Model Based Automatic Code Generation for Nonlinear Model Predictive Control

Behzad Samadi(✉)

Maplesoft, Waterloo, ON, Canada
bsamadi@maplesoft.com

Abstract. This paper demonstrates a symbolic tool that generates C code for nonlinear model predictive controllers. The optimality conditions are derived in a quick tutorial on optimal control. A model based workflow using MapleSim for modeling and simulation, and Maple for analysis and code generation is then explained. In this paper, we assume to have a control model of a nonlinear plant in MapleSim. The first step of the workflow is to get the equations of the control model from MapleSim. These equations are usually in the form of differential algebraic equations. After converting the equations to ordinary differential equations, the C code for the model predictive controller is generated using a tool created in Maple. The resulting C code can be used to simulate the control algorithm and program the hardware controller. The proposed tool for automatic code generation for model predictive controllers is open and can be employed by users to create their own customized code generation tool.

Keywords: Model predictive control · Code generation · Model based

1 Introduction

A systematic workflow for model based automatic code generation for Model Predictive Control (MPC) is introduced in this paper (Fig. 1). MPC started as the alternative advanced control method to PID controllers in chemical process industries. With the progress of MPC algorithms and the computational power of hardware controllers, other industries such as the robotics, automotive and aerospace industries are looking into MPC controllers. The majority of current MPC controllers use linear models because in that case, the controller needs to solve a Quadratic Programming (QP), which is a convex optimization problem that can be solved efficiently. However, linear models are not adequate for highly nonlinear systems. MPC controller synthesis based on physical modeling is the approach that can be employed to overcome the inefficiency of linear models. Working with physical models is on the other hand not easy. Control engineers need high-level modeling tools to save time, improve reliability and prevent human error getting into the controller code.

MapleSim [1], as an advanced high-level tool for physical modeling and simulation, and Maple [2], as an advanced symbolic analysis and synthesis tool, are

S. Bogomolov et al. (Eds.): NSV 2016, LNCS 10152, pp. 85–95, 2017.
DOI: 10.1007/978-3-319-54292-8_8

natural building blocks of the proposed workflow for automatic code generation for MPC. The workflow, depicted in Fig. 1, consists of the following steps:

- *Modeling:* A physical model of the system is built by connecting components in MapleSim. MapleSim contains more than 550 built-in components from many different domains. Users can also create their own custom components.
- *Extracting the equations:* The dynamic equations of this model are then extracted into Maple. In general, automatically generated dynamic equations of a physical model are in the form of differential algebraic equations (DAE).
- *Converting DAEs to ODEs:* Most MPC algorithms assume that the equations are given in the form of ordinary differential equations (ODE). Maple is employed in the workflow to convert DAEs to ODEs. The conversion can be exact or an approximation in order to simplify the resulting ODEs. Currently, this step is not systematic and it may be different for each system.
- *Generating Code:* The MPC algorithm for the given the ODE equations can be generated using Maple's symbolic computation engine. The resulting Maple code can be converted to C code automatically using the CodeGeneration package.
- *Simulating the Closed Loop System:* The generated C code can be embedded in a MapleSim model as an External C Block. The closed loop system with the original DAE equations can then be simulated in MapleSim.

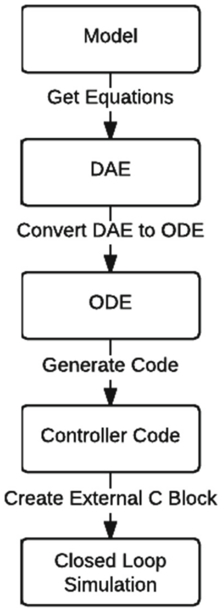

Fig. 1. Model-based automatic code Generation for MPC

An MPC controller consists of two main parts: problem formulation and the solver. Both of these parts can be generated in Maple and then translated to C code. In the Sect. 2, we will explain the approach we have followed to design an MPC controller. Mathematical details of the approach are described in Sect. 3. As an example, a system with electrical, mechanical and hydraulic components is considered in Sect. 4. The importance of this example is that it demonstrates automatic generation of dynamic equations, MPC problem formulation and an MPC solver. Corresponding MapleSim models and Maple worksheets will be provided to interested readers upon request.

2 Model Predictive Control

Consider a driver on a road. For a safe trip, the driver needs to look forward (prediction), follow the road and avoid any collision (constraints). On top of that, the driver might want to spend as little time and fuel as possible to reach the destination (optimization).

MPC can be employed to drive a self-driving car. The MPC controller minimizes a cost function at each time instant. Some of the optimization constraints that should be respected are listed below:

- Vehicle's dynamic behavior
- Limited power
- No skidding
- Following the road
- Avoiding collision

MPC is in fact a control method that computes the control input of the system by solving an Optimal Control Problem (OCP) in real time. The aim of the OCP is to optimize the trajectory of the system with respect to a cost function that is defined based on economic or performance measures. An MPC controller uses a model of the system to predict its behavior within a finite horizon. The following steps describe what an MPC controller performs in real time:

1. Measure/estimate the current state.
2. Solve the OCP to compute a sequence of control inputs.
3. Apply the first step in the control input sequence to the system.
4. Go to step 1.

MPC controllers were originally developed to control chemical processes [3]. PID controllers have been common in process control for a long time. However, there are a few issues with using PID controllers:

- *Highly nonlinear systems:* PID controllers do not perform well for highly nonlinear systems.

- *Multiple coupled loops:* To control a multi-input multi-output (MIMO) system, many PID controllers in different loops are required and it is hard to compensate the coupling effect.
- *Tuning:* Tuning PID controllers is not an easy task especially when there are limits on the control input.

MPC controllers have the following features:

- MPC can minimize a cost function defined based on economic or performance properties of the system.
- MPC can handle constraints on inputs and states.
- MPC can be designed for MIMO nonlinear systems.
- Tuning an MPC controller is more intuitive compared to a set of PID controllers.
- MPC is a systematic, model based approach.
- There are tools for generating code for MPC controllers automatically.

2.1 Applications

MPC was first employed to control chemical processes in 1970s. In a survey of industrial applications of MPC, reference [3] mentions thousands of applications in refining, petrochemicals, chemicals, pulp and paper, polymer and food processing industries. Chemical processes are usually very slow and expensive. Therefore, there is plenty of time and computational power available to the controller. The largest size for plants reported in [3] is 603 inputs and 283 outputs. The majority of the MPC controllers in process control use linear models for the plant. The main reason is that the OCP in an MPC controller for a linear system with constraints can be formulated as a Quadratic Programming (QP) problem. Even large QP problems can be solved efficiently.

For nonlinear systems, the OCP is a nonlinear optimization problem, which are hard to solve in general. In [4], it is stated that, in process control, the use of linear MPC decreases with increasing process nonlinearity. In recent years, computational power of microprocessors has increased and new algorithms have been proposed for implementing MPC controllers. The result has been very fast implementations of MPC and the increasing interest in robotics, automotive and aerospace industries to use MPC controllers.

3 Mathematical Description

Consider the following nonlinear system:

$$\begin{aligned} \dot{x}(t) =& f(x(t), u(t)) \\ x(t_0) =& x_\circ \end{aligned}$$

where:

- $x(t)$ is the state vector
- $u(t)$ is the input vector

The objective of the MPC controller is to solve the following OCP:

$$\begin{aligned}
\underset{u}{\text{minimize}}\ & J(x_0, t_0) = \phi(x(t_f)) + \int_{t_0}^{t_f} L(x(\tau), u(\tau))d\tau \\
\text{subject to}\ & \dot{x}(t) = f(x(t), u(t)) \\
& x(t_0) = x_\circ \\
& g_i(x(t), u(t)) = 0,\ \text{for}\ \ i = 1, \dots, n_g \\
& h_i(x(t), u(t)) \leq 0,\ \text{for}\ \ i = 1, \dots, n_h
\end{aligned}$$

This OCP is in general a nonconvex optimization problem and it is hard to solve. A particular *interior-point algorithm*, the *barrier method* [5], is employed here to convert the inequality constraints to equality constraints [6]:

$$\begin{aligned}
\underset{u,\sigma}{\text{minimize}}\ & \phi(x(t_f)) + \int_{t_0}^{t_f} \left(L(x(\tau), u(\tau)) - r^{\mathrm{T}}\sigma(\tau)\right) d\tau \\
\text{subject to}\ & \dot{x}(t) = f(x(t), u(t)) \\
& x(t_0) = x_\circ \\
& g_i(x(t), u(t)) = 0,\ \text{for}\ \ i = 1, \dots, n_g \\
& h_i(x(t), u(t)) + \sigma_i(t)^2 = 0,\ \text{for}\ \ i = 1, \dots, n_h
\end{aligned}$$

where $\sigma(t) \in \mathbb{R}^{n_h}$ is a vector slack variable and the entries of $r \in \mathbb{R}^{n_h}$ are small positive numbers. The vector r is chosen by the user. To put more penalty on the solution when it gets close to $h_i(x(t), u(t)) = 0$, the user should increase the corresponding entry of r.

If the problem is discretized into N steps from t_0 to t_f, we have:

$$\begin{aligned}
\underset{u,\sigma}{\text{minimize}}\ & \phi_d(x_N) + \sum_{k=0}^{N-1} \left(L(x_k, u_k) - r^{\mathrm{T}}\sigma_k\right) \\
\text{subject to}\ & x_{k+1} = f_d(x_k, u_k) \\
& x_0 = x_\circ \\
& g_i(x_k, u_k) = 0,\ \text{for}\ \ i = 1, \dots, n_g \\
& h_i(x_k, u_k) + \sigma_{ik}^2 = 0,\ \text{for}\ \ i = 1, \dots, n_h
\end{aligned}$$

where $\Delta\tau = \frac{t_f - t_0}{N}$ and:

$$\begin{aligned}
\phi_d(x_N) &= \frac{\phi(x(t_f), t_f)}{\Delta\tau} \\
f_d(x_k, u_k) &= x_k + f(x_k, u_k)\Delta\tau
\end{aligned}$$

By putting all the equality constraints into a vector G, we have:

$$\begin{aligned}\underset{u,\sigma}{\text{minimize}}\ & \phi_d(x_N) + \sum_{k=0}^{N-1} \big(L(x_k,u_k) - r^{\mathrm{T}}\sigma_k\big)\\ \text{subject to}\ & x_{k+1} = f_d(x_k,u_k)\\ & x_0 = x_\circ\\ & G(x_k,u_k,\sigma_k) = 0\end{aligned}$$

where:

$$G(x_k,u_k,\sigma_k) = \begin{bmatrix} g_1(x_k,u_k)\\ \vdots\\ g_{n_g}(x_k,u_k)\\ h_1(x_k,u_k)+\sigma_{1k}^2\\ \vdots\\ h_{n_h}(x_k,u_k)+\sigma_{n_hk}^2\end{bmatrix}$$

To solve an optimization problem with equality constraints, we can use Lagrange multipliers. The Lagrangian is defined as:

$$\begin{aligned}\mathcal{L}(x,u,\sigma,\lambda,\mu) = {} & \phi_d(x_N,N) + (x_\circ - x_0)^{\mathrm{T}}\lambda_0\\ & + \sum_{k=0}^{N-1} \big(L(x_k,u_k) - r^{\mathrm{T}}\sigma_k\\ & + (f_d(x_k,u_k) - x_{k+1})^{\mathrm{T}}\lambda_{k+1}\\ & + G(x_k,u_k,\sigma_k)^{\mathrm{T}}\mu_k\big)\end{aligned}$$

Necessary conditions for a point to be an extremum is the following partial derivatives to be zero:

$$\begin{aligned}\frac{\partial\mathcal{L}}{\partial x_k} &= 0\\ \frac{\partial\mathcal{L}}{\partial \lambda_k} &= 0 \qquad\qquad \text{for } k = 0,\ldots,N\end{aligned}$$

$$\begin{aligned}\frac{\partial\mathcal{L}}{\partial \sigma_k} &= 0\\ \frac{\partial\mathcal{L}}{\partial u_k} &= 0\\ \frac{\partial\mathcal{L}}{\partial \mu_k} &= 0 \qquad\qquad \text{for } k = 0,\ldots,N-1\end{aligned}$$

The Lagrangian $\mathcal{L}$ can be rewritten as:

$$\begin{aligned}\mathcal{L}(x,u,\sigma,\lambda,\mu) = &\,\phi_d(x_N) + x_\circ^{\mathrm{T}}\lambda_0 - x_N^{\mathrm{T}}\lambda_N \\ &+ \sum_{k=0}^{N-1}\left(\mathcal{H}(x_k,u_k,\sigma_k,\lambda_{k+1},\mu_k) - x_k^{\mathrm{T}}\lambda_k\right)\end{aligned}$$

where the Hamiltonian is defined as:

$$\begin{aligned}\mathcal{H}(x_k,u_k,\sigma_k,\lambda_{k+1},\mu_k) = &\,L(x_k,u_k) - r^{\mathrm{T}}\sigma_k \\ &+ f_d(x_k,u_k)^{\mathrm{T}}\lambda_{k+1} + G(x_k,u_k,\sigma_k)^{\mathrm{T}}\mu_k\end{aligned}$$

Now, the optimality conditions can be summarized as in the following table. These conditions are known as Pontryagin's Maximum Principle.

Partial derivatives	Optimality conditions
$\frac{\partial\mathcal{L}}{\partial\lambda_{k+1}} = 0$	$x_{k+1}^\star = f_d(x_k^\star, u_k^\star)$
$\frac{\partial\mathcal{L}}{\partial\lambda_0} = 0$	$x_0^\star = x_\circ$
$\frac{\partial\mathcal{L}}{\partial x_k} = 0$	$\lambda_k^\star = \mathcal{H}_x(x_k^\star, u_k^\star, \sigma_k^\star, \lambda_{k+1}^\star, \mu_k^\star)$
$\frac{\partial\mathcal{L}}{\partial x_N} = 0$	$\lambda_N^\star = \frac{\partial}{\partial x_N}\phi_d(x_N^\star)$
$\frac{\partial\mathcal{L}}{\partial u_k} = 0$	$\mathcal{H}_u(x_k^\star, u_k^\star, \sigma_k^\star, \lambda_{k+1}^\star, \mu_k^\star) = 0$
$\frac{\partial\mathcal{L}}{\partial\sigma_k} = 0$	$\mathcal{H}_\sigma(x_k^\star, u_k^\star, \sigma_k^\star, \lambda_{k+1}^\star, \mu_k^\star) = 0$
$\frac{\partial\mathcal{L}}{\partial\mu_k} = 0$	$G(x_k^\star, u_k^\star, \sigma_k^\star) = 0$

For $k = 0, \ldots, N-1$ where $\star$ denotes the optimal solution and $\mathcal{H}_u = \frac{\partial\mathcal{H}}{\partial u_k}$ and $\mathcal{H}_\sigma = \frac{\partial\mathcal{H}}{\partial\sigma_k}$. In the next section, we introduce an algorithm to find a solution of the optimality conditions.

3.1 Continuation GMRES Method

Following [7], optimality conditions can be written as the equation $F(x_n, U) = 0$ where

$$F(x_n,U) = \begin{bmatrix} \mathcal{H}_u(x_0,u_0,\sigma_0,\lambda_1,\mu_0) \\ \mathcal{H}_\sigma(x_0,u_0,\sigma_0,\lambda_1,\mu_0) \\ G(x_0,u_0,\sigma_0) \\ \vdots \\ \mathcal{H}_u(x_{N-1},u_{N-1},\sigma_{N-1},\lambda_N,\mu_{N-1}) \\ \mathcal{H}_\sigma(x_{N-1},u_{N-1},\sigma_{N-1},\lambda_N,\mu_{N-1}) \\ G(x_{N-1},u_{N-1},\sigma_{N-1}) \end{bmatrix}$$

and:

$$U = [u_0^{\mathrm{T}}, \ldots, u_{N-1}^{\mathrm{T}}, \sigma_0^{\mathrm{T}}, \ldots, \sigma_{N-1}^{\mathrm{T}}, \mu_0^{\mathrm{T}}, \ldots, \mu_{N-1}^{\mathrm{T}}]^{\mathrm{T}}$$

Dynamics equations of the states and costates (Lagrange multipliers corresponding to state equations) are given as:

$$\begin{aligned} x_{k+1} &= f_d(x_k, u_k) \\ x_0 &= x_n \\ \lambda_k &= \mathcal{H}_x(x_k, u_k, \sigma_k, \lambda_{k+1}, \mu_k) \\ \lambda_N &= \frac{\partial}{\partial x_N}\phi_d(x_N) \end{aligned}$$

Using the *Continuation method*, instead of solving $F(x, U) = 0$, we find U such that:

$$\dot{F}(x, U) = A_s F(x, U)$$

where A_s is a matrix with negative eigenvalues. Now, we have:

$$F_x \dot{x} + F_U \dot{U} = A_s F(x, U)$$

To compute $\dot{U}$ using the following equation, which is linear in $\dot{U}$, we use the generalized minimum residual (*GMRES*) algorithm.

$$F_U \dot{U} = A_s F(x, U) - F_x f(x, u)$$

To compute U at any given time, we need to have an initial value for U and then use the above $\dot{U}$ to update it.

4 Example

In this section, we will apply the proposed workflow for MPC controller design to an Electro Hydraulic Servo System (EHSS). The EHSS system, depicted in Fig. 2, consists of mechanical system with two arms. The pressure inside a hydraulic cylinder creates the force to move the arms up and down. The model of this system is built in MapleSim by connecting components from Multibody and Hydraulics libraries (Fig. 3). The implementation of the EHSS model in MapleSim, generating code for the MPC controller in Maple and simulating the MPC controller in MapleSim are available upon request.

In a Maple worksheet, equations of motion of the EHSS system are extracted from the MapleSim model. For this example, the extracted equations represent an implicit ODE. In order to convert the equations into an explicit ODE, we will assume that the input of the system (the input pressure to the hydraulic circuit) and the derivative variables (state variables) are given. Then, we need to create a procedure to compute the derivatives of the state variables. In this particular case, the implicit ODE contains piecewise expressions that lead to a fairly complicated piecewise explicit ODE. To simplify the resulting ODE, a few of the expressions are approximated with rational polynomial functions.

Once the explicit ODE is computed, the MPC algorithm can be generated in Maple. One of the advantages of a symbolic computation engine such as Maple

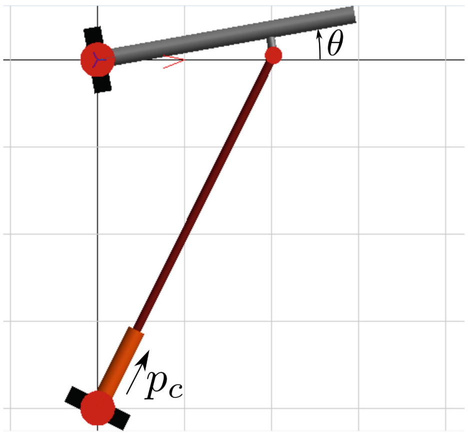

Fig. 2. EHSS system

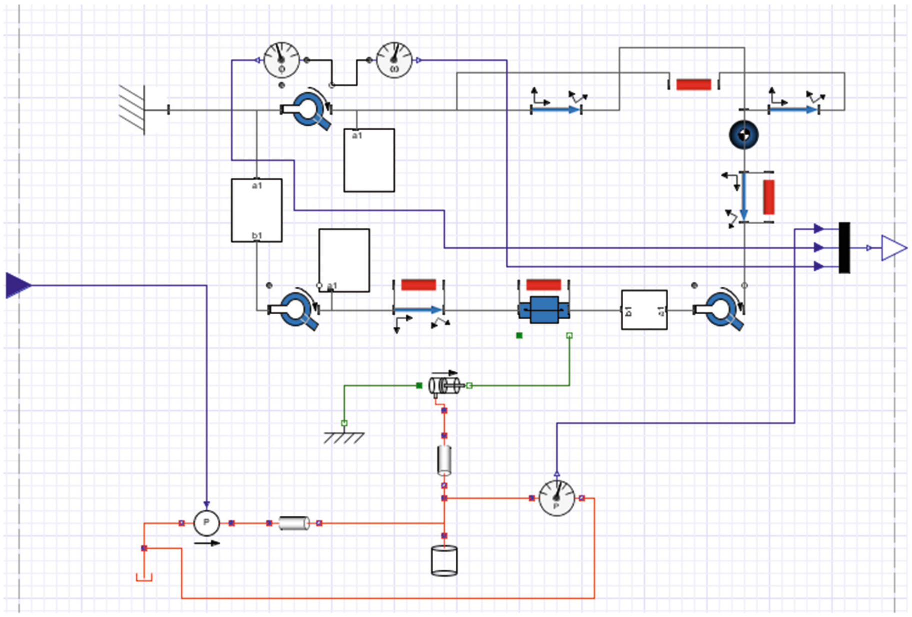

Fig. 3. EHSS MapleSim model

is that it can work on procedures as opposed to numbers. For example, by applying the codegen:-optimize command in Maple to a procedure, you can create a new procedure that computes the same result as the original procedure but it performs fewer operations. This optimization is done by recognizing common subexpressions that appear in the equations several times and computing them as intermediate variables. The intermediate variables are then only used, not recomputed, in the rest of the equations. In the MPC workflow, a procedure is generated that computes the optimality conditions. To that end, partial derivatives of the Hamiltonian with respect to states, inputs, Lagrange multipliers and slack variables are needed. Also, depending on the solver, we might need the derivatives of the optimality conditions. Depending of the dynamic equations of the system, the symbolic representation of the optimality conditions and their derivatives might become very long expressions. To avoid working with complicated expressions, we create Maple procedures that compute the optimality conditions. The partial derivatives of a procedure with respect to its input arguments can be computed using the codegen:-JACOBIAN command. The result is returned as a procedure that computes the corresponding derivative. The main advantage of using Maple in the workflow is that the procedures and their derivatives are computed automatically based on the original ODE equation. Users of numerical tools have to write several procedures by hand to implement an MPC controller. Manually writing procedures takes time and is always prone to human errors. With Maple, the workflow is automatic and the result is an optimized procedure. The MPC algorithm can also be translated into C code using the CodeGeneration package in Maple.

The generated C code for the MPC controller can then be embedded into the MapleSim model of the EHSS as an External C Block to simulate the closed loop system. Figure 4 shows the result of the closed simulation of the EHSS system with the MPC controller. The value of θ (Fig. 2) is shown in solid red and its desired value is shown in dashed green. Although, in the design of the MPC controller, we assumed that the setpoint was constant, the MPC controller has acceptable tracking performance.

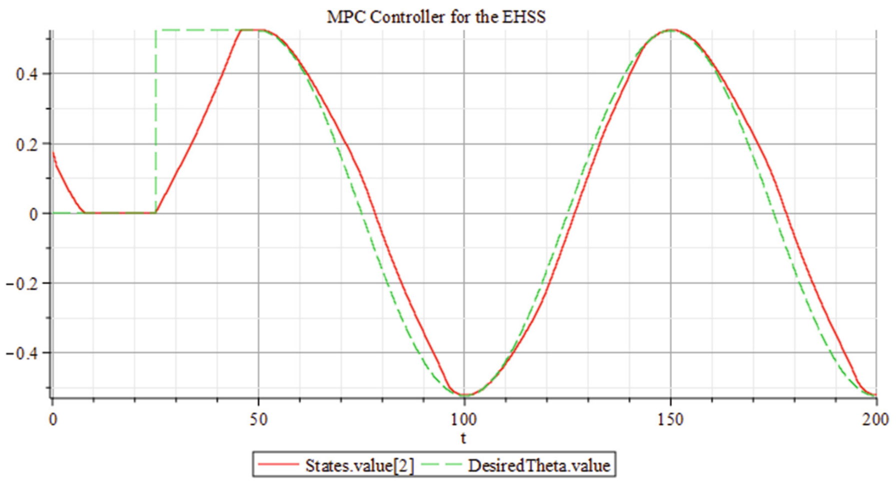

Fig. 4. Closed loop simulation

5 Summary

In this paper, we reviewed a workflow for implementing MPC controllers. MapleSim, a high-level modeling tool, and Maple, a high-level and symbolic computation engine, were employed to create an automatic workflow from a high-level model of the system to the C code of the MPC controller. The MPC problem formulation and solver were generated in Maple and then converted to C code. The main advantage of using a symbolic tool in the workflow is that the required procedures and their corresponding derivatives are computed and optimized automatically. This workflow saves time and it is less sensitive to human errors.

References

1. Maplesoft, a division of Waterloo Maple Inc.: MapleSim (2016)
2. Maplesoft, a division of Waterloo Maple Inc.: Maple (2016)
3. Qin, S.J., Badgwell, T.A.: A survey of industrial model predictive control technology. Control Eng. Pract. **11**, 733–764 (2003)
4. Qin S.J., Badgwell T.A.: An overview of nonlinear model predictive control applications. In: Allgöwer F., Zheng A. (eds.) Nonlinear Model Predictive Control. Progress in Systems and Control Theory. Springer, Switzerland (2000)
5. Boyd, S., Vandenberghe, L.: Convex Optimization. Cambridge University Press, Cambridge (2004)
6. Diehl, M., Ferreau, H.J., Haverbeke, N.: Efficient numerical methods for nonlinear MPC and moving horizon estimation. In: Magni, L., et al. (eds.) Nonlinear Model Predictive Control. LNCIS, vol. 384, pp. 391–417. Springer, Heidelberg (2009)
7. Ohtsuka, T.: A continuation/GMRES method for fast computation of nonlinear receding horizon control. Automatica **40**, 563–574 (2004)