}$$

where $\delta > 0$ is a small regularization parameter (Chachuat 2009). The system is simulated for a time span of 50 minutes with sampling time of 6 seconds. The prediction and the control horizons are set to $H_p = 1.5$ and $H_c = 1$ min, respectively. The control inputs are bounded as $0 \leq q_{in1},\ q_{in2},\ q_{in3} \leq 0.3$. Note that a discretized linear model is applied as the prediction model is updated at each sample time.

The results of the SLMPC are presented in the sequel. Figure 4 shows trajectories of the measured outlet flow rates that are used to estimate the tank levels.

Figure 5 depicts the actual and estimated trajectories of the tank levels. The initial guess for the state estimator is $\hat{h}_0^+ = [0.1,\ 0.1,\ 0.1]^T$, which is different than the actual values $h_0 = [0,\ 0,\ 0]^T$. Despite this difference, the EKF is able to track the actual state trajectories successfully. It is also seen that the actual tank levels approach the set points shortly after the SLMPC is executed. The optimal

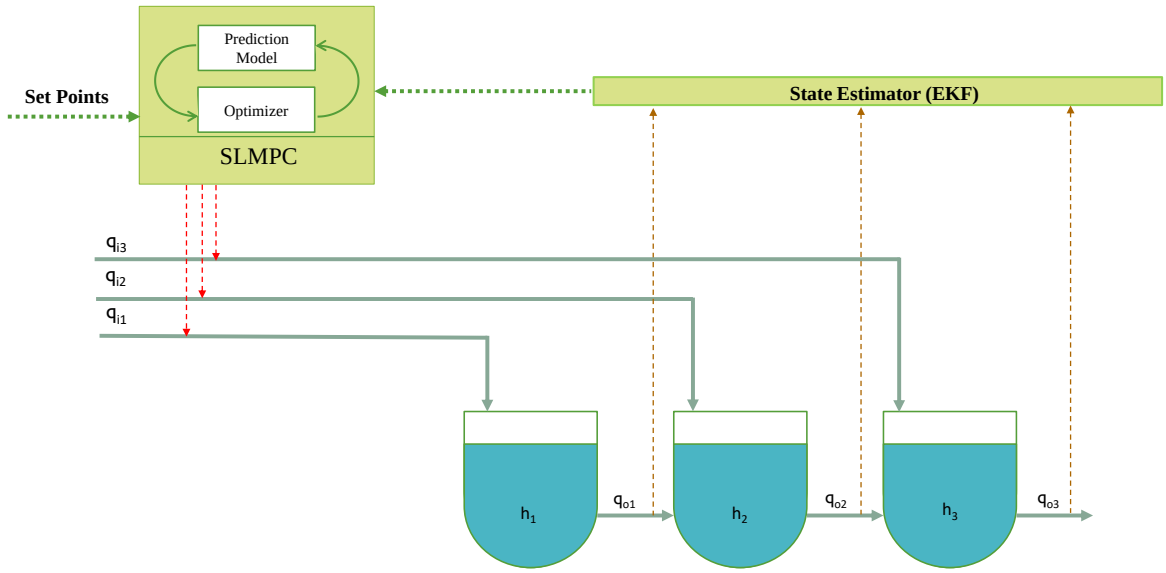

Figure 3. Schematic of the system with the SLMPC and estimator blocks.

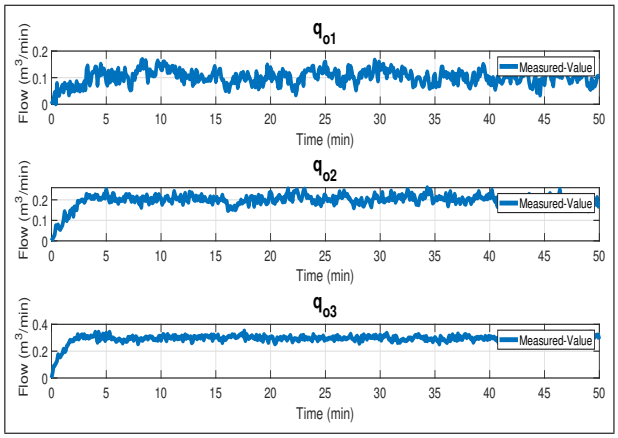

Figure 4. Measured system outputs.

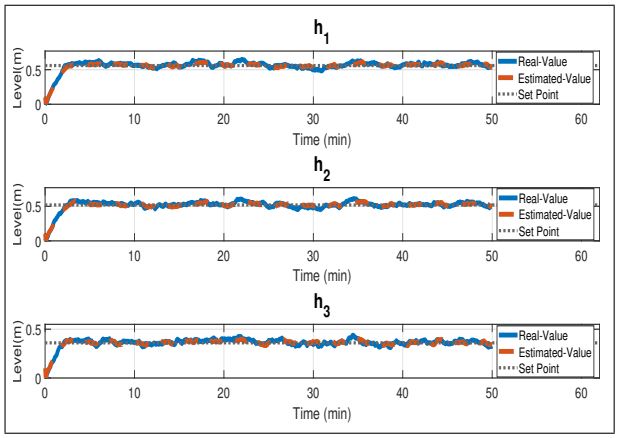

Figure 5. Actual and estimated system states.

control inputs are plotted in Figure 6. It is observed that the control inputs are slightly oscillatory as a result of the process noise.

It is worth noting that the total simulation takes only 25 minutes by the enhanced OMMatlab, while it would take around 7 hours when employing the previous OMMatlab versions. This proves that the linearization part is the main bottleneck in the SLMPC procedure.

The code for the case-study presented in this manuscript can be found on GitHub at `https://github.com/pseAUT/SLMPC_OMMatlab`.

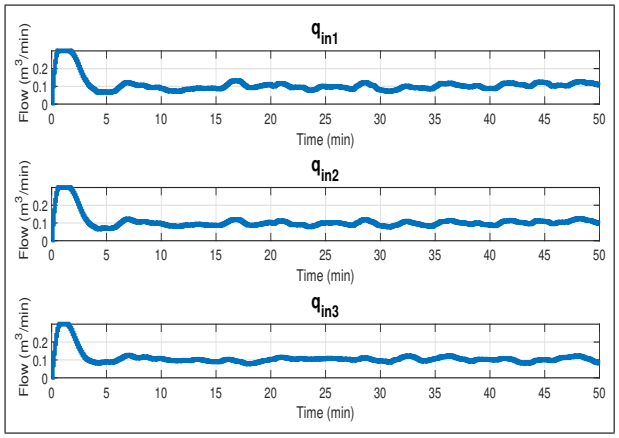

Figure 6. Optimal control inputs.

8 Conclusion

The implementation of the SLMPC algorithm using OpenModelica and MATLAB has been demonstrated in this paper. The OMMatlab API is upgraded to avoid repeated compilation of the OpenModelica model into an executable file at each sample time. In this way, the system can be linearized efficiently and the model matrices can be obtained directly in a `.m` file, thanks to the existing OpenModelica flags that make it possible to generate the linearized model in different formats. Therefore, the successive linearization runtime is optimized consid-

DOI 10.3384/ecp204293

erably. These enhancements can be implemented on other OpenModelica interfaces such as OMOctave, OMPython, and OMJulia to facilitate fast prototyping of control algorithms that require successive linearization.

References

Åkesson, Johan (2008). "Optimica—an extension of modelica supporting dynamic optimization". In: *In 6th International Modelica Conference*. Citeseer, pp. 57–66.

Andersson, Christian et al. (2018). *JModelica.org User Guide*. English. Version Version 2.2. Modelon AB.

Bachmann, Bernhard et al. (2012). "Parallel multiple-shooting and collocation optimization with openmodelica". In: *Proceedings of the 9th International MODELICA Conference; September 3-5; 2012; Munich; Germany*. 076. Linköping University Electronic Press, pp. 659–668.

Barton, P.I., J.R. Banga, and S. Galán (2000). "Optimization of hybrid discrete/continuous dynamic systems". In: *Comput. Chem. Eng* 24.9, pp. 2171–2182. ISSN: 0098-1354. DOI: https://doi.org/10.1016/S0098-1354(00)00586-X. URL: http://www.sciencedirect.com/science/article/pii/S009813540000586X.

Braun, Willi, Lennart Ochel, and Bernhard Bachmann (2011). "Symbolically derived Jacobians using automatic differentiation-enhancement of the OpenModelica compiler". In: *Proceedings of the 8th International Modelica Conference; March 20th-22nd; Technical Univeristy; Dresden; Germany*. 063. Linköping University Electronic Press, pp. 495–501.

Brembeck, Jonathan (2019). "A Physical Model-Based Observer Framework for Nonlinear Constrained State Estimation Applied to Battery State Estimation". In: *Sensors* 19.20. ISSN: 1424-8220. DOI: 10.3390/s19204402. URL: https://www.mdpi.com/1424-8220/19/20/4402.

Brembeck, Jonathan, Martin Otter, and Dirk Zimmer (2011). "Nonlinear observers based on the functional mockup interface with applications to electric vehicles". In: *Proceedings of the 8th International Modelica Conference; March 20th-22nd; Technical Univeristy; Dresden; Germany*. 63. Linköping University Electronic Press, pp. 474–483.

Cannon, Mark, Desmond Ng, and Basil Kouvaritakis (2009). "Successive linearization NMPC for a class of stochastic nonlinear systems". In: *Nonlinear model predictive control: towards new challenging applications*, pp. 249–262.

Casella, Francesco (1998). "Modeling, simulation and control of a geothermal power plant". PhD thesis. Politecnico di Milano, Italy, p. 72.

Chachuat, B.C. (2009). *Nonlinear and Dynamic Optimization: From Theory to Practice - IC-32: Spring Term 2009*. Polycopiés de l'EPFL. EPFL. URL: http://books.google.com/books?id=%5C_JOHYgEACAAJ.

Cortinovis, Andrea et al. (2014). "Safe and efficient operation of centrifugal compressors using linearized MPC". In: *53rd IEEE Conference on Decision and Control*. IEEE, pp. 3982–3987.

Ellis, Matthew, Jinfeng Liu, and Panagiotis D. Christofides (2017). "Two-Layer EMPC Systems". In: *Economic Model Predictive Control: Theory, Formulations and Chemical Process Applications*. Cham: Springer International Publishing, pp. 171–232. ISBN: 978-3-319-41108-8. DOI: 10.1007/978-3-319-41108-8_6. URL: https://doi.org/10.1007/978-3-319-41108-8_6.

Franke, Rüdiger (2002). "Formulation of dynamic optimization problems using Modelica and their efficient solution". In: *Proceedings 2nd International Modelica Conference*, pp. 315–323.

Fritzson, Peter et al. (2020). "The OpenModelica Integrated Environment for Modeling, Simulation, and Model-Based Development". In: *Modeling, Identification and Control* 41.4, pp. 241–295. DOI: 10.4173/mic.2020.4.1.

Gräber, Manuel et al. (2012). "Using functional mock-up units for nonlinear model predictive control". In: *Proceedings of the 9th International MODELICA Conference; September 3-5; 2012; Munich; Germany*. 076. Linköping University Electronic Press, pp. 781–790.

Heidarinejad, Mohsen, Jinfeng Liu, and Panagiotis D. Christofides (2013). "Algorithms for improved fixed-time performance of Lyapunov-based economic model predictive control of nonlinear systems". In: *Journal of Process Control* 23.3, pp. 404–414. ISSN: 0959-1524. DOI: https://doi.org/10.1016/j.jprocont.2012.11.003. URL: https://www.sciencedirect.com/science/article/pii/S0959152412002545.

Imsland, L., P. Kittilsen, and T.S. Schei (2010). "odel-Based Optimizing Control and Estimation Using Modelica Model". In: *Modeling, Identification and Control: A Norwegian Research Bulletin* 31.3, pp. 107–121. DOI: 10.4173/mic.2010.3.3. URL: https://doi.org/10.4173/mic.2010.3.3.

Imsland, Lars, Pål Kittilsen, and Tor Steinar Schei (2009). "Using modelica models in real time dynamic optimization–gradient computation". In: *Proc. of Modelica*.

Jorissen, F., W. Boydens, and L. Helsen (2019). "TACO, an automated toolchain for model predictive control of building systems: implementation and verification". In: *Journal of Building Performance Simulation* 12.2, pp. 180–192. eprint: https://doi.org/10.1080/19401493.2018.1498537. URL: https://doi.org/10.1080/19401493.2018.1498537.

Lie, Bernt et al. (2019). "Omjulia: An openmodelica api for julia-modelica interaction". In: *Proceedings of the 13th International Modelica Conference, Regensburg, Germany, March 4–6, 2019*. 157. Linköping University Electronic Press.

Liu, Changchun et al. (2015). "Stochastic predictive control for lane keeping assistance systems using a linear time-varying model". In: *2015 American Control Conference (ACC)*. IEEE, pp. 3355–3360.

Magnusson, Fredrik and Johan Åkesson (2015). "Dynamic Optimization in JModelica.org". In: *Processes* 3.2, pp. 471–496. ISSN: 2227-9717. DOI: 10.3390/pr3020471. URL: https://www.mdpi.com/2227-9717/3/2/471.

OpenModelica (2021). https://openmodelica.org/doc/OpenModelicaUsersGuide/1.16. Accessed: 2023-07-23.

Pandey, Madhusudhan et al. (2021a). "Formulation of Stochastic MPC to Balance Intermittent Solar Power with Hydro Power in Microgrid". In: *The First SIMS EUROSIM Conference on Modelling and Simulation, SIMS EUROSIM 2021*.

Pandey, Madhusudhan et al. (2021b). "Using MPC to Balance Intermittent Wind and Solar Power with Hydro Power in Microgrids". In: *Energies* 14.4. ISSN: 1996-1073. DOI: 10.3390/en14040874. URL: https://www.mdpi.com/1996-1073/14/4/874.

Pippia, Tomas et al. (2021). "Scenario-based nonlinear model predictive control for building heating systems". In: *Energy and Buildings* 247, p. 111108. ISSN: 0378-7788. DOI: https://doi.org/10.1016/j.enbuild.2021.111108. URL: https://www.sciencedirect.com/science/article/pii/S0378778821003923.

Romero, Alberto, Alejandro Goldar, and Emanuele Garone (2019). "A Model Predictive Control Application for a Constrained Fast Charge of Lithium-ion Batteries". In: *International Modelica Conference*.

Ruge, Vitalij et al. (2014). "Efficient implementation of collocation methods for optimization using openmodelica and ADOL-C". In: *Proceedings of the 10 th International Modelica Conference; March 10-12; 2014; Lund; Sweden*. 096. Linköping University Electronic Press, pp. 1017–1025.

Sahlodin, Ali M (2022). "Optimally safe tank changeover operation using a smooth optimization formulation". In: *ACS omega* 7.39, pp. 34974–34989.

Seki, Hiroya, Satoshi Ooyama, and Morimasa Ogawa (2002). "Nonlinear Model Predictive Control Using Successive Linearization Application to Chemical Reactors". In: *Transactions of the society of instrument and control engineers* 38.1, pp. 61–66.

Vrlić, Martin, Daniel Ritzberger, and Stefan Jakubek (2020). "Safe and Efficient Polymer Electrolyte Membrane Fuel Cell Control Using Successive Linearization Based Model Predictive Control Validated on Real Vehicle Data". In: *Energies* 13.20. ISSN: 1996-1073. DOI: 10.3390/en13205353. URL: https://www.mdpi.com/1996-1073/13/20/5353.

Zhakatayev, Altay et al. (2017). "Successive linearization based model predictive control of variable stiffness actuated robots". In: *2017 IEEE International Conference on Advanced Intelligent Mechatronics (AIM)*, pp. 1774–1779. DOI: 10.1109/AIM.2017.8014275.